PLD+: Accelerating LLM Inference by Leveraging Language Model Artifacts

Shwetha Somasundaram **Anirudh Phukan** **Apoorv Saxena**
Adobe Research, India
{shsomasu, phukan, apoorvs}@adobe.com

Abstract

To reduce the latency associated with autoretrogressive LLM inference, speculative decoding has emerged as a novel decoding paradigm, where future tokens are drafted and verified in parallel. However, the practical deployment of speculative decoding is hindered by its requirements for additional computational resources and fine-tuning, which limits its out-of-the-box usability. To address these challenges, we present PLD+, a suite of novel algorithms developed to accelerate the inference process of LLMs, particularly for input-guided tasks. These tasks, which include code editing, text editing, summarization, etc., often feature outputs with substantial overlap with their inputs—an attribute PLD+ is designed to exploit. PLD+ also leverages the artifacts (attention and hidden states) generated during inference to accelerate inference speed. We test our approach on five input-guided tasks and through extensive experiments we find that PLD+ outperforms all tuning-free approaches. In the greedy setting, it even outperforms the state-of-the-art tuning-dependent approach EAGLE on four of the tasks. (by a margin of upto 2.31 in terms of avg. speedup). Our approach is tuning free, does not require any additional compute and can easily be used for accelerating inference of any LLM.

1 Introduction

Large language models have emerged as the foundational building blocks for a wide array of user-facing applications, enabling unprecedented capabilities in natural language processing and generation (Liu et al., 2023). However, the autoregressive decoding approach employed by these large language models introduces significant inference latency, a challenge that becomes more severe as the model size and generation length increase (Xia et al., 2024). This latency can pose a barrier to the integration of these models into interactive applications, underscoring the importance of developing efficient decoding strategies to address this fundamental limitation.

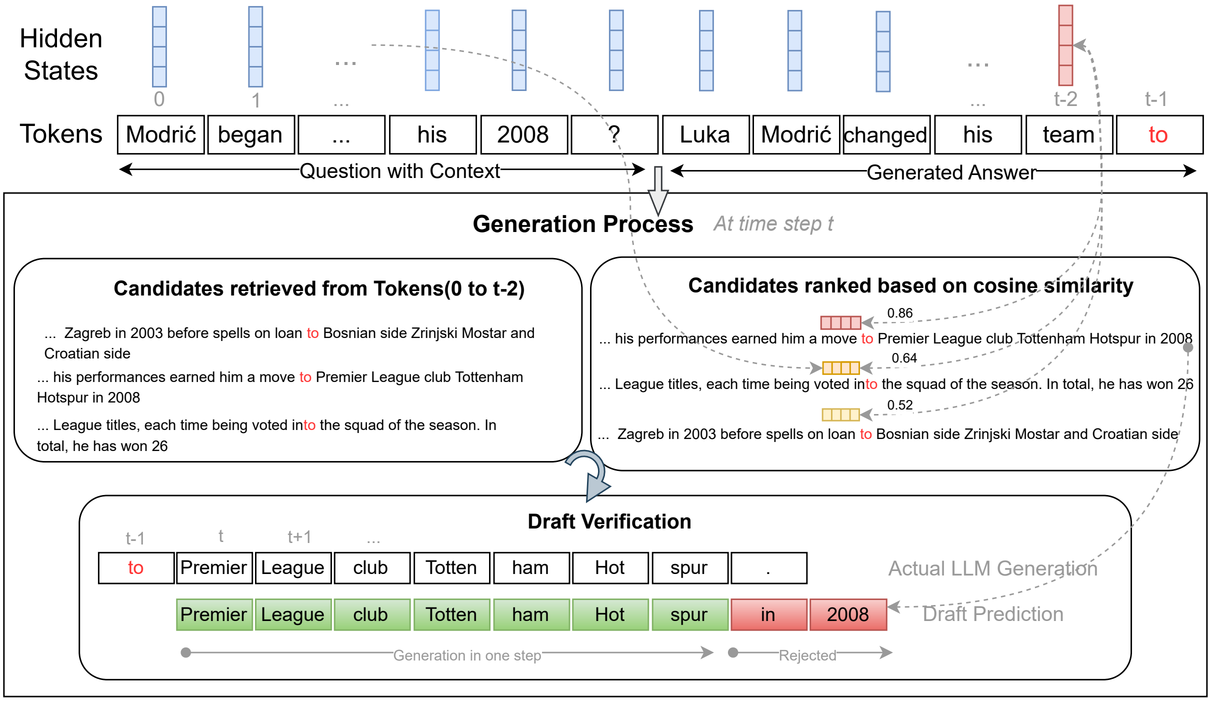
One strategy that has been proposed to mitigate the inference latency challenge faced by large language models is the Speculative Decoding paradigm, which operates based on the Draft and Verify principle to accelerate the inference process (Stern et al., 2018; Leviathan et al., 2023). The two key steps in this paradigm are (a) efficient generation of multiple future tokens in the drafting step and (b) parallel verification of the drafted tokens using the target Language Model to ensure quality and alignment.

Classic drafting strategies usually either employ a smaller independent model to efficiently draft tokens or leverage the target LLM itself, utilizing techniques such as incorporating additional FFN heads (Stern et al., 2018; Cai et al., 2024) or layer skipping (Zhang et al., 2023b). However, these methods often require extensive tuning, which needs to be performed for every new model, and can be time and resource-intensive.

In contrast, the PLD/LLMA approach gets rid of the need for any additional draft model by simply selecting text spans from the input as drafts, thus being extremely simple and demonstrating speedup on "input-guided tasks" (Saxena, 2023; Yang et al., 2023a). Input-guided tasks are those with context-rich prompts, where the output is directly informed by or closely aligned with the input information, such as summarization, retrieval-augmented generation, and code/text editing. The simplicity and effective nature of PLD has resulted in its integration with the `transformers` library, underlining the importance of a plug-and-play method that achieves speedup on "input-guided tasks".

In this paper, we build upon PLD and propose Prompt Lookup Decoding+ (PLD+), which leverages the information present in the model artifacts (attentions and hidden states generated during inference) to generate better drafts. In essence, we

Findings of the Association for Computational Linguistics: NAACL 2025, pages 6090–6104
April 29 - May 4, 2025 ©2025 Association for Computational Linguistics



... Zagreb in 2003 before spells on loan to Bosnian side Zrinjski Mostar and Croatian side

... his performances earned him a move to Premier League club Tottenham Hotspur in 2008

... League titles, each time being voted into the squad of the season. In total, he has won 26

Figure 1: Overview of PLD+. During the generation of token t, possible drafts are retrieved from the context by searching for the same tokens as token $t-1$ ("to"). These candidates are then ranked using the information present in the model artifacts (hidden states in the figure) corresponding to token $t-2$ ("team"). The text following "to" in the highest ranked candidate is proposed as the draft. The draft is verified against the actual LLM generation and all successfully verified tokens (Premier, League, club, Totten, ham, Hot, spur) are generated at once, resulting in speedup.

propose a tuning-free plug-and-play speculative decoding technique that leverages the model artifacts computed during the generation process. We outline the key contributions of our paper below:

- We introduce PLD+, a suite of plug and play tuning-free speculative decoding algorithms that exploit the overlap between the input and the output generations. PLD+ leverages model artifacts to intelligently select draft spans and accelerate LLM inference.

- The strengths of our method are that it does not require any finetuning, it can be easily applied to any LLM and it relies solely on model artifacts that are computed during the generation process.

- Through extensive experiments, we show that PLD+ outperforms tuning-free baselines for both greedy decoding and sampling and that it can sometimes even outperform the best tuning-dependent approach EAGLE. PLD+ outperforms tuning-free baselines across model families as well.

2 Related Work

Speculative Decoding, introduced by Stern et al. (2018), represents a novel paradigm aimed at enhancing the efficiency of LLM inference. Instead of generating tokens sequentially, Speculative Decoding drafts multiple future tokens efficiently and then verifies them in parallel.

Drafting strategies are distinctly categorized into Independent and Self-Drafting methodologies. Independent Drafting leverages external models for the drafting process. This method often necessitates special training or fine-tuning to align with the target LLM's output characteristics. Examples include the use of smaller models within the same model family to draft tokens for their larger counterparts (Chen et al., 2023; Leviathan et al., 2023), capitalizing on inherent architectural and training similarities to boost alignment and reduce additional training requirements.

PLD/LLMA (Saxena, 2023; Yang et al., 2023b) is a simple yet highly effective independent drafting strategy for speeding up LLM inference in "input-guided tasks". It leverages string matching to generate candidates, capitalizing on the n-gram overlap between input and output. By replacing external models with string matching, this approach benefits from being tuning-free and model agnostic.

Self-Drafting strategies employ the target LLM itself for drafting, introducing innovative methods like appending multiple [PAD] tokens to simulate future tokens (Monea et al., 2023; Santilli et al., 2023) or employing early exiting (Yang et al., 2023a) and layer skipping techniques within the LLM (Zhang et al., 2023b) to expedite the drafting phase. These approaches minimize the overhead associated with integrating separate drafter models and ensure closer inherent alignment with the target LLM's output patterns.

Substantial research efforts have focused on improving speculative decoding, yet the specific and ubiquitous setting of "input-guided tasks" has seen limited advancement beyond the simple PLD method. REST (He et al., 2023) while similar in flavor to PLD, retrieves drafts from an external datastore to avoid being limited to a small context. In the REST framework, a large datastore constructed using LLM generations is required for the best results, resulting in a considerable overhead. Our approach improves PLD by ranking the retrieved candidates from the input context based on semantics (rather than heuristics) using the model artifacts. Also by focusing on "input guided tasks", we get rid of the additional overhead of an external datastore required by REST.

