Dynamic Speculation Lookahead Accelerates Speculative Decoding of Large Language Models

Jonathan Mamou[♤] **Oren Pereg**[♤] **Daniel Korat**[♤]
Moshe Berchansky[♤] **Nadav Timor**[◊] **Moshe Wasserblat**[♤] **Roy Schwartz**[♧]
[♤] Intel Labs, Israel
[◊] Weizmann Institute of Science, Israel
[♧]School of Computer Science & Engineering, Hebrew University of Jerusalem
Correspondence: jonathan.mamou@intel.com

Abstract

Speculative decoding is commonly used for reducing the inference latency of large language models. Its effectiveness depends highly on the *speculation lookahead* (*SL*)—the number of tokens generated by the draft model at each iteration. In this work we show that the common practice of using the same SL for all iterations (*static SL*) is suboptimal. We introduce DISCO (**D**ynam**I**c **S**pe**C**ulation lookahead **O**ptimization), a novel method for dynamically selecting the SL. Our experiments with four datasets show that DISCO reaches an average speedup of 10% compared to the best static SL baseline, while generating the exact same text.

1 Introduction

Large language models (LLMs) generate tokens autoregressively [Radford et al., 2019, Brown et al., 2020], which often leads to slow generation. Speculative decoding algorithms Leviathan et al. [2023], Miao et al. [2024], Chen et al. [2023] reduces the inference latency by splitting the inference of a given model into two steps. First, a fast *draft* model generates tokens autoregressively. Then a more accurate (*target*) model validates all generated draft tokens simultaneously. See Fig. 1 for an example. The effectiveness of speculative decoding depends on the *speculation lookahead* (*SL*)—the number of tokens generated by the draft model at each iteration. An SL too small leads to too many target forwards steps; SL values too large add redundant draft forward passes. Yet, existing speculative decoding approaches use a *static SL*—an SL that remains constant across all iterations [Leviathan et al., 2023, Miao et al., 2024, Chen et al., 2023, Cai et al., 2024, Li et al., 2024]. This work starts by defining an oracle SL—a method that assigns each iteration its optimal SL. We observe that the optimal SL shows a high variance across iterations (Fig. 2). We then use this oracle to estimate an upper bound of the expected speedup (compared to using static SLs), showing a potential gain of up to 39% speedup. We then propose *DISCO*, a novel method for selecting the SL before each iteration. DISCO estimates the likelihood of the next draft token being accepted by the target model, and halts the draft model if this likelihood is too small. We evaluate DISCO across various tasks: code generation, text summarization, and instruction following. Our results show an average speedup of 10% compared to optimal static SL and 31% compared to a previously known heuristic for controlling SLs Gante [2023], all without modifying the output text (App. F). DISCO also transfers well across tasks from the same category: training it on one task and using it on another leads to similar speedups.

2 Background: Speculative Decoding

Speculative decoding expedites LLM generation while ensuring no accuracy loss by dividing it into two stages. In the first stage, a fast but less accurate *draft* model M_D autoregressively generates

4th NeurIPS Efficient Natural Language and Speech Processing Workshop (ENLSP-IV 2024).

a sequence of tokens. In the second stage, a large but more accurate *target* model M_T conducts parallelized verification over the generated draft tokens. This process allows the model to potentially produce multiple tokens per target forward pass.

Speculation lookahead (SL) The effectiveness of speculative decoding in accelerating the token generation process relies heavily on the SL parameter, which determines how many tokens are generated by the draft model before each validation step. The effect of the SL on the overall speedup is subject to a tradeoff; higher SLs potentially reduce the number of target model validations, but also increase the number of redundant draft generations (Fig. 1), and vice-versa. The majority of speculative decoding approaches use a static SL—the same number of draft tokens are generated per speculative iteration.[1] Chen et al. [2023] explored various static SLs, across different target-draft model pairs and tasks, and empirically showed that as the SL rises, the overall speedup increases until reaching a certain threshold, beyond which it either levels off or even regresses. To study the effect of the SL, Leviathan et al. [2023] defined the improvement factor (IF) as the expected token generation speedup:

