A Specialized Large Language Model for Electronic Design Automation

Iraj Shahzad
Electrical (Communication and Signal Processing) Engineering
The Islamia University, Bahawalpur, Pakistan
iraj_shahzad@yahoo.com

Muhammad Ali Raza
Electrical (Communication and Signal Processing) Engineering
The Islamia University, Bahawalpur, Pakistan
ali.bashir1947@gmail.com

Asjad Amin
Electrical (Communication and Signal Processing) Engineering
The Islamia University, Bahawalpur, Pakistan
asjad.amin@iub.edu.pk

***Abstract*—Integrating complex sets of Electronic Design Automation (EDA) tools for seamless information exchange has become a critical challenge for designers and engineers. Recent advancements in Large Language Models (LLMs) have driven significant progress in Natural Language Processing (NLP), enabling improved interaction with EDA tools. In this thesis, we propose *SmartEDA*, an agent and executor framework for EDA workflows, powered by LLMs and AutoMage. SmartEDA streamlines the design flow from RTL to GDSII by efficiently managing task decomposition, script generation, and task execution.**

***Index Terms*—AutoMage, Electronic Design Automation (EDA), Natural Language Processing (NLP), Large Language Models (LLMs), GPT-4, Register-Transfer Level (RTL), GDSII.**

I. Introduction

EDA tools possess a very complex design flow and programming interfaces. Designers use multiple tools for a single task and hence it becomes difficult for them to maintain these scripts. Currently, the implementation of LLMs with toolchain is generic and universal and cannot be fine-tuned [1]. For this reason user requirements cannot be met steadily and persistently. Due to limited understanding and familiarity between EDA and LLM in EDA domain, tools usage encounters consistent faults and inaccuracies. Work has commenced on the integration of tools and models with Large Language Models (LLMs) [2]. Toolformer is a method which integrates API tags in text sequences during which LLMs communicate with external tools as well [3]. This tool does not only increases the scope and capability of LLM, but it also expands and enlarges the logical reasoning and problem solving skills of LLMs. Table I summarizes prior work on EDA automation and large language models, including OpenROAD [4], iEDA [5], BabyAGI [6], LoRA [7], GPTQ [8], and ChatEDA [9].

This paper introduces SmartEDA: The first LLM-based Electronic Design Automation (EDA) tool. AutoMage Series (AutoMage and AutoMage2): Includes AutoMage and AutoMage2, designed to enhance the capabilities of EDA tools. Evaluations using SmartEDA Bench: Benchmarks to demonstrate the superior performance of the AutoMage series over other well-known LLMs,as shown in Table II.

The following diagram shows an overview of AutoMage powered SmartEDA, with AutoMage working as controller and EDA tools working as executors.

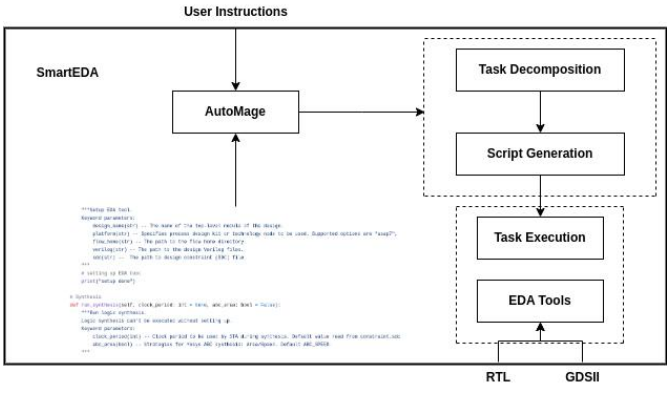

Fig. 1. Architecture of the proposed SmartEDA system.

II. Background

A. Generative Pretrained Language Model (GPLM)

GPLMs are a class of neural network architectures which are used to understand and generate human like text. They have completely changed the field of NLP and have enabled them to perform tasks like translation, summarizing, etc, with exceptional efficiency and accuracy. GPLM is the most successful point in LLM advancements.