3 Background: Autoregressive decoding and Speculative Decoding

Autoregressive Decoding: Given an input sequence $x = x_1, x_2, \dots x_t$, an autoregressive language model $\mathcal{M}_\mathrm{q}$ generates the next token x_{t+1} as, $x_{t+1} \sim q_{t+1} = \mathcal{M}_\mathrm{q}(x|x_1, x_2, ..., x_t)$, where q_{t+1} is the conditional probability calculated by the model $\mathcal{M}_\mathrm{q}$ over its vocabulary $\mathcal{V}$. Based on the sampling scheme, the token x_{t+1} is sampled from the probability distribution. The causal dependency of each decoding step on the results of previous decoding steps makes autoregressive decoding slow and inefficient due to its inability to fully utilize the parallelization capability of GPUs.

Speculative Decoding: Speculative Decoding (Stern et al., 2018; Leviathan et al., 2023), can be expressed as a Draft and Verify paradigm (Xia et al., 2024). For a given decoding step t, a drafting algorithm speculates K draft tokens $\hat{x_1}, \dots, \hat{x_\mathrm{K}}$ efficiently, which are then verified by the language model $\mathcal{M}_\mathrm{q}$. In the standard speculative decoding approach, the drafting algorithm used is a smaller language model $\mathcal{M}_\mathrm{p}$.

In the verification phase, the draft tokens $\hat{x_1}, \dots, \hat{x_\mathrm{K}}$ are verified by the model $\mathcal{M}_\mathrm{q}$ using a single forward pass. Given the input sequence $x = x_1, x_2, \dots x_t$ and the draft tokens $\hat{x_1}, \dots, \hat{x_\mathrm{K}}$, the model computes $\mathrm{K}+1$ probability distributions at once. Each drafted token is verified as per a verification strategy. Common verification criteria include rejection resampling (Leviathan et al., 2023; Chen et al., 2023) and greedy acceptance (Stern et al., 2018). Only the draft tokens that meet the verification strategy are retained in order to maintain consistency of generation with respect to standard autoregressive decoding using the model $\mathcal{M}_\mathrm{q}$.

4 Our approach: PLD+

In the following sections, we explain the drafting §4.2 and verification §4.4 algorithms of our approach. Figure 1 provides an overview of the generation process using PLD+ for inference. We also provide the algorithm for PLD+ in Appendix B.

4.1 Notations

Formally, given an input sequence $x = x_1, x_2, ...x_t$ passed to a language model $\mathcal{M}_\mathrm{q}$ for decoding step t, our approach predicts and verifies draft tokens $\hat{x_1}, \dots, \hat{x_K}$ leveraging model attentions $\mathbf{A}$ or model hidden states $\mathbf{H}$ computed during the generation process. The value K denotes the number of draft tokens that are predicted. The hidden states $\mathbf{H}$ is a vector with dimensions $(L, |x_{<t}|, d)$ and the model attention states $\mathbf{A}$ is a vector with dimensions $(L, G, |x_{<t}|, |x_{<t}|)$, where, L is the number of layers in the model, G is the number of attention heads per layer, $|x_{<t}|$ is the sequence length of the input tokens before decoding step t, d is the embedding size.

4.2 Drafting Algorithm

Our goal is to select an optimal token span from x such that we exploit the overlap between the input sequence and the generation sequence.

To achieve this, we first identify a set of positions P, where the last generated token x_t occurs in the input token sequence x.

$$P = \{j \mid x_j = x_t, j < t\} \tag{1}$$

To maximize inference speedup, we need to **rank** these occurrences and select the occurrence, j^*, that is most likely to yield the highest overlap with the subsequent tokens in the generation sequence. We hypothesize that the artifacts computed during the generation process captures contextual information which can be utilized to choose the optimal occurrence. In the following two sections, we describe how model hidden states and attentions are used to find the best occurrence j^*, for accelerated inference.

4.2.1 Ranking occurrences using model attentions

The model attentions $\mathbf{A}$ computed by the model across different layers l and heads g is available for the sequence $x_{<t}$. The most straightforward method of ranking occurrences is to aggregate all attention maps across l and g for token x_{t-1} using max or sum operation and choose the position j^* which has the highest value. Recent work in mechanistic interpretability (Olsson et al., 2022; Bansal et al., 2023) indicate that there exists induction heads which drive the in context learning ability of models. Induction heads are defined as attention heads that engage in two specific behaviors: prefix matching, which involves locating a previous instance of the current token within the context, and copying, which entails duplicating the sequence that follows the identified token. The behavior of induction heads can be highly useful for accelerating inference for input-guided tasks.

Identification of relevant attention heads: We identified heads that can be relevant by first generating the outputs $o_t \in O$ and attentions $\mathbf{A}$ for a set of prompts. For each generated token o_t present in the input x, we find the set of positions P, from where the token could have been "copied", i.e, positions where the generated token is present in the input. We then choose the position r^* which has the maximum overlapping suffix with the generated output. We then iterate over all of the attention heads $L \times G$ and keep track of the attention heads where the token position x_{r^*} has the maximum attention. We repeat this process for every prompt and for a given model $\mathcal{M}_\mathrm{q}$ we identify relevant attention heads , G^* from layers L^*.

After identifying the relevant attention heads, we aggregate the attention scores from the heads in G^* across layers L^* using the max operation, and then we select the position j^* that has the highest value.

$$j^* = \underset{j \in P}{\mathrm{argmax}} \left(\max_{l \in L^*, g \in G^*} \mathbf{A}_{t-1}^{(l,g)} \right) \tag{2}$$

4.2.2 Ranking occurrences using hidden states

The hidden states $\mathbf{H}$ computed by the model across different layers l is available for the sequence $x_{<t}$. The hidden states corresponding to the last generated token x_t is not available. Therefore, for each position $j \in P$, we compute the cosine similarity between $\mathbf{H}_{j-1}^{(l)}$ and $\mathbf{H}_{t-1}^{(l)}$ and select the occurrence j^* which results in the highest similarity value.

$$j^* = \underset{j \in P}{\mathrm{argmax}}\, \mathrm{cos\_sim}(\mathbf{H}_{j-1}^{(l)}, \mathbf{H}_{t-1}^{(l)}) \tag{3}$$

4.3 Draft Prediction

We predict $\hat{x}_i, \ldots, \hat{x}_{i+\mathrm{K}}$ as the future tokens, where K denotes the number of predicted draft tokens. The number of draft tokens K, the layer l and the head g are the hyper parameters.

$$\textit{draft tokens} \quad \hat{x}_i = x_{j^*+i}, i = 1, \ldots \mathrm{K}$$

4.4 Verification Algorithm

The goal of the verification phase is to ensure that the tokens generated by PLD+ are the same as those generated by standard autoregressive decoding. To achieve this, we first pass the input sequence x along with the draft tokens $\hat{x}_i$ to obtain conditional probabilities for future positions $(t + i, i = 1, \ldots K)$ using $\mathcal{M}_\mathrm{q}$. Using these probabilities, we sample new tokens at the future positions. We verify if the sampled tokens match with the draft tokens at each position. After the first mismatch we discard the subsequent draft tokens.

4.5 Application Scenarios

Our motivation is to accelerate generation in tasks where the generation outputs have significant overlaps with the input context. As enumerated by Yang et al. (2023a): Multi-turn conversation, Retrieval Augmented Generation and Cache assisted Generation naturally fall under the input-guided tasks paradigm. We conduct our experiments on a broader range of tasks: code editing, text editing (short), text editing (long), multiturn conversation, and summarization.

5 Experimental Setup

5.1 Datasets

For the tasks mentioned in Section 4.5, we sample data from the following datasets.