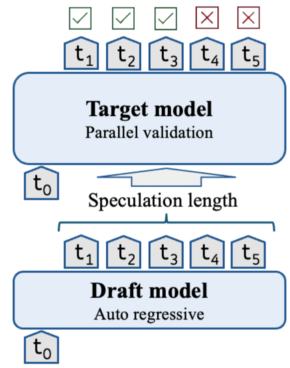


Figure 1: An illustration of a single speculative decoding iteration with Speculation Lookahead (SL) = 5. Given a prompt t_0, a draft model autoregressively generates 5 tokens $t_1, \ldots, t_5$. The target model validates them all in parallel and accepts only t_1, t_2, t_3. As t_4 and t_5 are rejected, the SL is suboptimal (too large).

$$IF = \frac{1 - \alpha^{\gamma+1}}{(1 - \alpha)(\gamma c + 1)} \tag{1}$$

where α denotes the acceptance rate, indicating the expected probability of a draft token to be accepted by the target model; c represents the cost coefficient, indicating the ratio between the walltime of a forward pass run of the draft model M_D and the wall time of a forward pass run of the target model M_T; and γ represents the static SL value.[2] While both α and c are important to the selection of the target-draft model pair, finding the optimal γ is fundamental to the effectiveness of the system.

3 Dynamic Speculation Lookahead

The IF function (Eq. (1)) is based on the simplifying assumption that the probability of accepting draft tokens by the target model is i.i.d. Nevertheless, in practical scenarios, different tokens may have varying levels of predictability, which challenges this i.i.d. assumption, and suggests that using a static SL might be suboptimal. Below we consider an oracle experiment, which applies the optimal dynamic γ value at each iteration. We then propose a method for dynamically setting γ, showing that it strongly outperforms a static selection method for any choice of a static SL.

Finding the optimal SL per iteration We start by employing an oracle for detecting the optimal value of SL (γ) for each speculative iteration. The oracle uses the draft model to autoregressively generate tokens until a mismatch occurs between the predicted tokens of the draft and target models. This process is repeated for each speculative iteration, ultimately returning the optimal (maximum) number of accepted draft tokens per iteration. The mismatch between the tokens is determined by using the rejection sampling algorithm introduced by Leviathan et al. [2023] with zero temperature. This oracle fulfills the speculative decoding potential: generating the maximal number of valid draft tokens at each iteration, and making a minimal number of calls to both draft and target model. Figure 2 shows the oracle SL values across the speculative iterations for one MBPP example. Compared to the static SL, we observe a lower number of both draft and target forward passes. Figure 3 shows the average oracle SL over the speculative iterations for the Alpaca dataset Taori et al. [2023]. Both figures show a high variance of oracle SL values, implying that astatic SL is likely to be suboptimal. See App. A for further analysis.

[1]A notable exception is Gante [2023], who applies a heuristic for dynamic SL adaptation by modifying the SL based on the acceptance rate of previous iterations.

[2]IF computation assumes enough compute resources for increased concurrency as γ rises.

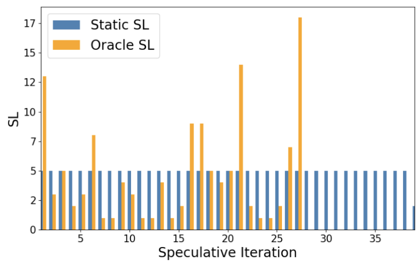


Figure 2: Oracle and static SL values for different speculative iterations on one MBPP example. For static SL, we run 38 target forward passes and 192 draft forward passes, while for oracle SL, we only run 27 target forward passes and 129 draft forward passes. We observe a high variance of oracle SL values.

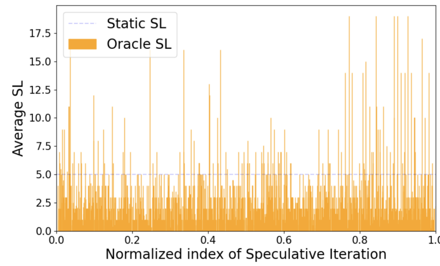


Figure 3: The average oracle SL over the normalized index of the speculative iterations for the Alpaca dataset. We observe a high variance of oracle SL values.

