iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** VIII **Month of publication:** August 2024

DOI: https://doi.org/10.22214/ijraset.2024.63985

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue VIII Aug 2024- Available at www.ijraset.com

Open-AI model Efficient Memory Reduce Management for the Large Language Models (LLMs) Serving with Paged Attention of sharing the KV Cashes

Dr. K. Naveen Kumar[1], Mr. Sreedhar Ambala[2], Dr. M. B Raju[3], Dr. Sai Hareesh Anamandra[4], Mr. Dhanunjaya Rao Kodali[5]

[1]*M.Tech., Ph.D., Professor & Head of the Department of CSE-Cyber Security, Pallavi Engineering College (PEC), Kuntlur (V), Abdullapurmet (M), Hyderabad, R.R (Dist)- 501 505*

[2]*M.Tech., (Ph.D)., Asst. Professor of the Department of CSE. Pallavi Engineering College (PEC), Kuntlur (V), Abdullapurmet (M), Hyderabad, R.R (Dist)- 501 505*

[3]*M.Tech., Ph.D., Professor of the Department of CSE, Principal, Pallavi Engineering College (PEC), Kuntlur (V), Abdullapurmet (M), Hyderabad, R.R (Dist)- 501 505*

[4]*M.Tech., Ph.D., Associate Professor & Head of the Department of CSE, Pallavi Engineering College (PEC), Kuntlur (V), Abdullapurmet (M), Hyderabad, R.R (Dist)- 501 505*

[5]*M.Tech CSE., M.Sc Maths., B.Ed., Asst. Professor of the Department of CSE, Pallavi Engineering College (PEC), Kuntlur (V), Abdullapurmet (M), Hyderabad, R.R (Dist)- 501 505*

Abstract: ***High throughput serving of large language models (LLMs) requires batching sufficiently many requests at a time. However, existing systems struggle because the key-value cache (KV cache) memory for each request is huge and grows and shrinks dynamically. When managed inefficiently, this memory can be significantly wasted by fragmentation and redundant duplication, limiting the batch size. To address this problem we proposed a Paged Attention. An alternative algorithm inspired by the classical virtual memory and paging techniques in operating systems. An LLM serving system that achieves (1) near-zero waste in KV cache memory and (2) flexible sharing of KV cache within and across requests to further reduce memory usage. Our evaluations show that LLM improves the throughput of popular LLMs by 2-4× with the same level of latency compared to the state-of-the-art systems, such as Faster Transformer and Orca. The improvement is more pronounced with longer sequences, larger models, and more complex decoding algorithms.***

Keywords: LLMs, KV Cashes, Open Ai, Chat Bots, GPU Memory and Paged Attention.

I. INTRODUCTION

The emergence of large language models (LLMs) like GPT [5] and PaLM [9] have enabled new applications such as programming assistants [6, 18] and universal chat bots [5] that are starting to profoundly impact our work and daily routines. Many cloud companies [4,] are racing to pro- vide these applications as hosted services. However, running these applications is very expensive, requiring a large number of hardware accelerators such as GPUs. According to recent estimates, processing an LLM request can be 10×more expensive than a traditional keyword query [4]. Given these high costs, increasing the throughput—and hence reducing

At the core of LLMs lies an autoregressive Transformer model [5]. This model generates words (tokens), one at a time, based on the input (prompt) and the previous sequence of the output's tokens it has generated so far. For each re- quest, this expensive process is repeated until the model out- puts a termination token. This sequential generation process makes the workload memory-bound, underutilizing the computation power of GPUs and limiting the serving throughput. Improving the throughput is possible by batching multiple requests together. However, to process many requests in a batch, the memory space for each request should be efficiently managed. For example, Fig. 1 (left) illustrates the memory distribution for a 13B-parameter LLM on an NVIDIA A100 GPU with 40GB RAM. Approximately 65% of the memory is allocated for the model weights, which remain static during serving. Close to 30% of the memory is used to store the dynamic states of the requests. For Transformers, these states consist of the key and value tensors associated with the attention mechanism, commonly referred to as KV cache [41], which represent the context from earlier tokens to generate new output tokens in sequence.

