Large Language Models related Hardware Optimizations

RAHUL VIGNESWARAN K*, CS23MTECH02002, Indian Institute of Technology Hyderabad, India
NIKHIL KUMAR PATEL*, CS23MTECH11013, Indian Institute of Technology Hyderabad, India

Large Language Models (LLMs) have revolutionized automation across various sectors with their human-like efficiency. Yet, their real-time application deployment is constrained by substantial computational demands due to their large parameter sets. This paper reviews hardware and software optimization strategies that enhance the operational efficiency of LLMs, focusing specifically on transformer architectures. We explore advanced hardware accelerators and innovative software techniques such as FPGA, and in-memory processing. Our analysis assesses their impact on reducing latency and energy consumption. The findings emphasize the benefits of integrating these methods, suggesting a unified approach could lead to more robust and efficient LLM implementations. This study highlights significant advancements and sets the stage for future innovations in LLM optimization.

Additional Key Words and Phrases: Large Language Models, Hardware accelerators, FPGAs, GPU, Survey, Transformer, Energy efficiency

ACM Reference Format:
Rahul Vigneswaran K and Nikhil Kumar Patel. 2024. Large Language Models related Hardware Optimizations. 1, 1 (June 2024), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Large Language Models (LLMs) such as the Generative Pre-trained Transformer (GPT) series have significantly transformed artificial intelligence, demonstrating exceptional capabilities in generating human-like text across various applications. The exponential growth in model size—from GPT-3's 175 billion parameters to GPT-4's even greater scalability—has correspondingly escalated computational demands. This escalation presents substantial challenges in processing speed, energy consumption, and operational efficiency, all of which are critical for the deployment and accessibility of LLM technologies.

LLMs have been profoundly integrated into sectors such as healthcare [5], finance [9], and customer service, automating responses, synthesizing information, and generating creative content. However, their deployment in real-time applications is constrained by high latency and significant power requirements. For instance, processing large inputs or generating extensive content using GPT models necessitates robust hardware and substantial energy resources.

Recent research has focused on optimizing hardware specifically for LLMs by developing specialized accelerators and techniques that enhance computational hardware and optimize the unique characteristics of LLMs to improve performance metrics.

*Both authors contributed equally to this term paper. Done as part of the coursework for Hardware Architecture for Deep Learning (CS6490).

Authors' Contact Information: Rahul Vigneswaran K, cs23mtech02002@iith.ac.in, CS23MTECH02002, Indian Institute of Technology Hyderabad, India; Nikhil Kumar Patel, cs23mtech11013@iith.ac.in, CS23MTECH11013, Indian Institute of Technology Hyderabad, India.

Transformers. Within the domain of Large Language Models (LLMs), transformers play a crucial role in processing and generating text by identifying complex patterns in extensive datasets. These models efficiently process sequences of data—primarily text—through a series of self-attention mechanisms and position-wise feed-forward networks, thereby enabling deep semantic understanding. Typically, the input to these models consists of tokenized text, which is then converted into numerical data that the model can interpret. The output from transformers is usually a sequence of tokens that correspond to various natural language processing tasks, such as translation, summarization, or content generation. Detailed discussions on the transformer architecture are beyond the scope of this paper; for an in-depth exploration, refer to [8].

Despite existing accelerators designed for Convolutional Neural Networks (CNNs) which optimize typical CNN operations, LLMs present unique challenges for hardware optimization. LLM-specific accelerators must efficiently manage memory and perform high-throughput matrix multiplications and additions, crucial for transformer architectures.

This paper explores various methodologies and innovations in optimizing LLMs. We categorize the analyzed methods into two segments:

- **Hardware Level Techniques**: These involve innovations in physical computing infrastructure and architecture designs to enhance LLM performance, including specialized accelerator designs and reconfigurable FPGA systems. These improvements significantly enhance speed, energy efficiency, and reduce computational costs.
- **Software Level Techniques**: This category includes algorithmic enhancements and novel dataflows tailored to LLMs, aiming to reduce the computational burden and enhance speed and energy efficiency. In theory, these techniques can sit on top of the hardware techniques for a multiplied effect.