	Methods	Summarization	Code Editing	Text Editing (Short)	Text Editing (Long)	Multi-turn Conversation	Avg. Throughput (#tokens/s)
Vicuna-7B	Autoregressive Decoding	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	28.95
	Medusa$^{\blacklozenge}$	1.46×$_{\pm0.27}$	2.45×$_{\pm0.12}$	1.87×$_{\pm0.03}$	2.15×$_{\pm0.04}$	2.21×$_{\pm0.09}$	58.77
	EAGLE$^{\blacklozenge}$ (Li et al., 2024)	2.43×$_{\pm0.05}$	3.16×$_{\pm0.07}$	2.58×$_{\pm0.02}$	2.85×$_{\pm0.06}$	2.77×$_{\pm0.1}$	79.84
	Hydra$^{\blacklozenge}$ (Ankner et al., 2024)	1.79×$_{\pm0.26}$	3.11×$_{\pm0.12}$	2.2×$_{\pm0.03}$	2.55×$_{\pm0.04}$	2.81×$_{\pm0.04}$	72.24
	SpS (Chen et al., 2023)	1.75×$_{\pm0.06}$	1.97×$_{\pm0.08}$	2.04×$_{\pm0.06}$	1.78×$_{\pm0.05}$	1.73×$_{\pm0.06}$	53.58
	Lookahead (Fu et al., 2024)	1.16×$_{\pm0.24}$	1.7×$_{\pm0.07}$	1.39×$_{\pm0.03}$	1.47×$_{\pm0.04}$	1.55×$_{\pm0.04}$	42.05
	REST (He et al., 2023)	1.41×$_{\pm0.02}$	1.84×$_{\pm0.03}$	1.43×$_{\pm0.04}$	1.6×$_{\pm0.03}$	1.69×$_{\pm0.02}$	46.25
	PLD (Yang et al., 2023a; Saxena, 2023)	2.62×$_{\pm0.03}$	2.43×$_{\pm0.04}$	2.73×$_{\pm0.02}$	3.11×$_{\pm0.06}$	1.63×$_{\pm0.02}$	72.48
	PLD+ (a)	3.32×$_{\pm0.07}$	3.69×$_{\pm0.17}$	3.88×$_{\pm0.16}$	5.09×$_{\pm0.17}$	1.85×$_{\pm0.04}$	103.23
	PLD+ (h)	**3.39**×$_{\pm0.07}$	**3.83**×$_{\pm0.1}$	**4.01**×$_{\pm0.01}$	**5.16**×$_{\pm0.1}$	**1.92**×$_{\pm0.04}$	**106.05**
Vicuna-13B	Autoregressive Decoding	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	22.7
	Medusa$^{\blacklozenge}$ (Cai et al., 2024)	1.82×$_{\pm0.04}$	2.49×$_{\pm0.02}$	1.92×$_{\pm0.06}$	2.25×$_{\pm0.06}$	2.23×$_{\pm0.13}$	48.69
	EAGLE$^{\blacklozenge}$ (Li et al., 2024)	2.51×$_{\pm0.04}$	3.32×$_{\pm0.14}$	2.73×$_{\pm0.05}$	3.03×$_{\pm0.07}$	2.95×$_{\pm0.03}$	66.08
	Hydra$^{\blacklozenge}$(Ankner et al., 2024)	2.24×$_{\pm0.09}$	3.01×$_{\pm0.46}$	2.37×$_{\pm0.09}$	2.82×$_{\pm0.04}$	2.89×$_{\pm0.05}$	60.62
	SpS (Chen et al., 2023)	1.84×$_{\pm0.02}$	2.19×$_{\pm0.06}$	2.1×$_{\pm0.06}$	1.78×$_{\pm0.08}$	1.74×$_{\pm0.07}$	43.81
	Lookahead (Fu et al., 2024)	1.39×$_{\pm0.02}$	1.68×$_{\pm0.09}$	1.33×$_{\pm0.02}$	1.44×$_{\pm0.04}$	1.52×$_{\pm0.04}$	33.49
	REST (He et al., 2023)	1.44×$_{\pm0.01}$	1.95×$_{\pm0.05}$	1.41×$_{\pm0.04}$	1.74×$_{\pm0.06}$	1.7×$_{\pm0.02}$	37.46
	PLD (Yang et al., 2023a; Saxena, 2023)	2.47×$_{\pm0.03}$	3.04×$_{\pm0.04}$	2.68×$_{\pm0.08}$	2.76×$_{\pm0.04}$	1.61×$_{\pm0.02}$	57.06
	PLD+ (a)	**2.75**×$_{\pm0.05}$	5.2×$_{\pm0.18}$	3.82×$_{\pm0.11}$	3.77×$_{\pm0.12}$	1.72×$_{\pm0.03}$	78.47
	PLD+ (h)	2.73×$_{\pm0.02}$	**5.37**×$_{\pm0.15}$	**3.88**×$_{\pm0.04}$	**3.85**×$_{\pm0.11}$	**1.79**×$_{\pm0.01}$	**80.1**
Vicuna-33B	Autoregressive Decoding	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	14.76
	Medusa$^{\blacklozenge}$ (Cai et al., 2024)	1.87×$_{\pm0.05}$	2.69×$_{\pm0.02}$	1.92×$_{\pm0.04}$	2.14×$_{\pm0.02}$	2.26×$_{\pm0.02}$	32.13
	EAGLE$^{\blacklozenge}$ (Li et al., 2024)	2.65×$_{\pm0.05}$	3.74×$_{\pm0.11}$	2.85×$_{\pm0.09}$	3.04×$_{\pm0.09}$	2.94×$_{\pm0.05}$	44.96
	Hydra$^{\blacklozenge}$	2.33×$_{\pm0.05}$	3.41×$_{\pm0.11}$	2.32×$_{\pm0.05}$	2.8×$_{\pm0.02}$	2.92×$_{\pm0.06}$	40.7
	SpS (Chen et al., 2023)	1.87×$_{\pm0.05}$	2.42×$_{\pm0.05}$	2.34×$_{\pm0.12}$	1.87×$_{\pm0.01}$	1.8×$_{\pm0.03}$	30.38
	Lookahead (Fu et al., 2024)	1.34×$_{\pm0.03}$	1.62×$_{\pm0.03}$	1.36×$_{\pm0.02}$	1.36×$_{\pm0.02}$	1.51×$_{\pm0.05}$	21.24
	REST (He et al., 2023)	1.51×$_{\pm0.04}$	2.0×$_{\pm0.06}$	1.42×$_{\pm0.05}$	1.8×$_{\pm0.01}$	1.74×$_{\pm0.04}$	25.04
	PLD (Yang et al., 2023a; Saxena, 2023)	**2.13**×$_{\pm0.0}$	2.77×$_{\pm0.04}$	2.87×$_{\pm0.22}$	2.45×$_{\pm0.03}$	1.56×$_{\pm0.02}$	34.75
	PLD+ (a)	2.07×$_{\pm0.04}$	3.71×$_{\pm0.07}$	4.2×$_{\pm0.04}$	**3.22**×$_{\pm0.01}$	1.59×$_{\pm0.06}$	43.65
	PLD+ (h)	2.09×$_{\pm0.02}$	**3.8**×$_{\pm0.11}$	**4.29**×$_{\pm0.1}$	3.18×$_{\pm0.03}$	**1.59**×$_{\pm0.01}$	**44.12**

Table 1: **Comparison of PLD+ against various speculative decoding baselines across 5 input guided tasks (T=0).** ♦ indicates tuning-dependent baselines. Mean speedup across 3 runs is reported. **Bold** represents best tuning-free and Underline represents best overall. **Note:** Hyperparameters were chosen for PLD+ using the summarization task.

	Methods	Summarization	Code Editing	Text Editing (Short)	Text Editing (Long)	Multi-turn Conversation	Avg. Throughput (#tokens/s)
Vicuna-7B	Autoregressive Decoding	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	27.02
	EAGLE$^{\blacklozenge}$ (Li et al., 2024)	2.02×$_{\pm0.05}$	2.75×$_{\pm0.06}$	2.08×$_{\pm0.0}$	2.37×$_{\pm0.01}$	2.37×$_{\pm0.16}$	62.61
	REST (He et al., 2023)	1.39×$_{\pm0.08}$	1.67×$_{\pm0.06}$	1.31×$_{\pm0.02}$	2.02×$_{\pm0.03}$	**1.61**×$_{\pm0.1}$	42.78
	PLD (Yang et al., 2023a; Saxena, 2023)	2.04×$_{\pm0.1}$	1.78×$_{\pm0.04}$	2.26×$_{\pm0.05}$	1.57×$_{\pm0.03}$	1.45×$_{\pm0.07}$	49.47
	PLD+ (a)	2.28×$_{\pm0.09}$	**2.52**×$_{\pm0.04}$	**2.75**×$_{\pm0.06}$	2.06×$_{\pm0.03}$	1.56×$_{\pm0.09}$	60.7
	PLD+ (h)	**2.33**×$_{\pm0.06}$	2.2×$_{\pm0.02}$	2.7×$_{\pm0.04}$	**2.26**×$_{\pm0.04}$	1.54×$_{\pm0.05}$	**59.66**
Vicuna-13B	Autoregressive Decoding	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	22.08
	EAGLE$^{\blacklozenge}$ (Li et al., 2024)	2.25×$_{\pm0.04}$	3.03×$_{\pm0.14}$	2.32×$_{\pm0.02}$	2.53×$_{\pm0.09}$	2.53×$_{\pm0.07}$	55.83
	REST (He et al., 2023)	1.4×$_{\pm0.02}$	1.98×$_{\pm0.08}$	1.38×$_{\pm0.02}$	1.76×$_{\pm0.05}$	**1.72**×$_{\pm0.05}$	36.32
	PLD (Yang et al., 2023a; Saxena, 2023)	2.0×$_{\pm0.02}$	2.02×$_{\pm0.1}$	2.32×$_{\pm0.06}$	1.82×$_{\pm0.01}$	1.41×$_{\pm0.05}$	42.27
	PLD+ (a)	**2.24**×$_{\pm0.02}$	2.4×$_{\pm0.06}$	3.16×$_{\pm0.05}$	**2.21**×$_{\pm0.03}$	1.64×$_{\pm0.04}$	**51.37**
	PLD+ (h)	2.09×$_{\pm0.03}$	**2.53**×$_{\pm0.16}$	**3.27**×$_{\pm0.06}$	2.15×$_{\pm0.08}$	1.54×$_{\pm0.01}$	51.01
Vicuna-33B	Autoregressive Decoding	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	1.00×$_{\pm0.00}$	14.41
	EAGLE$^{\blacklozenge}$ (Li et al., 2024)	2.36×$_{\pm0.04}$	3.23×$_{\pm0.11}$	2.46×$_{\pm0.06}$	2.67×$_{\pm0.14}$	2.63×$_{\pm0.01}$	38.66
	REST (He et al., 2023)	1.51×$_{\pm0.03}$	1.99×$_{\pm0.04}$	1.46×$_{\pm0.04}$	1.84×$_{\pm0.07}$	**1.68**×$_{\pm0.03}$	24.43
	PLD (Yang et al., 2023a; Saxena, 2023)	1.96×$_{\pm0.06}$	2.47×$_{\pm0.01}$	2.68×$_{\pm0.05}$	2.21×$_{\pm0.05}$	1.43×$_{\pm0.02}$	31.01
	PLD+ (a)	**1.97**×$_{\pm0.03}$	**3.28**×$_{\pm0.08}$	3.85×$_{\pm0.03}$	**2.57**×$_{\pm0.06}$	1.47×$_{\pm0.01}$	**38.11**
	PLD+ (h)	1.89×$_{\pm0.03}$	3.14×$_{\pm0.08}$	**3.86**×$_{\pm0.1}$	2.39×$_{\pm0.06}$	1.52×$_{\pm0.02}$	36.91

Table 2: **Comparison of PLD+ against various speculative decoding baselines across 5 input guided tasks (T=1)** ♦ indicates tuning-dependent baselines.Mean speedup across 3 runs is reported. **Bold** represents best tuning-free and Underline represents best overall. **Note:** Hyperparameters were chosen for PLD+ using the summarization task.