Conventional models like BERT [15] and XLNet [16] contain encoder-decoder architecture. GPLM is based on neural network structure which includes decoder blocks based on the architecture [10], [11], [13], [12]. It has more advantages over the conventional architecture [17]. It can capture long range dependencies skillfully leading to the synthesis of logical prose. It is also called auto regressive decoder. This decoder is used in NLP and machine learning. It generates text token by token (word by word or character by character) in which it considers already generated tokens.

DOI: https://doi.org/10.67535/tsp.000001.041
Published under the ICACNC 2025 Proceedings

TABLE I
LITERATURE REVIEW

Ref	Title & Authors	Year	Key Contribution	Limitation
[4]	OpenROAD - T. Ajayi et al.	2019	Proposed a open-source,autonomous RTL-to-GDSII flow for IC needed.	Manual Scripting still needed; lacks natural langugae interaction
[5]	iEDA - X. Li et al.	2023	Introduced an open-source, intelligent physical design toolkit with APIs and automation features.	API knowledge required; lacks human-like interaction.
[1]	Scripting for EDA Tools - P. Chen et al.	2001	Demonstrated scripting methods to automate EDA tool workflows.	Focused on scripting, not AI-driven automation.
[2]	GPT-3 - T. Brown et al.	2020	Showed few-shot learning and powerful text generation in general NLP.	Not fine-tuned for domain-specific (EDA) tasks.
[10]	GPT-4 Technical Report - OpenAI	2023	Improved reasoning, few-shot performance, and context window.	Still struggles in domain-specific tool execution like EDA.
[11]	Claude2 - Anthropic	2023	Designed to be helpful, honest, harmless; good at reasoning tasks.	Less open and less controllable for domain tuning.
[12]	LLaMA - H. Touvron et al.	2023	Open-source foundation models with decoder-only transformer design.	Lacks native EDA-specific knowledge; needs tuning for domain use.
[13]	LLaMA 2 - H. Touvron et al.	2023	Open-source foundation models with fine-tuned chat capability.	Requires instruction-tuning for domain tasks.
[3]	Toolformer - T. Schick et al.	2023	Introduced self-supervised LLM training to use external APIs.	Proof-of-concept; not specialized for EDA.
[6]	BabyAGI	2023	Experimental LLM agent simulating cognitive planning and execution loops.	Experimental and unstable for EDA tasks.
[**?**]	BERT - J. Kenton et al.	2019	Introduced deep bidirectional transformer pretraining for language understanding.	General NLP model; not designed for EDA tasks.
[14]	XLNet - Z. Yang et al.	2019	Generalized autoregressive pretraining for improved language modeling.	Not EDA-specific; requires fine-tuning for domain tasks.
[**?**]	GPT - A. Radford et al.	2018	Introduced generative pretraining for language understanding.	Limited context window; not optimized for EDA.
[7]	LoRA - E. J. Hu et al.	2021	Proposed low-rank adaptation for efficient LLM fine-tuning.	Only improves adaptation; full model training still costly.
[8]	GPTQ - E. Frantar et al.	2022	Accurate post-training quantization for generative transformers.	Focused on compression; not directly improving reasoning for EDA.
[9]	ChatEDA - H. Wu et al.	2024	LLM-powered autonomous agent for EDA, enabling code generation and tool automation.	Early implementation; may require adaptation for diverse EDA tools.

GPLMs are trained through self-supervised learning so that they can represent broad and vast languages [18]. Model parameters are refined during the training so that they can predict the ensuing tokens correctly without any error in sequence. Examples include GPT-4, Palm, Llama, etc. Difference between Encoder-Decoder Architecture and Auto-Regressive Model is given below:

B. Low Rank Adaptions of LLMs

LLMs have vast and large parameters due to which it becomes difficult to train each parameter during the training process. An efficient solution to this problem is Low rank adaptions of LLMs (LORA) [7]. In this method, weights of pre-trained models are preserved and are introduced in each layer of low rank decomposition matrices. This reduces the need of trainable parameters.