©IJRASET: All Rights are Reserved

II. BACKGROUND

In this section, we describe the generation and serving procedures of typical LLMs and the iteration-level scheduling used in LLM serving.

the same model weights, the overhead of moving weights is amortized across the requests in a batch, and can be over- whelmed by the computational overhead when the batch size is sufficiently large. However, batching the requests to an LLM service is non-trivial for two reasons. First, the requests may arrive at different times. A naive batching strategy would either make earlier requests wait for later ones or delay the incoming requests until earlier ones finish, leading to significant queueing delays. Second, the requests may have vastly different input and output lengths (Fig. 11). A straightforward batching technique would pad the inputs and outputs of the requests to equalize their lengths, wasting GPU computation and memory.

To address this problem, fine-grained batching mechanisms, such as cellular batching [16] and iteration-level scheduling [60], have been proposed. Unlike traditional methods that work at the request level, these techniques operate at the iteration level. After each iteration, completed requests are removed from the batch, and new ones are added. There- fore, a new request can be processed after waiting for a single iteration, not waiting for the entire batch to complete. Moreover, with special GPU kernels, these techniques eliminate the need to pad the inputs and outputs. By reducing the queueing delay and the inefficiencies from padding, the fine-grained batching mechanisms significantly increase the throughput of LLM serving.

III. MEMORY CHALLENGES IN LLM SERVING

Although fine-grained batching reduces the waste of computing and enables requests to be batched in a more flexible way, the number of requests that can be batched together is still constrained by GPU memory capacity, particularly the space allocated to store the KV cache. In other words, the serving system's throughput is memory-bound. Overcoming this memory-bound requires addressing the following challenges in the memory management:

Large KV cache. The KV Cache size grows quickly with the number of requests. As an example, for the 13B parameter OPT model [62], the KV cache of a single token demands 800

KB of space, calculated as 2 (key and value vectors) ×5120

(hidden state size) × 40 (number of layers) × 2 (bytes per FP16). Since OPT can generate sequences up to 2048 tokens, the memory required to store the KV cache of one request can be as much as 1.6 GB. Concurrent GPUs have memory capacities in the tens of GBs. Even if all available memory was allocated to KV cache, only a few tens of requests could be accommodated. Moreover, inefficient memory management can further decrease the batch size, as shown in Fig. 2. Additionally, given the current trends, the GPU's computation speed grows faster than the memory capacity [17]. For example, from NVIDIA A100 to H100, The FLOPS increases by more than 2x, but the GPU memory stays at 80GB maximum. Therefore, we believe the memory will become an increasingly significant bottleneck.

Complex decoding algorithms. LLM services offer a range of decoding algorithms for users to select from, each with varying implications for memory management complexity. For example, when users request multiple random samples from a single input prompt, a typical use case in program suggestion [18], the KV cache of the prompt part, which accounts for 12% of the total KV cache memory in our experiment can be shared to minimize memory usage. On the other hand, the KV cache during the autoregressive generation phase should remain unshared due to the dif- ferent sample results and their dependence on context and position. The extent of KV cache sharing depends on the specific decoding algorithm employed. In more sophisticated algorithms like beam search [9], different request beams can share larger portions (up to 55% memory saving Scheduling for unknown input & output lengths. The requests to an LLM service exhibit variability in their input and output lengths. This requires the memory management system to accommodate a wide range of prompt lengths. In addition, as the output length of a request grows at decoding, the memory required for its KV cache also expands and may exhaust available memory for incoming requests or ongoing generation for existing prompts. The system needs to make scheduling decisions, such as deleting or swapping out the KV cache of some requests from GPU memory.

A. Memory Management in Existing Systems

Since most operators in current deep learning frameworks [33, 39] require tensors to be stored in contiguous memory, previous LLM serving systems [31, 60] also store the KV cache of one request as a contiguous tensor across the differ- ent positions. Due to the unpredictable output lengths from the LLM, they statically allocate a chunk of memory for a request based on the request's maximum possible sequence length, irrespective of the actual input or eventual output length of the request.

The chunk pre-allocation scheme in existing systems has three primary sources of memory wastes: re- served slots for future tokens, internal fragmentation due to over-provisioning for potential maximum sequence lengths, and external fragmentation from the memory allocator like the buddy allocator. The external fragmentation will never be used for generated tokens, which is known before serving a request. Internal fragmentation also remains unused, but this is only realized after a request has finished sampling. They are both pure memory waste. Although the reserved memory is eventually used, reserving this space for the en- tire request's duration, especially when the reserved space is large, occupies the space that could otherwise be used to process other requests. We visualize the average percentage of memory wastes in our experiments in Fig. 2, revealing that the actual effective memory in previous systems can be as low as 20.4%.