While training LLMs presents significant challenges, this work primarily *focuses on the inference phase*, exploring optimization techniques that enhance performance during this stage. We also study the *energy efficiency* and *speed-up* offered by each method, acknowledging that each was tested in vastly different setups. We endeavor to draw a fair comparison. Through this analysis, we aim to provide a comprehensive overview of current advancements and identify potential pathways for future research. This exploration is intended to contribute to the development of more sustainable and efficient LLM technologies that can be broadly implemented across diverse computing environments.

2 SOFTWARE LEVEL TECHNIQUES

2.1 Flash

A significant challenge with large language models (LLMs) is their vast size, often several orders of magnitude larger than typical convolutional neural networks (CNNs). This scale makes it impractical to store all the model's weights in dynamic random-access memory (DRAM). The approach detailed in [1] addresses this issue by proposing an innovative method for performing inference with LLMs under DRAM constraints. Specifically, it involves storing the entirety of the LLM in flash memory and dynamically loading only necessary subsets of weights into DRAM as needed for inference tasks. This method not only alleviates the memory bottleneck but also facilitates the practical deployment of LLMs in resource-constrained environments.

Implementing this strategy, however, comes with its own set of challenges, largely due to the inherent limitations of flash memory. Although flash memory provides significantly higher storage capacity, it suffers from lower bandwidth compared to DRAM, which can severely bottleneck operations. Furthermore, the research highlighted in the paper reveals that flash memory exhibits higher throughput when reading data in larger blocks as opposed to smaller ones. Based on these insights, the paper proposes two novel techniques to optimize performance:

Windowing. This method strategically manages memory by selectively loading and caching subsets of model parameters that are relevant to a specific "window" of input tokens. In the context of LLM inference, particularly during sequence generation or processing, not all components of the model are required simultaneously. The windowing technique leverages this by focusing on segments of the model that are most pertinent to the current and upcoming input tokens.

During the inference process, elements such as neurons involved in attention mechanisms are consistently kept in DRAM due to their continuous relevance. Conversely, components like the Feed-Forward Network (FFN) sections, which are characterized by sparsity in LLMs, are handled more dynamically. Specifically, only those portions of the FFN that are dense (i.e., non-sparse) are dynamically loaded into DRAM. This selective loading is guided by a sophisticated low-rank predictor, which evaluates incoming tokens to predict which elements will not be zeroed out by subsequent ReLU activations. Only these predicted non-zero elements are loaded into the memory.

As the inference progresses and the input sequence advances, the window of tokens shifts forward. This continuous update ensures that only relevant data is retained in memory, enhancing efficiency by minimizing unnecessary data transfers from slower storage solutions like flash memory. The windowing technique thus not only optimizes computational speed and power efficiency but also plays a crucial role in overcoming the bandwidth limitations of flash memory, enabling more effective deployment of LLMs in resource-constrained settings.

Row-Column Bundling. This method involves bundling rows and columns of weights together, which significantly improves the efficiency of read operations from flash memory to DRAM. The underlying principle of this technique is to capitalize on larger data reads, which empirically have been shown to increase throughput per read operation.

The research detailed in [1] identifies that the usage of the ith column and the ith row of the projection module typically coincides with the activation of the ith neuron. Leveraging this insight, a new storage structure is devised to store the rows and columns corresponding to the same neuron in contiguous segments of memory. This configuration allows these elements to be accessed simultaneously, effectively reducing the number of read operations required and increasing the size of each data chunk retrieved.

By implementing this bundling strategy, the technique not only minimizes the frequency of memory accesses—which is crucial given the lower bandwidth of flash memory—but also maximizes data throughput by taking advantage of larger block reads. This dual benefit streamlines the computational process, reducing latency and enhancing overall efficiency during the inference phase of large language models. These innovations are crucial for mitigating the bandwidth issues of flash memory and enhancing the feasibility of deploying large models in environments with limited DRAM.

2.2 UltraFastBERT

The feedforward layers hold the majority of the parameters in large language models (LLMs), yet not all neurons within these layers need to be active during the computation of the output at inference time for every input.

The study presented in [2] introduces a novel variant of BERT, termed UltraFastBERT, which utilizes only 0.03% of its neurons during inference while achieving performance comparable to the original model. This significant reduction in active neurons is made possible by replacing traditional Feed Forward Networks (FFs) with Fast Feed Forward Networks (FFFs) [3].