Transformer architecture has various fully connected layers which conduct multiplication with full rank weight matrices. Pre-trained language models are complex but still they make representation of low intrinsic models (to represent a complex data or model structure into fewer dimensions) easier and they

TABLE II
SUMMARY OF KEY CONTRIBUTIONS

Contribution	Description
SmartEDA	The first LLM-based Electronic Design Automation (EDA) tool.
AutoMage Series (AutoMage and AutoMage2)	Includes AutoMage and AutoMage2, designed to enhance the capabilities of EDA tools.
Evaluations using SmartEDA Bench	Benchmarks to demonstrate the superior performance of the AutoMage series over other well-known LLMs.

TABLE III
COMPARISON OF ENCODER-DECODER ARCHITECTURE AND AUTO-REGRESSIVE MODEL

Feature	**Encoder-Decoder Architecture**	**Auto-Regressive Model**
Architecture Type	Encoder-Decoder	Decoder-only
Input Requirement	Requires full input sequence	Generates from prompt (few tokens)
Output Generation	Uses context vector from encoder	Uses previous output tokens
Attention Mechanism	Cross-attention and self-attention	Only causal self-attention
Example Models	T5, BART, Transformer (original)	GPT-2, GPT-3, GPT-4, LLaMA
Direction of Attention	Bidirectional (encoder) full	Unidirectional (left to right)
Use Cases	Translation, Summarization	Text generation, code, Q&A

also learn efficiently on random projection [19]. We can also update a pre-trained model using low rank decomposition.

Low rank decomposition is the process of breaking down a large matrix into smaller, lower dimensional matrices, whose product approximately matches the original matrix. Consider the following equation:

$$y = (W + \Delta W)x \tag{1}$$

The above equation is the formula to run the model after it is trained. In this equation, y is the output vector, x is the input vector, W is pre-trained (frozen) weight matrix, which is divided into two small matrices i.e L_1 and L_2. L_1 and L_2 are then trained whereas the original trained matrix W remains fixed. ΔW is the change added to W learned using LORA.

L_1 and L_2 are low rank matrices with dimension $d \times r$ and $r \times k$. Initially ΔW is initialized with random Gaussian distribution and is set to zero, making sure that ΔW is set to zero at outset. During production deployment, the equation is used and stored for inference. This method has no additional latency and is efficient as well. LoRA in this context is used for fine-tuning process.

C. In-Context Learning

In context learning is a technique in which LLMs like GPT-4 or Llama are trained by just learning through examples given through prompt, without being fine-tuned [20].

Model is provided with some examples through prompt through which it understands the user requirement/task and performs accordingly. For example: If we provide two examples saying "Oxford University is in England." "UC Berkeley is in California." "Stanford University is in _____."

ICL is done when it gets long range coherence in documents. Long range is coherency or consistency between different parts of document or text no matter how far they are. In the pre-training phase, LLM has to understand the hidden content (latent content) in the multiple sentences. Consider the following equation:

$$p(y|x) = \int p(y|x,\theta)p(\theta|x)d\theta \tag{2}$$

Here y is the output prediction we require. x is the input prompt (examples given by the user). θ represents the latent content. $p(y|x)$ is the probability of output given context. $p(y|x,\theta)$ is probability of output given fixed model and prompt. $p(\theta|x)$ is the posterior over model parameters given prompt. In-Context Learning (ICL) means figuring out how likely different possible completions are, based on the examples given in the prompt using what the model has already learned [21].