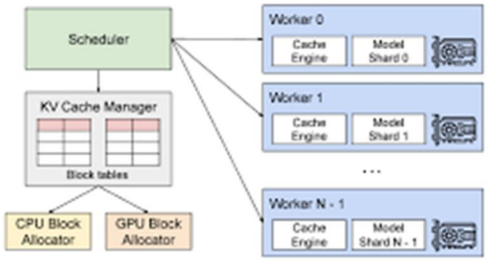


Figure 4. vLLM system overview.

IV. METHODOLOGY

In this work, we develop a new attention algorithm, Page d Attention, and build an LLM serving engine, vLLM, to tackle the challenges outlined in §3. The architecture of vLLM is shown in Fig. 4. vLLM adopts a centralized scheduler to coordinate the execution of distributed GPU workers. The KV cache manager effectively manages the KV cache in a paged fashion, enabled by Paged Attention. Specifically, the KV cache manager manages the physical KV cache memory on the GPU workers through the instructions sent by the centralized scheduler.

Next, We describe the Paged Attention algorithm in §4.1. With that, we show the design of the KV cache manager in §4.2 and how it facilitates Paged Attention in §4.3, respectively. Then, we show how this design facilitates effective memory management for various decoding methods (§4.4) and handles the variable length input and output sequences (§4.5). Finally, we show how the system design of vLLM works in a distributed setting (§4.6).

A. Paged Attention

To address the memory challenges in §3, we introduce Page d Attention, an attention algorithm inspired by the classic idea of paging [25] in operating systems. Unlike the traditional attention algorithms, Paged Attention allows storing continuous keys and values in non-contiguous memory space. Specifically, Paged Attention partitions the KV cache of each sequence into KV blocks. Each block contains the key and value vectors for a fixed number of tokens,1 which we denote as KV

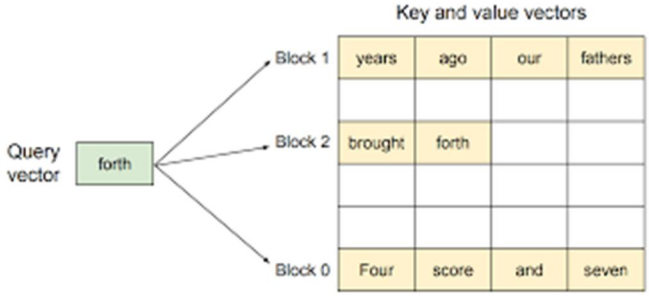


Illustration of the Paged Attention algorithm, where the attention key and values vectors are stored as continuous blocks in the memory.

During the attention computation, the Paged Attention kernel identifies and fetches different KV blocks separately. We show an example of Paged Attention in Fig. The key and value vectors are spread across three blocks, and the three blocks are not contiguous on the physical memory.

In summary, the Paged Attention algorithm allows the KV blocks to be stored in non-contiguous physical memory, which enables more flexible paged memory management in vLLM.

B. KV Cache Manager

The key idea behind vLLM's memory manager is analogous to the virtual memory [25] in operating systems. OS partitions memory into fixed-sized pages and maps user programs' logical pages to physical pages. Contiguous logical pages can correspond to non-contiguous physical memory pages, al- lowing user programs to access memory as though it were contiguous. Moreover, physical memory space needs not to be fully reserved in advance, enabling the OS to dynamically allocate physical pages as needed. vLLM uses the ideas be- hind virtual memory to manage the KV cache in an LLM service. Enabled by Paged Attention, we organize the KV cache as fixed-size KV blocks, like pages in virtual memory. A request's KV cache is represented as a series of logical

KV blocks, filled from left to right as new tokens and their KV cache are generated. The last KV block's unfilled positions are reserved for future generations. On GPU workers, a block engine allocates a contiguous chunk of GPU DRAM and divides it into physical KV blocks (this is also done on CPU RAM for swapping; see §4.5). The KV block manager also maintains block tables—the mapping between logical and physical KV blocks of each request. Each block table entry records the corresponding physical blocks of a logical block and the number of filled positions. Separating logical and physical KV blocks allows vLLM to dynamically grow the KV cache memory without reserving it for all positions in advance, which eliminates most memory waste in existing systems, as in Fig.