Fast Feed Forward Networks (FFFs). Introduced by [3], FFFs represent a groundbreaking architecture designed to expedite the inference process while maintaining high predictive performance. FFFs organize neurons into a hierarchical binary tree structure, enabling $O(\log n)$ time access to neurons during inference, compared to the $O(n)$ time required by traditional FFs. Each node in this binary tree determines whether to pass the computation downstream to its left or right child based on the input received. This method efficiently segments the input space into discrete regions, each managed by a specific subset of neurons, thereby minimizing the number of neurons activated for any given input.

In their implementation, [2] replaces the dense layers in a transformer with FFFs, capitalizing on the accelerated inference speeds afforded by this architecture. Moreover, the allowance for multiple binary trees to coexist in parallel and the aggregation of their outputs at the end of the process further enhances the representation capacity of the model.

2.3 LLMA

Autoregressive decoding. This is a common method in language modeling where the prediction of each new word or token in a sequence is based on the previously generated words or tokens. This process is inherently sequential; the model generates one token at a time, with each new token dependent on the preceding outputs. Predicting each new token involves calculating the probability of all possible tokens given the previous context and selecting the token with the highest probability.

This sequential process is computationally intensive during inference, as it cannot fully utilize the parallel computational capabilities of GPUs. [10] exploits this inefficiency in scenarios where the large language model (LLM) is merely quoting or using sentences from the provided source material. Despite the word-for-word reproduction, traditional autoregressive decoding still performs the entire compute-intensive task. By exploiting this characteristic, LLMA achieves a twofold increase in speed over traditional methods without compromising the quality of the generated text.

LLMA begins by identifying text spans in reference documents that align with the expected output based on the input prompt. This strategy leverages the observation that in many practical scenarios, such as document retrieval or multi-turn conversations, significant portions of the output can be directly derived from available reference materials. Once a matching text span is identified, LLMA selectively copies these tokens directly into the output buffer of the decoder. This approach circumvents the standard token-by-token generation process for portions of text that are present in the reference documents, substantially accelerating the overall generation process.

After incorporating the text spans into the output, LLMA conducts a parallel verification of the appropriateness of each copied token, instead of processing them sequentially. This verification occurs within a single decoding step, drastically speeding up the inference process. This approach not only ensures that the output remains contextually appropriate and coherent but also optimizes the use of computational resources, making it more efficient than traditional sequential decoding. If the text from the reference materials is found to be unsuitable, LLMA automatically reverts to the default autoregressive decoding. The versatility of LLMA, coupled with its ability to integrate into existing architectures without requiring additional models or extensive modifications, makes it highly adaptable for various large language model applications.

3 HARDWARE LEVEL TECHNIQUES

3.1 X-Former

The study presented in [6] categorizes the matrix-vector multiplication (MVM) operations essential to transformers into two distinct types:

- **MVMStatic**: These operations involve matrix-vector multiplications where at least one operand, typically the matrix, remains constant across all inputs during the computation. This characteristic is prevalent in many traditional neural network operations, where the network's weights (the matrices) do not change during the forward pass of an input batch.
- **MVMDynamic**: These operations involve matrix-vector multiplications where both the matrix and the vector operands dynamically change with each new input. In the context of transformers, this dynamic

nature is particularly relevant because the matrices used in the self-attention mechanism (namely, Query and Key) are derived directly from the input sequences, which vary from one input to another.

Unlike traditional deep learning architectures, which predominantly rely on MVMStatic operations (over 95%), transformers have a substantial proportion of their operations as MVMDynamic (approximately 35%). This significant presence of MVMDynamic operations presents unique challenges in traditional crossbar-based architectures. Each new input requires the crossbar to be reconfigured for the self-attention layers due to the dynamic nature of the weights, exacerbating the inherent issues associated with non-volatile memory (NVM) devices. These issues include increased overall latency, higher energy consumption, and limited endurance, which can significantly affect the efficiency and longevity of the hardware used.

The study in [6] proposes a dual-component solution to address the distinct computational challenges in transformers. This solution consists of a Projection Engine and an Attention Engine, each optimized for different types of matrix-vector multiplications (MVM).

