FACULTY OF ENGINEERING AND ARCHITECTURE

Context-Aware Speculative Decoding for Retrieval Augmented Generation with a case study at UZGent

Gianni Van de Velde
Student number: 02008064

Supervisors: Prof. dr. ir. Thomas Demeester, Prof. dr. ir. Chris Develder
Counsellor: Ir. Cédric Goemaere

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2024-2025

GHENT UNIVERSITY

Preface

For this thesis, I had the opportunity to write my own proposal. I wanted to work on the energy efficiency of LLMs and more specifically in RAG scenarios. Needless to say, I am very excited about the topic. There are three passions of mine, that I could combine in this thesis: a passion for AI, efficiency and environmental friendliness.

More people helped me with this paper than I can list here. I would like to thank all of them. Especially, I would like to thank my counselor ir. Cédric Goemaere and promotors for this thesis. They gave me much freedom while researching, while still following closely and guiding me with valuable and actionable feedback. Also, Christiaan, Julie, Tom and Corine of UZGent deserve a special mention, because they all put significant effort into helping me get results in time. The project at UZGent could have easily taken years if it had not been for the great efforts they made. Lastly, I would like to thank my friends with whom I shared countless hours on the 11th floor of IGent, each working on their own thesis. Certainly Arthur, Bram and Quinten helped much by thinking and discussing with me, and they also helped me to forget about my thesis for a second during breaks.

Permission of use on loan

The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In all cases of other use, the copyright terms have to be respected, in particular with regard to the obligation to state explicitly the source when quoting results from this master dissertation.

Gianni Van de Velde, May 2025

Explanatory note

This master's dissertation is part of an exam. Any comments formulated by the assessment committee during the oral presentation of the master's dissertation are not included in this text.

Context-Aware Speculative Decoding for Retrieval Augmented Generation with a case study at UZGent

Gianni Van de Velde

Supervisors: Prof. dr. ir. Thomas Demeester, Prof. dr. ir. Chris Develder

Counsellor: Cédric Goemaere

Master's dissertation submitted in order to obtain the academic degree of Master of Science in Computer Science Engineering

Ghent University

Academic year 2024-2025

Abstract

In recent years, Large Language Models (LLMs) have taken over the world, leading to spectacular applications but also concerns about their energy consumption. In an ideal world, the models would provide the same answers faster, while using less energy. This thesis works towards that ideal world. Speculative decoding is a proven method to increase time and energy efficiency for LLMs and this thesis builds further on the current state-of-the-art (SOTA) to achieve even more efficiency. We present Context-Aware Speculative Decoding (CASD), a hybrid method that can augment any other speculator to achieve better performance in high-copy environments. Retrieval-Augmented Generation (RAG) is a technique that is very popular in the business world and it yields such a high-copy environment. However, limited benchmarks are available for RAG and certainly in niche domains (e.g. medical, Dutch), where the SOTA underperforms. For this reason, we deliver a new benchmark UZGentRAG with data from a real world use case: RAG at UZGent. This benchmark not only proves the strength of CASD, but also the gap that current methods seem to leave open. In the future, we see great potential for CASD to augment speculative decoders in RAG use cases, certainly in low-resource languages.

Keywords

{Speculative Decoding, RAG, LLM}

Context-Aware Speculative Decoding: Accelerating Inference in High-Copy Environments

Gianni Van de Velde
gianni.vandevelde@ugent.be

Cédric Goemaere
cedric.goemaere@ugent.be

Chris Develder
chris.develder@ugent.be

Thomas Demeester
thomas.demeester@ugent.be

Abstract

In recent years, Large Language Models (LLMs) have proven to be useful for many tasks, leading to wide adoption. Yet, in pace with the increasing model sizes, energy consumption went up, whereas inference speeds went down. Speculative decoding is a proven method that can alleviate both issues. It parallelizes the decoding of the next tokens by speculating multiple decoding steps. Many speculators have been made to try and predict what the generator might generate next. For some applications, the generator may literally copy phrases from the context. Building on this intuition, this paper suggests a new method: Context-Aware Speculative Decoding (CASD), which is a hybrid method that augments the current state-of-the-art (SOTA) with a simple context copying mechanism. Tests on Llama 3 (Grattafiori et al., 2024) using EAGLE-2 (Li et al., 2024) show that CASD speeds up decoding with 8-17% for the Retrieval-Augmented Generation (RAG) tasks. This while adding limited overhead, seen by a slowdown of only 4% when applied to general benchmarks. In the future, we see great potential for CASD to augment speculative decoders in RAG use cases, certainly in low-resource languages.

1 Introduction

In recent years, the scaling of Large Language Models (LLMs) improved their capabilities tremendously. However, growing model sizes comes at a cost. Among others, inference time goes up and so does the energy consumed at inference. To compensate for these added costs, speculative decoding emerged as a way to break the autoregressive nature of generating text with an LLM. Speculative decoding tries to parallelize LLM calls, to reduce the time and energy necessary to generate an answer (Qin et al., 2024). In practice, this means that a small model drafts many samples of what the generator could generate in the coming tokens. These drafts are then verified in parallel, in one forward pass of the model. So it becomes possible to generate multiple tokens with only one forward pass. This method has already been shown to reduce inference time and energy.

Many small draft models have been proposed to guess the next generated tokens. There are two large categories of draft models: statistical models (He et al., 2023; Yang et al., 2023) and neural models (Leviathan et al., 2023; Cai et al., 2024; Li et al., 2024). This paper presents a new model-agnostic statistical model: Context-Aware Speculative Decoding (CASD). It is naturally suited to augment neural models, and as such achieves a new state-of-the-art (SOTA) for specific use cases. To the best of our knowledge, we are the first to combine the drafts of statistical and neural models.

CASD leverages the context given in the prompt to make good speculations. It checks whether the LLM is copying text from the context and if so, it provides the continuation as found in the context. This method works best if the generator is expected to copy many passages from the context. With the recent popularity of Retrieval-Augmented Generation (RAG) applications, this is a reasonable assumption. We test CASD's performance by using it to augment the current SOTA, EAGLE-2 (Li et al., 2024). For the RAG benchmarks, CASD showed a 1.08 speedup on SQuAD (Rajpurkar et al., 2016) and 1.17 on our private UZGentRAG benchmark compared to EAGLE-2. CASD was also tested on the widely used benchmarks for speculative decoding: Alpaca (Taori et al., 2023), GSM8K (Cobbe et al., 2021), HumanEval (Chen et al., 2021), MT-bench (Zheng et al., 2023), Natural Questions a.k.a. QA (Kwiatkowski et al., 2019) and CNN/DM (Nallapati et al., 2016). Since these general speculative decoding benchmarks do not use RAG, we naturally see no inference speedup. Yet, we validate that even in these cases, the overhead introduced by adopting CASD remains minimal.

Besides the added performance in RAG-based

use cases, CASD offers the following advantages:

- **LLM agnostic.** While modern speculators are usually trained to work well with one LLM, CASD can be plugged into any LLM for generating long, high quality speculations.
- **Independent predictions.** Today, neural models are the standard of speculative decoding. These models typically have good performance for the initial upcoming tokens, but each step accumulates noise compared to what the actual LLM would output. In contrast, CASD works in a fundamentally different way, also generating high quality long length speculations. This is a built-in independence, which makes it likely that a CASD speculation is not present in the drafts of the base speculator. Thus, CASD will probably augment future speculators well too.
- **Training-free.** CASD requires no training of a model and works out of the box with any input data. This means it does not require retraining if it is applied to new language or a new niche domain.

This paper is structured as follows: we start with some preliminaries necessary to understand the rest of the paper. Then, our new method, CASD, is introduced, followed by the experiments performed and their results. Afterwards, the related work is briefly discussed and finally, we form a conclusion for this paper.

2 Preliminaries

As this paper augments neural speculators with a statistical method, it is important to first understand how speculators work, what their strengths are and their weaknesses.

2.1 Speculative Decoding

Speculative decoding is a technique to generate the same number of tokens with less forward passes through the LLM. This not only reduces the time to generate, but also the energy consumption (Qin et al., 2024). To achieve this goal, the speculator, which is a model to come up with candidate answers, starts with generating such answers, shortly referred to as drafts. Afterwards, in the verification phase, all drafts are sent through the LLM in one forward pass, essentially parallelizing the workload for more efficiency. This makes it possible to break the autoregressive nature, as the generator can attend to draft tokens and thus verify tokens multiple steps in the future. After verification, the real LLM distributions are known for each draft token. Standard autoregressive sampling can be simulated on those distributions and as long as the sampled token was already in the draft tokens, the next distribution is available. This means potentially multiple consecutive tokens can be added to the actual answer under construction, within the same forward pass. This process is repeated until the stop condition is met, progressing multiple tokens per step rather than just one.

2.2 EAGLE-2

We use the current state-of-the-art, EAGLE-2 (Li et al., 2024), as the main speculative decoder in this paper. Hence, we introduce it briefly below.

Most research in speculative decoding is about the drafting method, as this is the component with most freedom of implementation. EAGLE-2 innovates the drafting method by trying to extrapolate the embedding of the LLM autoregressively. This is a task that a neural network can be trained to do, and it seems to be an easier task than extrapolating tokens, allowing for smaller networks. The intuition is that the embedding layer contains the "reasoning" or "thought" of the model and thus EAGLE-2 has more valuable information as input than a standard token-predicting model. This is further clarified in Figure 1.

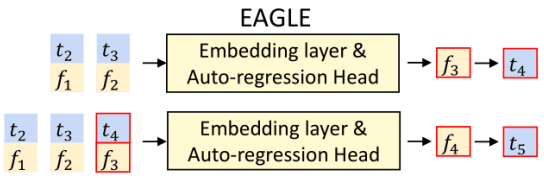


Figure 1: EAGLE-2 uses the LLM's features and previous tokens to predict the next step. It is used autoregressively, to draft multiple tokens ahead. (Li et al., 2024)

Although EAGLE-2 achieves SOTA results in speculative decoding, there are some limitations:

- **Poor out-of-distribution performance.** Neural networks tend to perform badly when the input is completely different from the training data (out-of-distribution). For neural speculators, this is the case when the task contains specific jargon or a different language.

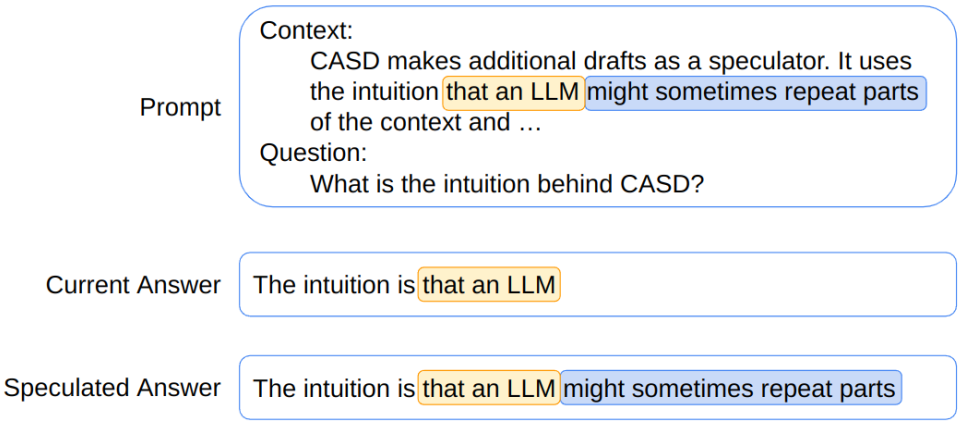


Figure 2: CASD applied to a simplified RAG prompt. The prefix ("that an LLM") is found in the context and thus the continuation ("might sometimes repeat parts") is drafted by CASD.

- **Drift.** EAGLE-2 autoregressively drafts tokens, so a token five steps ahead already contains the accumulated noise of five extrapolations. In practice, this means the drafts at many steps into the future are already of much lower quality and have lower chances of being accepted.

3 Context-Aware Speculative Decoding

CASD makes additional drafts as a speculator. It uses the intuition that an LLM might sometimes replicate parts of the context. Thus, when it has already copied a small part, it might continue copying further tokens, as illustrated in Figure 2. To put this intuition into practice, the first step is to detect when the LLM might be copying literally from the context. This is easily done by substring-matching the last generated tokens with any part of the context. To limit the overhead, only the top-k matches are kept, based on the length of the match. For each of these matches, the continuation in the context is added as a draft.

While CASD itself is not the best speculator, it can augment SOTA speculators to achieve even better performance. In a practical setup, the SOTA speculator is asked to generate its drafts and then CASD adds some more drafts. After that, the default speculative decoding is performed, calling the LLM for each token and sampling from the distribution.

4 Experimental validation of CASD

In this section, we validate our new method empirically on niche and standard benchmarks, focusing mostly on RAG. Then, we further analyze how CASD works and what makes it perform well.

4.1 Experimental setup

Models. The experiments use only Llama-3-8B-Instruct (Grattafiori et al., 2024) as an LLM, as only this model fits our following constraints. The first constraint is the hardware: 8B is the largest model that fits on the used GPU. Since one of our benchmarks has Dutch content (UZGentRAG, see further), we also need a Dutch-capable LLM. Lastly, EAGLE-2 also has to support the LLM. This yields Llama-3-8B-Instruct as the chosen model.

Tasks. CASD is evaluated on UZGentRAG benchmark (see further), SQuAD (Rajpurkar et al., 2016) and the classical speculative decoding benchmarks (Alpaca (Taori et al., 2023), GSM8K (Cobbe et al., 2021), HumanEval (Chen et al., 2021), MT-bench (Zheng et al., 2023), Natural Questions a.k.a. QA (Kwiatkowski et al., 2019) and CNN/DM (Nallapati et al., 2016)). All experiments were performed with batch size 1, conform with most related work (Leviathan et al., 2023; Li et al., 2024).

Metrics. CASD, being an augmenting method, focuses on improvements on the baseline speculator. The usual speculative decoding metrics, applied to CASD are:

- *Walltime speedup ratio*: The speedup ratio relative to the baseline method, EAGLE-2.
- *Average acceptance length* τ : The average number of tokens accepted per forward pass of the target LLM.

4.2 UZGentRAG benchmark

We hypothesized that EAGLE-2 would underperform in multilingual settings and niche domains.

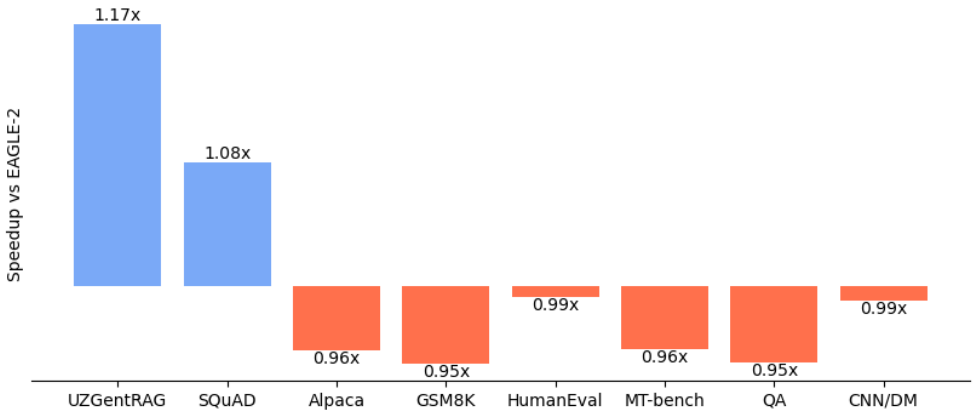


Figure 3: Speedup of CASD compared to the baseline, EAGLE-2. CASD yields improvements for both RAG benchmarks. On standard speculative decoding benchmarks the overhead is still rather limited.

To prove this, we had to go beyond the standard English speculative decoding benchmarks. However, low-resource languages such as Dutch have limited benchmarks available and for RAG use cases we did not find any. In collaboration with Ghent University Hospital (UZGent), we constructed UZGentRAG, a private niche RAG benchmark in Dutch and for the medical domain. This benchmark contains the queries logged from the daily work of doctors and nurses, by recording their questions to a chatbot. Internal UZGent documents found by the chatbot were combined with the query in a typical RAG prompt. These prompts form the benchmark, as that is the only data necessary to test a speculative decoding method.

4.3 Results

Figure 3 shows the comparison between CASD and EAGLE-2. The most relevant benchmarks are the RAG benchmarks, UZGentRAG and SQuAD, for which CASD achieved a speedup of 1.17 and 1.08 respectively. This is quite interesting, because both benchmarks are RAG use cases, yet the results differ strongly. We hypothesize that the Dutch UZGentRAG is harder for EAGLE-2, so it is easier for CASD to find better continuations.

Next to the RAG benchmarks, we also list the general benchmarks. Since these are not RAG-based, and EAGLE-2 was designed for these settings, performance is now greater for EAGLE-2. The key takeaway here is that even when applying CASD to non-RAG input results it still has limited overhead (1-5%).

4.4 Acceptance length

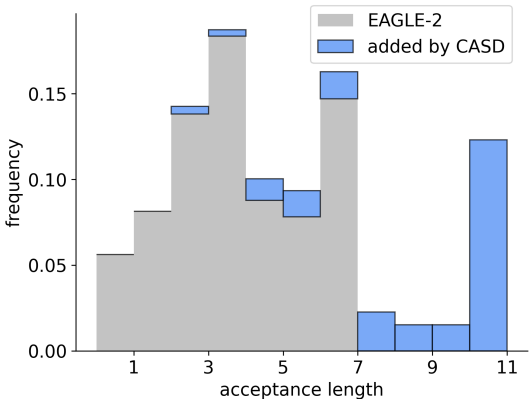


Figure 4: Distribution of acceptance lengths with CASD augmenting EAGLE-2. CASD supplements EAGLE-2 with long-length drafts.

To understand why CASD yields improvements on neural speculators, Figure 4 shows how the acceptance length is distributed. EAGLE-2 works quite well and rather consistently: most often it has 4 to 7 tokens accepted. On the other hand, CASD's added speculations show a strong peak at 11. So if CASD gives the best speculation, it is a high quality speculation for a long length. This also explains intuitively why CASD works so well with neural speculators: CASD has occasional speculations that are extremely good, while the neural speculator works well consistently when nothing is copied. As a last remark, it might stand out that there is a hard cap at 7 and 11 for EAGLE-2 and CASD respectively. Both methods implement this cap to balance overhead with more accepted tokens.

	CASD		CASD_{prompt}		EAGLE-2	
	Tokens/s	τ	Tokens/s	τ	Tokens/s	τ
UZGentRAG	**35**	**2.73**	34	2.65	30	2.09
SQuAD	**46**	**7.78**	**46**	**7.78**	43	5.88
MT-bench	61	**4.35**	61	4.30	**63**	4.18

Table 1: Performance of CASD_{prompt} compared to CASD and EAGLE-2 as the baseline. Best scores in bold.

4.5 Ablation study

CASD copies drafts from the entire context, but also from the tokens that were already generated. However, for the original problem statement (RAG), CASD only had to copy from the prompt and not from the answer that the LLM had generated up until that point. If CASD did that, it could be better if the LLM almost never copies from the generated tokens, as it has a smaller overhead. The opposite could as well be true, where CASD benefits now from an LLM repeating itself. The following empirical tests show whether the extra LLM tokens actually provide benefits: we compare our original CASD to CASD_{prompt}, which only considers prompt tokens.

Table 1 shows how CASD_{prompt} performs. For the most relevant benchmark, UZGentRAG, a slight improvement justifies the choice of CASD over CASD_{prompt}. For SQuAD, τ stayed the same and so did the throughput. Finally, MT-bench shows very similar performance. So it seems that CASD performs better than CASD_{prompt}, mostly on the UZGentRAG benchmark. As this benchmark contains longer answers than SQuAD, we intuitively think that the LLM starts repeating generated fragments more the longer the answer gets.

5 Related work

The original paper that introduced speculative decoding (Leviathan et al., 2023) already mentioned the idea of copying from the context, without elaborating. Inference with Reference (Yang et al., 2023) actually implemented it, copying only the best match as a single speculation. While this speculator cannot compete with the current state-of-the-art, CASD proves the principle of copying yields benefits as an augmenting technique on top of the current state-of-the-art.