D. Block k-bit Quantization

Quantization is a process of converting high information data into low information data such as conversion of 32-bit floats into 8-bit fewer bit integers [22].

k-bit levels make this process better and it makes sure that the full range of low-bit data type is used effectively. In this approach, data is scaled so that it can fit between the limits of target data type. For example, to quantize a 32-bit floating point tensor (FP32) into Int8 tensor within the range of [-127,127]. This can be expressed as:

$$q = \text{round}\left(\frac{x}{s}\right) \tag{3}$$

$$s = \frac{\max(|x|)}{127} \tag{4}$$

Here x is the original 32-bit floating point data. $\max(|x|)$ is the biggest value in the data. 127 is the maximum value that fits in 8-bit signed integer. q is the final quantized value.

$$X \in \mathbb{R}^{b \times h} \tag{5}$$

When we have an input tensor X which is a matrix with size (b, h) (rows b, columns h), we make a 1D array of it. Now we divide this line into small blocks with each block size B. Then we calculate total blocks and every block is then quantized independently [1]. Each block has individual scale constant s_i, where $X \in \mathbb{R}$ and can be 2-dimensional.

III. Methodology

A. SmartEDA-LLM-Powered Framework for EDA

SmartEDA is specially designed for automation of RTL-GDSII flow. It understands the user requirements and responds back to them. To achieve these goals SmartEDA breakdowns complex user requirements in to smaller and more manageable subtasks and uses proper EDA tools to address these tasks. First the user gives input to AutoMage as user requirements/natural language requirements. It understands these requirements and afterwards it decomposes the tasks in to subtasks. AutoMage then generates python scripts for execution of the subtasks on the basis of tasks given to it and on the basis of tools specifications. Finally SmartEDA executes the generated scripts to get final output for user requirements.

1) Task Decomposition: Since RTL-GDSII automation is a complex nature task, hence it requires to divide longer tasks in to smaller subtasks to achieve the desired outcome. For this purpose task decomposition is introduced as a primary stage in this thesis.

2) Script Generation: Once the process of task decomposition is complete and subtasks are defined, now every subtask is executable through APIs. But now we require a script which will invoke these APIs for task execution. This phase is called script generation. Figure 3 shows the scripts generated.

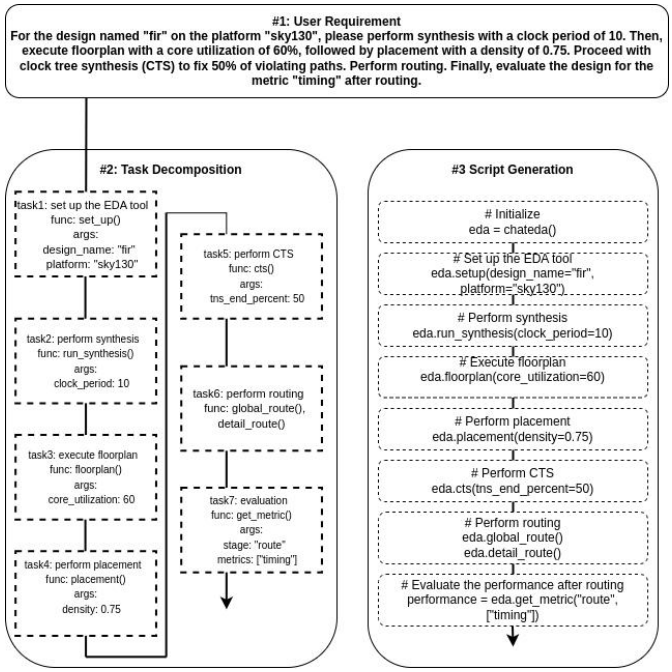

Fig. 2. Script Generation and Task Decomposition

3) Task Execution: After script generation, SmartEDA executes those scripts and subtasks using python interpreter and EDA Tools like OpenROAD.

B. AutoMage-LLM-Based Controller of SmartEDA

AutoMage model is a fine tuned version of Llama2. This model is based on design of a standard transformer model architecture in decoder-only setup. This means that at every step this model will only look up to current states or previous inputs instead of future states. CodeLlama is also an LLM model which performs efficiently in coding due to incremental pre-training. But this training results in the loss of general knowledge, including knowledge related to EDA [23]. AutoMage model balances these aspects. It is necessary for SmartEDA to have basic knowledge in order to understand user requirements on EDA tools. For this reason, we will used Llama2 model in this research work.