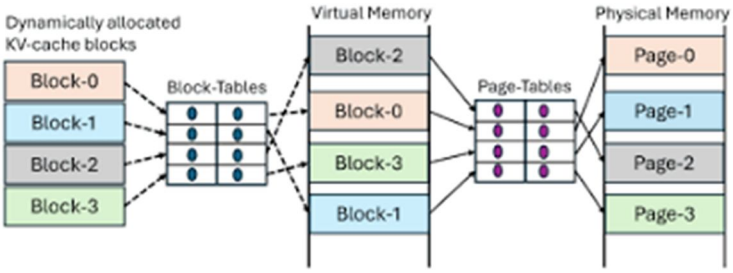


Fig: Block table translation in vLLM.ors are stored.

C. Decoding with Paged Attention and vLLM

Next, we walk through an example, as in Fig. , to demonstrate how vLLM executes Paged Attention and manages the memory during the decoding process of a single input sequence: ○1 As in OS's virtual memory, vLLM does not require reserving the memory for the maximum possible generated sequence length initially. Instead, it reserves only the necessary KV blocks to accommodate the KV cache generated during prompt computation. In this case, The prompt has 7 tokens, so vLLM maps the first 2 logical KV blocks (0 and

1) to 2 physical KV blocks (7 and 1, respectively). In the prefill step, vLLM generates the KV cache of the prompts and the first output token with a conventional self-attention algorithm (e.g., [13]). vLLM then stores the KV cache of the first 4 tokens in logical block 0 and the following 3 tokens in logical block 1. The remaining slot is reserved for the time in vLLM requests and the latest tokens for generation phase requests) as one sequence and feeds it into the LLM. During LLM's computation, vLLM uses the Paged Attention kernel to access the previous KV cache stored in the form of logical KV blocks and saves the newly generated KV cache into the physical KV blocks. Storing multiple tokens within a KV block (block size > 1) enables the Paged Attention kernel to process the KV cache across more positions in parallel, thus increasing the hardware utilization and reducing latency. However, a larger block size also increases memory fragmentation. Again, vLLM dynamically assigns new physical blocks to logical blocks as more tokens and their KV cache are generated. As all the blocks are filled from left to right and a new physical block is only allocated when all previous blocks are full, vLLM limits all the memory wastes for a request within one block, so it can effectively utilize all the memory, as shown in Fig. 2. This allows more requests to fit into memory for batching—hence improving the throughput. Once a request finishes its generation, its KV blocks can be freed to store the KV cache of other requests. In Fig., we show an example of vLLM managing the memory for two sequences. The logical blocks of the two sequences are mapped to different physical blocks within the space reserved by the block engine in GPU workers. The neighbouring logical blocks of both sequences do not need to be contiguous in physical GPU memory and the space of physical blocks can be effectively .

V. IMPLEMENTATION

vLLM is an end-to-end serving system with a Fast API [1] frontend and a GPU-based inference engine. The frontend extends the OpenAI API [4] interface, allowing users to customize sampling parameters for each request, such as the maximum sequence length and the beam width. The vLLM engine is written in 8.5K lines of Python and 2K lines of C++/CUDA code. We develop control-related components including the scheduler and the block manager in Python while developing custom CUDA kernels for key operations such as Paged Attention. For the model executor, we implement popular LLMs such as GPT [5], OPT [2], and LLaMA [2] using

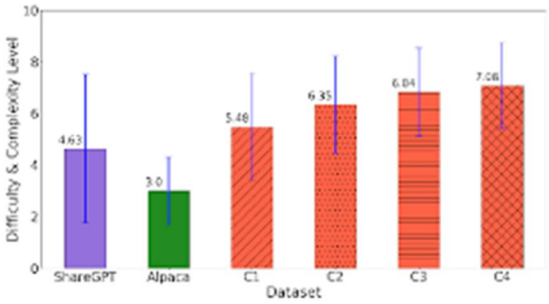


Fig: Share GPT and Alpaca.