Code Editing: We leverage CodeEditorBench_Plus, [1] one of the two datasets introduced by (Guo et al., 2024) to test the performance of LLMs in code editing tasks: debugging, translating, polishing and requirement switching. We randomly select 20 instances for each task, resulting in a total of 80 samples.

Text Editing (short): We leverage the text editing benchmark XATU [2], introduced by (Zhang et al., 2023a) to test the capabilities of LLMS for fine-grained instruction-based text editing. The benchmark accounts for tasks of various difficulties such as text simplification, grammar error correction, style transfer and information update.We

[1] The dataset was downloaded from https://github.com/CodeEditorBench/CodeEditorBench. It is licensed under Apache License 2.0

[2] The dataset was downloaded from https://github.com/megagonlabs/xatu. It is licensed under CC-BY-NC 4.0 license

randomly select 30 samples for each of the nine datasets resulting in a total of 270 samples.

Text Editing (long): We leverage the ArgRewrite V.2 corpus[3] (Kashefi et al., 2022) which contains annotated argumentative revisions, collected over 2 revision cycles on essays about self-driving cars. In particular, we make use of the first draft of the essay and the expert feedback (human feedback) given to the first draft. We sample 80 instances from this dataset to conduct our experiments.

Multi-turn Conversation: We leverage the MT-Bench benchmark [4] (Zheng et al., 2024) which consists of 80 multi-turn questions across the following categories: writing, roleplay, extraction, reasoning, math, coding, knowledge I (STEM), and knowledge II (humanities/social science).

Summarization : We leverage the summarization subset of the Spec-Bench benchmark [5](Xia et al., 2024) for our experiments, as it has a high overlap between the input prompts and the generation outputs. This subset has 80 samples that were randomly sampled from the CNN/Daily Mail corpus (Nallapati et al., 2016).

5.2 Baselines

We compare our method with seven speculative decoding approaches leveraging the Spec-Bench benchmark (Xia et al., 2024). In particular we compare our approach with Medusa (Cai et al., 2024), EAGLE (Li et al., 2024), Hydra (Ankner et al., 2024) (finetuning dependent approaches) , and SpS (Chen et al., 2023), Lookahead (Fu et al., 2024), LLM-A/PLD (Yang et al., 2023a; Saxena, 2023), REST (He et al., 2023) (tuning-free approaches).

5.3 Metrics

Our evaluation centers on three key metrics: average throughput (tokens/sec), speedup (speculative vs. standard decoding), and average acceptance length (tokens accepted per generation).

5.4 Implementation Details

We conduct our main experiments using the `Vicuna-1.3` model series leveraging the code base of the Spec-Bench benchmark (Xia et al., 2024). For SpS, we used `Vicuna-68m-v1.3` as the draft language model. We follow the default parameters for all of our speculative decoding baselines. All our experiments have been conducted on an NVIDIA A100 GPU (with 80GB memory), with CUDA Version 12.2 and torch version 2.0.1.

5.5 Hyperparameter Tuning

We used the summarization task as our validation set for hyperparameter tuning with `Vicuna-7b-1.3`. As explained in Section 4.2, PLD+ has two key hyperparameters: the number of draft tokens (K) and the layer (l) if model hidden states are used to accelerate inference. We first determined the optimal value for l by fixing K=10, then, for the selected layer, we tuned K. Using the average acceptance length metric, the best values were K=70 and l=9. When using model attentions for accelerating inference, PLD+ has two hyperparameters: the number of draft tokens (K) and the attention heads (g). As described in Section 4.2.1, we identified 379 induction heads (37% of the total heads in the model) and ranked them based on prefix matching and copying frequency. We found that using the top 50 heads gave the best average acceptance length. After fixing g as the top 50 heads, we tuned K and found the optimal value to be K=70.

6 Results and Analysis

6.1 Main Experimental Results

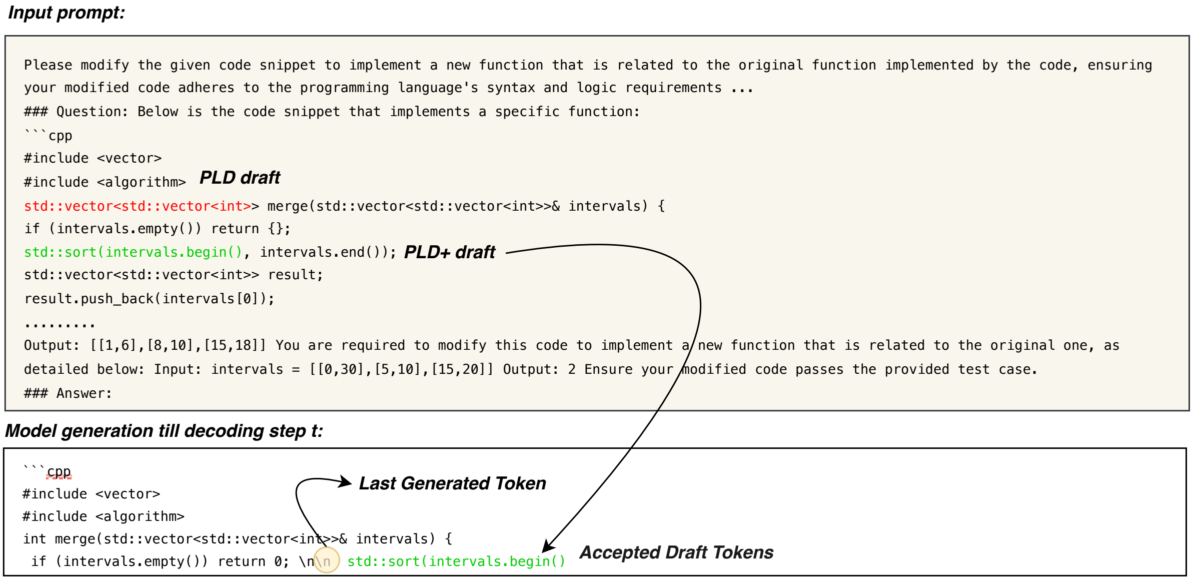
Tables 1 (T=0, greedy decoding) and 2 (T=1, sampling) compare PLD+ with baselines across five input-guided tasks. We denote our approach as PLD+ (a) when using model attentions and PLD+ (h) when using hidden states. In the greedy decoding scenario, PLD+ either exceeds or matches performance of tuning-free baselines in all five tasks. For code editing, short text editing, and long text editing tasks, PLD+ even surpasses the best-performing fine-tuned approach, EAGLE (Li et al., 2024), by margins ranging from 0.24× to 2.85× depending on model size. These results support our hypothesis that leveraging overlap between input and generation accelerates inference.

In the sampling scenario, speedups for all methods decrease compared to greedy decoding due to the randomness introduced by sampling. Despite this, PLD+ still outperforms all tuning-free baselines except on the Multi-turn Conversation task, where it ranks second, trailing REST (He

[3] The dataset was downloaded from https://argrewrite.cs.pitt.edu/#corpus. It is licensed under GNU General Public License

[4] The dataset was downloaded from https://huggingface.co/spaces/lmsys/mt-bench. It is licensed under Apache License 2.0

[5] The dataset was downloaded from https://github.com/hemingkx/Spec-Bench. It is licensed under Apache License 2.0



~~~
Please modify the given code snippet to implement a new function that is related to the original function implemented by the code, ensuring
your modified code adheres to the programming language's syntax and logic requirements ...
### Question: Below is the code snippet that implements a specific function:
```cpp
#include <vector>
#include <algorithm>
std::vector<std::vector<int>> merge(std::vector<std::vector<int>>& intervals) {
if (intervals.empty()) return {};
std::sort(intervals.begin(), intervals.end());
std::vector<std::vector<int>> result;
result.push_back(intervals[0]);
.........
Output: [[1,6],[8,10],[15,18]] You are required to modify this code to implement a new function that is related to the original one, as
detailed below: Input: intervals = [[0,30],[5,10],[15,20]] Output: 2 Ensure your modified code passes the provided test case.
### Answer:
~~~

~~~
```cpp
#include <vector>
#include <algorithm>
int merge(std::vector<std::vector<int>>& intervals) {
 if (intervals.empty()) return 0; \n\n std::sort(intervals.begin()
~~~

Figure 2: As shown in the figure, the last generated token is \n. PLD has proposed draft tokens by selecting the candidate span with the longest matching prefix and only 'std::' of the draft span can be accepted during verification. In contrast, PLD+ proposes the candidate span with the highest semantic relevance resulting in a larger draft span 'std::sort(intervals.begin()' being accepted and resulting in higher token throughput.

et al., 2023) by a maximum of 0.18. We manually reviewed the dataset and found that certain categories (roleplay, STEM, humanities) in the multi-turn conversation task may not qualify as input-guided tasks. The first-turn questions often lack sufficient information for generating second-turn answers, which impacts PLD+'s performance as it relies on input-output overlap.