DynamIc SpeCulation lookahead Optimization We introduce DISCO, a simple method for dynamically setting the SL value at each iteration. To estimate the correct SL value at each step, we employ a simple classifier as follows. Immediately after generating any draft token, our classifier decides whether the draft model M_D should proceed and generate the next token or switch to the target model M_T for verification. The classifier takes as inputs the probability vector of the draft model (y_i^D) and the token position (i), generating a confidence score (C_i) used for the decision-making as follows:

$$C_i = FFN(Concat(Top_k(y_i^D), Ent(y_i^D), i)) \tag{2}$$

where $Top_k()$ selects the top k values and $Ent()$ is the entropy function.[3] At inference time, C_i is compared against a predetermined threshold τ to decide whether the draft model should continue to generate the next token or turn to the target model for validation. In addition, we limit the number of draft generated tokens to SL_{max}.[4] Note that our method adapts the rejection sampling algorithm which preserves the distribution of M_T, thus ensuring no quality degradation.

4 Experiments

Datasets and Models We evaluate our method on four datasets spanning three tasks: code generation using MBPP Austin et al. [2021] and HumanEval Chen et al. [2021]; text summarization using CNN-DailyMail Nallapati et al. [2016]; and instruction-following using Alpaca. We use the training sets for training the SL classifier and the validation sets for setting the threshold τ and the SL_{max} hyperparameter. For HumanEval, which has no training and validation sets, we evaluate transfer learning from MBPP. For code generation tasks, we use the `Starcoder` model family Li et al. [2023]—15B for target and 168M for draft. For the other tasks, we use `Vicuna` models—13B as target Chiang et al. [2023] and 68M as draft Yang et al. [2024]. See Apps. B and C for more details.

SL classifier training To train the classifier we extract features from the training sets of our datasets MBPP, CNN-DM, and Alpaca. For minimal overhead, we use a shallow 2-layer FFN classifier and train it based on the extracted features to predict the agreement between the draft and target models. The training employs cross-entropy loss with Total Variance as distance measure: $TV(y_i^D, y_i^T)$, where y_i^D and y_i^T represent the vocabulary distribution of M_D and M_T respectively at the i^{th} token position. We evaluate the quality of the classifier by measuring its F1 score on the validation set. The F1 results obtained on the datasets are relatively high; for instance, 95% on MBPP, compared to 85% using the optimal static SL. See Apps. D and E for more details.

Baselines and Results We compare the LLM inference latency of DISCO to both static SL (*static SL-5*) and dynamic heuristic SL setups (*dynHeur SL*; Gante, 2023).

[3]We use $k = 10$ in all experiments.

[4]SL_{max} enables optimized execution with LLMs using fixed tensor shapes.

We also consider the optimal static SL baseline tuned on our validation sets (*static SL-opt*). Finally, we also report results for our oracle (Sec. 2), which represents the lower bound on latency. Table 1 presents our results using the rejection sampling scheme with greedy decoding (temperature=0) since baselines get higher speedup Leviathan et al. [2023]. Employing an SL classifier consistently outperforms all other baselines across all benchmarks. Average latency improvements of DISCO over the optimal static SL and the dynamic heuristic baselines are 10.3% and 31.4% respectively, while preserving the same output as the target model. Importantly, our improvement does not come only from our training data: the optimal static SL (as fit by that data) is still underperformed by DISCO. Finally, DISCO transfers well across tasks: when trained on MBPP, it is still outperforms all baselines on HumanEval. See App. G for further analysis.

Benchmark	Method	Latency	Speedup
MBPP	Target	23.21	1.00x
	dynHeur SL	20.07	1.16x
	static SL-5	15.88	1.46x
	static SL-opt	14.16	1.64x
	DISCO (ours)	12.58	**1.84x**
	oracle	10.18	2.28x
HumanEval (transfer learning)	Target	23.46	1.00x
	dynHeur SL	22.57	1.04x
	static SL-5	14.43	1.63x
	static SL-opt	14.42	1.63x
	DISCO (ours)	12.78	**1.84x**
	oracle	10.59	2.22x
CNN-DM	Target	38.29	1.00x
	dynHeur SL	21.18	1.81x
	static SL-5	19.74	1.94x
	static SL-opt	20.66	1.85x
	DISCO (ours)	17.85	**2.15x**
	oracle	15.41	2.48x
Alpaca	Target	47.67	1.00x
	dynHeur SL	31.83	1.50x
	static SL-5	23.79	2.00x
	static SL-opt	23.65	2.02x
	DISCO (ours)	22.49	**2.12x**
	oracle	20.04	2.38x

Table 1: Average latency results (in milliseconds) on different benchmarks. HumanEval results use a classifier trained on MBPP (transfer learning). All results are provided with greedy decoding (temperature=0).