A. Kernel-level Optimization

Since Paged Attention introduces memory access patterns that are not efficiently supported by existing systems, we develop several GPU kernels for optimizing it. (1) Fused re- shape and block write. In every Transformer layer, the new KV cache are split into blocks, reshaped to a memory layout optimized for block read, then saved at positions specified by the block table. To minimize kernel launch overheads, we fuse them into a single kernel. (2) Fusing block read and attention. We adapt the attention kernel in Faster Transformer [1] to read KV cache according to the block table and perform attention operations on the fly. To ensure coalesced memory access, we assign a GPU warp to read each block. More- over, we add support for variable sequence lengths within a request batch. (3) Fused block copy. Block copy operations, issued by the copy-on-write mechanism, may operate on discontinuous blocks. This can lead to numerous invocations of small data movements if we use the Open AI models. To mitigate the overhead, we implement a kernel that batches the copy operations for different blocks into a single kernel launch.

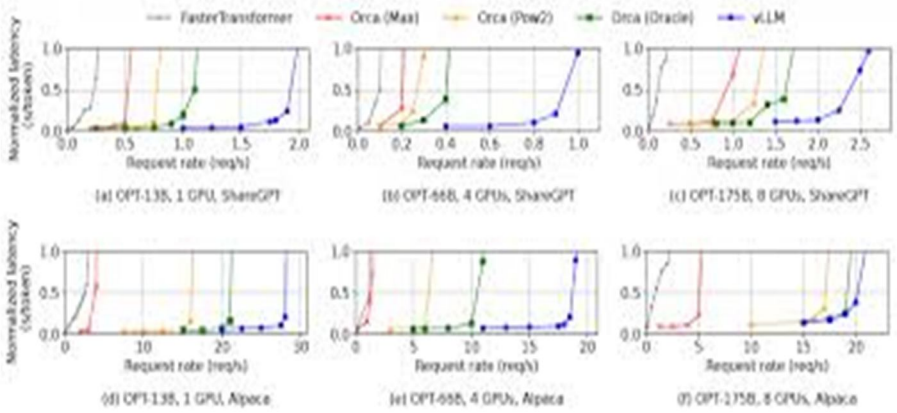


Fig: Paged Attention and vLLM serve Large Language Models faster and cheaper.

VI. CONCLUSION AND FUTURE SCOPE

This paper proposes Paged Attention, a new attention algorithm that allows attention keys and values to be stored in non-contiguous paged memory, and presents vLLM, a high-throughput LLM serving system with efficient memory management enabled by Paged Attention. Inspired by operating systems, we demonstrate how established techniques, such as virtual memory and copy-on-write, can be adapted to efficiently manage KV cache and handle various decoding algorithms in LLM serving. Our experiments show that vLLM achieves 2-4×throughput improvements over the state-of-the-art systems.

©IJRASET: All Rights are Reserved

Applying the virtual memory and paging technique to other GPU workloads. The idea of virtual memory and paging is effective for managing the KV cache in LLM serving because the workload requires dynamic memory allocation (since the output length is not known a priori) and its performance is bound by the GPU memory capacity. However, this does not generally hold for every GPU workload. For example, in DNN training, the tensor shapes are typically static, and thus memory allocation can be optimized ahead of time. For another example, in serving DNNs that are not LLMs, an increase in memory efficiency may not result in any performance improvement since the performance is primarily compute-bound. In such scenarios, introducing the vLLM's techniques may rather degrade the performance due to the extra overhead of memory indirection and non-contiguous block memory. However, we would be excited to see vLLM's techniques being applied to other workloads with similar properties to LLM serving.

In the future we can go with LLM-specific optimizations in applying virtual memory and paging. vLLM reinterprets and augments the idea of virtual memory and paging by leveraging the application- specific semantics. One example is vLLM's all-or-nothing swap-out policy, which exploits the fact that processing a request requires all of its corresponding token states to be stored in GPU memory. Another example is the re computation method to recover the evicted blocks, which is not feasible in OS. Besides, vLLM mitigates the overhead of memory indirection in paging by fusing the GPU kernels for memory.

REFERENCES

[1] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Am- mar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, et al. 2022. DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. arXiv preprint arXiv:2207.00032 (2022).

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. arXiv preprint arXiv:1607.06450 (2016).