While speculative decoding methods with powerful draft models or additional tuning could achieve even better alignment between the target LLM and the draft model, PLD+ offers an out-of-the-box, training-free solution for faster LLM inference, requiring no additional RAM for a draft model.

	Methods	Speedup	Avg. Throughput
L2C-7B	Autoregressive Decoding	1.00× $_{\pm 0.00}$	28.73 $_{\pm 0.26}$
	EAGLE♦ (Li et al., 2024)	2.35× $_{\pm 0.04}$	67.6 $_{\pm 0.88}$
	REST (He et al., 2023)	1.48× $_{\pm 0.01}$	42.45 $_{\pm 0.38}$
	PLD (Yang et al., 2023a; Saxena, 2023)	1.87× $_{\pm 0.02}$	53.64 $_{\pm 0.96}$
	PLD+ (h)	**2.12**× $_{\pm 0.04}$	**61.01** $_{\pm 0.7}$
L2C-13B	Autoregressive Decoding	1.00× $_{\pm 0.00}$	22.25 $_{\pm 0.23}$
	EAGLE♦ (Li et al., 2024)	2.49× $_{\pm 0.03}$	55.5 $_{\pm 0.3}$
	REST (He et al., 2023)	1.5× $_{\pm 0.01}$	33.31 $_{\pm 0.23}$
	PLD (Yang et al., 2023a; Saxena, 2023)	1.82× $_{\pm 0.02}$	40.41 $_{\pm 0.76}$
	PLD+ (h)	**1.93**× $_{\pm 0.05}$	**42.89** $_{\pm 1.01}$
M-7B	Autoregressive Decoding	1.00× $_{\pm 0.00}$	26.37 $_{\pm 0.68}$
	EAGLE♦ (Li et al., 2024)	-	-
	REST (He et al., 2023)	-	-
	PLD (Yang et al., 2023a; Saxena, 2023)	3.32× $_{\pm 0.05}$	87.45 $_{\pm 1.86}$
	PLD+ (h)	**4.78**× $_{\pm 0.02}$	**126.04** $_{\pm 3.59}$

Table 3: Comparison of PLD+ with speculative decoding baselines on the summarization task using Mistral-7B-Instruct (M-7B), Llama-2-7B-Chat (L2C-7B) and Llama-2-13B-Chat (L2C-7B) . Experiments were performed using with the greedy decoding strategy. Mean speedup across 3 runs is reported. **Bold** represents best tuning-free and Underline represents best overall.

6.2 Differences and Improvements: PLD+ versus PLD

For input-guided tasks, PLD leverages string matching to generate candidates and ranks them using the length of the prefix match. Our motivation is to exploit the intermediate artifacts available during generation (as they capture contextual information) to rank candidates to maximize token throughput. The quantitative improvement of PLD+ over PLD is reflected in Tables 1 and 2. Figure 2 illustrates how PLD+ selects semantically relevant draft span compared to PLD. For both methods, we search for occurrences of last generated token '\n'. Draft selected by PLD has the longest matching prefix (heuristic), and only 'std::' of the draft span is accepted. PLD+ selects draft based on semantic relevance allowing a larger draft span 'std::sort(intervals.begin()' to be accepted. In tasks like code editing, where repeated n-grams (e.g., indents) are common, correct ranking of drafts is crucial for performance. PLD+ surpasses PLD by ranking retrieved candidates based on semantic relevance using model artifacts rather than simple heuristics, yielding superior speedup.

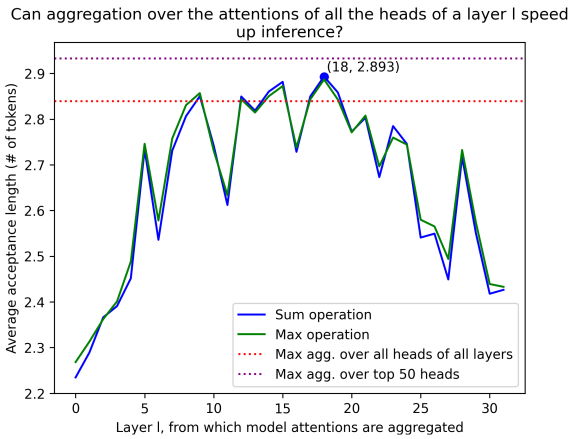


Figure 3: In this figure, we display the results of aggregating attentions across the heads of a specific layer. We experiment with two aggregation operations - summation and maximum aggregation.

6.3 Plug and play nature of PLD+

To demonstrate the versatility of PLD+, we conducted experiments on the summarization task using newer models like Mistral-7B-Instruct, Llama-2-7B-Chat, and Llama-2-13B-Chat with the greedy decoding strategy. Unlike many speculative decoding methods that require additional compute/-tuning and don't support all LLMs out of the box, PLD+ works seamlessly across models. We compare against baselines that support these models, and as shown in Table 3, PLD+ consistently outperforms all tuning-free baselines, reinforcing our claim that it can be used out of the box with any model, achieving notable speedups.

7 Ablation study

To gain a better understanding of our approach, we conduct a series of experiments and analyse their results in this section. We also present additional experiments in Appendix C.

7.1 How to choose attention heads for ranking occurrences?

In Section 4.2.1, we detailed how we identify relevant attention heads. Using `Vicuna-7b-1.3` for the summarization task and average acceptance length as our metric, we tested other methods by aggregating attention from all heads in a given layer and across all layers, using both summation and maximum aggregation. Figure 3 shows the results, with the best performance achieved by selecting specific attention heads (as described in Section 4.2.1) and applying maximum aggregation.

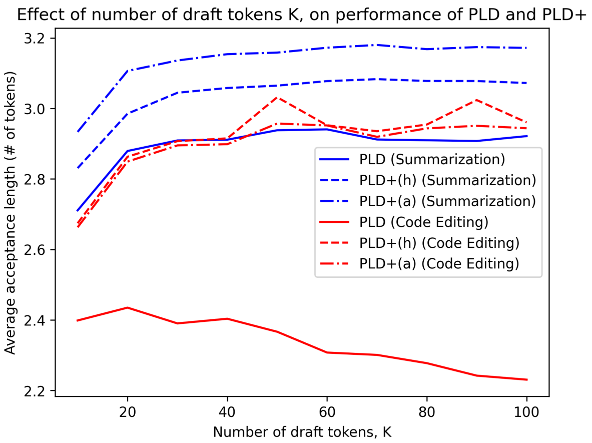


Figure 4: Effect of K,the number of draft tokens on the performance of PLD, PLD+

7.2 Effect of number of draft tokens, K on performance of PLD+ and PLD

In this section we analyse the impact of K, the number of draft tokens on the performance of PLD and PLD+. We vary K from 10 to 100 and run experiments on the summarization task and use the average acceptance length metric to evaluate the performance. Figure 4 depicts the performance of PLD and PLD+ across different K values. From this figure, we can clearly see that PLD+ benefits from larger K values, and it would be more suitable for tasks where large spans of the output overlap with the input (Code Editing, Text Editing). We hypothesize that using model artifacts (hidden states / attentions) to select the draft tokens helps us find longer matching spans compared to simple n-gram matching.

7.3 Effect of layer l, on performance of PLD+(h)

In this section we analyse the impact of the l from which the hidden states are obtained for PLD+. For the `Vicuna-1.3` family of models, we fix K=10 and compute the performance for every l on the summarization task and use the average acceptance length metric to evaluate the performance. Figure 4 depicts performance of PLD+(h) across different layers. From this figure, we can see that across models PLD+ benefits from using early layers (between 9 and 13).

8 Conclusion

In this work, we propose Prompt Lookup Decoding+ (PLD+), a plug-and-play speculative decoding approach that can be easily integrated into the gen-

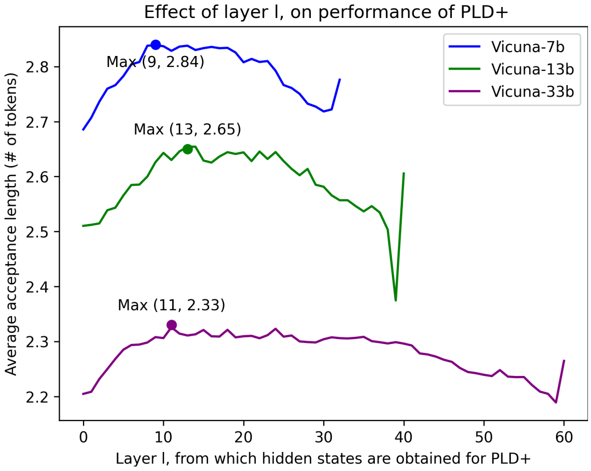


Figure 5: Effect of layer l on performance of PLD+ (h)

eration process of any language model without additional training. Instead of using a small language model as our drafter, we intelligently choose spans from the input as draft tokens. At each decoding step, we find candidate draft spans and choose the best span leveraging the model artifacts computed during generation. Through extensive experiments we found that PLD+ performs on par or even outperforms fine-tuning dependent speculative decoding approaches for the ubiquitous setting of input guided tasks.

9 Limitations and Future Work

Though PLD+ is a plug-and-play tuning-free decoding technique, it is important to note that the speedup is directly influenced by the nature of the task. PLD+ is best suited for the tasks where there is a significant overlap between the input and the model generations. In Appendix D, we provide some qualitative examples illustrating examples, where our algorithm does not propose the optimal draft completion (which has maximal overlap with the subsequent generation sequence).