A dataset of 1500 instances was created for self-instruction and manually refined to ensure accuracy. In the instruction fine tuning process, each instance includes detailed composition process and script, and requirements and responses are combined in a single sequence through concatenation to ensure precise and accurate training.

C. Efficient Fine-Tuning of Quantized LLMs

To fine tune LLMs efficiently, a technique called QLORA is used. QLORA uses 4-bit Normal Float (NF4) to reduce the precision of weights of models from 32-bit to 4-bit, resulting in reduced memory requirement and increased efficiency. Double quantization further optimizes memory usage by compressing scale factors. QLORA combines quantization and Low-Rank Adaption (LoRA) to fine-tune models efficiently without touching original weights.

D. AutoMage2: Upgraded Version of AutoMage

AutoMage2 addresses limitations of AutoMage, including overfitting on 1500 tool instructions and lack of coding and logical reasoning [9]. AutoMage2 encompasses the same model architecture with several advancements: enriched corpus, instruction training with explanation, and chain of thoughts (COT). The enriched corpus includes 1500 refined EDA tool instructions combined with 110,000 code-related instructions from open repositories, forming a hybrid dataset. Instruction tuning with explanation uses step-by-step guidance from a teacher model (GPT-3.5/4) to teach reasoning. Chain of thoughts (COT) enables the model to think step-by-step to logically solve complex problems using zero-shot COT prompts, resulting in more accurate and logical answers [24].

IV. Experiments and Results

A. Setup

A special training setup is used to train AutoMage2 efficiently, in which the learning rate is set constant, i.e learning speed is fixed. But to slow start it in the beginning, warm-up ratio of 0.03 was given. This method uses Paged AdamW 8-bit optimizer, which is an efficient training optimizer [25]. It fine tunes models by using less memory. Adam W Optimizer is a popular optimizer which trains deep learning models. It is an improved version of Adam optimizer in which W-weights decay is handled separately. In 8-bit optimizer, some parts are compressed in 8-bit precision which reduces memory consumption 4x times keeping performance unchanged. It is difficult to store a large model with billion + parameters in

a GPU. Page optimizer breaks the data in to pages (chunks) and loads/unloads the needed page into the memory, just like paging. In this way, we can fine tune a larger model with even limited GPU memory. The training starts at a learning rate of 0.0001 with no weight decay. Batch size is 128 and the size of each input was 4096 tokens. The model was trained on the entire data set only once, and this training was done using 16 powerful GPUs. When the model is being used (inference phase), user gives instructions in simple english. To test the performance of this model, it is compared with famous language models like Claude2, GPT-3.5/4.

B. SmartEDA-Bench

A test set named as SmartEDA-Bench has been made in order to test effectiveness and performance of AutoMage. It includes total 50 tasks which are divided in to three categories: simple flow calls, complex flow calls and parameter flow calls. Simple flow calls include basic API instructions. Complex flow calls require logical reasoning like using multiple parameters correctly, understanding API arguments. Parameter flow calls test parameter tuning and optimization, meaning the model must adjust specific values to get the correct result.

C. Evaluation of LLMs

An evaluation system is made to test AutoMage, an AI model. Evaluation process comprises of automated testing and manual evaluation. The python script generated by model is executed through EDA tool to check either the script will run or give errors. If it runs successfully, it means the model has worked correctly and has generated results as required. In manual evaluation, humans act as judges to check whether the model has understood the user's problem and whether the code it provided meets the user's requirements. To make this process fair and bias free, judges are not told about which model has generated which result (blind review). Three level grading system is used in this process: Grade A indicates model has worked correctly, broken down the tasks correctly and generated correct script. Grade B indicates the thinking pattern of model was correct but there were some mistakes found in script generation. Grade C indicates it generated errors in either of the tasks. SmartEDA-Bench is used with 50 different tasks and 3 categories to evaluate the models.