Projection Engine. This component primarily handles the MVMStatic operations, which involve static matrix-vector multiplications typical in layers where the weights (parameters) do not change with each input. Optimized for tasks that can leverage in-memory computing, the Projection Engine reduces data movement and increases processing speed by storing the weights of the Transformer model in non-volatile memory (NVM) arrays. This configuration minimizes the need for frequent data transfers between memory and processing units, significantly reducing latency and power consumption.

Attention Engine. This engine is tailored specifically to manage MVMDynamic operations, which are pivotal in the self-attention mechanism of the Transformer, where both operands (queries and keys) are dependent on the input data and change dynamically. It efficiently handles the dynamically altering matrices such as the Query, Key, and Value matrices essential in self-attention calculations. The engine incorporates CMOS processing tiles that are specially optimized for high endurance, low write latency, and low write energy, enabling rapid adaptation to data changes. This optimization ensures that the engine can perform intensive computational tasks efficiently while maintaining robust performance under continuous operation.

To address the inefficiency of traditional dataflow where the attention engine might sit idle while the projection engine is running, [6] introduces a novel *sequence blocking* dataflow. This approach efficiently manages the operations between the Projection Engine and the Attention Engine by dividing the entire input sequence into smaller sequential blocks. While one block of data is processed for static weight multiplications in the Projection Engine, the Attention Engine can begin processing dynamic operations on a previously prepared block. This coordinated approach not only maximizes hardware efficiency but also minimizes the latency and energy consumption typically associated with large-scale Transformer computations.

3.2 FlexRun

Large language models (LLMs) continue to make significant impacts while undergoing constant evolution. Though the core architecture largely remains transformer-based, frequent architectural modifications are made to meet new demands or to enhance performance. This dynamic evolution can make existing LLM accelerator designs appear rigid and inflexible. [7] addresses these challenges by developing FlexRun, a flexible platform designed to accelerate a diverse range of architectures. Departing from the traditional approach of creating highly specialized accelerators, FlexRun supports a broad spectrum of operations, dimensions, and parameter configurations, providing a comprehensive end-to-end solution that enhances both flexibility and efficiency of FPGA accelerators.

FlexRun's innovative approach is structured around three interconnected modules: FlexRun:Architecture, FlexRun:Algorithm, and FlexRun:Automation, which collaboratively work to fulfill the system's objectives.

FlexRun:Architecture. This module offers a versatile and reconfigurable template, enabling rapid adaptation to various architectural needs. The design is inherently modular, requiring minimal adjustments to suit a wide array of configurations. This flexibility addresses the limitations of traditional FPGA architectures, which often struggle to keep pace with the complex and varied requirements of modern NLP tasks.

FlexRun:Algorithm. This component facilitates design space exploration, crucial for determining the optimal arrangement of compute units within the FPGA based on specific configurations. It evaluates different configurations to optimize performance for particular tasks, reflecting the diverse operational, dimensional, and configurational needs of contemporary applications. By doing so, it not only ensures that the architecture fits the desired template but also maximizes the efficiency of FPGA resource utilization.

FlexRun:Automation. Integrating the architecture and algorithm modules, this component accelerates the implementation of changes on actual FPGA systems. It automates the configuration process outlined by FlexRun:Algorithm, which is essential for applications requiring quick and reliable adaptation to new architectures. Additionally, it eliminates the need for manual intervention or technical expertise by automatically updating instruction decoders to manage new configurations, ensuring accurate interpretation of model-specific instructions by the FPGA.

Together, these modules make FlexRun an adaptive and efficient solution for deploying FPGA accelerators in environments demanding high flexibility and rapid update capabilities.

4 COMPARATIVE ANALYSIS OF OPTIMIZATION METHODS

This section aims to compare and contrast each of the methods discussed so far and their interconnected contributions toward enhancing model performance.