REST (He et al., 2023) is another method that has similarities with this paper. Rather than drafting from the context, REST queries a large set of pre-existing text to find relevant pieces to copy.

6 Conclusion

We have introduced CASD, a model-agnostic augmentation to any modern speculator for speculative decoding. We found that CASD improves performance of the SOTA speculators, with a very simple method. CASD yields high quality speculations by copying relevant tokens from the context. Like most other speculative decoding methods, CASD allows generating tokens faster, while still reducing the energy necessary to generate. Our tests, augmenting the current SOTA, EAGLE-2, with CASD, demonstrate the added value of its speculations. The experiments show that our method improves speculations in the intended domains, while minimizing the overhead when applied to other domains.

Limitations

CASD is tailored specifically for high-copy environments, such as RAG-based use cases. As seen in the results section, one should only apply it for such cases, because then the gains outweigh the overhead significantly.

Also, the current implementation is not overly optimized. Thus, it is likely that further optimizations such as specialized CUDA kernels could help CASD achieve even higher speedups.

Acknowledgments

We would like to thank the ICT department of the UZGent for their collaboration on making the UZGentRAG benchmark.

References

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*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph,

		CASD		EAGLE-2		CASD vs EAGLE-2	
		Tokens/s	τ	Tokens/s	τ	Speedup	$\tau_{CASD}/\tau_{EAGLE-2}$
RAG benchmarks	UZGentRAG	**35**	**2.73**	30	2.09	1.17	1.30
	SQuAD	**46**	**7.78**	43	5.88	1.08	1.32
General benchmarks	Alpaca	61	**4.15**	**64**	4.12	0.96	1.01
	GSM8K	64	**4.56**	**67**	4.42	0.95	1.03
	HumanEval	75	**5.55**	**76**	5.05	0.99	1.10
	MT-bench	61	**4.35**	**63**	4.18	0.96	1.04
	QA	52	**3.56**	**54**	3.53	0.95	1.01
	CNN/DM	52	**4.14**	**52**	3.74	0.99	1.11

Table 2: Performance of CASD compared with the baseline EAGLE-2. The best score is in bold.

Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

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.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. Eagle-2: Faster inference of language models with dynamic draft trees. *arXiv preprint arXiv:2406.16858*.

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

Zongyue Qin, Ziniu Hu, Zifan He, Neha Prakriya, Jason Cong, and Yizhou Sun. 2024. Optimized multi-token joint decoding with auxiliary model for llm inference. *arXiv preprint arXiv:2407.09722*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

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

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

A Appendix results and reproducibility

Results. Table 2 shows the full results of the benchmarks, comparing CASD with EAGLE-2.

Hardware. The tests in Table 2 used an NVIDIA A40 with 44 GB RAM. The results were gathered in one session on the GPU to improve consistency between results.

Benchmarks. Most benchmarks are publicly available, however UZGentRAG is not open for publication as it contains possibly sensitive data. UZGent has indicated that they are open to give access to the dataset in collaborations.

Hyperparameters. The maximal substring length considered is 10. Only the top-2 matches are kept when there are more matches. Of those matches, a continuation of 10 tokens long is chosen as a draft.

Contents

Chapter 1

Introduction

In recent years, the quality of Large Language Models (LLMs) has made significant progress [86]. With it, the widescale usage of LLMs has grown so much, up to the point that their energy consumption has raised major concerns [4, 7, 70]. The larger players in the market even started building nuclear plants, just to be able to keep powering the servers that host these LLMs [54, 57, 69]. Or to look at it from another perspective: each time a user gives a long prompt to a modern LLM, so much energy is used on the server as one would need to charge their phone from 0% to 100% [85]. So it is clear that the adoption of LLMs in the daily life should come with extra steps to counter the explosion of energy consumption. One component of the solution is to use fewer LLMs. While often neglected, this may have the largest impact [3]. Do we actually need a state-of-the-art (SOTA) LLM to generate us a poem, to have a chat with or to suggest which movie to watch? Another part of the solution is to make sure the least possible energy is used to generate responses of the same quality. But how would that work?

Many methods have successfully reduced the energy consumption of LLMs. Some popular methods are quantizing [24, 81], distilling smaller models [27, 52] and switching between small and large models [1]. All of these methods try to minimize the loss in quality for maximal energy gains. Complementary to these methods, speculative decoding promises to generate exactly the same answers faster, using less energy [46, 64]. It works by using a small model to draft many samples of what the generator could generate in the coming tokens. Then, these drafts are verified in parallel by the LLM. This parallelization can lead to speedups of up to 4x [49]. Surprisingly, this even leads to decreased energy consumption [64]. While speculative decoding is not new, there is much freedom in how the speculator (i.e. the draft model) works and this is where current research focuses on.

The main goal of this thesis is to develop a new speculator, Context-Aware Speculative Decoding (CASD), that augments the current SOTA to increase efficiency of LLMs in high-copy environments. With the popularity of Retrieval-Augmented Generation (RAG) [35, 78], such high-copy environments become more and more common and increasing LLM efficiency for those cases has the potential to make a large impact. To prove that a new method works, relevant benchmarks are necessary, but the number of RAG benchmarks is very limited. Even more, we found only one well known English benchmark, that can be adapted to be a RAG benchmark for speculative decoding [66]. To fill the gap in more niche benchmarks, **the second goal is to make a novel benchmark, UZGentRAG, that contains real-world questions and context to evaluate speculative decoders on.** To gather realistic data, we needed a live RAG system that actual healthcare professionals can direct their questions to. As the UZGent is interested in such a RAG application, **the third goal is to make a functioning RAG system at the UZGent**, as a case study.

With these goals in mind, the following research questions follow naturally:

Research Questions

RQ1: Can a new speculative decoding strategy be designed, relying on the repetition of phrases in prompt and answer, resulting in increased inference efficiency of LLMs?

- RQ1.1: Which energy gains does CASD make?
- RQ1.2: Which speedups does CASD deliver?

RQ2: Does a real-world benchmark in a niche domain show significantly different results from existing speculative decoding benchmarks?
RQ3: Can a RAG system provide additional value at UZGent?

In this paper, the context is built up logically, before diving into the research questions. The structure is as follows: First, in Chapter 2, we summarize the relevant literature around RAG for the UZGent use case. Chapter 3 applies this knowledge in the practical implementation of the RAG system. Chapter 4 describes speculative decoding, which is the main academic topic of this thesis. In Chapter 5 we then introduce our new method, CASD, to improve speculative decoding. This is followed by Chapter 6, which formulates answers to the research questions and discusses the most important opportunities for future work. Finally, Chapter 7 looks into the societal impact these results can have.

Chapter 2

Retrieval-Augmented Generation

When the first LLMs were released, people were blast away by the numerous skills these models had learned. Yet, after a while it became clear that producing factually correct answers was a skill the models had not learned. It is not what the models were trained for and certainly when a fact was not in the training data, the LLM has no way of knowing that fact. Even worse, the LLM was trained to always give an answer, no matter how incorrect it is and it does so confidently. When an LLM makes up something rather than being based on facts, it is called a hallucination. These hallucinations can make the answers unreliable and Retrieval-Augmented Generation (RAG) is made to solve that problem (partially).

2.1 What is RAG?

Figure 2.1: Diagram of the basic RAG pipeline.

Retrieval-Augmented Generation (RAG) is a system where the LLM is provided with factual information in a prompt, such that it can answer correctly even if it has never seen that fact before or already forgot about it. Practically, a simple RAG pipeline has two stages: retrieval and generation. First the user asks a question and the most relevant pieces of text are retrieved from a database of facts. Then, this information is given to the LLM as part of the prompt, before it answers the question. This is visualized in Figure 2.1.

2.1.1 Retriever

The retriever in a RAG pipeline refers to the component that takes query text as an input and returns the most relevant document chunks. In Figure 2.1, the example context is drawn from Wikipedia [17], but any source of text can be used. Needless to say, a good retriever is the basis for a good RAG system. In practice, we see that good retrievers are hybrid retrievers, that combine the advantages of a lexical search and a semantic search. Lexical search works on keywords and is good to find literal matches between the question and documents. BM25 [68] is typically used for lexical search, because it evaluates the lexical matches with a robust scoring function. This lexical search is complemented with dense retrieval, using an embedding model. An embedding model is a neural network that is trained to take text as input and transform it into a vector that contains the "meaning" of the text. For a first time reader, this might not sound intuitive. However, for the purposes of this thesis, it is only important to understand that this model creates an output for each piece of text, which we can easily compare. For example, given a query and a piece of context, we can say: these strings are 20% similar. In other words, this embedding model will allow us to efficiently find which pieces of text in the entire database are semantically similar to the query.

While lexical search is practically set in stone, there is still much freedom in the semantic search. There are many strong embedding models available for general use cases, such as the Splade models [22, 23, 45], Contriever [32], GTR [59], etc. These could serve as a basis for fine-tuning on a specific domain, such as the medical domain for this thesis. However, within the scope of this thesis, we limit ourselves to using existing embedders directly. In that case, it will be important to find the right embedder for the specific use case.

2.1.2 Generator

The generator is the component that takes the question and context as input, to formulate an answer using the information in the context. In practice, the question and context are combined in a prompt and that prompt is fed to an LLM. The first step is optimized through prompt engineering. This has become a whole field of research in se and there are many ways to incentivize the LLM to give better answers. Common prompt engineering methods include: adding structure, adding examples, telling the LLM what not to answer, defining roles, etc. [8, 9, 80] The second step is just the LLM generating an answer, which can also be optimized strongly by choosing the right LLM and/or fine-tuning [28, 83]. Again, for the scope of this thesis fine-tuning costs too much time and compute. Thus, we limit ourselves to selecting the best LLM out of the publicly available LLMs.

2.2 RAG optimizations

A modern RAG pipeline may consist of many more stages than a simple retrieve and generate step. As each application requires different tricks, there is an endless amount of extra steps one could add. In the following literature review, we list the popular RAG optimizations that could be interesting for this thesis.

2.2.1 Pre-Retrieval

In an additional pre-retrieval step, the input of the user is first processed. As a normal user is not an experienced prompt engineer, the questions users ask are often incomplete, ambiguous, etc. To better interpret the question, three different methods have emerged: abstracting the question, retrieving based on a hypothetical answer and splitting the question.

Abstracting the question

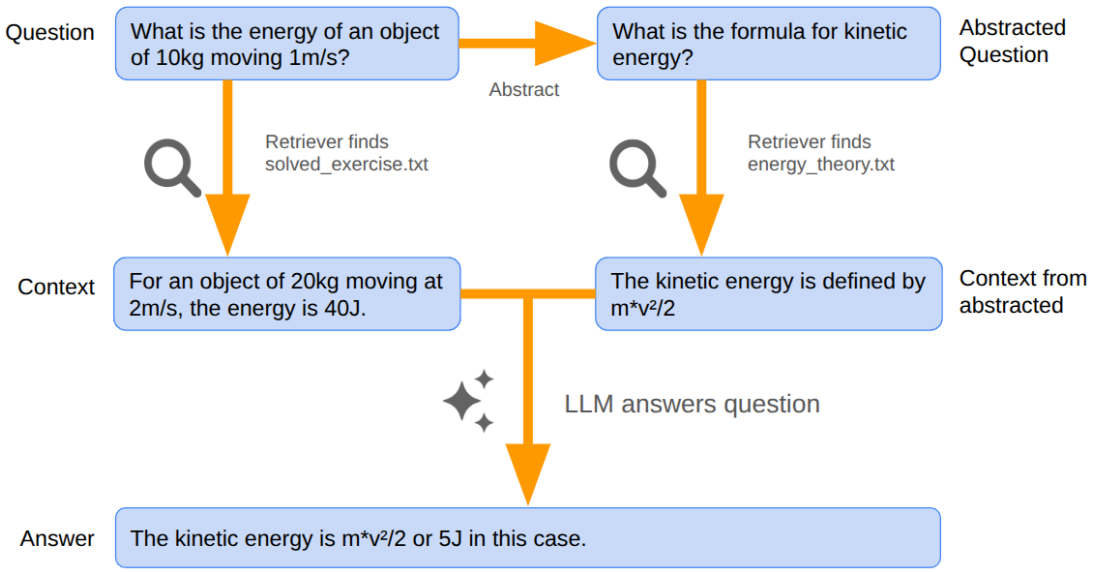


Figure 2.2: Diagram of prompt abstraction. First, the prompt is abstracted to the higher level reasoning that is necessary to answer the question. Then the relevant context is searched for both the original question and the abstracted question. The contexts are combined to form the prompt for the LLM. So the left part of the diagram is basic RAG and the right part adds the abstracted question for retrieval. Note that this example is cherry-picked such that the "context from abstracted" is much better, but oftentimes the direct retrieval also yields relevant context, depending on the question.

Step-Back Prompting [87] tries to make the retrieval easier by first abstracting the question. The idea is that highly specific questions with uncommon words can confuse the retriever, preventing the it from finding semantically similar, relevant documents. This is also shown in Figure 2.2.

Retrieving based on hypothetical answer

Retrieval is based on semantic similarity between the query and the context in the database. However, the question a user asks can be completely different from the expected answer. To narrow the gap between the query and the retrieved document, HyDE [25] builds a hypothetical answer to the question. This hypothetical answer can contain hallucinations, but that is not a problem. The point is that the hypothetical answer should be semantically closer to the real answer than the question is. This is further clarified by Figure 2.3.

Splitting the question

A question often consists of multiple subquestions. Demonstrate-Search-Predict [36] and IRCoT [74] try to leverage the Chain of Thought (CoT) strategy, where the question is reduced to a simpler question iteratively. Figure 2.4 clarifies this principle with an example. Later, each question will then be answered with its own relevant retrieved documents. More parallel methods like ToC [37] and RAG-Fusion [65] break down the question recursively into multiple questions. The difference with CoT is that there are now multiple branches of thoughts.

2.2.2 Post-retrieval

The retrieval step typically yields multiple chunks, pieces of text, in which the answer is hopefully present. But sometimes the retriever fails and it adds an irrelevant chunk or it succeeds, but the relevant part is only a small fraction of the entire chunk [20]. This all adds noise to the input of the LLM and LLMs

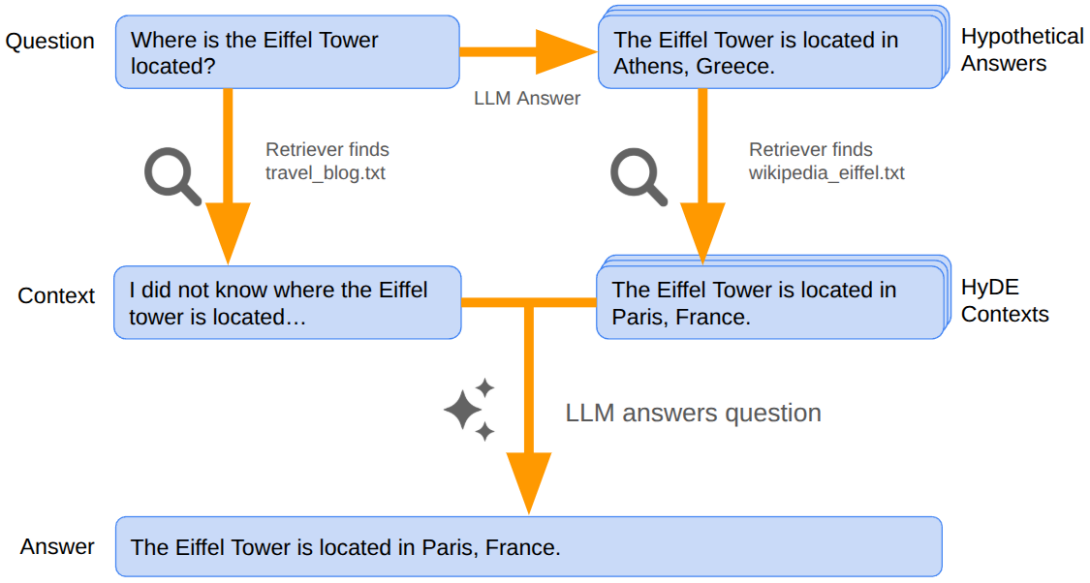


Figure 2.3: Diagram of HyDE. First, the LLM is asked to answer the question. Then the relevant context is searched for both the original question and the hypothetical answers. The contexts are combined to form the prompt for the LLM. So the left part of the diagram is basic RAG and the right part adds the hypothetical answers for retrieval. Note that the example is again cherry-picked to clarify where HyDE excels.

often get confused by it. That is why a post-retrieval step is sometimes necessary to filter the abundance of information. The two post-retrieval methods relevant to this thesis are prompt compression and reranking. Prompt compression is discussed later, as it also falls under the energy optimization methods.

Reranking is the post-retrieval method that takes the retrieved contexts and reorders them based on relevance. The difference with the embedder of a retriever, is that it does not go to an intermediary embedding space to define similarity. This usually makes it much more compute intensive, but also more accurate. For this reason, the reranker is only used after only a limited amount of candidate chunks remain. QLM [90] is a recent method that uses the modern LLM's zero-shot capabilities to rerank. The score of the context is the probability that the question would be generated given the context. The idea is that more relevant contexts should make the question more likely. Another method is to train a cross-encoder to output the relevance of the context given the query directly. One such cross-encoder is "BGE-Reranker-v2-M3" [10].

2.3 Generator energy optimization

While the previous optimization techniques were about generating better answers, this section discusses how the best answer is generated with the least amount of energy. Speculative decoding is the method that is the main topic of this thesis, but we also list alternative methods that are equally valid to reduce the energy consumption of the generator.

2.3.1 Speculative decoding

Speculative decoding tries to reduce the energy an LLM takes to generate an answer by reducing the number of forward passes ($\approx$ the number of times the LLM is activated). Less forward passes in the LLM mean not only less latency, but also less energy consumed [64]. The key idea to speculative decoding is that when the output of an LLM is known, the LLM can verify all generated tokens in parallel, in one forward pass. However, to know the output, we need to use the LLM. This creates a chicken and

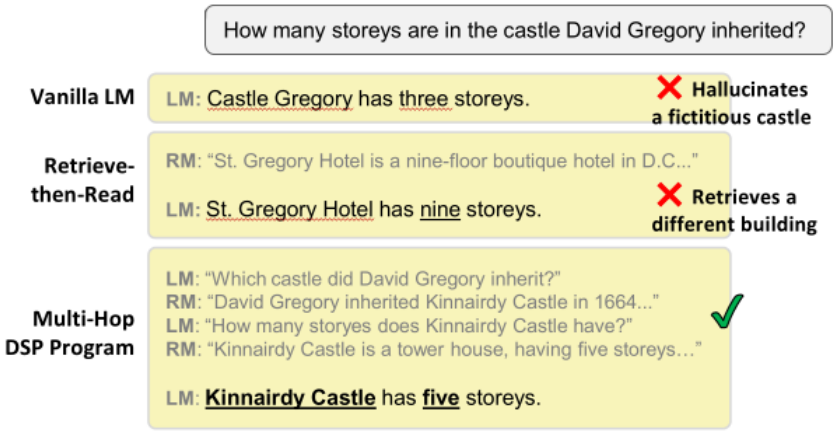


Figure 2.4: Illustration from Demonstrate-Search-Predict [36]. Demonstrate-Search-Predict (DSP) performs multi-hop reasoning, retrieving the necessary documents in each step. This is indicated by the interleaving of LM (language model) asking questions and the RM (retrieval model) answering those questions by providing the right pieces of text.

egg problem, but it can be solved by guessing. First we guess the output of the LLM, not only for the next token, but also for further tokens. With that guess, all tokens are checked in parallel and only the correctly guessed tokens are kept. Two open questions remain: how is a good guess made without calling the LLM itself and how much does that estimation itself cost. This is what recent research in speculative decoding tries to answer [30, 49]. As speculative decoding is the main topic of this thesis, the details go in to a separate Chapter 4.