AutoMage2 performed best among all with A grade in 82% tasks. This shows that AutoMage2 not only writes code accurately, but also performs correct reasoning. GPT-4 achieved 62% Grade A, Claude2 46%, GPT-3.5 28%, Distil GPT-2 20.5%, EleutherAI_gpt-neo-1.3B 43.9%, BigScience BLOOM-560M 39.3%, Local_Benchmark_BLOOM-560M 33.9%, as shown in the benchmark graph below:

These results clearly prove that performance of AutoMage2 is way better and reliable than general-purpose LLMs. Please consider the following table for better understanding:

D. Results and Discussions

Nine cases were analyzed where users have worked with EDA through the conversational interface. In quantitative evaluation, AutoMage2 solved 82% of tasks in Grade A, 11–12% of tasks in Grade B, and 6.7% of tasks in Grade C. In comparison, GPT-4 achieved 62% Grade A, 16% Grade B, and 22% Grade C, Claude2 46% Grade A, and GPT-3.5 only 28% in Grade A.

AutoMage2 completed 41/50 tasks successfully, whereas GPT-4 could solve only 31/50 tasks. The overall results give a clear ranking by categories. In simple flow calls, AutoMage2 generated consistently executable scripts, while ChatGPT-4 missed important steps that resulted in incomplete scripts. In complex flow calls, AutoMage2 showed stronger logical reasoning, correct sequence of operations, and correct interpretation of API arguments, whereas GPT-4 made more mistakes in parameter dependencies. In parameter flow calls, AutoMage2 efficiently optimized parameters, whereas GPT-4 struggled.

Error analysis indicates that Grade B cases correspond to small syntax mistakes and Grade C cases arise when completely new constraints are provided. These results demonstrate that domain-specific fine-tuning is essential for EDA applications. SmartEDA, based on AutoMage2, reduces manual efforts, errors, and improves automation. General-purpose LLMs like GPT-4 failed multiple times in managing optimization and task dependencies.

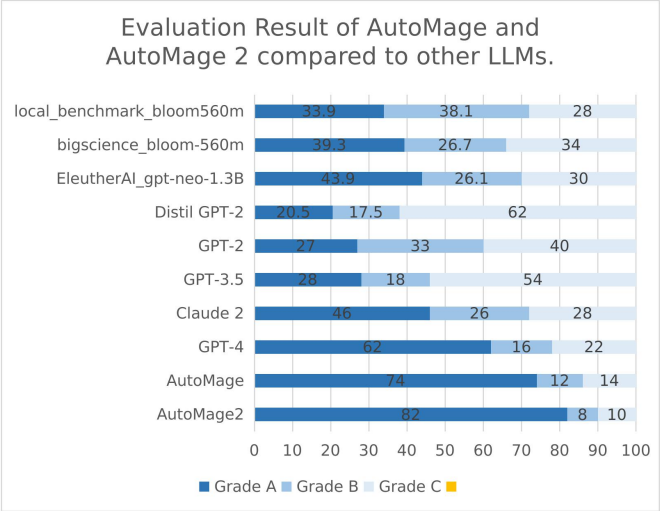

Fig. 3. Low Rank Adaption (LoRA) Process

TABLE IV
BENCHMARK RESULTS OF AUTOMAGE2 AND OTHER LLMS

Model	% Grade A	% Grade B	% Grade C
AutoMage2	82%	~11–12%	~6.7%
GPT-4	62%	16%	22%
Claude 2	46%	-	-
GPT-3.5	28%	-	-
GPT-2	27%	-	-
Distil GPT-2	20.5%	-	-
EleutherAI_gpt-neo-1.3B	43.9%	-	-
BigScience BLOOM-560M	39.3%	-	-
Local_Benchmark_BLOOM-560M	33.9%	-	-