An important feature of PLD+ is the ability to point to the location in the context from which copying occurs. The task of identifying which parts of the context resulted in the generation of a token is known as input attribution and is of particular interest to the community (Li et al., 2023).

In the Appendix E, we provide preliminary results on the ability of our method to perform attribution. Our approach to attribution has a similar flavor to the approach employed by Phukan et al. (2024), who successfully use LLM hidden states to perform attribution of verbatim copied spans. We intend to perform a detailed analysis of attribution quality in the future.

We want to highlight that our method focuses on speeding up inference and guarantees the same generation as standard autoregressive decoding. Our method does not introduce any additional risks than those associated with LLMs such as bias, malicious use, etc.

References

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

Hritik Bansal, Karthik Gopalakrishnan, Saket Dingliwal, Sravan Bodapati, Katrin Kirchhoff, and Dan Roth. 2023. Rethinking the role of scale for in-context learning: An interpretability-based case study at 66 billion scale. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11833–11856, Toronto, Canada. 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. *arXiv 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. *arXiv preprint arXiv:2302.01318*.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. 2024. Break the sequential dependency of llm inference using lookahead decoding. *Preprint*, arXiv:2402.02057.

Jiawei Guo, Ziming Li, Xueling Liu, Kaijing Ma, Tianyu Zheng, Zhouliang Yu, Ding Pan, Yizhi LI, Ruibo Liu, Yue Wang, Shuyue Guo, Xingwei Qu, Xiang Yue, Ge Zhang, Wenhu Chen, and Jie Fu. 2024. Codeeditorbench: Evaluating code editing capability of large language models. *Preprint*, arXiv:2404.03543.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. 2023. Rest: Retrieval-based speculative decoding. *arXiv preprint arXiv:2311.08252*.

Omid Kashefi, Tazin Afrin, Meghan Dale, Christopher Olshefski, Amanda Godley, Diane Litman, and Rebecca Hwa. 2022. Argrewrite v. 2: an annotated argumentative revisions corpus. *Language Resources and Evaluation*, pages 1–35.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Dongfang Li, Zetian Sun, Xinshuo Hu, Zhenyu Liu, Ziyang Chen, Baotian Hu, Aiguo Wu, and Min Zhang. 2023. A survey of large language models attribution. *arXiv preprint arXiv:2311.03731*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*.

Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, et al. 2023. Summary of chatgpt-related research and perspective towards the future of large language models. *Meta-Radiology*, page 100017.

Giovanni Monea, Armand Joulin, and Edouard Grave. 2023. Pass: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulcehre, 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.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2022. In-context learning and induction heads. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Anirudh Phukan, Shwetha Somasundaram, Apoorv Saxena, Koustava Goswami, and Balaji Vasan Srinivasan. 2024. Peering into the mind of language models: An approach for attribution in contextual question answering. *arXiv preprint arXiv:2405.17980*.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. 2023. Accelerating transformer inference for translation via parallel decoding. *arXiv preprint arXiv:2305.10427*.

Apoorv Saxena. 2023. Prompt lookup decoding.

Tal Schuster, Adam D Lelkes, Haitian Sun, Jai Gupta, Jonathan Berant, William W Cohen, and Donald Metzler. 2023. Semqa: Semi-extractive multi-source question answering. *arXiv preprint arXiv:2311.04886*.

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

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. *arXiv preprint arXiv:2401.07851*.

Nan Yang, Tao Ge, Liang Wang, Binxing Jiao, Daxin Jiang, Linjun Yang, Rangan Majumder, and Furu Wei. 2023a. Inference with reference: Lossless acceleration of large language models. *arXiv preprint arXiv:2304.04487*.

Seongjun Yang, Gibbeum Lee, Jaewoong Cho, Dimitris Papailiopoulos, and Kangwook Lee. 2023b. Predictive pipelined decoding: A compute-latency trade-off for exact llm decoding. *arXiv preprint arXiv:2307.05908*.

Haopeng Zhang, Hayate Iso, Sairam Gurajada, and Nikita Bhutani. 2023a. Xatu: A fine-grained instruction-based benchmark for explainable text updates. *arXiv preprint arXiv:2309.11063*.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. 2023b. Draft & verify: Lossless large language model acceleration via self-speculative decoding. *arXiv preprint arXiv:2309.08168*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

A Further analysis on A100 40GB GPU

Using `Vicuna-7b-v1.3` as the LLM, we conduct all of our experiments on an NVIDIA A100 GPU (with 40GB memory), with CUDA Version 12.2 and torch version 2.0.1. Implementation details are mentioned in Section 5.4. Table 4 and Table 5 indicate that PLD+ outperforms all of the tuning-free baselines on 4 input guided tasks, Code Editing, Summarization, Text editing (short), Text editing (long). In Table 4 PLD+ even outperforms all baselines in the greedy setting on the same four tasks.

B Algorithm for PLD+

In Section 4.2, we explain the steps involved in the drafting and verification phases of PLD+. In this appendix, we provide Algorithm 1 which provides a comprehensive end-end view of the generation

	Methods	Summarization	Code Editing	Text Editing (Short)	Text Editing (Long)	Multi-turn Conversation	Avg. Throughput (#tokens/s)
Vicuna-7B	Autoregressive Decoding	$1.00\times_{\pm 0.00}$	$1.00\times_{\pm 0.00}$	$1.00\times_{\pm 0.00}$	$1.00\times_{\pm 0.00}$	$1.00\times_{\pm 0.00}$	28.53
	Medusa♦ (Cai et al., 2024)	$1.8\times_{\pm 0.06}$	$2.51\times_{\pm 0.06}$	$1.78\times_{\pm 0.06}$	$2.2\times_{\pm 0.04}$	$2.22\times_{\pm 0.01}$	59.97
	EAGLE♦ (Li et al., 2024)	$2.48\times_{\pm 0.08}$	$3.19\times_{\pm 0.05}$	$2.51\times_{\pm 0.05}$	$2.84\times_{\pm 0.05}$	$2.74\times_{\pm 0.03}$	78.59
	Hydra♦ (Ankner et al., 2024)	$2.09\times_{\pm 0.11}$	$3.08\times_{\pm 0.11}$	$2.19\times_{\pm 0.03}$	$2.67\times_{\pm 0.08}$	$2.74\times_{\pm 0.03}$	72.94
	SpS (Chen et al., 2023)	$1.84\times_{\pm 0.06}$	$2.08\times_{\pm 0.02}$	$2.02\times_{\pm 0.02}$	$1.79\times_{\pm 0.08}$	$1.74\times_{\pm 0.01}$	53.94
	Lookahead (Fu et al., 2024)	$1.44\times_{\pm 0.07}$	$1.7\times_{\pm 0.04}$	$1.36\times_{\pm 0.02}$	$1.5\times_{\pm 0.05}$	$1.51\times_{\pm 0.04}$	42.92
	REST (He et al., 2023)	$1.41\times_{\pm 0.02}$	$1.83\times_{\pm 0.03}$	$1.38\times_{\pm 0.01}$	$1.65\times_{\pm 0.03}$	$1.61\times_{\pm 0.04}$	45.09
	PLD (Yang et al., 2023a; Saxena, 2023)	$2.59\times_{\pm 0.17}$	$2.39\times_{\pm 0.04}$	$2.63\times_{\pm 0.04}$	$3.14\times_{\pm 0.08}$	$1.64\times_{\pm 0.01}$	70.53
	PLD+ (a)	$\mathbf{3.43}\times_{\pm 0.09}$	$3.7\times_{\pm 0.19}$	$\mathbf{3.84}\times_{\pm 0.02}$	$\mathbf{5.37}\times_{\pm 0.12}$	$\mathbf{1.89}\times_{\pm 0.04}$	**103.9**
	PLD+ (h)	$3.27\times_{\pm 0.14}$	$\mathbf{3.76}\times_{\pm 0.09}$	$3.71\times_{\pm 0.05}$	$5.16\times_{\pm 0.1}$	$1.83\times_{\pm 0.08}$	101.04

Table 4: **Comparison of PLD+ against various speculative decoding baselines across 5 input guided tasks (T=0).** Experiments were performed using Vicuna-7b-v1.3 with the greedy decoding strategy. ♦ indicates tuning-dependent baselines. Mean speedup across 3 runs is reported. **Bold** represents best tuning-free and Underline represents best overall. **Note:** Hyperparameters were chosen for PLD+ using the summarization task.