2.3.2 Prompt compression

Prompt compression intends to make the prompt, the input of the LLM, shorter, while still maintaining the meaning behind the prompt. There are multiple reasons to keep the length of the input for an LLM limited. Firstly, longer input increases the necessary computation, which leads to longer latency and more energy consumption [38]. Secondly, each model has a limited context window and it is impossible for the LLM process the full input when the input gets longer. Lastly, longer input can degrade the quality of the generation. This is because typically LLMs are trained with fixed length inputs, much shorter than the context window. Even though they are then trained somewhat to work with longer inputs, the performance usually stays subpar [47, 53]. To limit these effects, there are three categories of prompt compression, each with their own benefits and weaknesses: extractive compression, abstractive compression and token pruning.

Extractive compression

Extractive compression is the highest performing type of compression, and it is based on extracting relevant parts from the original text. These parts can have a granularity of phrases, sentences or passages. In RAG systems, this extraction can be based on the semantic similarity between a part of the context and the input prompt.

Below, we list the most prominent methods for extractive prompt compression: CPC [51], RECOMP [82] and Selective Context [48]. CPC uses contrastive learning to fine-tune an embedder, such that it can find similarity between context sentences and the question asked. The authors noticed that embedders are good at finding direct answers to a question, but not components of answers, so they trained the embedder to learn similarity between questions and parts of answers. Only the relevant sentences are then kept and the irrelevant ones are cut out. RECOMP has a similar method of contrastive learning,

but they get their positive samples from sentences that lower the perplexity of the correct answer the most. Selective Context uses a whole different strategy, where they combine the token-level perplexities to get a phrase level perplexity. The lowest perplexity phrases are then cut out.

Abstractive compression

In abstractive compression, the prompt is summarized by a smaller language model. This summary can contain new tokens that were not in the original prompt, unlike the other techniques. RECOMP [82] approaches this task by training a small model to summarize like the modern LLMs do. It is important that the summarizing model is many times smaller than the LLM to actually make gains, rather than losses through overhead.

Token pruning

Token pruning exploits the fact that an LLM can handle input that is not exactly natural language. By cutting away tokens that were "obvious" to an LLM anyway, the compressor can condense information by cutting only the obvious tokens and leaving the surprising (read high perplexity) tokens as is. LLMLingua [33] laid the foundations of token compression for both LLMLingua 2 [60] and LongLLMLingua [34], which are currently the best token pruners. They are all based on the same principle of using a smaller model's perplexity to estimate the relevance of a token. Then, the low perplexity tokens are cut iteratively, recalculating perplexity every time. To get to the optimal compression level, a budget controller decides what perplexity is too low. While this method yields high compression ratios, a known issue is that an LLM often copies literal parts of the prompt. This means that the answer also contains text that is rather incoherent to humans, where some tokens were just dropped.

2.3.3 Hybrid generation

Certain pieces of text are easier to generate than others. In a RAG context, that could be when the LLM is quoting excerpts from the context. AutoMix [1] tries to improve efficiency by generating these easier parts with a smaller LLM. This hybrid generation method works as follows. First, the smaller LLM generates verifiable statements from the RAG context. Then it is given a verification prompt and the probabilities of the "Correct" and "Incorrect" token are measured. If the probability that the statement is correct is too low, the task is passed to the large LLM to do it better. Originally, this method was posed as a way to cut monetary costs, where the smaller LLM is local and the large LLM is paid for through API. Yet, this method has potential to reduce both latency and energy consumption.

Chapter 3

Use case UZGent

To put the theory of speculative decoding into practice, this thesis looks at a use case for RAG at the UZGent. This chapter starts with a description of the use case. Then the technical implementation is uncovered, detailing how the architecture of the RAG system works. Afterwards, we discuss the test results of healthcare professionals interacting with the RAG system. Finally, we list some future work and ethical considerations specifically for this RAG system.

3.1 Description of use case

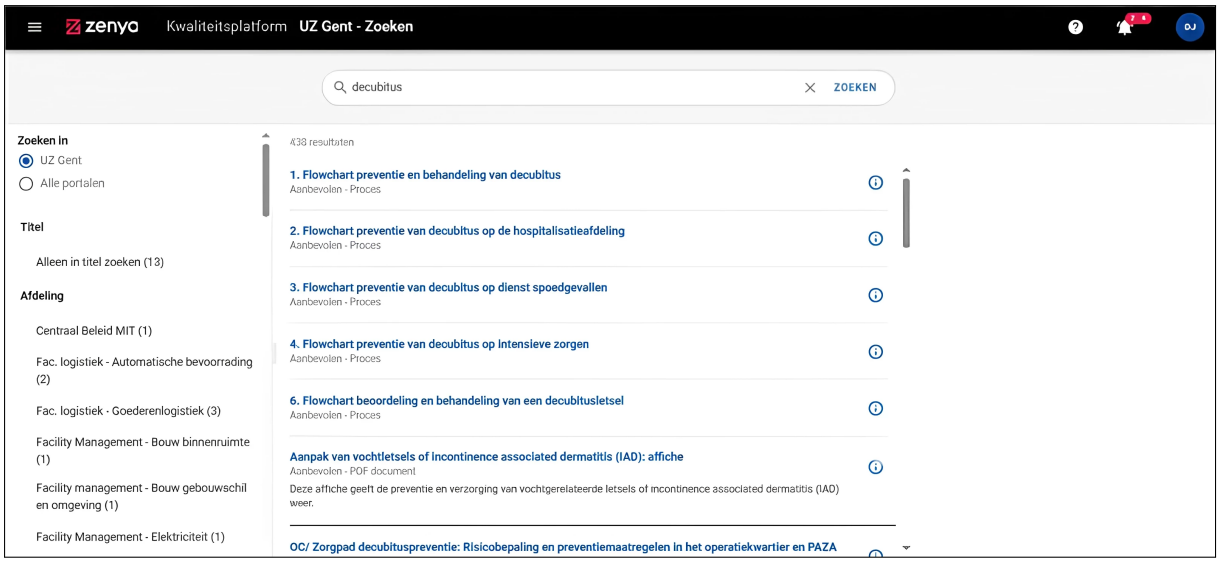


Figure 3.1: Screenshot of Zenya, the current system healthcare professionals use to search procedure documents.

In the UZGent, there is a system, Zenya and it contains many documents often needed by healthcare professionals. To work with the tool, one must either be guided or have enough experience themselves in order to get the document they need to read. Also, the documents can be quite lengthy, while often only a very specific part or summary is necessary. This makes the perfect use case for a RAG system. In this RAG system, the user first poses a question. Then the retrieval fetches the relevant documents based on a deeper understanding than the current keyword-only search. Finally, the relevant part of the document is read by the LLM and used to answer the question concisely. This description was rather

general about RAG, however each use case is different and has different requirements. To get a grasp of the details of this use case, many involved people were interviewed. These interviews led to the following list of specifications for the RAG use case at the UZGent.

- The user can ask questions to the RAG chatbot and the answer is found in one single document. This already simplifies the RAG complexity. It also means that the retriever has only one document to find as the ground truth. However, this can still be multiple chunks long, sometimes requiring a summary of large parts of the document.
- The question will be in natural language, rather than keywords as it used to be with Zenya. However, the language can contain abbreviations and short, incomplete sentences. This is because the healthcare professional often needs the answer quickly.
- The system should know when it does not know. It is very important that when there is no answer in the database, it is able to detect this and answer accordingly.

3.2 Technical implementation

In this section, we list the technical details of our implementation. First, some necessary metrics are defined to evaluate the system. Then, we follow chronological order: the documents need to be pre-processed first. Afterwards, the retriever must find the right chunks and the generator will produce an answer. Finally, some post-processing will make the result more accessible to the end user. As a guide through this section, Figure 3.3 shows the high level data flow graph that makes RAG possible.

3.2.1 Metrics

To objectively compare different models, it is important to choose the right metric for the job. We will focus most on the retriever, as that is the most valuable component for UZGent. In this thesis, the goal of the retriever is to get one piece of information and to make sure that piece is ranked as high as possible. The logical metrics to use in this case are Hit@k and Mean Reciprocal Rank (MRR). Hit@k is the probability that a relevant document is in the top k results. So Hit@3 = 0.7 means that a relevant document is in the top 3 retrieved documents for 70% of the times. The MRR is defined as $\frac{1}{U}\sum_{u=1}^{U}\frac{1}{rank_i}$, with the rank defined as the place of first document that is relevant, as shown in Figure 3.2. In the best case scenario, a relevant document is retrieved on place 1, resulting in rank 1. As the MRR is the mean of many such ranks, the best possible MRR is 1. Thus, the MRR ranges between 0 and 1.

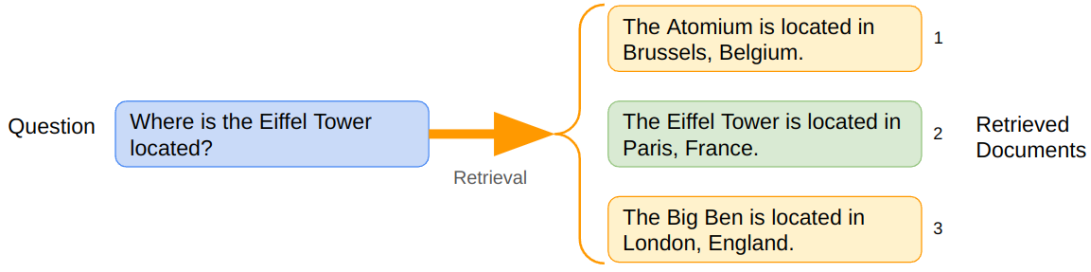


Figure 3.2: Example of a retrieval to explain the rank. For this question, there is clearly one relevant document retrieved: the second one. This makes the rank of this retrieval 2.

3.2.2 Preprocessing

In the preprocessing step, the goal is to extract the text from documents. To start this preprocessing, one must first know what format the data will have. The database of Zenya contains many file formats:

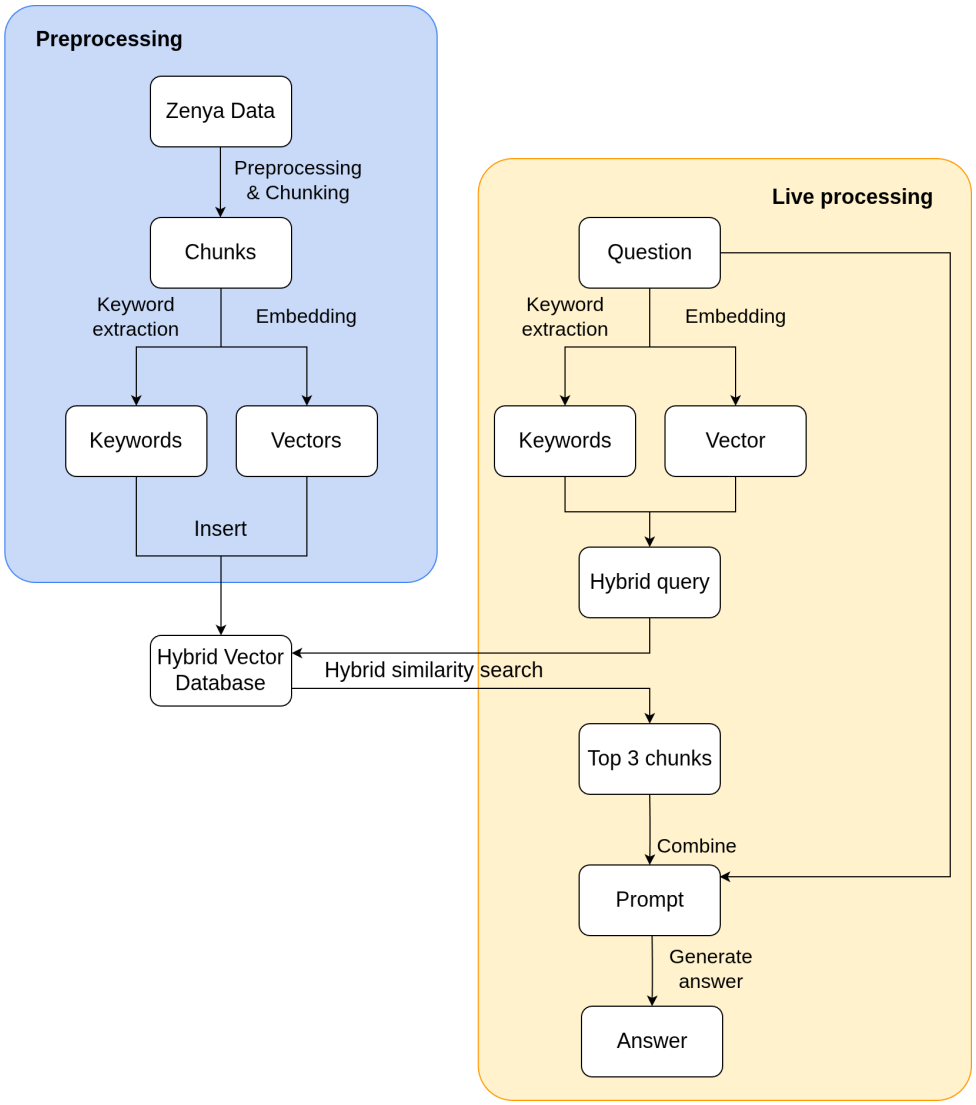


Figure 3.3: The data flow graph, which shows how the data and input question lead to a final answer. The graph is visually split to indicate the difference in time. The blue part fills the database and as this data is constant, it can be calculated in before. So practically, this preprocessing is done before the system goes live. The orange part shows the live processing that is repeated for each new question that comes in.

pdf, docx, excel, etc. This heterogeneity is not desirable for building a quick proof of concept (POC) in the scope of this thesis. Yet, the majority of the files are pdfs and docx, most of which contain primarily text. This is visualized in Figure 3.4. The first conclusion is that pdfs and docx are important, while the rest can be ignored in a simple POC. Also, the data exporter made it possible to convert docx to pdf and for simplicity, this was used, such that the input data was as homogeneous as possible. The python module PyPDFDirectoryLoader of the langchain_community package [43] provides very concise code to parse these pdfs and provide the text in them. This text is not directly used, because pdfs (like most other formats) contain much textual noise. One example is the character "\" which often appears in places one cannot find visually inspecting the pdf, as shown in Figure 3.5. To handle this textual noise, a pdf text cleaner scans all text and strips the text down to the real content. After this cleaning, mostly natural language remains in the text.

With the actual text available, the next steps are those typical to RAG.

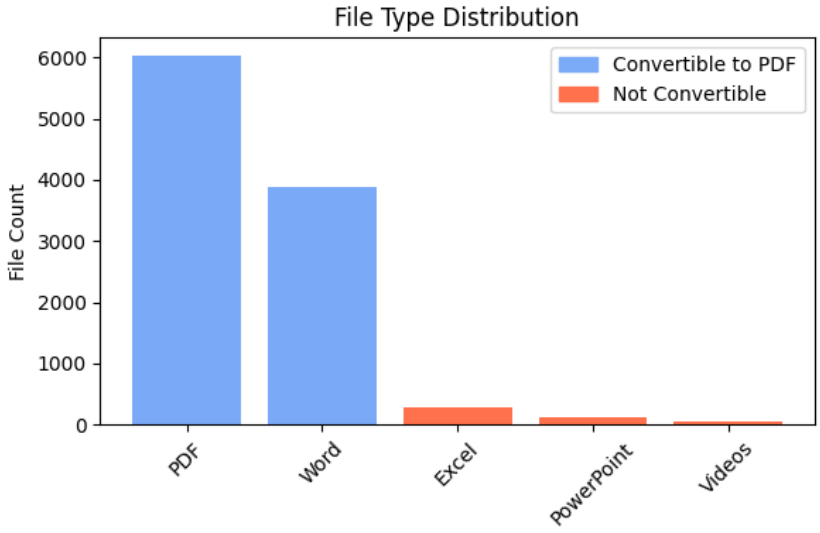


Figure 3.4: The file type distribution of Zenya documents.

3.2.3 Chunking

First, the text of each pdf is split into smaller pieces, chunks. This chunking helps in multiple ways. Good chunking can make the pieces of text more bite-sized, such that the embedding model can interpret it better. Also, RAG does not always need to read the full document, but rather the relevant part, so chunking also limits the amount of noise around the retrieved information. There are some choices necessary to do chunking successfully. The optimal chunk size, the first choice, depends on both the dataset and the embedder. It is usually helpful to match the chunk size with how long one thought is in the dataset. Next to that, an embedder is trained on a certain input size and they work better when working in similar sizes of new input. Empirically, it was found that approximately 800 characters yields the optimal score (MRR) for the use case.

The next option, is to use overlap. When splitting text it is impossible not to break an ongoing thought at any point. To make sure this cut in the text does not lose the information in that passage, overlap is often used to make sure this context helps both the previous and following chunk. The exact number did not seem to matter enough to make significant changes to the validate score, so we chose a 20% or 160 characters overlap.

With this target chunk size and overlap, we still need to decide where to cut the text. While deterministically stepping by 800 characters at a time would be a simple method, it can cut through thoughts, or even worse: words. Figure 3.6 shows the difference between a good and a bad split. To solve this, the RecursiveCharacterTextSplitter of langchain_text_splitters [44] splits the text recursively. Each time, it tries to find the most splittable characters until the desired chunk size is approximately met. Practically, this means that a hierarchy is followed where, for example, the new line character "\n" is preferred over a space " " to split on. With this last choice made, the chunks can be extracted.

3.2.4 Retrieval

With the text split into chunks, the next step is to retrieve the right chunks for a query. This is the hardest part for a reasonable proof of concept, because the retriever just cannot fail. If the retriever fails, the answer will never be satisfactory anymore, regardless of the LLM. We list the techniques applied to the retriever, making a clear distinction between what worked and what did not.

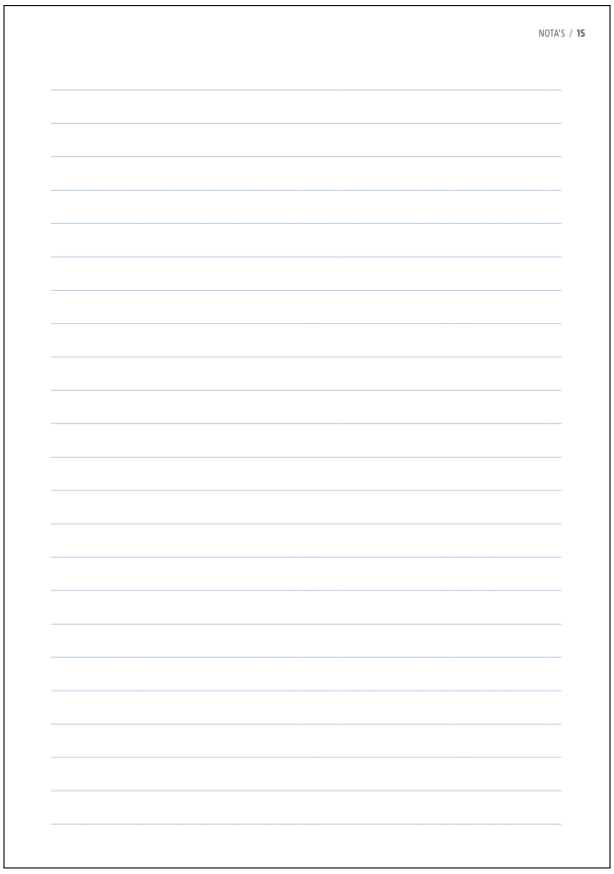


Figure 3.5: Example of a non-informative pdf page in Zenya. Despite being almost empty, the pdf parser extracts over 2500 hidden characters.

Effective retrieval techniques

In this part, the retrieval techniques are listed that are actually used in the final RAG system. In other words, the methods below all successfully made the retriever return better documents.

As explained in Chapter 2, retrieval is always based on a similarity search. In this case, the similarity search is a hybrid search, meaning classical methods (BM25 [68]) and dense retrieval (embeddings) are combined. This hybrid search was supported by the python package weaviate, with accompanying vector database (see Figure 3.9). To combine the scores of two methods, the separate scores are first scaled such that the minimum is 0 and maximum is 1. Then they are combined with a weight: $\alpha \cdot \text{dense score} + (1 - \alpha) \cdot \text{BM25 score}$. α has a default value of 0.7, focusing most on the embedder. By coincidence, this was found to be the optimum for the use case too. For BM25, some parameters are available to tune, but typically the defaults suffice. In this case the defaults were also used as implemented by weaviate.