5 Related Work

Pioneering studies on speculative decoding Leviathan et al. [2023], Chen et al. [2023] introduced a rejection sampling scheme that preserves the distribution of the target model, guaranteeing that speculative decoding maintains the quality of the target model. Subsequent work Miao et al. [2024] elevated the average number of accepted tokens by using several draft models. Most recently, Timor et al. [2024] introduced DSI, a distributed variation of speculative decoding that is provably faster than non-distributed methods and does not require additional training or architectural changes. To eliminate the need for a separate draft model, Li et al. [2024], Cai et al. [2024], Bhendawade et al. [2024], Yang et al. [2024] train additional, specialized draft layers on top of the transformer decoder. DISCO transfers well within domains and does not require classifier training per dataset whereas these methods necessitate training per dataset. At inference time, they employ a static SL; we believe that DISCO can be beneficially applied to these approaches, we leave this research for future work. Zhang et al. [2023] proposed draft-exiting with an adaptive threshold for self-speculative decoding using a rule-based approach that compares a confidence to a predetermined threshold. This method seems suitable to approaches where the draft is a subset of the target model, whereas our approach is more generic. A very recent concurrent work by S et al. [2024] enhanced the draft model's accuracy by granting it access to the target model's representations. In addition, it employed a classifier to determine whether to halt or continue the speculation process. Our work delves into the impact of the SL on the efficiency of speculative decoding, encompassing comparisons between static and dynamic SL approaches, as well as the upper bound of improvement represented by the oracle SL.

6 Conclusion

We have shown that using the same speculation lookahead parameter across speculative decoding iterations is suboptimal. We introduced DISCO, a dynamic speculation lookahead optimization method. The method uses a classifier that determines whether the draft model should continue to generate the next token or pause and transition to the target model for validation. We evaluated DISCO's effectiveness using four benchmarks and demonstrated average speedup gains of 10.3% and 31.4% relatively to the optimal static SL and dynamic heuristic baselines. Our results highlight the potential of further reducing inference cost by using simple, efficient techniques.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL `https://arxiv.org/abs/2108.07732`. arXiv:2108.07732.

Loubna Ben Allal, Niklas Muennighoff, Logesh Kumar Umapathi, Ben Lipkin, and Leandro von Werra. A framework for the evaluation of code generation models. `https://github.com/bigcode-project/bigcode-evaluation-harness`, 2022.

Nikhil Bhendawade, Irina Belousova, Qichen Fu, Henry Mason, Mohammad Rastegari, and Mahyar Najibi. Speculative streaming: Fast llm inference without auxiliary models, 2024. URL `https://arxiv.org/abs/2402.11131`. arXiv:2402.11131.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024. URL `https://arxiv.org/abs/2401.10774`. arXiv:2401.10774.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL `https://arxiv.org/abs/2302.01318`. arXiv:2302.01318.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL `https://arxiv.org/abs/2107.03374`. arXiv:2107.03374.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality, 2023. URL `https://lmsys.org/blog/2023-03-30-vicuna/`.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen tau Yih, Luke Zettlemoyer, and Mike Lewis. Incoder: A generative model for code infilling and synthesis. In *Proc. of ICLR*, 2023. URL `https://arxiv.org/abs/2204.05999`.