The **Flash [1]** method addresses LLMs' substantial memory demands by innovatively using flash memory to store model weights, effectively mitigating slow access speeds and enhancing memory efficiency, critical for environments where conventional high-speed memory solutions are impractical. Concurrently, **UltraFastBERT [2]** introduces a sparsity-driven modification to feedforward networks within transformers, significantly reducing the computational load during inference without compromising accuracy, demonstrating how architectural innovations can streamline LLM operations. In a complementary manner, **LLMA [10]** optimizes the decoding process by leveraging source material to expedite traditional greedy decoding, thus speeding up response generation while maintaining output quality, showcasing the synergy between heuristic strategies and conventional decoding techniques. Meanwhile, **X-Former [6]** merges the strengths of Non-Volatile Memory (NVM) and CMOS technologies to balance memory efficiency with processing speed, addressing limitations inherent to each technology when used independently. Lastly, **FlexRun [7]** offers a holistic framework for the flexible and efficient adaptation, testing, and deployment of LLMs on FPGAs, illustrating the importance of adaptable infrastructure.

Across these methodologies, we observed common recurring bottlenecks such as memory technology limitations, and latency are repeatedly addressed. These challenges often arise from the inherent properties of LLMs themselves, including the large number of weights, the sequential nature of the generation process, and the dynamic nature of the attention layers.

4.1 Speedup and Energy Efficiency

Despite the diversity in the experimental setups used to test each method, some observations can be drawn from Table 1 to understand the effectiveness of each optimization approach. Among the software level solutions, UltraFastBERT [2] demonstrates the most significant speedup, an interesting result given its relatively simple software tweaks compared to other methods. This highlights the effectiveness of focused optimizations within the model's architecture itself. On the hardware side, X-Former [6] stands out by providing the most substantial speedup among all evaluated methods, as illustrated in Figure 1. This observation underscores the impact of

Table 1. Comparison between different LLM related optimization techniques discussed in this work. Note that we did not reproduce any of the metrics and the values were directly borrowed from the respective papers.

Year	Method	Optimization level	Technology	Baseline	Speedup	Energy efficiency
2023	Flash[1]	Software	Flash	CPU/GPU	25x/5x	—
2023	UltraFastBERT[2]	Software	CPU	CPU	78x	—
2023	LLMA[10]	Software	GPU	GPU	2x	—
2023	X-Former[6]	Hardware	In-memory	GPU	85x	7.5x
2023	FlexRun[7]	Hardware	FPGA	GPU	2.7x	—

hardware-level optimizations, despite the higher inherent costs associated with such solutions. The substantial gains from X-Former reinforce the value of investing in specialized hardware modifications to enhance the performance of LLMs, indicating that strategic hardware improvements can yield significant efficiency benefits even in the face of substantial initial investments.

While X-Former [6] provides clear data on energy efficiency, it is challenging to infer the energy efficiency of other methods due to the absence of comprehensive metrics. The operational principles of the methods discussed suggest a potential increase in energy efficiency, as each aims to optimize some aspect of LLM processing, whether by reducing computational overhead, streamlining memory usage, or enhancing processing speeds. However, without concrete numerical data, we refrain from drawing definitive conclusions about the energy efficiency improvements of these methods. Thus, while there is an implicit suggestion that these optimizations might lead to better energy management, this discussion remains speculative without robust empirical support. We acknowledge this gap in the analysis and suggest that future research should aim to provide detailed energy consumption metrics to fully assess the impact of these optimization strategies.

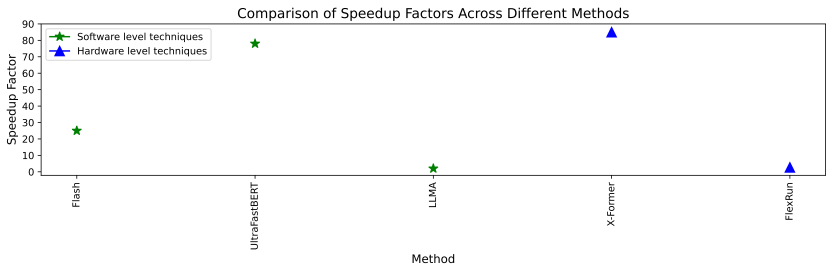

Fig. 1. Comparison of Speedup Factors Across Different Methods.