[3] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A neural probabilistic language model. Advances in neural information process- ing systems 13 (2000)

[4] Ond rej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Gra- ham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Car- olina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 Conference on Machine Trans- lation. In Proceedings of the First Conference on Machine Translation. Association for Computational Linguistics, Berlin, Germany, 131–198. http://www.aclweb.org/anthology/W/W16/W16-2301

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

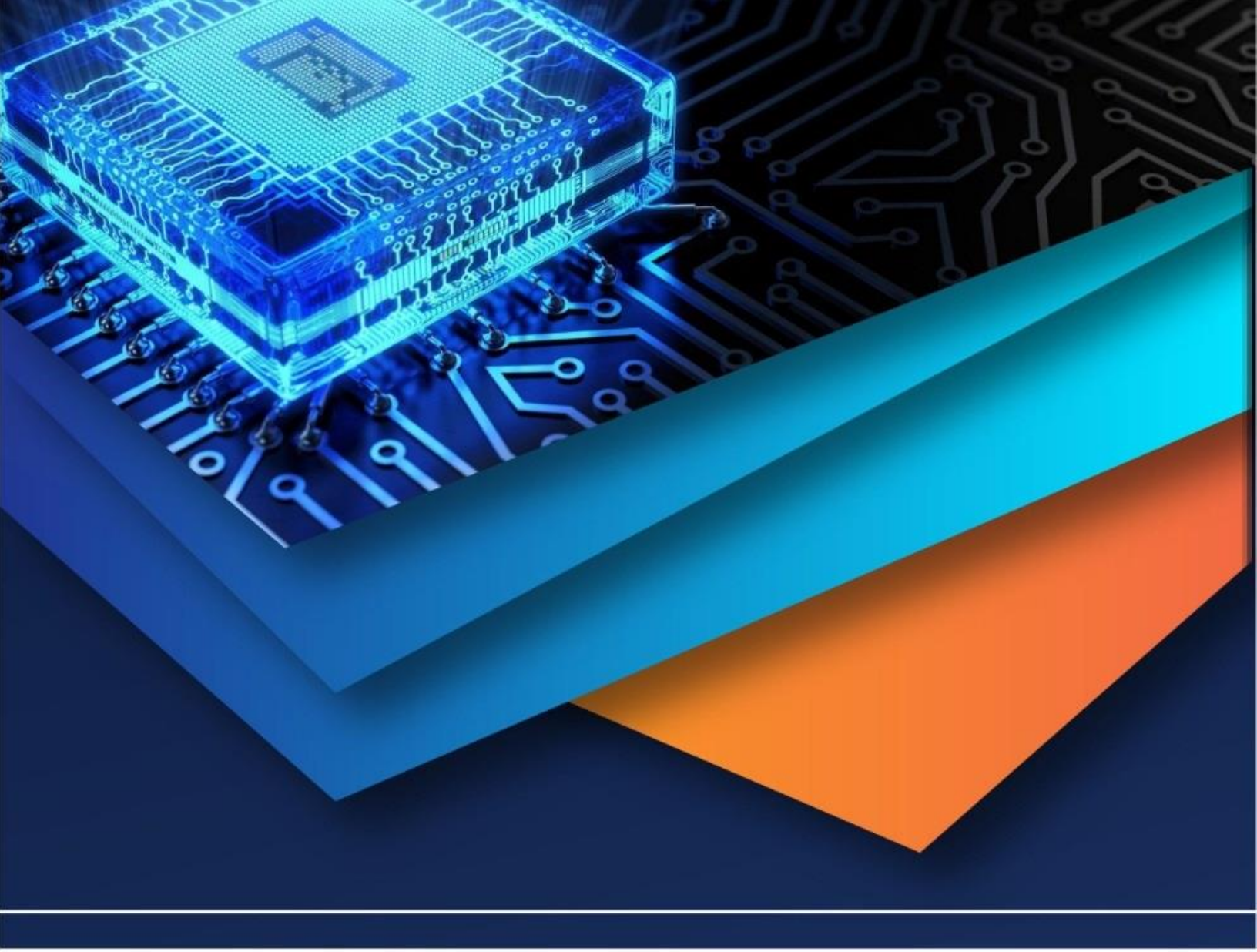
[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374 (2021).

[7] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. arXiv preprint arXiv:1604.06174 (2016).

[8] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality. https://lmsys. org/blog/2023-03-30-vicuna/

[9] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling lan- guage modeling with pathways. arXiv preprint arXiv:2204.02311 (2022).

[10] Dr. K. Naveen Kumar1 , Mr. R. Mallikharjun2 , Mr. L. Jagdeesh Nayak3 , Survey of Machine Learning Applications of Convolutional Neural Networks to Medical Image Analysis . International Journal for Research in Applied Science & Engineering Technology (IJRASET) ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429 Volume 9 Issue XI Nov 2021- Available at www.ijraset.com.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)