The largest effort went into dense retriever. Not only are there typical parameters to tune, but since the rise of LLMs a whole new realm of tricks has allowed retrieval systems to push it to the limits. The discussion starts with the classical choices.

To start, we first need to find a good embedding model. A pragmatic approach is the only way not to get lost in the thousands of embedders, available on HuggingFace. To compare these embedders, a good benchmark is necessary that represents the Dutch, medical use case somewhat. Beyond MTEB [55], no reputable benchmarks were found to give a fair and insightful comparison between embedders. And even MTEB differs much from the intended use case, as it is a rather general, English benchmark. As no benchmarks represent the use case well, leaderboards using those benchmarks are not too reliable. For this reason, we hand-picked some embedders and tested them directly on our data. Surprisingly, Reddit

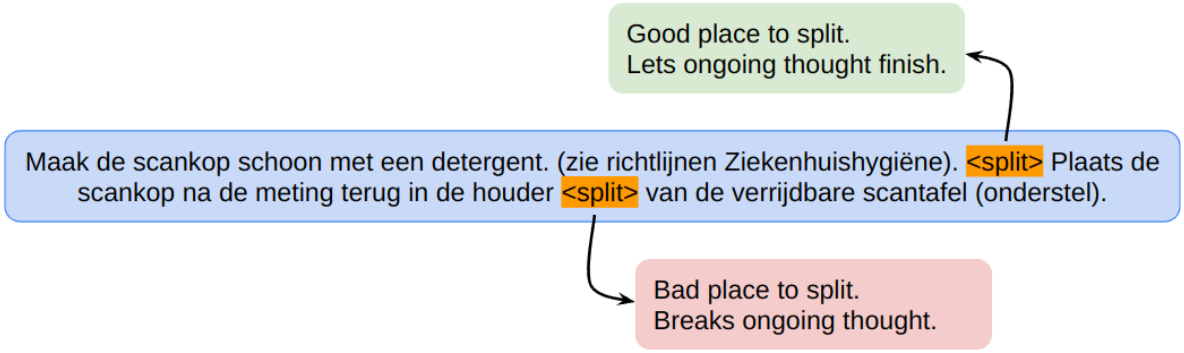


Figure 3.6: The difference between a good and bad point to split, applied on content from Zenya.

turned out to be the most useful source of evaluations of some hand-picked models [75]. This is likely, because Dutch is a niche language and a more professional comparison is typically not worthwhile. In that Reddit discussion, they mention the embedder that performs best on our tests. The model is called BGE-M3 [10] and it is a multilingual embedder. This property has proven to be the most important among all tested embedders. While a specialized model exists specifically for multilingual medical data, BioLORD-2023-M [67], it performed significantly worse, with a drop in MRR of more than 0.2. We hypothesize that the BGE-M3 model performs better based on sheer size. BGE-M3 is 568M parameters large, while BioLORD-2023-M has only 109M parameters. Other factors could be at play too, but no deep analysis was done as to why the performance of BGE-M3 was so much better: whatever works works.

With this setup, we ran the first tests to analyze the performance and the weak spots of the current model. A recurring error was that the embedder did not match on chunks that were in the middle of a large explanation. A typical example would be a document where the title contains the important keywords such as "hematology" or "oncology". As these documents were made for human readers, the rest of the full pdf assumes that the reader remembers that context. An example of a chunk with too little context inside is shown in Figure 3.7. So the chunks in se do not contain enough information to properly match with the query. As a rather quick solution, we appended the title to each chunk before being embedded. The structure is as follows: `f'Titel: {title}\nTekst: {cleaned_text}'`. This simple method yielded approximately a 0.08 gain in MRR, which is an enormous boost.

- Voeding

Maagsonde mag uit op advies van arts en pat mag maximaal 200ml water over 24 uur drinken
Starten van kauwgom naar wens

Metoclopramide (Primperan®) of domperidone (Motilium®) IV 3 x daags
Start Ranitidine (Zantac®) IV 3x daags bij systematisch NSAID gebruik of andere risicofactoren

- Infuusbeleid Glucose 5% met 2gr Nacl/liter 2000ml over 24uur

Bij perifeer infuusbeleid is het belangrijk om de toediening voldoende te monitoren

- Pijnbeleid

NSAID en paracetamol

- *Start kinesitherapie*

Figure 3.7: Chunk of a pdf with the title "Zorgpad invasieve blaastumor robotgeassisteerde cystectomie met neo-blaas". The chunk does not contain enough information to place it into context. The title clearly adds some necessary context.

We then repeated the analysis to find the common causes of errors again. The most common source of mistakes was medical synonyms and abbreviations. This is not too surprising as the model was not

trained for this. In fact, it is somewhat of a miracle that the model works this well without actually having a clue what many questions are about. Practically, this means that questions with uncommon terms, medical abbreviations or even abbreviations specific to the UZGent often did not match well with the intended chunks. This issue is quite fundamental and to the best of my knowledge, the only real solution is to fine-tune the embedder. This was out of scope for this thesis, so it is listed as future work.

Ineffective retrieval techniques

Of course, many methods were tried that did not lead to improvements. These methods are listed below. It is important to note that these methods do work in other settings and that it is specifically for this use case that the methods did not work. So either the method was not fit for this specific dataset, or it required larger models than what could be run locally on the UZGent servers.

One such method that did not work is to **search chunks hierarchically**. When searching hierarchically, the retriever tries to find the right document (or subset of documents) first and then it locates the right chunks in that document. This can help explainability, but it can also allow incorporating metadata into the search. To see whether this was a technique worth considering, the implementation started with a very simple approach. First, the retriever fetches the chunks as usual, but then the scores are aggregated per document. Afterwards, the idea is to find the right document first before taking the best chunks out of that document. The document-MRR was lower with aggregation than without it, which means it made the retrieval worse. So this idea was dropped. However, the principle of hierarchical search is still promising for future work, when more time and data is available. UZGent has metadata for each document (that was not available during the thesis), which could strengthen this method. For example, during practical usage multiple doctors and nurses gave the feedback that the retrieved chunks were very reasonable and contained the right information, but for another function. E.g. a pediatrician got very relevant results, but for adults rather than kids. To help guide the model, the function of the user could be given to the retriever and the metadata could say for whom which document is intended. This way, documents can either be filtered or given extra score for being fit for the specific user asking the question. In conclusion, gathering this data and implementing hierarchical search is also future work.

The **small-to-large method** also did not yield any improvements. The idea behind small-to-large is that the matched chunk is similar to the question, but not necessarily the answer. So certainly for small chunk sizes, the answer is to be expected before or after the chunk. And thus the solution is to retrieve a small chunk and then just expand, gathering also a piece before and after the retrieved chunk. During the process, this was first relevant, because the first tested models required small chunk sizes. Then, the final embedder we used was optimal for large chunk sizes, so this technique became less relevant. However, it later seemed that even for the large chunk sizes, the answer was often hidden before or after the chunk. This time, though, the chunks started crossing page boundaries and the used pdf parser made it harder to cross that border. While this is not the largest effort to fix, it was still not implemented. The most important reason was that the expected gains were for the generator rather than the retriever and at the time it already became clear that the generator would not add the most value (as discussed in Section 3.6).

Another method that failed in this particular use case was **reranking**. The reranking method starts as usual: the query is embedded and a similarity search finds the top-k most relevant chunks. Then the reranker model compares the query to each single chunk to give a relevance score. So the original order is dropped and replaced by the order that the reranker finds most relevant. The reason that this should work better, is that classical retrieval has an intermediary step of an embedding which implicitly encodes similarity. Learning the similarity directly should be an easier task and thus these models should perform better. The reranker BGE-Reranker-v2-M3 [10] was tested, as the embedding model of the same publisher performed so well, but the MRR went down when reranking. Further analysis was beyond the

scope of this thesis and thus the idea of reranking was also left out.

While **rechunking** seemed promising too, it did not deliver improved scores. Rechunking works as follows: first the classical retrieval is performed and then the retrieved chunk(s) are split again in even narrower parts. These new parts are again scored based on their similarity with the query to find the best match with a much smaller granularity. This works well in cases where the goal is to find one small piece of text. However, this seemed to conflict with the goals of the application. Consider the following dummy question that is based on a real Dutch question a doctor asked: "How much Vancomycin do I give to a 46-years old patient with a viral infection?" The answer could be as simple and short as "15 mg/kg", but if we single out that specific piece in the retriever, we might miss crucial information that the LLM would not miss. One problem could be that this answer came from the document "Vancomycin for patients younger than 18". Even common metrics such as Ragas' [19] faithfulness would praise this answer as it was grounded in the retrieved chunks (though it was the wrong chunk and wrong document). To further dive into the possible problems, the question was a trick question. Vancomycin is for bacterial infections rather than viral infections [77]. Even though this is a dummy example, it is based on a real example, and we found that in real cases the critical context was often found far above of below the retrieved chunk. So the perfect answer of "Do not use Vancomycin for viral infections." has very low chances of actually being the output of the chatbot with short contexts. To solve the problem of missing context, one could just try to give the whole document to the generator. This is not the solution, because the missing context is oftentimes implicit as the writer of the document thought of it as common sense, which the generator does not have. Specifically in this case, the limitations of a 9B model also became clear when too much context was given.

HyDE [25], **Step-Back Prompting** [87] and **splitting the question** (methods discussed in Section 2.2.1) all could not live up to the expectations. While these are proven methods in general use cases, this particular case was somewhat different. We analyzed why these methods failed and found out that the added value was much smaller than the noise the LLM added. Oftentimes the LLM did a good job at making extra queries for the retriever, but sometimes it was confused by either a lack of general understanding or a lack of medical knowledge. In the end, the average retrieval score went down, proving that the added noise was too large.

3.2.5 Generation

After the retriever has found a list of possibly relevant chunks, the generator uses that information to generate an answer. To optimize a generator, there are broadly speaking two common options: making the prompt better and making the model better. The latter option is usually work-intensive and resource-intensive, making it out of scope for this thesis. By consequence, the following paragraphs will discuss only prompt engineering techniques used in the application.

The first technique is to properly separate pieces of information. The pragmatic approach was to use the first separation technique that the generator seemed to handle well. This turned out to be the following style:

```
[begin context]
Jij bent een dokter. Jij kent alle medische procedures van het UZGent.
[einde context]
[begin documenten]
{context}
[einde documenten]
[begin vraag]
{query}
```

```
[einde vraag]
[begin antwoord]
```

This is of course the prompt for the G in RAG. As stated before, the generator was also used in other settings, but the same style with brackets was kept. As a sidenote, the context of "You know all medical procedures of the UZGent" might seem weird, motivating the generator to answer possibly more often than it should. However, without this part the model seemed to be fine-tuned to say it cannot answer medical questions. So it is a little hack to get an answer at all.

The second technique is few-shot prompting [8]. This uses in-context learning to show the model what typical input is and the corresponding model answer. At the time of development, the preferred answer of the full pipeline was still unclear, so few-shot prompting was not used. But for other techniques, such as HyDE, it did yield improved results (though still not good enough).

3.2.6 Post-processing

The last part of the pipeline is to make the output presentable for the end user. Figure 3.8 shows the user interface with an example question and answer. Not only do we present the answer, but also the used context and where it came from. To this end, the plain text is searched again in the original document (pdf) and highlighted. And as a nice addition, the pdf is automatically scrolled to the highlighted part.

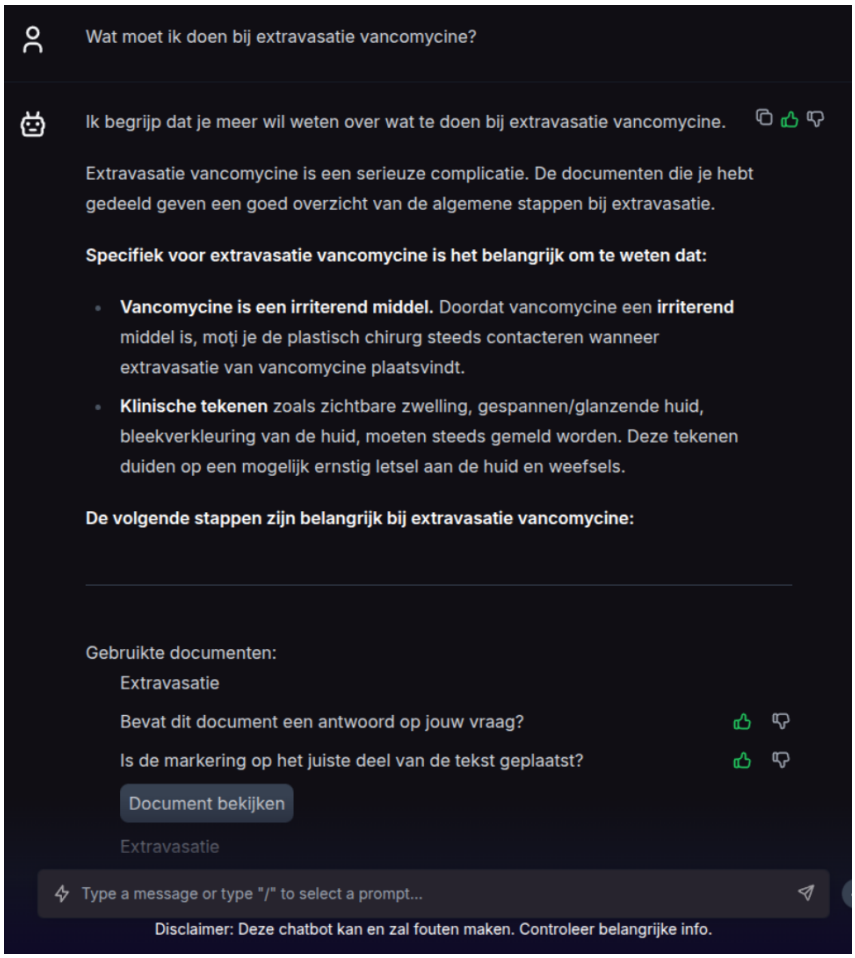


Figure 3.8: Screenshot of the user interface of our chatbot.

3.3 Application diagrams

The department of ICT at UZGent was very enthusiastic about this project and they showed interest to understand how the web application was made. We made the following diagram to clarify what was going on in the background. Figure 3.9 explains the whole application in high level components. While a simple diagram now, this used to be different. We put significant effort into containerizing each component, making them as independent as possible and building those efficiently to reduce build time. The orchestrator is simply a GitHub repository that contains a docker compose file and some necessary configurations. This docker compose will then spin up the docker containers. Most of them are straight from an image, but some of them are built with local code. The RAG container is built from local python code and it runs the API. The API's input is a question and it returns the answer with corresponding chunks. The website is also built from local code and it is a typical chat interface that forwards questions to the API. The reverse proxy uses the well known Nginx [58] in Docker [15] to allow https connections to the website. The vector database is also a default Weaviate [79] container, just like the database is a default PostgreSQL [63] container. Finally, vLLM [42] also supports a containerized version making it very easy to host LLMs in a scalable way.

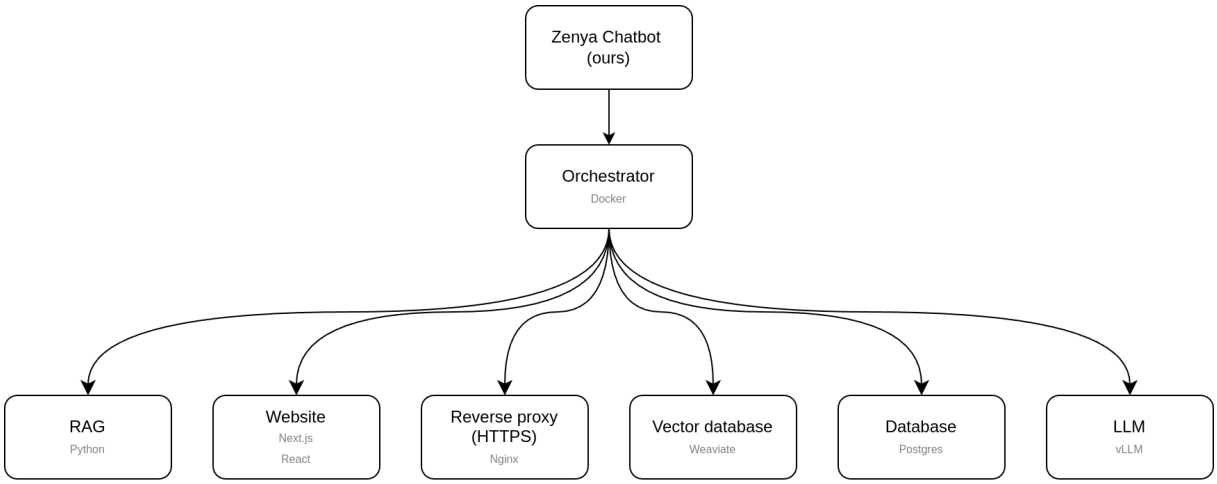


Figure 3.9: A high level diagram that lists the different docker containers of our application.

3.4 Test results and feedback

To evaluate the RAG system for practical value, we organized a test session where a few dozen healthcare professionals of UZGent asked the chatbot questions they would typically ask Zenya during their daily work. For these questions, the relevant documents are retrieved and an answer is generated. The healthcare professionals then evaluate the output by indicating which documents contained essential information and which questions were answered correctly. This section discusses both the test results and the informal feedback gathered during these sessions.

3.4.1 Objective results

Before going into the test results, some caveats must be pointed out. First of all, this test data is important for future use at the UZGent, as they want to use it to evaluate all possible competitors that claim they can do better than the current system, Zenya. This means a validate set is necessary to hand over to the competitor and a test set to keep for themselves. As the writer of this thesis is both the person gathering the data, making the split and also testing his system, this is something of a violation against best practices. Thus, the UZGent will have to trust us not to make any decisions based on the

test set.

The next problem, is that the test set turned out to be quite unclean. For example 20% of the ground truth documents were derived from the context. While this would work if every tester followed the instructions perfectly, testers are human and make mistakes. Thus, sometimes this derived ground truth is not actually right. Additionally, 8% of the questions were left without any clue what the ground truth would be. Ideally, these questions should be revised by an unbiased person at the UZGent itself. This was done in a quick way, but further cleaning will be necessary.

That being said, the results are a mix of multiple interpretations that try to keep these issues into account. First, the direct results of the full test session are given. These yield the most pessimistic view, because of the noise discussed before and there is no bias towards the writer's opinion at all. Table 3.1 shows the most high level question: did the user find the answer or not? The first thing that becomes clear is that the answer is almost always found in Zenya. The discussion about why this is biased is given in the paragraph about the informal feedback. The second conclusion is that 54% of the questions were properly answered by the RAG system. This is not exactly a great result, but let us consider all metrics before drawing conclusions.

Figure 3.10 analyzes how many unique liked documents a conversation had. A conversation is defined as a sequence of messages to the chatbot until the healthcare professional is comfortable with the answer and they would use the information in real life. A liked document is a document in which the healthcare professional found essential information to answer the question. The most important insight in the writer's opinion is that about 35% of the conversations had no right document found, yet 46% of the conversations were labeled as "answer not found". This further indicates the need for cleaning of the seemingly inconsistent dataset. While many other conclusions could be drawn here, the discussion is limited to the essentials. This is because focusing on the details could be focusing on the noise as described in the caveats. On a finer granularity, Figure 3.11 looks at the number of liked documents per question. The first interesting feature is that sometimes multiple documents are labeled as good. This information is invaluable to the UZGent, as they expected one document to always suffice. This could show the existence duplicate information across documents or it could indicate complexer questions have their answer scattered across documents. A detailed investigation of this is left to the experts of the Zenya data at the UZGent. Next to that, it can be seen that 48% of the questions did not get a relevant document returned. This number does not really impress, though it is the pessimistic view of course.

	Not found with RAG	Found with RAG
Found with Zenya	39	50
Not found with Zenya	4	1

Table 3.1: The distribution of all conversations, indicating whether the answer was found or not. This is shown for both the RAG system and the old system, Zenya.