	Methods	Summarization	Code Editing	Text Editing (Short)	Text Editing (Long)	Multi-turn Conversation	Avg. Throughput (#tokens/s)
Vicuna-7B	Autoregressive Decoding	$1.00\times_{\pm 0.00}$	$1.00\times_{\pm 0.00}$	$1.00\times_{\pm 0.00}$	$1.00\times_{\pm 0.00}$	$1.00\times_{\pm 0.00}$	28.45
	EAGLE♦ (Li et al., 2024)	$2.07\times_{\pm 0.04}$	$2.63\times_{\pm 0.02}$	$2.05\times_{\pm 0.04}$	$2.34\times_{\pm 0.08}$	$2.29\times_{\pm 0.05}$	64.77
	REST (He et al., 2023)	$1.43\times_{\pm 0.04}$	$1.78\times_{\pm 0.05}$	$1.34\times_{\pm 0.02}$	$1.69\times_{\pm 0.04}$	$\mathbf{1.63}\times_{\pm 0.04}$	44.81
	PLD (Yang et al., 2023a; Saxena, 2023)	$2.08\times_{\pm 0.09}$	$1.84\times_{\pm 0.04}$	$2.2\times_{\pm 0.1}$	$1.56\times_{\pm 0.03}$	$1.35\times_{\pm 0.02}$	51.3
	PLD+ (a)	$\mathbf{2.32}\times_{\pm 0.11}$	$\mathbf{2.57}\times_{\pm 0.04}$	$\mathbf{2.67}\times_{\pm 0.05}$	$1.68\times_{\pm 0.03}$	$1.57\times_{\pm 0.07}$	**61.44**
	PLD+ (h)	$2.27\times_{\pm 0.06}$	$2.27\times_{\pm 0.06}$	$2.59\times_{\pm 0.08}$	$\mathbf{1.89}\times_{\pm 0.08}$	$1.52\times_{\pm 0.03}$	59.89

Table 5: **Comparison of PLD+ against various speculative decoding baselines across 5 input guided tasks (T=1)** ♦ indicates tuning-dependent baselines. Experiments were performed using Vicuna-7b-v1.3 with sampling and temperature=1. Mean speedup across 3 runs is reported. **Bold** represents best tuning-free and Underline represents best overall. **Note:** Hyperparameters were chosen for PLD+ using the summarization task.

process using PLD+. For the sake of clarity, we demonstrate the algorithm using PLD+ (h) where the model artifact used is the hidden states from layer l, however it is important to note that apart from the ranking methodology (detailed in 4.2.1), all the other steps are applicable for PLD+ (a) .

C Additional ablations

C.1 How many top-k induction heads should we use for PLD+ (a)?

Table 6 summarizes the effect of the number of induction heads on the performance of PLD+(a). We vary the number of top-k induction heads and perform experiments on the summarization dataset using `Vicuna-7b-v1.3` as the LLM. We can see that we get the best performance when the number of induction heads is 50. We separately choose the top-50 attention heads for all models.

Number of top-k induction heads	Average Accept. Length
10	3.17
20	3.18
30	3.2
40	3.21
50	**3.21**
100	2.93
200	2.9
300	2.9
All	2.9

Table 6: Effect of the number of top-k induction heads on the performance of PLD+ (a)

C.2 Does using averaged hidden state representations improve performance of PLD+ (h)?

In equation 3, averaged hidden state representations $\overline{\mathbf{H}}^{(l)}$ can be used to find the best occurrence. Let m denote the prefix length used when computing the averaged representation, $\overline{\mathbf{H}}_t^{(l)} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{H}_i^{(l)}$. We conduct experiments on the summarization task using `Vicuna-7B-v1.3` by varying the prefix length m. Table 7 indicates that using averaged hidden state representations does not improve performance and that as m increases performance decreases. In the generation setup, the hidden state of x_{t+1} is contextualized by x_t by design and $\mathbf{H}_{t+1}^{(l)}$ would already be similar to $\mathbf{H}_t^{(l)}$. We hypothesize that averaging the hidden states, $\mathbf{H}_t^{(l)}$ might lead to a less discriminative representation.

C.3 Does thresholding improve performance of PLD+ (h)?

In equation 3, we can additionally impose a threshold, where we consider an occurence j only if the cosine similarity $\cos(\mathbf{H}_{j-1}^{(l)}, \mathbf{H}_{t-1}^{(l)})$ is greater than

Algorithm 1 PLD+ Decoding.

```
Requires: Language Model $\mathcal{M}_{\text{q}}$, Input sequence $\boldsymbol{x}$, maximum generation length $N$, model hidden states $\mathbf{H}$ from layer $l$, number of draft tokens K
$\boldsymbol{y} \leftarrow []$
for $t = 1$ upto $N$ do
    $positions \leftarrow$ FIND_MATCH$(\boldsymbol{x_t}, \boldsymbol{x_{<t}})$ ▷ Identify positions where $x_t$ occurs in input sequence $x$.
    if $\neg positions$ then
        $o = \mathcal{M}_{\text{q}}(\boldsymbol{x})$
        APPEND$(\boldsymbol{x}, o)$
        continue
    end if
    similarities $\leftarrow$ []
    for each $j$ in $positions$ do ▷ Iterate over positions.
        cos_sim $\leftarrow$ COSINE_SIMILARITY$(\mathbf{H}^{(l)}_{j-1}, \mathbf{H}^{(l)}_{t-1}))$ ▷ Compute cosine similarity.
        APPEND(similarities, cos_sim)
    end for
    $j^* \leftarrow \arg\max_{\text{j}}$ similarities ▷ Find position with maximum similarity.
    $draft\_tokens, \hat{x} \leftarrow [x_{j^*+1}, \ldots, x_{j^*+\text{K}}]$ ▷ Propose draft tokens.
    $(o_0, o_1, \ldots, o_{\text{K}}) \leftarrow \mathcal{M}_{\text{q}}(x, \hat{x})$ ▷ Obtain conditional probabilities for future positions using $\mathcal{M}_{\text{q}}$ and using a sampling strategy sample new tokens.
    APPEND$(\boldsymbol{y}, o_0)$
    for $i$ in $0, \ldots,$ K do
        if $o_i$ neq $\hat{x}_i$ then
            break
        end if
        APPEND$(\boldsymbol{y}, o_{i+1})$
    end for
end for
```

Experiment	Average Acceptance Length
PLD+ (No averaging)	**3.14**
PLD+ ($m = 2$)	3.05
PLD+ ($m = 3$)	3.04
PLD+ ($m = 4$)	3.03
PLD+ ($m = 5$)	3.01
PLD+ ($m = 6$)	3.00

Table 7: Effect of using averaged hidden state representations for PLD+ (h), where m is the prefix length used for calculating the averaged representations.

the threshold θ. Setting the optimal θ value can help in reducing the number of occurences that need to be ranked in equation 3. We vary the value of θ from -1 to 1 and perform experiments on the summarization dataset using `Vicuna-7b-v1.3` as the LLM. Table 8 shows that $\theta = 0.0$ gives the best performance. As $\theta > 0.0$, the average throughput reduces indicating that the optimal draft span might have been discarded due to thresholding.

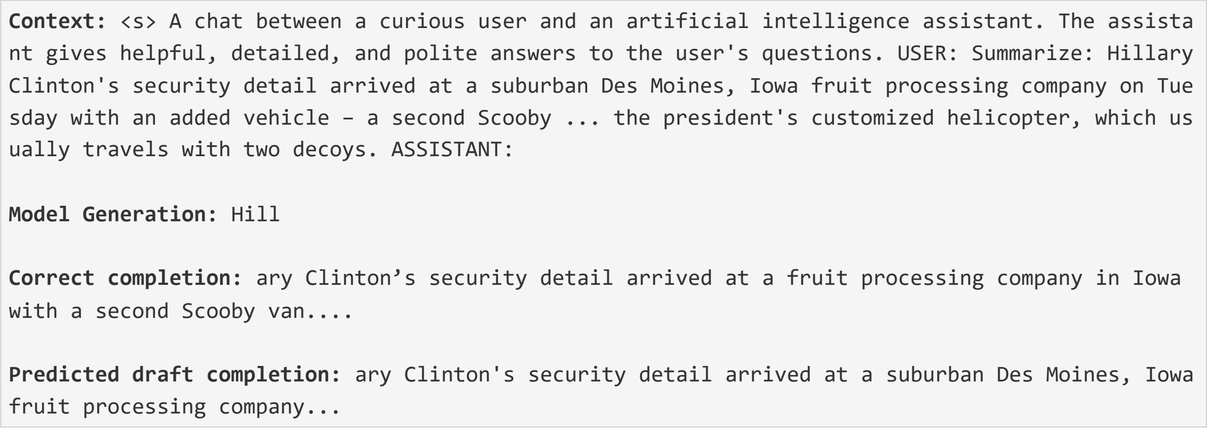
Experiment	Average Throughput (Tok/sec)
PLD+, (No thresholding)	92.26
PLD+, $\theta = -0.75$	94.44
PLD+, $\theta = -0.5$	93.79
PLD+, $\theta = 0$	**95.05**
PLD+, $\theta = 0.5$	76.86
PLD+, $\theta = 0.75$	57.73

Table 8: Effect of threshold θ on performance of PLD+ (h)

D Error Analysis

It is important to note that PLD+ proposes drafts from the input context and therefore the proposed draft completion will not align completely with the model generation as the model can paraphrase information from the input context.