Joao Gante. Assisted generation: a new direction toward low-latency text generation, 2023. URL `https://huggingface.co/blog/assisted-generation`.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proc. of ICML*, 2023. URL `https://arxiv.org/abs/2211.17192`.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. StarCoder: may the source be with you!, 2023. URL https://arxiv.org/abs/2305.06161. arXiv:2305.06161.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty, 2024. URL https://arxiv.org/abs/2401.15077. arXiv:2401.15077.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating generative llm serving with speculative inference and token tree verification. In *Proc. of ASPLOS*, 2024. URL https://api.semanticscholar.org/CorpusID:258740799.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In Stefan Riezler and Yoav Goldberg, editors, *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/K16-1028. URL https://aclanthology.org/K16-1028.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Aishwarya P S, Pranav Ajit Nair, Yashas Samaga, Toby Boyd, Sanjiv Kumar, Prateek Jain, and Praneeth Netrapalli. Tandem transformers for inference efficient llms, 2024. URL https://arxiv.org/abs/2402.08644. arXiv:2402.08644.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Alpaca: A strong, replicable instruction-following model. Stanford Center for Research on Foundation Models, 2023. https://crfm.stanford.edu/2023/03/13/alpaca.html.

Nadav Timor, Jonathan Mamou, Daniel Korat, Moshe Berchansky, Oren Pereg, Moshe Wasserblat, Tomer Galanti, Michal Gordon, and David Harel. Distributed speculative inference of large language models. *arXiv preprint arXiv:2405.14105*, 2024.

Sen Yang, Shujian Huang, Xinyu Dai, and Jiajun Chen. Multi-candidate speculative decoding, 2024. URL https://arxiv.org/abs/2401.06706. arXiv:2401.06706.

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

Datasets	Target Model	Draft Model	oracle SL
MBPP	`Starcoder-15B`	`Starcoder-168M`	18.0 (± 46.2)
HumanEval	`Starcoder-15B`	`Starcoder-168M`	14.7 (± 42.6)
CNN-DM	`Vicuna-13B`	`Vicuna-68M`	3.2 (± 4.0)
Alpaca	`Vicuna-13B`	`Vicuna-68M`	2.4 (± 2.1)

Table 2: Average (and STD) of the oracle SL as determined by the oracle per dataset and target/draft models.

Datasets	Train	Validation	Test
MBPP	374	80	80
HumanEval	-	-	80
CNN-DM	500	80	80
Alpaca	500	80	80

Table 3: Number of samples per dataset and split.

A Oracle SL Analysis

Table 2 shows the oracle SL expectancy and standard deviation of the oracle measured on different datasets and models and Fig. 4 shows the probability distribution of the oracle SL for the different datasets. We observe a high variance of SL values.

We hypothesized that later tokens are more predictive but eventually found only a relatively weak correlation, as Fig. 2 and Fig. 5 show. The figures are bar charts of the average oracle SL over the *normalized index* of the Speculation Iteration. We calculate the bars as follows. For each prompt of a dataset, we have its corresponding sequence of oracle SLs. The length of the sequence is equal to the number of Speculation Iterations. For example, consider a prompt with oracle SLs $\langle 7, 3, 13, 21, 8 \rangle$. Its normalized index is $\langle 0, 0.25, 0.5, 0.75, 1 \rangle$. The bars represent the average oracle SL of buckets of size 0.0001.

B Datasets and Prompts Details

We use standard datasets from Hugging Face and standard prompts from the state-of-of-the-art. Tab. 3 summarizes the composition of the datasets. We provide more details per dataset in the next sections.

B.1 MBPP

For MBPP, we use the 'train', 'validation' and 'test' splits of the 'full' subset. The whole 'train' split is used for training, while 80 randomly selected samples of the 'validation' and 'test' splits are respectively used for validation and test. MBPP is distributed under the cc-by-4.0 License.

Concerning the prompt, we followed Ben Allal et al. [2022], Fried et al. [2023] and included the description of the programming task and a single test to verify solution, in order to help the model catch the signature of the function (see Fig. 6).

B.2 HumanEval

HumanEval dataset contains a single subset with a single split ('test' split). We use 80 randomly selected samples of that split for test. Note that since we evaluate transfer learning from MBPP, we don't need HumanEval training and validation sets. HumanEval is distributed under the MIT License.

Prompt contains only `prompt` field from the dataset.