5 CONCLUSION

This paper has provided a comprehensive review of recent advancements in both hardware and software optimizations for Large Language Models (LLMs), emphasizing the substantial progress made in addressing their computational and energy efficiency challenges. Through an in-depth examination of a variety of technologies, from cutting-edge accelerator designs to novel software methodologies, we have demonstrated how these developments significantly enhance the performance and practicality of LLMs across multiple deployment contexts. As LLMs increasingly integrate into diverse sectors, the need for robust, efficient, and flexible optimization strategies becomes ever more critical. This review highlights the importance of ongoing innovation in this domain, which promises to broaden the deployment of LLMs, thereby enhancing the speed, efficiency, and sustainability of AI technologies on a global scale.

6 FUTURE WORKS

Throughout our analysis, we noticed a significant lack of synergy between software and hardware optimization methods. Although the techniques discussed in this paper function effectively as standalone solutions, there is a substantial opportunity for integration that could enhance their effectiveness. For instance, in the **X-Former [6]** architecture, the projection engine's performance could be further accelerated by integrating the storage optimization techniques used in **Flash [1]**. Additionally, inspired by **UltraFastBERT [2]**, the feedforward layers in X-Former could be replaced with Fast Feed Forward layers, leading to a more sparse and speed-enhanced inference. The effectiveness of these combined optimizations could be further amplified by incorporating the inference with reference to source technique utilized in **LLMA [10]**. To bring all these enhancements together in a cohesive package, a **FlexRun [7]**-like suite could be developed to facilitate flexible deployment on FPGAs, offering a comprehensive solution that maximizes the efficiency and adaptability of LLMs.

This potential for a holistic integration of optimization techniques appears to be a glaring omission in current research efforts and represents a promising direction for future exploration. Encouraging the development of such integrated systems could drive significant advancements in the field, providing a blueprint for future efforts to enhance the efficiency and effectiveness of LLM technologies.

ACKNOWLEDGMENTS

We would like to express our gratitude to our course instructors, Dr. Rajesh Kedia and Dr. Shirshendu Das, for their invaluable guidance and support throughout the course. Their insights and expertise were instrumental in shaping both the direction and execution of this work. Additionally, we wish to acknowledge the contributions of [4], which provided substantial inspiration for organizing this survey paper cohesively.

REFERENCES

[1] Keivan Alizadeh-Vahid, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifard, Minsik Cho, Carlo C. Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2023. LLM in a flash: Efficient Large Language Model Inference with Limited Memory. *ArXiv* abs/2312.11514 (2023). https://api.semanticscholar.org/CorpusID:266362016

[2] Peter Belcák and Roger Wattenhofer. 2023. Exponentially Faster Language Modelling. *ArXiv* abs/2311.10770 (2023). https://api.semanticscholar.org/CorpusID:265294694

[3] Peter Belcák and Roger Wattenhofer. 2023. Fast Feedforward Networks. *ArXiv* abs/2308.14711 (2023). https://api.semanticscholar.org/CorpusID:261243851

[4] Christoforos Kachris. 2024. A Survey on Hardware Accelerators for Large Language Models. *ArXiv* abs/2401.09890 (2024). https://api.semanticscholar.org/CorpusID:267035076

[5] Malik Sallam. 2023. ChatGPT Utility in Healthcare Education, Research, and Practice: Systematic Review on the Promising Perspectives and Valid Concerns. *Healthcare* 11 (2023). https://api.semanticscholar.org/CorpusID:257650377

[6] Shrihari Sridharan, Jacob R. Stevens, Kaushik Roy, and Anand Raghunathan. 2023. X-Former: In-Memory Acceleration of Transformers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 31 (2023), 1223–1233. https://api.semanticscholar.org/CorpusID:257505181

[7] Hur Suyeon, Seongmin Na, Dongup Kwon, Kim Joonsung, Andrew Boutros, Eriko Nurvitadhi, and Jang-Hyun Kim. 2022. A Fast and Flexible FPGA-based Accelerator for Natural Language Processing Neural Networks. *ACM Transactions on Architecture and Code Optimization* 20 (2022), 1 – 24. https://api.semanticscholar.org/CorpusID:252669471

[8] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Neural Information Processing Systems*. https://api.semanticscholar.org/CorpusID:13756489

[9] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. BloombergGPT: A Large Language Model for Finance. *ArXiv* abs/2303.17564 (2023). https://api.semanticscholar.org/CorpusID:257833842

[10] 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* abs/2304.04487 (2023). https://api.semanticscholar.org/CorpusID:258048436