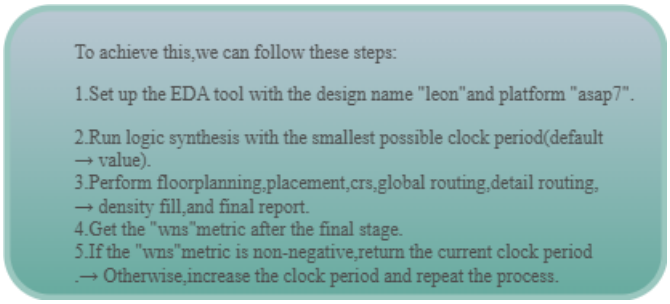

Fig. 4. Task Decomposition Process by AutoMage2

E. Implications and Future Work

Comparing results with existing research, AutoMage2 achieves domain-specific mastery with 82% success rate. Previous works like Toolformer and GPT-4 were limited to proof-of-concept. AutoMage2 outperformed general LLMs in parameter tuning and complex reasoning. Limitations include small dataset size, potential overfitting, and some unmeasured parameters such as GPU memory usage and latency. Future work directions include developing standardized large EDA datasets, testing SmartEDA on other EDA tools, deploying AutoMage2 in industrial projects to improve time-to-market and design quality, and extending AutoMage2 for multi-agent frameworks.

REFERENCES

[1] P. Chen, D. A. Kirkpatrick, and K. Keutzer, "Scripting for eda tools: A case study," in *Proc. IEEE 2nd Int. Symp. Qual. Electron. Des.*, 2001, pp. 87–93.
[2] T. Brown *et al.*, "Language models are few-shot learners," pp. 1–25, 2020.
[3] T. Schick *et al.*, "Toolformer: Language models can teach themselves to use tools," 2023.
[4] T. Ajayi *et al.*, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," in *Proc. Gov. Microcircuit Appl. Crit. Technol. Conf.*, 2019, pp. 1–6.
[5] X. Li *et al.*, "ieda: An open-source intelligent physical implementation toolkit and library," in *Proc. Int. Symp. Electron. Design Autom. (ISEDA)*, 2023, pp. 1–3.
[6] "Babyagi," https://github.com/yoheinakajima/babyagi, 2023.
[7] E. J. Hu *et al.*, "Lora: Low-rank adaptation of large language models," in *Proc. ICLR*, 2021, pp. 1–26.
[8] E. Frantar *et al.*, "Gptq: Accurate post-training quantization for generative transformers," *arXiv preprint arXiv:2210.17323*, 2022.
[9] H. Wu *et al.*, "Chateda: A large language model powered autonomous agent for eda," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 10, pp. 3184–3197, 2024.
[10] OpenAI, "Gpt-4 technical report," https://cdn.openai.com/papers/gpt-4.pdf, 2023.
[11] Anthropic, "Claude2," https://www.anthropic.com/, 2023.
[12] H. Touvron *et al.*, "Llama: Open and efficient foundation language models," 2023.
[13] ——, "Llama 2: Open foundation and fine-tuned chat models," 2023.
[14] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Proc. NIPS*, 2019, pp. 1–11.
[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "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 (NAACL)*, 2019, pp. 1–16.
[16] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
[17] Y. Tay, M. Dehghani, J. Gupta, V. Q. Tran, D. Bahri, T. Schuster, W. Fedus, H. W. Chung, S. Narang, D. Yogatama, A. Vaswani, and D. Metzler, "Ul2: Unifying language learning paradigms," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022, pp. 1–39.
[18] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf, 2018, preprint.
[19] A. Aghajanyan, S. Gupta, and L. Zettlemoyer, "Intrinsic dimensionality explains the effectiveness of language model fine-tuning," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021, pp. 1–11.
[20] S. Min, M. Lewis, L. Zettlemoyer, and H. Hajishirzi, "Metaicl: Learning to learn in context," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022, pp. 1–19.
[21] S. M. Xie, A. Raghunathan, P. Liang, and T. Ma, "An explanation of in-context learning as implicit bayesian inference," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021, pp. 1–25.
[22] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.
[23] B. Roziere and et al., "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.
[24] J. Wei *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," in *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS)*, 2022, pp. 1–14.
[25] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, "8-bit optimizers via block-wise quantization," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021, pp. 1–20.