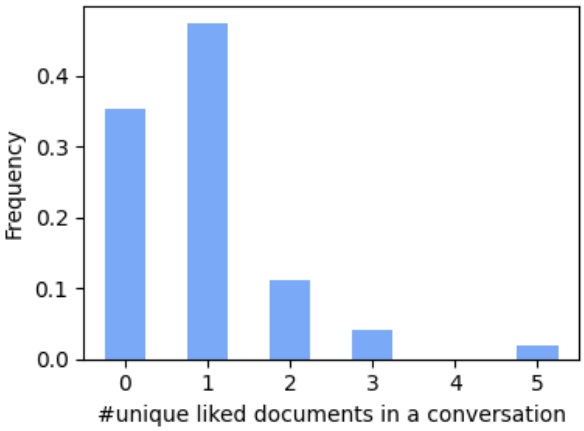


Figure 3.10: Distribution of the number of unique correct documents found in a whole conversation. 0 means that no relevant document was found before the conversation was finished. If 1 or more correct documents were found (liked), it was a successful conversation.

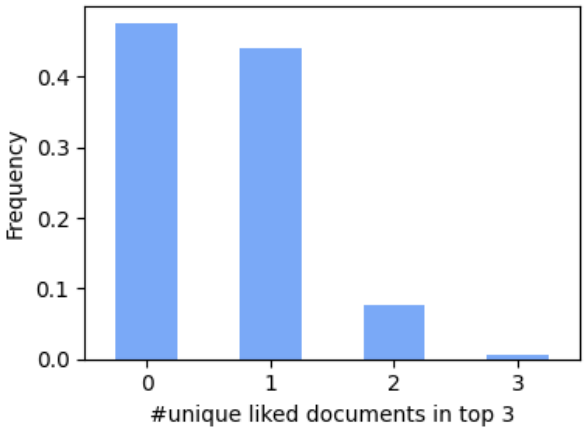


Figure 3.11: Distribution of the number of unique correct documents found per individual message. This is similar to the previous plot, but on the level of individual messages. If more than 1 document was correct (liked), it means that the question required parts from multiple documents to answer correctly and completely.

While the direct results are the most unfiltered, multiple red flags indicate that the statistics are not extremely reliable. For this reason, we filtered out the noise as follows, trying to remain completely unbiased in the process. All messages without any possible way of deducting the right document were left out, as approved by the UZGent. Combined with some manual labeling by the UZGent, the following results should be representative.

Figure 3.12 and 3.13 show the same information with different emphasis: how the rank of the first relevant document is distributed. Figure 3.12 shows the density. This visualization shows very well how most queries find the right document in the earlier ranks. When a document was not in the top 10, the chances clearly drop that the document is found at all. An important note to make is that a document not found is displayed as the maximal rank: 50, because by then we expect the healthcare professional to never find it. With this information, one can see that about 10% of the documents were totally not found. Figure 3.13 shows the cumulative distribution. This representation shows better what portion was already found by rank x. This shows the top 3 documents already have the right one 69% of the time. We mention the Hit@3, because if the correct document is so highly ranked, it could be considered a perfect retrieval. The Hit@7 is 77%, which again means 77% of the times we have the right document in the top 7. We chose Hit@7, because after a Zenya query, the healthcare professional sees 7 documents without scrolling any further. 90% is found in the top 50 documents and as said before, 10% are not found at all. The Hit@50 mostly shows how often a healthcare professional could find a document at all, because they do not have time to scan beyond 50 titles.

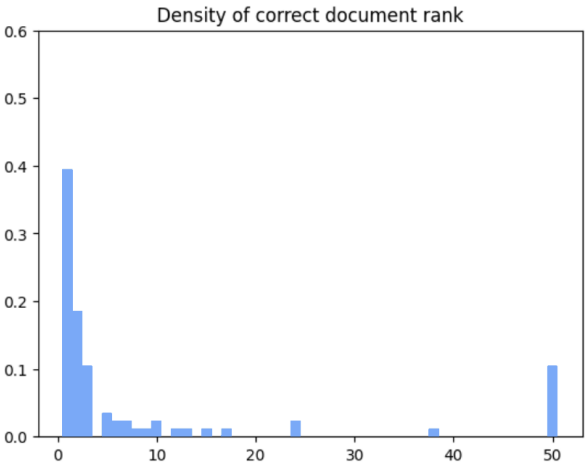


Figure 3.12: Distribution of the rank of the first correct document. More density to the left is better.

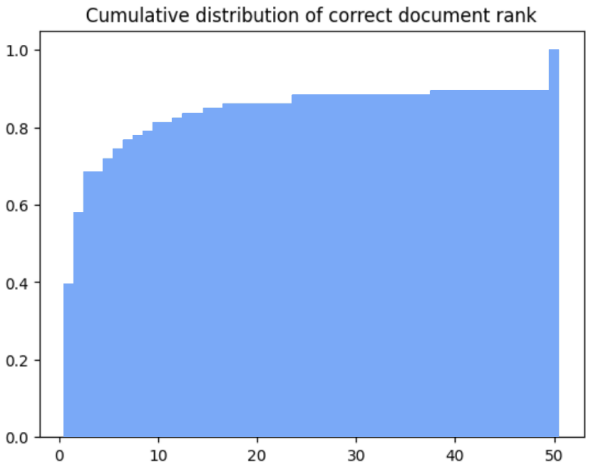


Figure 3.13: Cumulative distribution of the rank of the first correct document, where the value shows the chance that a relevant document was somewhere in the top-k documents. Higher values are better.

3.4.2 Informal feedback

The informal feedback provided great nuance to the objective feedback. The first and most important conclusion by most testers was that this system is not ready for production yet. As they were referring to the whole RAG system, this was a logical conclusion, given the large number of hallucinations. However, when asked about the retrieval only, the testers were more positive. Some were already convinced about the new model, while others still saw much work to be done. As a trend, the testers who work in a niche domain, were less convinced than the other ones. Interestingly, they said themselves that this problem could be very easily fixed by adding user information to the query. This means that a pediatrician should only get documents about children, or at least get them first. This information was not available, so it could not be implemented during the thesis. However, this is very important to be aware of for any future developer making retrievers for the UZGent.

One particular anecdote about the informal feedback really stood out: a nurse said that they are usually looking for five minutes in the current system, Zenya, before finding the right document, if it is found at all. While this was the most extreme example, the sentiment of the other testers seemed to be similar. They claimed that in the current system, they need to know what they are looking for exactly, before searching or else they will never find the right document. This is of course the opposite of how a good retrieval system should work. The main conclusion out of this is that the results in Table 3.1 are to be taken with a grain of salt. For a fair comparison of both systems, one should choose testers who do not know either system yet. This will probably shift the results towards "not found" in Zenya, possibly even indicating our RAG system as the winner. Such tests are future work though.

Another type of feedback was implicit. Some testers asked how to pose their questions or just struggled to find a good question by themselves. This means asking a logical, natural language question is not as easy as first thought. Now, it must be said that the testers were biased. As they are Zenya experts that were trained for years to use only keywords, giving a natural language question to a computer just felt off for them.

3.5 Future work at the UZGent and on the RAG system

In this section, a list of ideas is given to the UZGent, should they be interested to further develop the RAG system. The future steps are ordered by importance.

- The first step to do proper machine learning in general, is to have a high quality validate and test set. The sets we propose are a good start, but an increase in quantity is desirable. This is important, not only for further development, but also to compare other systems. For this reason, this is deemed to be the most important work, that needs to happen first.
- As said before, adding metadata to the retriever's input could provide context that is necessary to retrieve the right document. Both metadata about the documents (which exists, but was not available during the thesis) and metadata about the user can be used to filter documents.
- Another high impact improvement is to find a better embedder. While the currently used embedder is with reasonable certainty the best for the use case today, new great embedders pop up on HuggingFace occasionally. And as the difference between the best and second best embedder was immense, a new model might as well make such a large jump in the score.

As final recommendations for the UZGent we suggest the following. **Drop the generative model** and focus on improving the retriever. We see only advantages in this:

- No risk of hallucinations. A retriever alone does not generate text and it leaves the responsibility to the healthcare professionals, who are more than capable to critically read the documents and asses the relevance.
- Reduced computational cost, which reduces both the energy consumption and the latency.
- Reduced server requirements, which allows running the application on a server with a cheap (or even no) GPU.

Even if generative language models were to be used in critical applications such as healthcare, we would advise to be very critical and only bring it to a production environment when it is objectively proven to be extremely accurate.

3.6 Ethical considerations

While the critical domain of healthcare brings many ethical considerations, this section will focus on one specifically: the hallucinations of the generator. For clear context: the demand for a RAG application came from the UZGent itself, before being implemented for this thesis. However, in the presented use case, we cannot afford critical mistakes with odds one to a million. As a typical generator hallucinates with much higher odds, a production-ready product was far out of scope of this thesis. For this reason, it is proposed only to work further with the retriever and to be very wary about any application that uses a generator directly to hand critical information to a healthcare professional.

To make this suggestion more hands-on, we give an example of a good application and a risky application. A retriever using HyDE is good, because:

- The generator only helps the retriever.
- The final output is a retrieved, trustworthy document. In the worst case, it is an irrelevant document, which the healthcare professional can see and then they move on to the next document.

As a risky example, consider the RAG use case. Some documents are retrieved, then the generator makes an answer with them. This is risky, because:

- The generator is the last step in the pipeline, and it provides possibly critical information.
- The generator could hallucinate.
- The generator could ground its answer in the context, but the context could be wrong. For example, the generator once gave a seemingly perfect answer, using facts found in the document, but the document was for people older than 15, while the question said the patient was younger than 15. There were two documents for both cases and the contents were extremely similar, except for the drug doses. Needless to say, such confusion is dangerous.
- A common argument is that "The professional will notice" when the generator is hallucinating, but if that is the case, why are we generating an answer that the healthcare professional already knows?

While we do not advocate against any use of LLMs in these applications, we do urge to assess them with the necessary scrutiny. Only if objective measures are decisive, it can be brought to production.

Chapter 4

Speculative Decoding

The academic part of this thesis revolves around speculative decoding. This chapter provides details about speculative decoding and the current landscape of research. It serves as context for the next chapter, where our new model, CASD, is proposed. First we explain the context of speculative decoding, followed by how it works. Finally, we give an overview of the most recent advancements.

4.1 Context of speculative decoding

Figure 4.1: Autoregressive decoding by standard LLM.

To understand the need for speculative decoding, one must first understand the usual way of decoding with LLMs. Figure 4.1 shows the normal, autoregressive way of decoding. Autoregressive means that the LLM "reads" all previous tokens to generate the next token. So to construct an answer, the LLM takes the input and generates one new token. Then it takes the input and the new token and generates another new token. This is repeated until the LLM decides the answer is complete or the maximal output length is reached.

The inefficiency of this autoregressive decoding is that it does not use the full capacity of the GPU it is running on. While GPUs are very complex, we can still simplify their limitations to two constraints.

On the one hand, a GPU has limited computing power, which means it can only do so many operations per second. On the other hand, GPUs can only load a limited amount of data per second. Surprisingly, the latter constraint is the real bottleneck for LLMs. This is because each time the LLM is called (each generation step from before), the LLM parameters need to be loaded from the GPU's memory (VRAM) to the GPU. This large amount of data seems to be slower to transfer than the actual time the GPU needs to compute the results [26, 39]. In other words, the GPU could calculate faster, but loading the model each time keeps it from doing so.

There are two types of solutions for this bottleneck. The first one, batching, is relevant to the larger players in the market. Suppose an LLM host receives many requests per second. In this case, the questions can be batched, as shown in Figure 4.2. This means that the server waits for a certain number of questions to arrive, before processing them all together, in parallel. The effect is that the model is only loaded once into the GPU, while generating the next token for each of the questions. So if there are for example 16 questions in a batch, the GPU is calculating 16 times as much, while loading the same model only once. Thus, the calculation intensity is increased and much of the spare calculation-capacity is filled in. In conclusion, this method increases the GPU efficiency, by exploiting the many requests coming in per second.

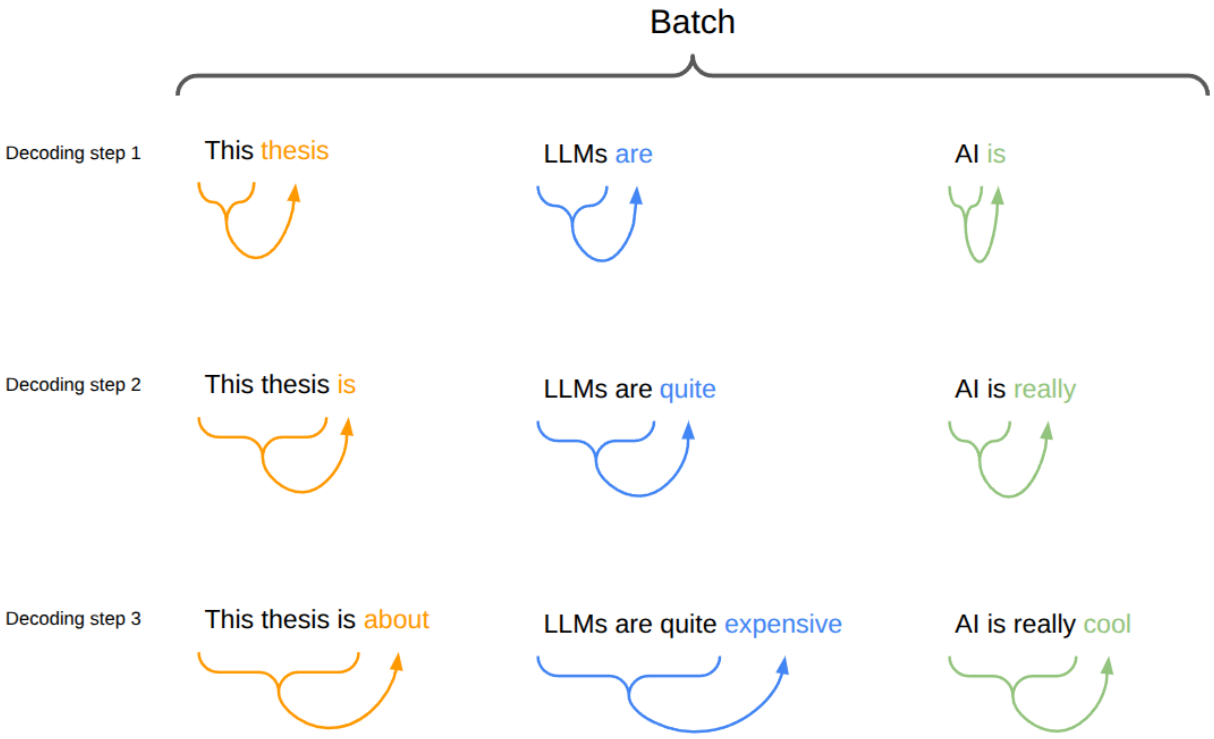


Figure 4.2: Batch decoding on a small example. The different users' prompts are indicated with different colors.

The second solution to the GPU memory bottleneck is speculative decoding. Speculative decoding does not need the large load of requests per second to achieve parallelism. Instead, it tries to break the autoregressive nature of LLMs and to parallelize it. The following section describes how that works in detail.

4.2 How does speculative decoding work?

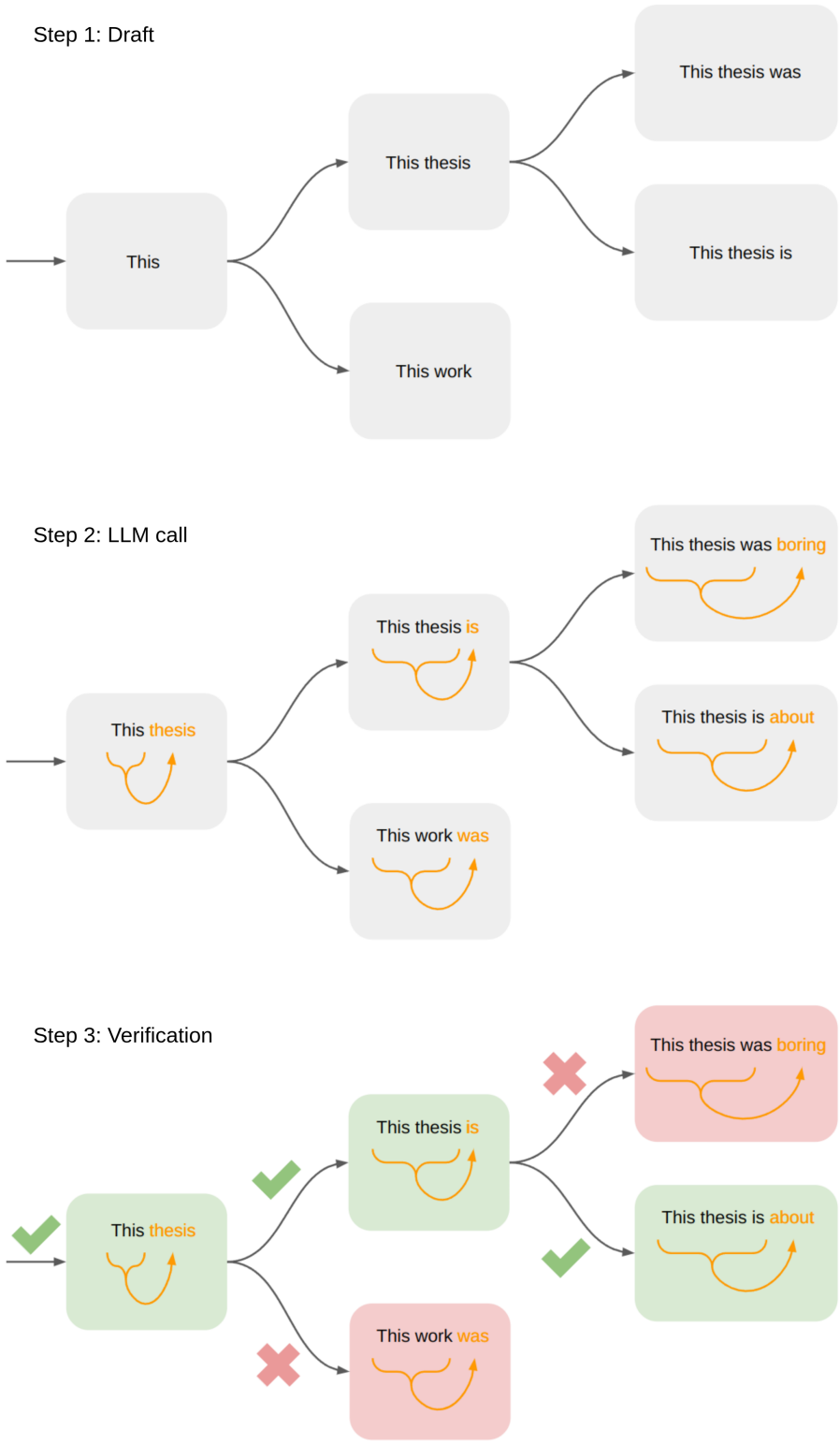


Figure 4.3: Step-by-step explanation of speculative decoding: in step 1, the drafter makes a tree of possible outcomes. Each draft is the same as its parent plus one speculated token. In step 2, the LLM call determines which token would be generated next for each of the speculations. In step 3 each speculated token is verified. The longest correct speculation is accepted. Here that is "This thesis is about".

Speculative decoding tries to form batches, but rather than looking at multiple users, it looks multiple decoding steps ahead. To do so, it follows a three-step plan. These steps are detailed below, with Figure 4.3 as a guide.

The first step is to make a good guess (a.k.a. draft) what the LLM will generate, without calling the LLM itself. This step is the hardest step and by consequence the focus of most research in the domain. Many clever methods have been found already and we provide a short overview of them in Section 4.3. This thesis also focuses on the drafting.

In the second step, the guesses are verified in parallel. To do so, a tree is made of all predictions. In this tree, guesses with a common prefix are joined, such that the prefix has to be verified only once. Once the tree is made, each node of the tree needs to be run through the LLM to see what the correct token is at that step. When sending all tokens to the LLM, it must be sure that tokens from different guesses do not interfere with each other. This is made possible with tree attention. Tree attention puts a mask over the attention, such that tokens from other branches in the tree are "invisible" to the current branch. So to conclude, tree attention makes it possible to call the LLM only once, while verifying many guesses.