B.3 CNN-DM

For CNN-DM, we use the 'train', 'validation' and 'test' splits of the '2.0.0' subset. 500 randomly selected samples of the 'train' split is used for training, while 80 randomly selected samples of the

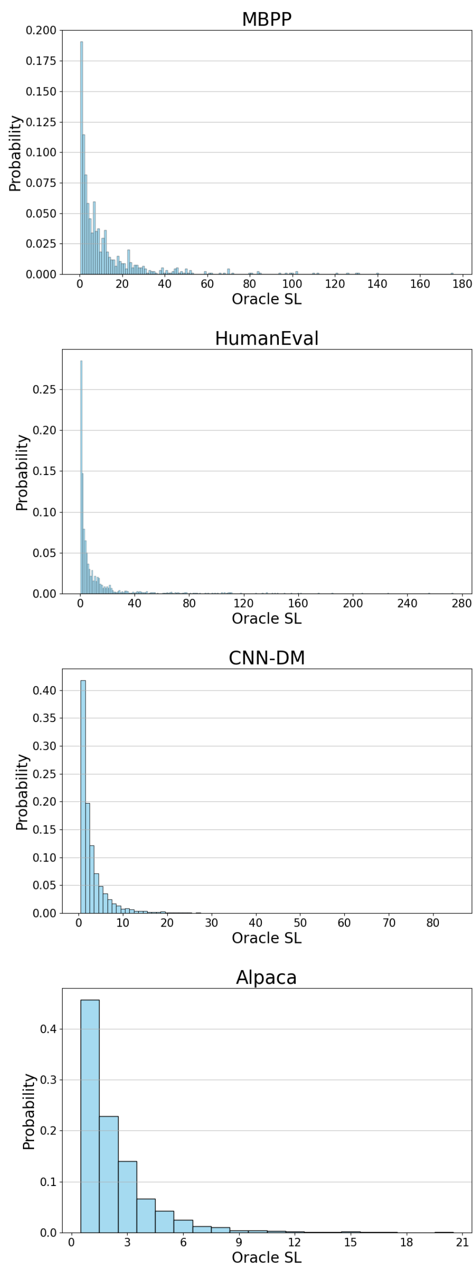
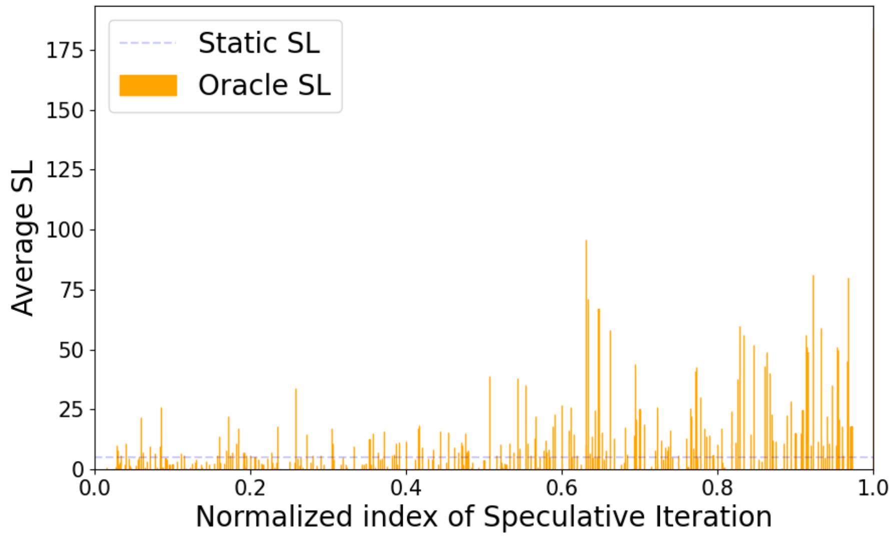
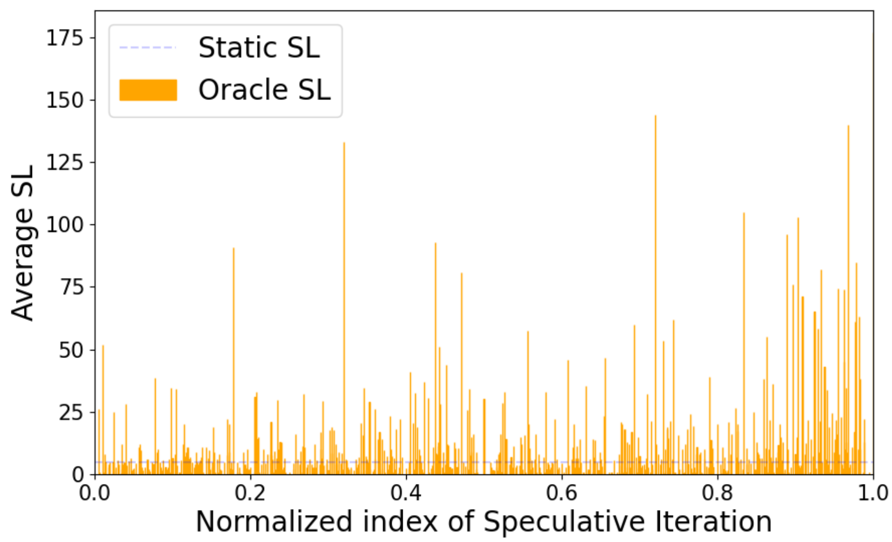