Following this observation, we noted that during verification we reject draft tokens that are semantically correct, as they are not exactly equal to the model predicted token. This is illustrated in Figure 6, where the semantically correct draft subsequence "ary Clinton's security detail arrived at a"



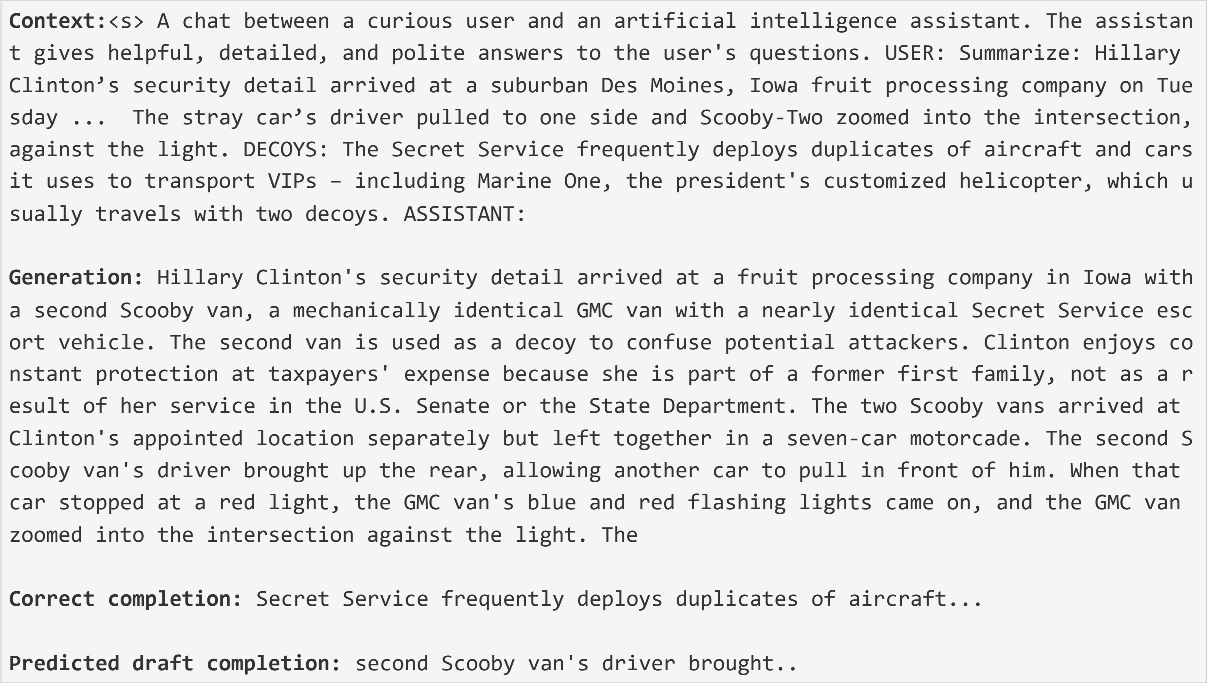
```
Context: <s> A chat between a curious user and an artificial intelligence assistant. The assista
nt gives helpful, detailed, and polite answers to the user's questions. USER: Summarize: Hillary
Clinton's security detail arrived at a suburban Des Moines, Iowa fruit processing company on Tue
sday with an added vehicle – a second Scooby ... the president's customized helicopter, which us
ually travels with two decoys. ASSISTANT:

Model Generation: Hill

Correct completion: ary Clinton’s security detail arrived at a fruit processing company in Iowa
with a second Scooby van....

Predicted draft completion: ary Clinton's security detail arrived at a suburban Des Moines, Iowa
fruit processing company...
```

Figure 6: Illustration of token rejection during verification due to mismatched punctuation. The expected token span "ary Clinton’s security detail arrived at a" was not accepted because the model predicted a different hyphen character (’ with token id 30010) than the one present in the context (' with token id 29915), leading to the rejection of an otherwise semantically correct draft completion.



```
Context:<s> A chat between a curious user and an artificial intelligence assistant. The assistan
t gives helpful, detailed, and polite answers to the user's questions. USER: Summarize: Hillary
Clinton’s security detail arrived at a suburban Des Moines, Iowa fruit processing company on Tue
sday ...  The stray car’s driver pulled to one side and Scooby-Two zoomed into the intersection,
against the light. DECOYS: The Secret Service frequently deploys duplicates of aircraft and cars
it uses to transport VIPs – including Marine One, the president's customized helicopter, which u
sually travels with two decoys. ASSISTANT:

Generation: Hillary Clinton's security detail arrived at a fruit processing company in Iowa with
a second Scooby van, a mechanically identical GMC van with a nearly identical Secret Service esc
ort vehicle. The second van is used as a decoy to confuse potential attackers. Clinton enjoys co
nstant protection at taxpayers' expense because she is part of a former first family, not as a r
esult of her service in the U.S. Senate or the State Department. The two Scooby vans arrived at
Clinton's appointed location separately but left together in a seven-car motorcade. The second S
cooby van's driver brought up the rear, allowing another car to pull in front of him. When that
car stopped at a red light, the GMC van's blue and red flashing lights came on, and the GMC van
zoomed into the intersection against the light. The

Correct completion: Secret Service frequently deploys duplicates of aircraft...

Predicted draft completion: second Scooby van's driver brought..
```

Figure 7: Candidate selection error in later steps of text generation. The algorithm has incorrectly selected the span "The second Scooby van’s driver brought.." over the contextually appropriate completion ("The Secret Service frequently deploys duplicates of aircraft...").When we try to generate candidates for frequently occurring tokens like (the, a, comma) the candidates often contain one or two ‘generated spans’ which have high similarity even though they are not the correct completion.

Table 9: Average number of words in the input for each input-guided task.

Task (# of samples)	Average # of words in input
Summarization (80)	570.08
Code Editing (80)	289.19
debug (20)	317.55
polish (20)	234.2
switch (20)	289.9
translate (20)	315.1
Text Editing (Short) (270)	190.41
Information update (120)	350.66
Style transfer (90)	66.57
Simplification (30)	52.87
Grammar correction (30)	58.5
Text Editing (Long) (86)	662.79
Multi-turn Conversation (80)	34.15
writing (10)	24.4
roleplay (10)	30.3
reasoning (10)	34.85
math (10)	23.05
coding (10)	32.35
extraction (10)	81.9
stem (10)	22.85
humanities (10)	23.5

was rejected after the quote **’**, because the model predicted a different hyphen character (**’** with token id 30010) than the one present in the context (**'** with token id 29915).

We also note that in the later steps of generation, when we try to generate candidates for frequently occurring tokens like (the, a, comma) the candidates often contain one or two ‘generated spans’ which have high similarity even though they are not the correct completion. This is illustrated in Figure 7, where the algorithm incorrectly selected the span "The second Scooby van’s driver brought.." over the contextually appropriate completion ("The Secret Service frequently deploys duplicates of aircraft...").

E Attribution

We utilise the ability of our method to point to the location in the context from where copying occurs to perform attribution. Following Phukan et al. (2024), who similar to us perform attribution of verbatim copied spans, we show results on the QuoteSum dataset (Schuster et al., 2023) in Table 10.

F Dataset details

In this section, we add additional details about the datasets used for each of the input-guided tasks. In Table 9, we report the average number of words in the input to the LLM for each task. We offer a representative sample prompt for each category in the following subsections to illustrate the tasks.

Methods	Precision	Recall	F1
GPT-4	0.96	0.87	0.90
Yi-6b (Phukan et al., 2024)	0.94	**0.99**	**0.96**
PLD+(h), threshold=2	0. 96	0.96	**0.96**
PLD+(h), threshold=5	0.98	0.85	0.90
PLD+(h), threshold=8	**0.99**	0.7	0.79

Table 10: Token level P, R & F1 scores for identifying output tokens copied from the context on QuoteSum test set

F.1 Summarization

```
Summarize: Hillary Clintons security
    detail arrived at a suburban Des
    Moines, Iowa fruit processing
    company on Tuesday with an added
    vehicle  a second Scooby. After her
    signature oversize black Chevy
    conversion van dropped her off at
    Capitol Fruit Company in Norwalk,
    Iowa, a visually identical GMC van
    drove up to the building with a
    nearly identical Secret Service
    escort vehicle. Both armored
    vehicles have raised roofs,
    deep-tinted windows and New York
    license plates...
```

F.2 Code Editing

```
### Instruction:
Please correct the errors in the buggy
    code snippet below, ensuring that
    your corrected code adheres to the
    specified programming language
    syntax and logic requirements.
    Validate your solution against the
    provided test\ncases to ensure its
    accuracy. Note that your solution
    should strictly consist of the
    corrected code only. Generate only
    the required code enclosed in "‘‘‘"
### Question:
Below is the java buggy code:
class Solution {\n    public int
    numberOfBeams(String[] bank) ...
Correct the code and ensure it passes
    the following test case:\nInput:
    bank = [ "011001 ", "000000",
    "010100", "001000"] Output:  8
### Answer:
```

F.3 Text Editing (Short)

```
Below is an instruction that describes
    a task, along with an input text.
    Please edit the input text based on
    the instruction. Your response
    should only include the edited
    output.
# Instruction:
Correct the grammar error in the
    sentence.
# Input:
```

```
When we talk about the so-called value
    of a product , we envision a
    scenario where dozens of products
    are available in supermarket
    shelves and when you switch on a
    television , there is an endless
    stream of commercials , each
    claiming exciting new features
    about the products advertised .
Response:
```

F.4 Text Editing (Long)

```
Essay: The article "Top 20 Pros and
    Cons Associated With
    Self-Driving Cars", provides an
    in depth look into the
    reasoning of whether or not we,
    as a society, should adopt
    self-driving vehicles.  Of the
    20 "pros" listed, an increase
    in safety was likely the
    leading factor, but also
    included were the benefits in
    commute times and city travel,
    the possibility for the driver
    ...
Feedback:  Your draft has been
    read, and feedback from an
    expert writing instructor is
    written below. We advise that
    you use this feedback when you
    revise.
The strengths of your essay include:
All of your sentences are clear
    because of word choice and
    sentence structure.
You respond to one, but not all
    parts of the prompt. However,
    your entire essay is focused on
    the prompt.
You provided specific and
    convincing evidence for each
    claim, and most evidence is
    given through relevant direct
    quotations or detailed examples
    from the provided reading.
Areas to improve in your essay
    include:
You provided a statement that
    somewhat show your stance for
    or against self-driving cars,
    but it is unclear, or is just a
    restatement of the prompt.
You made multiple, distinct, and
    clear claims that aligned with
    either your thesis, or the
    given reading, but not both.
Revise the essay using the expert
    feedback.
```

F.5 Multi-turn Conversation

```
TURN 1: Compose an engaging travel
    blog post about a recent trip
    to Hawaii, highlighting
    cultural experiences and
    must-see attractions.
TURN 2: Rewrite your previous
    response. Start every sentence
    with the letter A.
```