The third step is to accept the longest correct guess. A correct guess can have multiple interpretations. When the temperature is 0, it is easier to define. Temperature 0 means that the LLM will always, deterministically pick the most probable next word. In this case, correct means that the draft chose exactly the same tokens as the LLM. In the case of temperature different from 0, the answer is not deterministic anymore. So more calculations are involved to guarantee the output distribution is the same as the LLM would have chosen. As this sampling method is not the focus of this thesis, we do not explain it further and we limit the discussion to temperature equal to 0.

One last remark must be made to fully understand the accept step. The real accepted length is always one larger than the longest accepted guess. How is that possible? Suppose we had no correct guesses, then we still called the LLM on the original output, yielding one new token. Similarly, if we had n correct tokens, we also called the LLM on the n-th token and know what the n+1th token should be. This last part ensures one extra token is accepted each iteration.

4.3 Drafting methods

To understand the innovation of our new method, CASD, we must first look at the existing drafters and what their weaknesses are. The existing drafters can be divided into two categories: the statistical speculators and the neural speculators. Statistical techniques are more hand-crafted methods, while neural methods train a neural network to do the drafting. In this thesis, we will introduce a third category: the hybrids that combine statistical and neural methods. Below, we list some examples of recent work in the two standard categories.

Starting off with the statistical techniques: REST [30] makes drafts by scanning existing text to find probable snippets. Practically, they use a large dataset of existing documents to find continuations of what is currently being generated. This is visualized in Figure 4.4. We also mention Inference with Reference [84], not for its performance, but because it has similarities with the method we propose. Inference with Reference works similarly to REST, but instead of looking in a large dataset, it looks back at the prompt and the generated tokens to see if it is copying from its own context. Statistical models are typically very fast, but they do not generate the most accepted tokens on average. They are outperformed by the neural models, which are very good at predicting a few tokens very consistently.

The second category of drafters, the neural speculators, can be further subdivided based on their key ideas. The key idea of the earlier methods was to use a smaller LLM to speculate what the larger LLM would generate [46]. However, they do not exploit the features that are generated by the LLM "for free". These features contain valuable information, the thoughts of the LLM in a sense. So the key idea of most

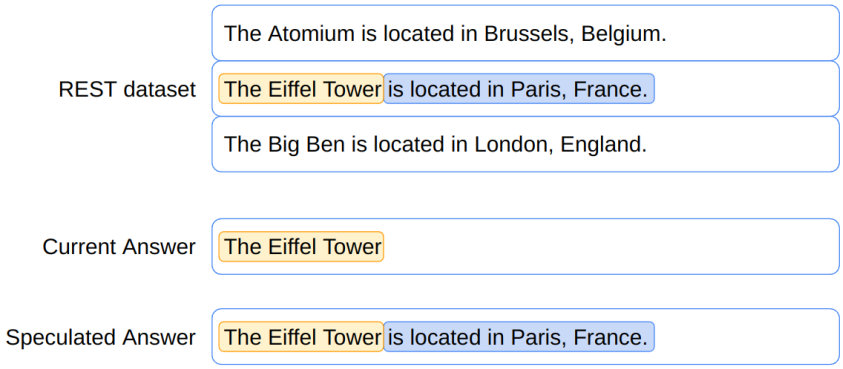


Figure 4.4: Example of REST [30] drafting (our figure). In the large dataset, REST found a match in a document and it speculates the words that follow it in the document.

new models is to use the features generated by the LLM too, as an extra source of input. For example, the current state-of-the-art, EAGLE-2 [49], tries to generate the next features autoregressively instead of generating the tokens directly. This is visualized in Figure 4.5. The core weakness of neural speculators is that they are defined by their training set. If they are trained on English, they will probably work well only on English (as we show later). Thus, anything out-of-distribution can be a weak spot.

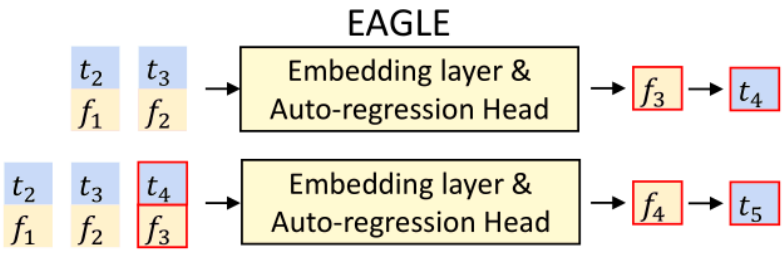


Figure 4.5: EAGLE-2 uses the LLM's features and previous tokens to predict the next step. It is used autoregressively, to draft multiple tokens ahead. Figure from EAGLE-2 [49].

Chapter 5

Context-Aware Speculative Decoding

There are many stand-alone techniques to do speculative decoding. Each technique tries to perform as good as it can by itself. However, what seems to be underexplored in the domain of speculative decoding is the possibility of augmenting techniques. An augmenting technique will use a strong baseline method and try to fill in the gaps, where the baseline model underperforms. This principle is shown in Figure 5.1. To have a good augmenting model, it must be fundamentally different from the baseline model. We present Context-Aware Speculative Decoding (CASD), which is such an augmenting technique, that generates drafts where the current SOTA (EAGLE-2 [49]) underperforms.

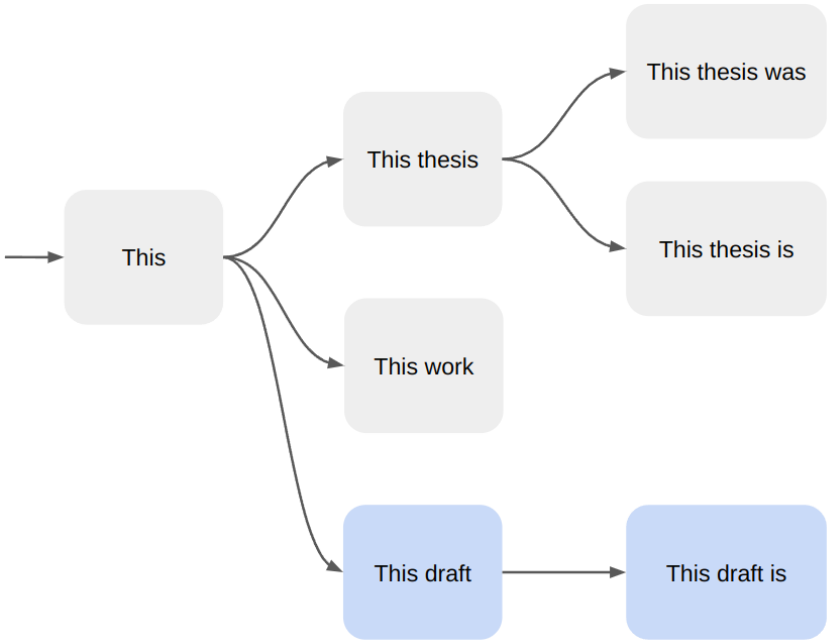


Figure 5.1: An example draft by a base speculator (e.g. EAGLE-2) and an augmenting method (e.g. CASD). The augmenting method has merged its drafts with the baseline's drafts to form one draft tree.

5.1 Research hypothesis

A common use case for LLMs is to use it in a RAG framework. As Section 2 explains, RAG uses an LLM to answer questions based on some relevant document chunks, so the context of the LLM is prefilled with chunks before answering. This context contains the facts that the LLM is expected to use in its answer.

To improve factuality, the preprompt often contains explicit or implicit instructions to use or even copy the context. This is what Context-Aware Speculative Decoding will try to pick up on. When the LLM will copy long sequences of tokens from the context, a reasonable draft could be made by just looking at the context. The hypothesis is that this yields long, good drafts, that are quite independent from neural-based methods.

5.2 Drafting method of CASD

CASD is a lightweight statistical method to generate speculative samples from the context itself. To do so, the last generated tokens are searched in the full context, as shown in Figure 5.2. If at least one token corresponds to the last generated tokens, we call it a match. When there are multiple matches found in the context, the top 2 are kept based on how long the match was. Then, up to 10 tokens after the match are copied as drafts. Finally, the drafts are merged with those of EAGLE-2.

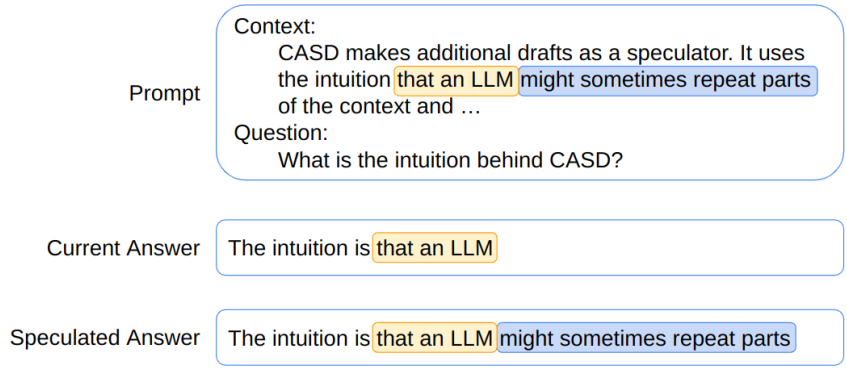


Figure 5.2: CASD applied to a simplified RAG prompt. The prefix ("that an LLM") is found in the context and thus the continuation ("might sometimes repeat parts") is drafted by CASD.

5.3 Results

In this section, we analyze the performance of CASD, mainly comparing with EAGLE-2.

5.3.1 Experimental setup

Models. The experiments use only Llama-3-8B-Instruct [27] as an LLM, as only this model fits our following constraints. The first constraint is the hardware: 8B is the largest model that fits on the used GPU. Since one of our benchmarks has Dutch content (UZGentRAG, see further), we also need a Dutch-capable LLM. Lastly, the SOTA baseline, EAGLE-2 also has to support the LLM. This yields Llama-3-8B-Instruct as the chosen model for all experiments.

Tasks. CASD is evaluated on our private UZGentRAG benchmark (see further), SQuAD [66] and the classical speculative decoding benchmarks (Alpaca [73], GSM8K [13], HumanEval [11], MT-bench [88], Natural Questions a.k.a. QA [41] and CNN/DM [56]). All experiments were performed with batch size 1, conform with most related work [40, 46, 49, 89].

Metrics. CASD, being an augmenting method, focuses on improvements on the baseline speculator. The usual speculative decoding metrics, applied to CASD are:

- *Walltime speedup ratio*: The speedup ratio relative to the baseline method, EAGLE-2.
- *Average acceptance length* τ : The average number of tokens accepted per forward pass of the target LLM.

- *Energy improvement*: The improvement in energy efficiency relative to the baseline method.

We approximate the energy consumption by the consumption of the GPU only. This is because our virtualized setup could not measure CPU and RAM energy accurately. However, the energy consumption of the whole system is primarily determined by the GPU [50], so this indicator should be reasonably accurate.

5.3.2 UZGentRAG benchmark

We hypothesized that EAGLE-2 would underperform in multilingual settings and niche domains. This is because EAGLE-2 is trained on mostly English data in rather general domains (ShareGPT [2]). That means niche domains are out-of-distribution and neural networks are known to have bad performance on out-of-distribution input. We must note that the neural network of EAGLE-2 works on top of the features of the LLM, so if the LLM is trained to have language-agnostic features that would not be the case. However, the latest LLMs are typically not trained in that way and thus the problem remains.

To prove our hypothesis, we had to go beyond the standard English speculative decoding benchmarks. However, low-resource languages such as Dutch have limited benchmarks available and for RAG use cases we did not find any. That is why we made our own benchmark. As mentioned in Section 3.4, we held a test session at UZGent. In that session, we not only evaluated the RAG system, but we also constructed UZGentRAG, a private niche RAG benchmark in Dutch and for the medical domain. This benchmark contains the queries logged from the daily work of doctors and nurses, by recording their questions to a chatbot. Internal UZGent documents found by the chatbot were combined with the query in a typical RAG prompt. These prompts form the benchmark, as that is the only data necessary to test a speculative decoding method.

While we cannot make the entire dataset public, we are allowed to give some insights. Table 5.1 shows some relevant statistics of UZGentRAG, compared with SQuAD. The number of prompts of UZGentRAG is rather limited for a benchmark, however we see that EAGLE-2 uses 80 questions per benchmark and for speculative decoding that order of magnitude seems acceptable. That is also the reason our RAG adaptation of SQuAD was capped at 80 questions. The prompt length might also surprise, because SQuAD has rather large prompts. This is because the adaptation required few-shot prompting to achieve satisfactory answers from the LLM and those few shots take up some space in every prompt. For the length in tokens, we used the Llama 3 [27] tokenizer, consistent with our experiments later. The last metric, BLEU, is good at measuring how much is copied between context and answer; exactly what CASD needs. It is clear that UZGentRAG often requires copying from the context, but that holds even more for SQuAD. As this thesis is not about the BLEU score, we do not further elaborate on how this metric checks for copying. For reproducibility, it needs to be said that we edited the metric to have no brevity penalty as this is not relevant to our use case.

	UZGentRAG	SQuAD (RAG)
language	Dutch	English
# prompts	165	80
# chunks per prompt	3	1
mean prompt length	2420 char - 780 tokens	2947 char - 615 tokens
mean answer length	849 char - 83 tokens	83 char - 18 tokens
BLEU score (no brevity penalty)	0.25	0.64

Table 5.1: Statistics of UZGentRAG benchmark, compared with the most similar benchmark: SQuAD (adapted for RAG with speculative decoding).

In order to give some idea what a typical UZGentRAG prompt might look like, one shortened example is given below. For the full prompt, we refer to Appendix B.

```
[begin context]
Jij bent een dokter. Jij kent alle medische procedures van het UZGent.
[einde context]
[begin documenten]
*chunk 1*
---
*chunk 2*
---
*chunk 3*
[einde documenten]
[begin vraag]
Wat moet ik doen bij een graad 1 immuun-gerelateerde hypothyroïdie?
[einde vraag]
[begin antwoord]
```

5.3.3 Performance of CASD

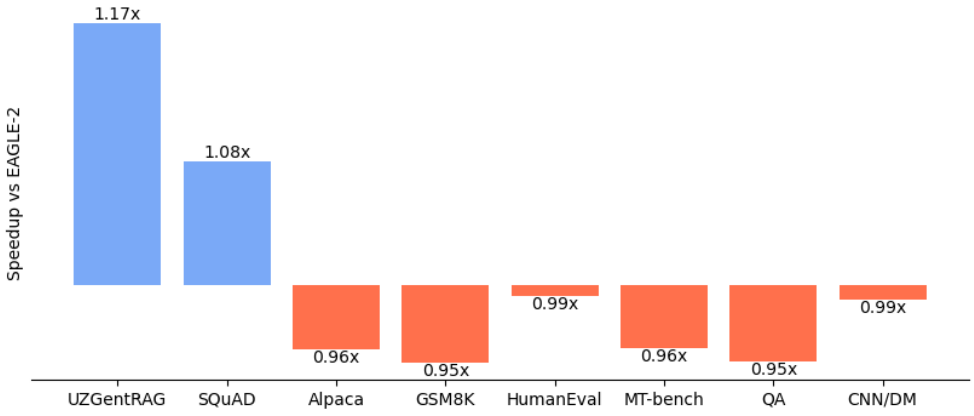


Figure 5.3: Speedup of CASD compared to the baseline, EAGLE-2. CASD yields improvements for both RAG benchmarks. On standard speculative decoding benchmarks the overhead is still rather limited.

Figure 5.3 shows the comparison between CASD and EAGLE-2. The most relevant benchmarks are the RAG benchmarks, UZGentRAG and SQuAD, for which CASD achieved a speedup of 1.17 and 1.08 respectively. This is quite interesting, because both benchmarks are RAG use cases, yet the results differ strongly. We hypothesize that the Dutch UZGentRAG is harder for EAGLE-2, so it is easier for CASD to find better continuations.

Next to the RAG benchmarks, we also list the general benchmarks. Since these are not RAG-based, and EAGLE-2 was designed for these settings, performance is now greater for EAGLE-2. The key takeaway here is that even when applying CASD to non-RAG input, it still has limited overhead (1-5%).

The full details of the performance can be seen in Table 5.2. The first thing that stands out is that the relative increase of the number of correctly speculated tokens ($\tau_{CASD}/\tau_{EAGLE-2}$) is higher for SQuAD than for UZGentRAG. In other words, the number of LLM calls were reduced most for SQuAD and nevertheless the speedup is much higher for UZGentRAG. This likely means that SQuAD had many more substring matches, increasing the overhead of CASD and thus slowing down the total system.

Another interesting deduction we can make is that UZGentRAG successfully addresses the pain points

of EAGLE-2. It has a τ of around 4 in general use cases, around 6 in the case of SQuAD (RAG) and only 2.09 for UZGentRAG (also RAG). This strong difference shows that EAGLE-2 struggles with the combination of Dutch and the medical domain. The scope of this thesis did not allow a further study to see whether Dutch or the medical domain was the dominating factor here.

		CASD		EAGLE-2		CASD vs EAGLE-2	
		Tokens/s	τ	Tokens/s	τ	Speedup	$\tau_{CASD}/\tau_{EAGLE-2}$
RAG benchmarks	UZGentRAG	**35**	**2.73**	30	2.09	1.17	1.30
	SQuAD	**46**	**7.78**	43	5.88	1.08	1.32
General benchmarks	Alpaca	61	**4.15**	**64**	4.12	0.96	1.01
	GSM8K	64	**4.56**	**67**	4.42	0.95	1.03
	HumanEval	75	**5.55**	**76**	5.05	0.99	1.10
	MT-bench	61	**4.35**	**63**	4.18	0.96	1.04
	QA	52	**3.56**	**54**	3.53	0.95	1.01
	CNN/DM	52	**4.14**	**52**	3.74	0.99	1.11

Table 5.2: Performance of CASD compared with the baseline EAGLE-2. The best score is in bold.

5.3.4 Energy improvements by CASD

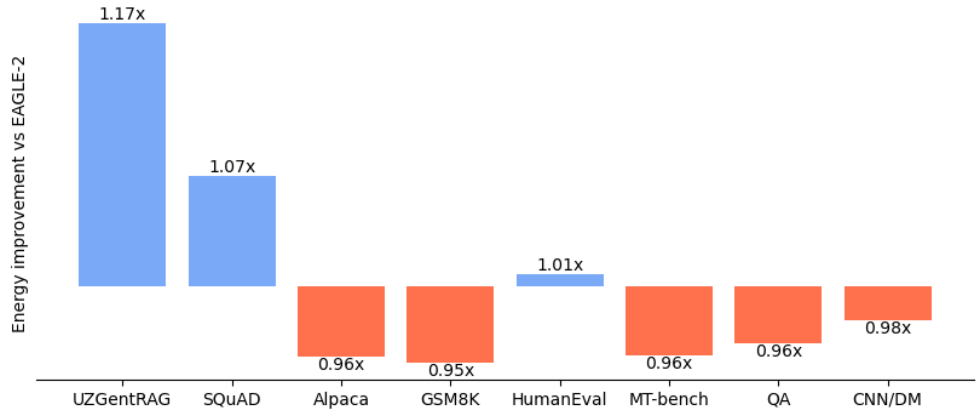


Figure 5.4: GPU Energy improvements of CASD compared to the baseline, EAGLE-2. CASD yields improvements for both RAG benchmarks. On standard speculative decoding benchmarks the overhead is again rather limited.

Figure 5.4 shows the improvement in GPU energy consumption. Interestingly, we see very similar numbers as for the speedups. This should not necessarily be the case and it depends on the speculator's speed-efficiency, power-efficiency and token-efficiency, so it is a pure coincidence. Practically, this means the conclusions are the same: applying CASD to RAG use cases yields improvements and applying it to other cases generally yields worse results. As a side note, HumanEval did see a slowdown of 1%, but also an energy improvement by 1%, making it the only benchmark that has opposite results. However, the difference is insignificant and might as well be noise on the measurements.