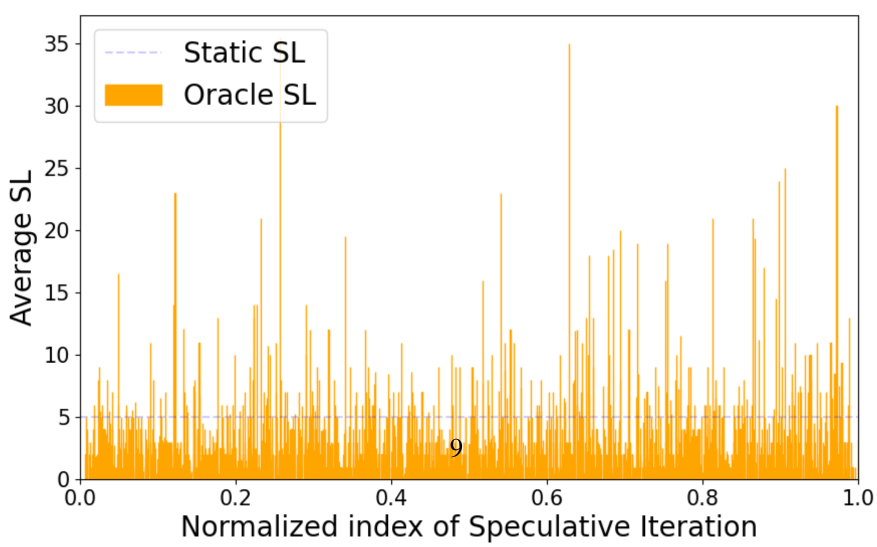
Figure 4: Oracle SL probability histogram on the different datasets. We observe a high variance of SL values.



(a) MBPP



(b) HumanEval



(c) CNN-DM

```
"""{text}
{test_list[0]}
"""
```

Figure 6: MBPP Prompt

```
"""Summarize:
{article}
Summary:
"""
```

Figure 7: CNN-DM Prompt

'validation' and 'test' splits are respectively used for validation and test. CNN-DM is distributed under the Apache License 2.0.

We included the `article` field in the prompt as in Fig. 7.

B.4 Alpaca

Alpaca dataset contains a single split ('train' split). As for CNN-DM, 500 randomly selected samples of the 'train' split is used for training, while 80 randomly selected samples of the 'validation' and 'test' splits are respectively used for validation and test. Alpaca is distributed under the cc-by-nc-4.0 License.

We follow Taori et al. [2023] to define the prompts. For samples with a non-empty input field, we use the prompt as in Fig. 8 while for samples with empty input field, we use the prompt as in Fig. 9.

C Models

For all models, we retrieve model weights from Hugging Face. For clarity and reproducibility, we provide the URLs for each model used:

- `Vicuna-13B`: `https://huggingface.co/lmsys/vicuna-13b-v1.3`, distributed under Non-Commercial License.

```
"""Below is an instruction that describes a
task, paired with an input that provides
further context. Write a response that
appropriately completes the request.

### Instruction:
{instruction}

### Input:
{input}

### Response:
"""
```

Figure 8: Alpaca prompt for samples with a non-empty input field.

```
"""Below is an instruction that describes a
task. Write a response that appropriately
completes the request.

### Instruction:
{instruction}

### Response:
"""
```

Figure 9: Alpaca prompt for samples with empty input field.