5.3.5 Acceptance length

To understand why CASD yields improvements on neural speculators, Figure 5.5 shows how the acceptance length is distributed. EAGLE-2 works quite well and rather consistently: most often it has 4 to 7 tokens accepted. On the other hand, CASD's added speculations show a strong peak at 11. So if CASD gives the best speculation, it is a high quality speculation for a long length. This also explains intuitively why CASD works so well with neural speculators: CASD has occasional speculations that are extremely

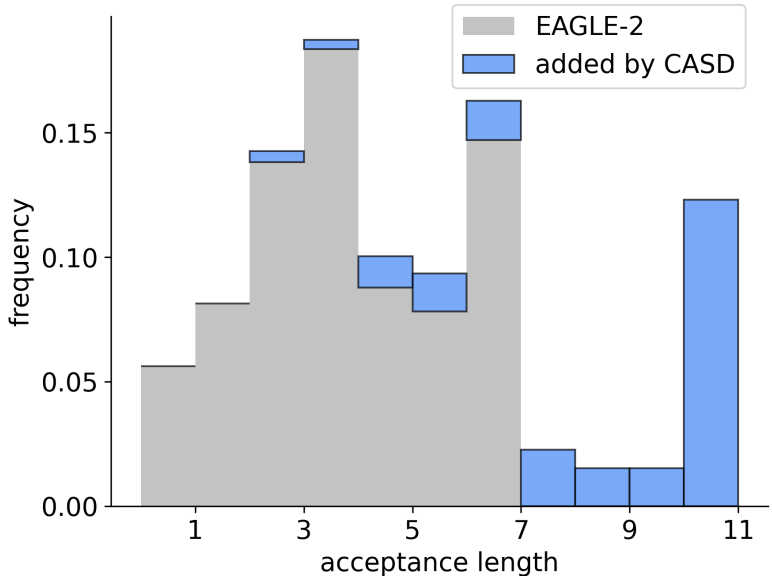


Figure 5.5: Distribution of acceptance lengths with CASD augmenting EAGLE-2. CASD supplements EAGLE-2 with long-length drafts.

good, while the neural speculator works well consistently when nothing is copied. As a last remark, it might stand out that there is a hard cap at 7 and 11 for EAGLE-2 and CASD respectively. Both methods implement this cap to balance overhead with more accepted tokens.

5.3.6 Ablation study

CASD copies drafts from the entire context, but also from the tokens that were already generated. However, for the original problem statement (RAG), CASD only had to copy from the prompt and not from the answer that the LLM had generated up until that point. If CASD did that, it could be better if the LLM almost never copies from the generated tokens, as it has a smaller overhead. The opposite could as well be true, where CASD benefits now from an LLM repeating itself. The following empirical tests show whether the extra LLM tokens actually provide benefits: we compare our original CASD to CASD_{prompt}, which only considers prompt tokens.

Table 5.3 shows how CASD_{prompt} performs. For the most relevant benchmark, UZGentRAG, a slight improvement justifies the choice of CASD over CASD_{prompt}. For SQuAD, τ stayed the same and so did the throughput. Finally, MT-bench shows very similar performance. So it seems that CASD performs better than CASD_{prompt}, mostly on the UZGentRAG benchmark. As this benchmark contains longer answers than SQuAD, we intuitively think that the LLM starts repeating generated fragments more the longer the answer gets.

	CASD		CASD_{prompt}		EAGLE-2	
	Tokens/s	τ	Tokens/s	τ	Tokens/s	τ
UZGentRAG	**35**	**2.73**	34	2.65	30	2.09
SQuAD	**46**	**7.78**	**46**	**7.78**	43	5.88
MT-bench	61	**4.35**	61	4.30	**63**	4.18

Table 5.3: Performance of CASD_{prompt} compared to CASD and EAGLE-2 as the baseline. The best score is indicated in bold.

5.4 Further analysis

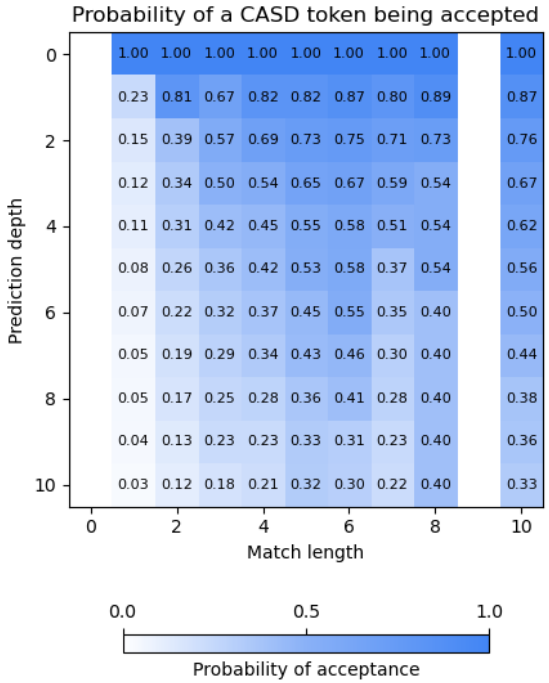
This section shows the reasoning behind some implementation choices, by further analyzing CASD.

In CASD, we only take the top-2 matches according to match length. This implicitly assumes that longer matches are better and Figure 5.6 (a) validates this assumption. The first interesting feature is that the further to the right, the higher the probability (typically), and this for all rows. This shows that the average accepted length grows longer with longer match lengths. But not only does this validate that longer matches generate better speculations, it also opens up an area of future work. Instead of taking the 10 next tokens of a match regardless of the match length, CASD could also work with adaptive length speculations. A good idea could be to put a threshold on a certain acceptance probability and only add tokens that have more chance of being accepted than that threshold.

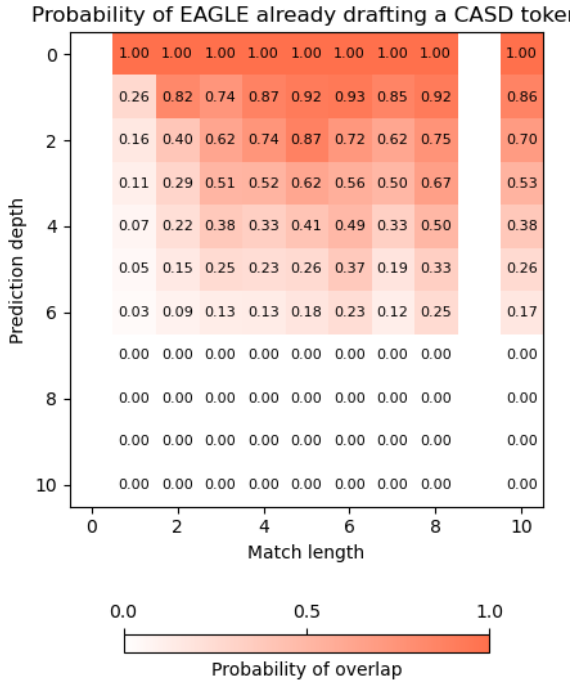
For completeness' sake, we also point out some details that might be confusing about the three plots in Figure 5.6. Firstly, at depth 0 the token is always accepted, because that is actually a token that is already accepted in our case. This is nothing more than an implementation detail inherited from the code of EAGLE-2, and it is kept for consistency. Another detail is that not all values are filled in. This is because the plot is generated with a limited amount of data (400 samples from SQuAD) and each value in the plot that is not supported by sufficient data is left out. Lastly, the plot ends at 10 match length, because this is the longest match that we check for, anything longer is also mapped to 10.

The most important part for an augmenting method such as CASD to work is that the speculations are quite independent. The fact that CASD actually improved upon EAGLE-2 on RAG benchmarks already proves the independence, but Figure 5.6 (b) adds some context as to why. The plot shows the chances that a token drafted by CASD is already drafted by EAGLE-2. The first thing to notice is that the further to the right, the higher the probabilities typically become and this for each row. What this means is that EAGLE-2 already has some notion of copying. The longer the text that was already copied, the more likely that EAGLE-2 will also predict the tokens that CASD predicts. Yet, we also see that the probabilities drop quite fast when the prediction depth grows larger. As EAGLE-2 does not predict beyond 7 depth, the chances are 0 there. And this is where the strength of CASD lies: it copies longer spans literally.

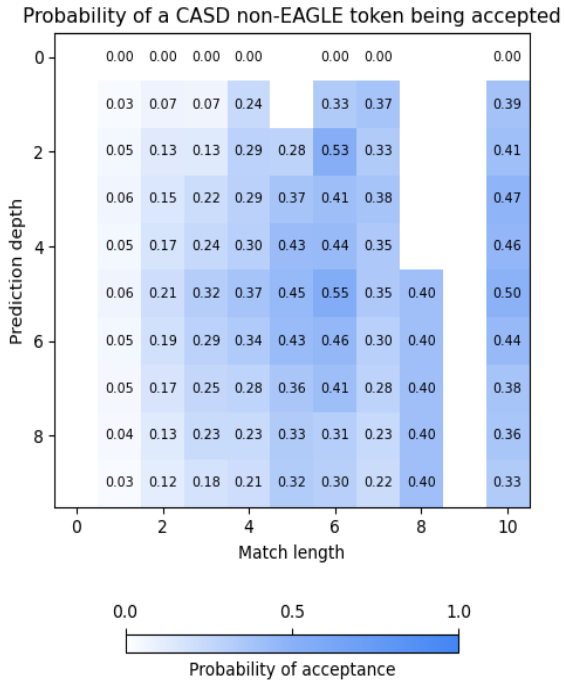
The last plot, Figure 5.6 (c), shows where CASD adds value on top of EAGLE-2. It shows only the tokens that CASD actually added to the draft tree and not the tokens that EAGLE-2 already drafted (i.e. duplicate drafts). There are two key takeaways from this plot. Firstly, it is clear that EAGLE-2 already picked the low-hanging fruit for small prediction depths, because the probabilities are much lower there than for subfigure (a). Secondly, from depth 5 and onwards, the probabilities did not change much, which means that CASD is drafting non-trivial tokens there. In other words, the peak of added value by CASD is around prediction depth 5. For smaller depths, the copying is too trivial and EAGLE-2 already notices these tokens. For larger depths, it becomes harder and harder to draft good tokens for any method, so also for CASD. Do note that these last results of subfigure (b) and (c) are specific for the interaction between CASD and EAGLE-2 and different baseline speculators will lead to different results.



(a) CASD has long predictions of high quality, certainly for long match lengths.



(b) For few steps into the future, CASD's tokens often overlap with EAGLE-2, but tokens deeper into the future barely overlap. So, EAGLE-2 already has some notion of copying, but only for the near future.



(c) The tokens drafted by CASD and not by EAGLE-2 are accepted frequently, certainly for long match lengths. CASD gets most tokens accepted around depth 5, when augmenting EAGLE-2.

Figure 5.6: Analysis of CASD's predictive power and its interaction with EAGLE-2.
Prediction depth: how many steps in the future is the predicted token.
Match length: number of tokens that correspond between the last generated tokens and the match in the context.

Chapter 6

Conclusions and future work

In this thesis, we introduced CASD, a model-agnostic augmentation to any modern speculator for speculative decoding. Multiple benchmarks proved the value of its additional speculations, achieving a new state-of-the-art in RAG scenarios, while overhead for other scenarios remains limited. We introduced one of those benchmarks ourselves; UZGentRAG is made of real-world questions that healthcare professionals asked a RAG chatbot, which we implemented at UZGent. To summarize the findings, we answer the research questions posed in the introduction:

RQ1: Can a new speculative decoding strategy be designed, relying on the repetition of phrases in prompt and answer, resulting in increased inference efficiency of LLMs?

Yes it can. In RAG use cases, our new method CASD successfully improves speculative decoding, by drafting copies from the context. As UZGentRAG and SQuAD [66] are benchmarks of the intended use cases, only these are mentioned below.

- **RQ 1.1** Which energy gains does CASD make? When applied to EAGLE-2 [49], CASD makes energy gains of 17% on UZGentRAG and 7% on SQuAD.
- **RQ 1.2** Which speedups does CASD deliver? The speedups are 1.17 for UZGentRAG and 1.08 for SQuAD.

Exploring this question led to some interesting insights. We found that the current state-of-the-art, EAGLE-2 already has some notion of copying implicitly learned. However, CASD adds much longer high quality drafts from the context with a high probability of being accepted. This is why CASD improves speculative decoding strongly in RAG use cases. Yet, in standard non-RAG settings, the performance went down as the LLM barely copies from the context. In the end, the key takeaway from this question is that fundamentally different speculators can be combined to transcend the speculators separately. On a personal note, my key takeaway is that even very simple ideas like CASD can be successful, even when the current state-of-the-art is so complex and seems unbeatable.

RQ2: Does a real-world benchmark in a niche domain show significantly different results from existing speculative decoding benchmarks?

Yes, the results from our new private UZGentRAG benchmark differ significantly from SQuAD and even more from the non-RAG benchmarks. This shows a gap of niche benchmarks in the current set of public benchmarks.

The answer to this question is important, not only for the academic world, but also for real-world applications. As we show that the performance of speculators can differ significantly for a niche domain, a one-size-fits-all approach with speculators might not be optimal. While CASD is specifically made for RAG, it is domain-agnostic of the content of that RAG system, solving a part of this difficulty.

RQ3: Can a RAG system provide additional value at UZGent?

Partially. The retriever has potential to save healthcare professionals much time that they could spend helping their patients. Yet, the generation step of RAG is currently no added value and it will probably require significant time, effort and money to extract value from such generator.

While the full RAG pipeline will probably not provide value for UZGent in the foreseeable future, working with AI there opened up interesting discussions. We found that UZGent, as probably many hospitals, has untapped potential to use AI in an ethical and safe manner. We believe that AI-based retrievers are the low-hanging fruit for multiple applications at UZGent, while being ethical and safe.

Future work

We see great potential for CASD to augment other speculators in RAG use cases, because of its cheap, high quality, long length speculations. The future might bring even more advanced strategies around the idea of CASD. This is the future work we see for CASD:

- Applying CASD to the baseline's speculated tokens. CASD currently works completely independent of the baseline speculator and merges the speculation trees afterwards. However, the results showed that EAGLE-2 was already reasonably good at copying shorter spans. If CASD could pick up on those shorter spans and expand on them, that could lead to even longer high quality speculations.
- Taking into account speculator confidence. Some speculators (e.g. EAGLE-2) have a built-in confidence metric, which can help decide which tokens to speculate. CASD does not have that, but in Section 5.4, we showed that combining match length with prediction depth provides a good heuristic for confidence. Combining this information, we can keep the top-k speculations according to confidence, and optimizing k for specific hardware could bring significant gains in efficiency [21].
- Optimization with CUDA kernels. The current implementation is not overly optimized. So, it is likely that further optimizations such as specialized CUDA kernels could help CASD achieve even higher speedups.

Also for the use case at UZGent we summarize the future work:

- On the practical side, we suggest that UZGent focuses on evaluation rather than implementation. UZGent is a hospital and its IT department works hard to make the life of their healthcare professionals easier. Developing in-house AI solutions is likely beyond the scope of that task. What seems more realistic is that the IT department builds up a good framework to evaluate all AI solution providers and to see which solution benefits their healthcare professionals the most. The test set for retrieval we made together is certainly a step in the right direction. As the retrieval of current systems is subpar compared to the possibilities of modern AI, we suggest to start there.
- On the technical side, many possibilities remain unexplored. The low-hanging fruit for the data in Zenya, might be to add metadata to the queries. However, fine-tuning the AI models seems to have the largest potential impact, as current embedders get confused by terms that were not part of their training data.

Chapter 7

Societal impact

Until now, this thesis focused on the academical value of our research, but we think it is important to also evaluate the potential societal impact. To do so, this chapter lists four different effects that our research can have, and we evaluate whether this could have a positive or negative impact.

7.1 Reduced energy consumption of LLMs

We are currently living in a climate-crisis, of which climate change is a major component. This climate change is caused by greenhouse gas emissions, where between 1.5% and 4% is attributed to the ICT sector [5]. This fraction is already quite large and AI is expected to drive up emissions even more. For example, datacenter electricity demand is expected to more than double by 2030, with AI as the largest cause [31]. All this to say that reducing the energy consumption by AI and thus LLMs could be a small part of tackling the climate-crisis we live in.

This thesis strives towards the goal of using less energy with LLMs. Our new method, CASD, successfully reduced the energy consumption of an LLM for RAG in multiple experiments. However, we must also take into account the secondary effects of this. Jevons' paradox [3] says that while CASD improves the energy efficiency, it might lead to more energy consumption through increased demand. So, we can only urge implementers of CASD and LLMs in general to use the technologies responsibly, with the energy consumption in mind.

As we mentioned responsible use, it is important to define what that means. First of all, the general LLM guidelines still hold.

- Is the answer of the LLM worth the energy consumption?
- Does this task/application actually need an LLM or do more efficient method exist?
- Is a more energy efficient (L)LM fit for this job?

Next to that, an implementer of an LLM can use these extra technical guidelines.

- When possible, use specialized tools such as vLLM [42] for inference. Such tools apply batching and speculative decoding to generate answers faster with less energy.
- Always use as much batching as possible and then fill the remaining GPU-power with speculative decoding, as this uses the least energy [21]. This is because with batching each decoded token is used, while in speculative decoding only part of the decoded tokens are accepted.

This last point already indicates that speculative decoding is more important when batching is not (or hardly) possible. This is typically the case for local LLMs, as there are not enough users to fill the GPU-power with batching only. A local LLM is an LLM that a company decides to run on their own servers. This has the advantage that the data (question, answer and documents for RAG) never leaves the company and thus it is the preferred method for handling more sensitive data. For this reason, local RAG systems are becoming more popular [16, 35, 61, 78], and CASD just so happens to reduce energy most for these applications.

7.2 Reduced LLM latency

Reducing the latency of an LLM is a double-edged sword. On the one positive side, it can improve the user experience, because they have to wait less long for their answer [6]. Also, this allows for complexer application pipelines that require more LLM steps, such as the recently popular agents [12, 71]. However, that already shows the negative side too: it can stimulate more usage. When an answer is generated faster, more answers can be generated, leading to increased energy consumption. This is already happening with large-scale LLM providers, who use speculative decoding over batching, which is more energy efficient [29, 72]. This type of speculative decoding then powers applications that would otherwise be unpopular due to the long latency, such as some reasoning LLMs, where many more tokens are generated than usual. This is why we advise again to always prefer batching over speculative decoding for optimal energy efficiency.

7.3 Democratization of LLMs

With LLMs becoming larger each year, the primary concern is energy consumption. Yet, in the shadow of this, another issue is often overlooked: the elitism of LLMs.

The state-of-the-art LLMs today are about a 1000 times larger than the state-of-the-art LMs were in 2018 (comparing DeepSeek V3 with BERT) [14, 52]. Needless to say, you also need much more expensive GPUs to run such models. This has even led to jokes about two new classes in society: the GPU-Poor and the GPU-Rich [18]. So, clearly there is a need for democratization of LLMs such that anyone can run their own LLM, not only the GPU-Rich.

Speculative decoding can also help in the democratization of LLMs. Recent advancements in the domain showed that running LLMs on much wider available, cheaper GPUs is also feasible thanks to speculative decoding [72]. While the generation speed is not the same as for the specialized GPUs, it at least allows the GPU-Poor to also run their own LLMs. Our work could further improve the speculations, making it possible to run the LLMs on even cheaper hardware.

7.4 Improved healthcare efficiency

The last point on societal impact is about the use case at UZGent of Chapter 3. There, we tried to improve the efficiency of the healthcare professionals with a RAG application, because we want them to spend less time searching documents in a database and more time helping patients. Fully achieving this goal was of course out of scope for this thesis, but it was a good start of a project that will continue beyond this thesis.

In dialogue with the CTO of UZGent, we found that bringing UZGent up to speed was the most impactful effect of this thesis. Next to the technical implementation of the RAG system, we put significant effort in creating awareness about what AI can and cannot do. UZGent now also feels more ready to step into the AI world, as they have a test set prepared to compare different AI vendors, who promise to deliver the best retrieval system.

Bibliography

[1] Pranjal Aggarwal et al. "Automix: Automatically mixing language models".
In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 131000–131034.

[2] Ryoko AI. "ShareGPT Dataset". In: ().
URL: https://huggingface.co/datasets/RyokoAI/ShareGPT52K.

[3] Blake Alcott. "Jevons' paradox". In: *Ecological economics* 54.1 (2005), pp. 9–21.

[4] Negar Alizadeh et al. "Analyzing the Energy and Accuracy of LLMs in Software Development".
In: *arXiv e-prints* (2024), arXiv–2412.

[5] Seth Ayers et al. *Measuring the Emissions and Energy Footprint of the ICT Sector : Implications for Climate Action ((English)*. Mar. 2024.
URL: https://documents.worldbank.org/en/publication/documents-reports/documentdetail/099121223165540890/p17859702a98880540a4b70d57876048abb.

[6] Himanshu Bamoria.
Solving Latency Challenges in LLM Deployment for Faster, Smarter Responses. Sept. 2024.
URL: https://medium.com/athina-ai/solving-latency-challenges-in-llm-deployment-for-faster-smarter-responses-64ff301e40d3.

[7] Emily M Bender et al. "On the dangers of stochastic parrots: Can language models be too big?"
In: *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*. 2021, pp. 610–623.

[8] Tom Brown et al. "Language models are few-shot learners".
In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[9] Banghao Chen et al. "Unleashing the potential of prompt engineering in large language models: a comprehensive review". In: *arXiv preprint arXiv:2310.14735* (2023).

[10] Jianlv Chen et al. "Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation". In: *arXiv preprint arXiv:2402.03216* (2024).

[11] Mark Chen et al. "Evaluating large language models trained on code".
In: *arXiv preprint arXiv:2107.03374* (2021).

[12] Sarah Chudleigh. *Complete Guide to LLM Agents (2025)*. Nov. 2024.
URL: https://botpress.com/blog/llm-agents.

[13] Karl Cobbe et al. "Training Verifiers to Solve Math Word Problems".
In: *arXiv preprint arXiv:2110.14168* (2021).

[14] Jacob Devlin et al.
"Bert: Pre-training of deep bidirectional transformers for language understanding".
In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 2019, pp. 4171–4186.

[15] Docker, Inc. *Docker*. Accessed: 2025-05-18. 2024. URL: https://www.docker.com.

[16] E-SPIN. *The Rise of Local LLMs: Why Everyone Is Building Their Own AI Models*. Feb. 2025. URL: https://www.e-spincorp.com/rise-of-local-llms-ai-models.

[17] *Eiffel Tower*. URL: https://en.wikipedia.org/wiki/Eiffel_Tower (visited on 04/30/2025).

[18] Envisioning. *GPU-Poor*. 2022. URL: https://www.envisioning.io/vocab/gpu-poor.

[19] Shahul Es et al. "Ragas: Automated evaluation of retrieval augmented generation". In: *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*. 2024, pp. 150–158.

[20] Wenqi Fan et al. "A survey on rag meeting llms: Towards retrieval-augmented large language models". In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2024, pp. 6491–6501.

[21] Jared Fernandez et al. "Energy Considerations of Large Language Model Inference and Efficiency Optimizations". In: *arXiv preprint arXiv:2504.17674* (2025).

[22] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. "SPLADE: Sparse lexical and expansion model for first stage ranking". In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2021, pp. 2288–2292.

[23] Thibault Formal et al. "SPLADE v2: Sparse lexical and expansion model for information retrieval". In: *arXiv preprint arXiv:2109.10086* (2021).

[24] Elias Frantar et al. "Gptq: Accurate post-training quantization for generative pre-trained transformers". In: *arXiv preprint arXiv:2210.17323* (2022).

[25] Luyu Gao et al. "Precise zero-shot dense retrieval without relevance labels". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2023, pp. 1762–1777.

[26] Amir Gholami et al. "AI and memory wall". In: *IEEE Micro* (2024).

[27] Aaron Grattafiori et al. "The llama 3 herd of models". In: *arXiv preprint arXiv:2407.21783* (2024).

[28] Zeyu Han et al. "Parameter-efficient fine-tuning for large models: A comprehensive survey". In: *arXiv preprint arXiv:2403.14608* (2024).

[29] Demis Hassabis, James Manyika, and Jeff Dean. *2024: A year of extraordinary progress and advancement in AI*. Jan. 2025. URL: https://blog.google/technology/ai/2024-ai-extraordinary-progress-advancement/.

[30] Zhenyu He et al. "Rest: Retrieval-based speculative decoding". In: *arXiv preprint arXiv:2311.08252* (2023).

[31] IEA. *AI is set to drive surging electricity demand from data centres while offering the potential to transform how the energy sector works*. Apr. 2025. URL: https://www.iea.org/news/ai-is-set-to-drive-surging-electricity-demand-from-data-centres-while-offering-the-potential-to-transform-how-the-energy-sector-works.

[32] Gautier Izacard et al. "Unsupervised dense information retrieval with contrastive learning". In: *arXiv preprint arXiv:2112.09118* (2021).

[33] Huiqiang Jiang et al.
"Llmlingua: Compressing prompts for accelerated inference of large language models".
In: *arXiv preprint arXiv:2310.05736* (2023).

[34] Huiqiang Jiang et al. "Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression". In: *arXiv preprint arXiv:2310.06839* (2023).

[35] K2view. *GenAI adoption 2024: Challenges with enterprise data.*
URL: `https://www.k2view.com/genai-adoption-survey/` (visited on 04/30/2025).

[36] Omar Khattab et al. "Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp". In: *arXiv preprint arXiv:2212.14024* (2022).

[37] Gangwoo Kim et al. "Tree of clarifications: Answering ambiguous questions with retrieval-augmented large language models".
In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 2023, pp. 996–1009.

[38] Sehoon Kim et al. "Full stack optimization of transformer inference: a survey".
In: *arXiv preprint arXiv:2302.14017* (2023).

[39] Sehoon Kim et al. "Full stack optimization of transformer inference: a survey".
In: *arXiv preprint arXiv:2302.14017* (2023).

[40] Sehoon Kim et al. "Speculative decoding with big little decoder".
In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 39236–39256.

[41] Tom Kwiatkowski et al. "Natural questions: a benchmark for question answering research".
In: *Transactions of the Association for Computational Linguistics* 7 (2019), pp. 453–466.

[42] Woosuk Kwon et al.
"Efficient Memory Management for Large Language Model Serving with PagedAttention".
In: *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*. 2023.

[43] langchain. *PyPDFDirectoryLoader*.
URL: `https://python.langchain.com/api_reference/community/document_loaders/langchain_community.document_loaders.pdf.PyPDFDirectoryLoader.html`.

[44] langchain. *RecursiveCharacterTextSplitter*.
URL: `https://python.langchain.com/api_reference/text_splitters/character/langchain_text_splitters.character.RecursiveCharacterTextSplitter.html`.

[45] Carlos Lassance et al. "SPLADE-v3: New baselines for SPLADE".
In: *arXiv preprint arXiv:2403.06789* (2024).

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

[47] Mosh Levy, Alon Jacoby, and Yoav Goldberg. "Same task, more tokens: the impact of input length on the reasoning performance of large language models".
In: *arXiv preprint arXiv:2402.14848* (2024).

[48] Yucheng Li et al. "Compressing context to enhance inference efficiency of large language models".
In: *arXiv preprint arXiv:2310.06201* (2023).

[49] Yuhui Li et al. "Eagle-2: Faster inference of language models with dynamic draft trees".
In: *arXiv preprint arXiv:2406.16858* (2024).

[50] Yuzhuo Li et al. "The unseen AI disruptions for power grids: LLM-induced transients".
In: *arXiv preprint arXiv:2409.11416* (2024).

[51] Barys Liskavets et al.
"Prompt compression with context-aware sentence encoding for fast and improved llm inference".
In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 23. 2025,
pp. 24595–24604.

[52] Aixin Liu et al. "Deepseek-v3 technical report". In: *arXiv preprint arXiv:2412.19437* (2024).

[53] Nelson F Liu et al. "Lost in the middle: How language models use long contexts".
In: *Transactions of the Association for Computational Linguistics* 12 (2024), pp. 157–173.

[54] Richard Luscombe. *Three Mile Island nuclear reactor to restart to power Microsoft AI operations*.
Sept. 2024. URL: https://www.theguardian.com/environment/2024/sep/20/three-mile-island-nuclear-plant-reopen-microsoft.

[55] Niklas Muennighoff et al. "MTEB: Massive text embedding benchmark".
In: *arXiv preprint arXiv:2210.07316* (2022).

[56] Ramesh Nallapati et al.
"Abstractive text summarization using sequence-to-sequence rnns and beyond".
In: *arXiv preprint arXiv:1602.06023* (2016).

[57] world nuclear news. *Facebook owner Meta seeks up to 4 GW nuclear capacity*. Dec. 2024.
URL: https://www.world-nuclear-news.org/articles/facebook-owner-meta-seeks-up-to-4gw-nuclear-capacity.

[58] NGINX, Inc. *NGINX*. Accessed: 2025-05-18. 2024. URL: https://nginx.org.

[59] Jianmo Ni et al. "Large dual encoders are generalizable retrievers".
In: *arXiv preprint arXiv:2112.07899* (2021).

[60] Zhuoshi Pan et al.
"Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression".
In: *arXiv preprint arXiv:2403.12968* (2024).

[61] Pieces. *Local large language models (LLMs) and their growing traction*. Mar. 2025. URL: https://pieces.app/blog/local-large-language-models-lllms-and-copilot-integrations.

[62] Soham Poddar et al. "Towards Sustainable NLP: Insights from Benchmarking Inference Energy in Large Language Models". In: *arXiv preprint arXiv:2502.05610* (2025).

[63] PostgreSQL Global Development Group. *PostgreSQL*. Accessed: 2025-05-18. 2024.
URL: https://www.postgresql.org.

[64] Zongyue Qin et al.
"Optimized Multi-Token Joint Decoding with Auxiliary Model for LLM Inference".
In: *arXiv preprint arXiv:2407.09722* (2024).

[65] Zackary Rackauckas. "Rag-fusion: a new take on retrieval-augmented generation".
In: *arXiv preprint arXiv:2402.03367* (2024).

[66] Pranav Rajpurkar et al. "Squad: 100,000+ questions for machine comprehension of text".
In: *arXiv preprint arXiv:1606.05250* (2016).

[67] François Remy, Kris Demuynck, and Thomas Demeester. "BioLORD-2023: semantic textual representations fusing large language models and clinical knowledge graph insights".
In: *Journal of the American Medical Informatics Association* 31.9 (2024), pp. 1844–1855.

[68] Stephen Robertson, Hugo Zaragoza, et al.
"The probabilistic relevance framework: BM25 and beyond".
In: *Foundations and Trends® in Information Retrieval* 3.4 (2009), pp. 333–389.

[69] João da Silva. *AI's Energy Appetite Sparks Environmental Concerns*. Oct. 2024. URL: https://www.bbc.com/news/articles/c748gn94k95o.

[70] Emma Strubell, Ananya Ganesh, and Andrew McCallum. "Energy and policy considerations for modern deep learning research". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 09. 2020, pp. 13693–13696.

[71] Tyler Suard. *A Quick Review of The Most Popular AI Agent Frameworks (June 2024)*. June 2024. URL: https://medium.com/@ceo_44783/a-quick-review-of-the-most-popular-ai-agent-frameworks-june-2024-ce53c0ef809a.

[72] Ruslan Svirschevski et al. "Specexec: Massively parallel speculative decoding for interactive llm inference on consumer devices". In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 16342–16368.

[73] Rohan Taori et al. *Stanford Alpaca: An Instruction-following LLaMA model*. https://github.com/tatsu-lab/stanford_alpaca. 2023.

[74] Harsh Trivedi et al. "Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions". In: *arXiv preprint arXiv:2212.10509* (2022).

[75] u/fictioninquire. *Best open source embedding models for EU languages*. Reddit thread. 2024. URL: https://www.reddit.com/r/LocalLLaMA/comments/1chqkph/best_open_source_embedding_models_for_eu_languages/.

[76] UGent. *VERANTWOORD GEBRUIK GENERATIEVE AI IN DE MASTERPROEF FACULTEIT INGENIEURSWETENSCHAPPEN EN ARCHITECTUUR*. URL: https://www.ugent.be/ea/nl/faculteit/studentenadministratie/masterproef/ai_2425.pdf (visited on 05/03/2025).

[77] *Vancomycin*. URL: https://en.wikipedia.org/wiki/Vancomycin (visited on 04/30/2025).

[78] Menlo Ventures. *2024: The State of Generative AI in the Enterprise*. URL: https://menlovc.com/2024-the-state-of-generative-ai-in-the-enterprise (visited on 04/30/2025).

[79] Weaviate. *Weaviate*. Accessed: 2025-05-18. 2024. URL: https://weaviate.io.

[80] Jules White et al. "A prompt pattern catalog to enhance prompt engineering with chatgpt". In: *arXiv preprint arXiv:2302.11382* (2023).

[81] Guangxuan Xiao et al. "Smoothquant: Accurate and efficient post-training quantization for large language models". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 38087–38099.

[82] Fangyuan Xu, Weijia Shi, and Eunsol Choi. "Recomp: Improving retrieval-augmented lms with compression and selective augmentation". In: *arXiv preprint arXiv:2310.04408* (2023).

[83] Lingling Xu et al. "Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment". In: *arXiv preprint arXiv:2312.12148* (2023).

[84] Nan Yang et al. "Inference with reference: Lossless acceleration of large language models". In: *arXiv preprint arXiv:2304.04487* (2023).

[85] Josh You. *How much energy does ChatGPT use?* Feb. 2025. URL: https://epoch.ai/gradient-updates/how-much-energy-does-chatgpt-use.

[86] Wayne Xin Zhao et al. "A survey of large language models". In: *arXiv preprint arXiv:2303.18223* 1.2 (2023).

[87] Huaixiu Steven Zheng et al.
"Take a step back: Evoking reasoning via abstraction in large language models".
In: *arXiv preprint arXiv:2310.06117* (2023).

[88] Lianmin Zheng et al. "Judging llm-as-a-judge with mt-bench and chatbot arena".
In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 46595–46623.

[89] Yongchao Zhou et al. "Distillspec: Improving speculative decoding via knowledge distillation".
In: *arXiv preprint arXiv:2310.08461* (2023).

[90] Shengyao Zhuang et al. "Open-source large language models are strong zero-shot query likelihood models for document ranking". In: *arXiv preprint arXiv:2310.13243* (2023).

Appendix A

Deliverables to UZGent

In this thesis we worked together with UZGent and we made sure both parties benefited from the collaboration. In this section, we list the documentation, code and data that we deliver to UZGent.

A.1 Documentation

The RAG application ran on the servers of UZGent. However, after the project the server was reclaimed for other purposes. This means the setup needs to happen again if UZGent decides to go further with the application and to help with that, we logged all commands that are necessary to set up the server again. After following the instructions, the requirements are installed, https is set up, the RAG API is live and the website is hosted on that server. This all works with the building blocks explained in Figure 3.9. The only thing that still needs to happen is to contact the networking person to open up the right ports of that server to the internal network of UZGent.

A.2 Code

The setup above requires access to the code of two GitHub repositories. One repository is for the web application. The other repository contains the code to run the RAG API, which also comes with a short demo notebook that shows how the different components work. We grant access to the ICT department of UZGent on demand.

A.3 Data

We held a test session to gather typical questions from healthcare professionals and the corresponding document in Zenya. This data shows how the system is currently used and allows for further "in vitro" experiments (as the experts of UZGent like to call it). By that, we mean that the usage of Zenya is not too dynamic and thus the test set can be used for experiments in the coming year(s), without further input required.

The most straightforward use of this data is to evaluate retrievers, in order to find the optimal way of presenting the Zenya data to healthcare professionals. To do this properly, we split the data in a validate set and a test set. The provider of the retriever can use the validate set to tune its retriever specifically for the use case. Then, the UZGent can test the retriever with their withheld test set. Both the validate and test set were delivered in the form of CSVs, as this was most practical for UZGent.

Appendix B

Example prompt of UZGentRAG

We introduced a new private benchmark, UZGentRAG. In order to give some idea what a typical UZGentRAG prompt might look like, one example is given below. It always starts with a role description. Then the documents are given, split by "—". Finally, the question of the user is added.

```
[begin context]
Jij bent een dokter. Jij kent alle medische procedures van het UZGent.
[einde context]
[begin documenten]
obij verhoogde nuchtere of at random glycemie: bijkomend bepalen HbA1c

4.5 Diagnostiek cont'd
Beeldvorming
-Hypofyse:
oI.g.v. anders onverklaarde hoofdpijn, gezichtsvelduitval (met name uitval lateraal),
hormonaal bilan suggestief voor hypofysitis / hypopituïtarisme : dringende MRI
↪  hypofyse
(hypofysitis? RIP?), in overleg met MRI radioloog van wacht.
oZo normale MRI en sterk suggestief klinisch beeld, nieuwe MRI na 1 maand.

-Schildklier:
oBij thyreotoxicosis, pijnlijke schildklieropzetting - te plannen via polikliniek
↪  Endocrinologie
(urgente poli-afspraak via D22137, na overleg met endocrinoloog van wacht).
5. Beleid en verloop
5.1 Behandeling hypothyreose
Graad

Graad 1
(TSH < 10 mU/L):

Graad 2-3
(TSH > 10 mU/L ): Graad 4
```

Behandeling

-Controle na 6 weken;

-Labo:
oHbA1c
oBloedgroep en Indirecte Coombs
oEPD > Plaatsen van Orders > Subset Klinische biologie Gyn/Materniteit > Prenataal 1ste consult
o(NB: enkel bepalen wat nog niet bekend is).
oZo combinatietest gewenst: EPD > Plaatsen van Orders > Subset Klinische biologie Gyn/Materniteit > Combinatietest 1ste trimester
oZo bekende schildklierpathologie: (zie richtlijn "Schildklierlijden en zwangerschap")
-Bepaal ook FT4, FT3.
-Bij gekende Morbus Graves: bepaal TSI-antilichamen (belang voor foetale en neonatale opvolging).
-Bij hypothyreoïdie en bekende diagnose Morbus Graves: geen verdere bepalingen van antistoffen nodig.
-Bij onduidelijke diagnose: bepaal TPO- en TSI-antilichamen.
-Bij gekende hypothyreoïdie: verhoog dosis levothyroxine met 25 tot 50\%

- Inname levothyroxine en zwangerschapsvitamines/ijzersuppletie met minstens 2 uur tussen (ivm verminderde absorptie bij gebruik met ijzerbevattende supplementen)
Nieuwe diagnose hypothyreoïdie
- Zie flowchart nieuwe diagnose hypothyreoïdie
- Indien substitutie aangewezen is, start een lage dosis levothyroxine bvb 50mcg per dag.
Gekende hypothyreoïdie
- Anamnese bij 1ste consultatie: Hoe lang bestaand? Ontstaan? Oorzaak? Uitsluiten behandelde Morbus Graves. Behandeling?
- Bij een positieve zwangerschapstest dosis met 25\% verhogen (dit kan vb. door 2 van de 7 dagen van de week een dubbele dosis in te nemen).
- Controle bij 1e bloedafname in de zwangerschap:
- Bepaling van TSH, vrij-T4 en vrij-T3
- Bepaling van TSI i.g.v. iatrogene hypothyreoïdie bij ziekte van Graves of bij
[einde documenten]
[begin vraag]
Wat moet ik doen bij een graad 1 immuun-gerelateerde hypothyroïdie?
[einde vraag]
[begin antwoord]

Appendix C

Use of Generative AI

The responsible use of AI was allowed for this thesis. Responsible use is defined as being aware of the limitations, risks, pitfalls of using generative AI, such as (un)reliability, bias, copyright, data security, privacy, social inequality, ethical and sustainable impact, etc. [76].

For this thesis, only boilerplate code was generated with generative AI. This improved coding efficiency, while being low risk. The boilerplate code is quite predictable and easily checked for errors (which I always did). For data security, no sensitive data was given to the generative AI tools. The sustainable impact was limited as much as possible, by keeping the generated answers short, which limits the energy consumption [62]. We did this with prompt additions such as "Answer shortly.", "Give code only." and "Only reply with the relevant part of the code."