Datasets	F1 static-SL	F1 DISCO
MBPP	85	95
CNN-DM	76	88
Alpaca	68	77

Table 4: Classifier F1 scores for both static SL and DISCO.

- `Vicuna-68M`: `https://huggingface.co/double7/vicuna-68m`, distributed under the Apache License 2.0.
- `Starcoder-15B`: `https://huggingface.co/bigcode/starcoder`, distributed under the Responsible AI License.
- `Starcoder-168M`: `https://huggingface.co/bigcode/tiny_starcoder_py`, also distributed under the Responsible AI License

D Classifier

D.1 Feature Extraction from Training Data for Classifier Training

To train the model for each dataset, MBPP, CNN-DM, and Alpaca, we used the corresponding training set of each dataset. For each token in each training set, we extracted a boolean label (accepted/rejected) and a list of features in the following manner: We ran the target model using the standard autoregressive approach for generating tokens based on the input prompt. In contrast, the draft model iteratively generated only a single token per iteration. During each iteration, the draft model generated this token based on the concatenation of the input prompt and the tokens subsequently generated by the target model. Draft tokens that resembled the target token at the same position were labeled "accepted," while others were labeled "rejected." Corresponding features were extracted for both draft and target tokens, encompassing the top-k probabilities of the vocabulary distribution, the entropy associated with these probabilities, and the token position value counted from the beginning of the generation process.

D.2 Classifier F1 Results

We evaluate the quality of the classifier by measuring its F1 score on the validation set and report in Tab. 4 F1 scores for both static SL and DISCO. F1 scores of DISCO always outperforms F1 scores of static SL.

F1 score measures the accuracy of the classifier in predicting the speculative length (SL) but does not account for how well the predictions align with the oracle's behavior in reducing latency. In particular, F1 is influenced by 2 different types of error FP and FN that have a different impact on the speedup. FP Errors lead to unnecessary speculative execution, which wastes resources but might not drastically reduce overall speedup. FN Errors lead to missed opportunities for speedup, having a more severe impact on overall latency reduction.

Dataset	static SL-opt	dynHeur SL
MBPP	11.2	37.3
HumanEval	11.4	43.4
CNN-DM	13.6	15.7
Alpaca	4.9	29.3
Average	**10.3**	**31.4**

Table 5: The latency improvement (percentage) of DISCO over static SL-opt and dynHeur SL across four datasets.

Benchmark	Method	Latency(ms)	Speedup
CNN-DM	Target	36.32	1.00x
	static SL-5	21.88	1.66x
	DISCO (ours)	19.96	**1.82x**
	ppl SL-opt	26.79	1.43x

Table 6: Additional average latency results for CNN-DM: temperature=1; perplexity-based baseline.

E Additional Implementation Details

Our implementation is based on the Transformers library of HuggingFace, distributed under the Apache License 2.0, and PyTorch Deep Learning library, distributed under the BSD License (BSD-3). Our code will be available upon publication under the Apache License 2.0.

For every dataset, DISCO classifier is trained on the train set; threshold τ and SL_{max} hyper-parameters are fine-tuned on the validation set optimizing the latency. Optimal static SL is estimated on the validation set. Latency results are reported on the test set.

All our experiments are run on a single A100 80GB GPU.

F Additional Results

Table 5 shows the percentage of improvement in latency of DISCO over static SL-opt and dynHeur SL baselines. The numbers are calculated based on the latency results shown in Table 1

G Further Latency Results Analysis

Concerning SL-opt and SL-5 speedup values on CNN-DM reported in Tab. 1, note that the optimal static SL is tuned on the validation set while the latency and speedup reported numbers are on the test set. This is similar to our setup, where the classifier is tuned on the validation set, and similarly reported on the test set. We observe that for CNN-DM, the optimal SL on the validation set is not as good as the SL-5 on the test set.

Since DISCO is sampling temperature agnostic, we present in Tab. 6 latency results for CNN-DM with non-zero temperature; we observe that DISCO speedup also improves in that case. In addition, we report latency results using a simple rule-based approach: we use the perplexity to measure the confidence in the sequence of tokens predicted by the draft, when lower perplexity indicates higher confidence. Optimal perplexity threshold is tuned on our validation sets (*ppl SL-opt*). We observe that it yields a lower speedup.