EPiC Series in Computing

Volume 98, 2024, Pages 189–199

Proceedings of 39th International Conference on Computers and Their Applications

A Case Study on the Generative AI Project Life Cycle Using Large Language Models

Ajay Bandi and Hemanth Kagitha

School of Computer Science and Information Systems
Northwest Missouri State University
Maryville, Missouri, USA
ajay@nwmissouri.edu, hemanthkagithaa@gmail.com

Abstract

Large Language Models represent a disruptive technology set to revolutionize the future of artificial intelligence. While numerous literature reviews and survey articles discuss their benefits and address security and compliance concerns, there remains a shortage of research exploring the implementation life cycle of generative AI systems. This paper addresses this gap by presenting the various phases of the generative AI life cycle and detailing the development of a chatbot designed to address inquiries from prospective students. Utilizing Google Flan LLM and a question-answering pipeline, we processed user prompts. In addition, we compiled an input file containing domain knowledge of the education program, which was preprocessed and condensed into vector embeddings using the HuggingFace library. Furthermore, we designed a chat interface for user interaction using Streamlit. The responses generated by the chatbot are both descriptive and contextually pertinent to the prompts, with their quality improving in response to more detailed prompts. However, a significant constraint is the size limit of the input file, given the processing power limitations of CPUs.

1 Introduction

LLM stands for Large Language Model, an artificial intelligence model capable of understanding and generating human-like text. These models are trained on vast amounts of text data and can perform tasks such as text generation, text summarization, machine translation, and question-answering [32]. LLMs have various applications across industries including, natural language processing (NLP), chatbots, content generation, and sentiment analysis [18]. They are crucial in advancing AI technology and revolutionizing human-computer interactions [11]. LLMs utilize a transformer architecture model, serving as a foundation for generative AI applications, along with generative adversarial networks (GANs) [26] and variational autoencoders (VAEs) [30].

Several researchers focus on reviewing the literature on generative AI and LLM technology in various application domains. Some of the application areas are dialog conversation in healthcare [27], detection of marine litter [20], drug detection [8], and in business and finance [12]. Meskó and Topol [25] presented potential risks in using LLMs in healthcare, categorizing them into three categories, and discussed the regulatory challenges of using LLMs. These risks include

data privacy, intellectual property rights, lack of informed consent, and bias, among several others [34]. However, there is a gap in the literature regarding the finding of evidence supporting the implementation of generative AI systems or their usage as case studies in research [9]. This paper focuses on implementing a generative AI chatbot for an educational program, that utilizes LLM to generate new responses for prospective graduate students.

The remainder of the paper is organized as follows. Section 2 describes the hardware, software, and user experience requirements of generative AI systems. Section 3 discusses the different phases of the generative AI project life cycle. Section 4 presents a case study of chatbot implementation using LLM and its results. Section 5 presents research conclusions.

2 Requirements of Generative AI systems

Understanding the requirements of implementing a generative AI system is essential. This section discusses the three types of requirements: hardware, software, and user experience [5].

LLMs have significant hardware requirements due to their computational intensity [33]. They typically necessitate high-performance CPUs with multiple cores and high clock speeds to efficiently process large text datasets. Graphics Processing Units (GPUs) are commonly employed to accelerate training and inference tasks by parallelizing computations [29]. Alternatively, Tensor Processing Units (TPUs), known for their exceptional speed and energy efficiency, are increasingly utilized, especially in large-scale deep learning applications. Sufficient memory, comprising both RAM and storage space, is essential for storing model parameters, input data, and intermediate results during computations, while fast storage solutions such as SSDs or NVMe drives play a critical role in housing large datasets, model checkpoints, and training logs, thereby reducing data loading times and improving overall training throughput. Also, high-speed networking capabilities may be necessary for accessing and transferring large datasets stored on remote servers or cloud platforms. Overall, meeting the hardware requirements of LLMs often involves investing in robust computing infrastructure or leveraging cloud-based solutions to ensure efficient training and deployment processes [16].

LLMs require a specific set of software tools and frameworks to facilitate their development, training, and deployment [3]. Key components include deep learning frameworks like TensorFlow, PyTorch, or Hugging Face's Transformers library, which provide the foundational infrastructure for building and training neural network models. Natural Language Processing (NLP) libraries such as NLTK, spaCy, and Hugging Face's Transformers offer specialized tools for text processing and analysis, essential for tasks like tokenization and feature extraction [10]. Hardware acceleration libraries like CUDA and software integrations for GPUs or TPUs enable efficient computation during training and inference [15]. In addition, access to model repositories and pre-trained models, along with development environments like PyCharm Jupyter Notebooks, or Google Colabs streamline the development process [24]. Deployment frameworks such as TensorFlow Serving and cloud computing platforms like Amazon Web Services (AWS) or Google Cloud Platform provide the infrastructure needed to deploy and manage LLMs in production environments. These platforms provide scalable and reliable [31] solutions for hosting LLMs, allowing organizations to handle increased workloads and ensure consistent performance seamlessly.

Generative AI systems require a user experience that prioritizes intuitive interaction, clear communication, and customization options. Users should find the interface user-friendly and easy to navigate, with clear communication about the system's capabilities and limitations [23]. Feedback mechanisms should indicate when the system is processing input and generating output, while robust error handling ensures a smooth experience [13]. Customization options allow

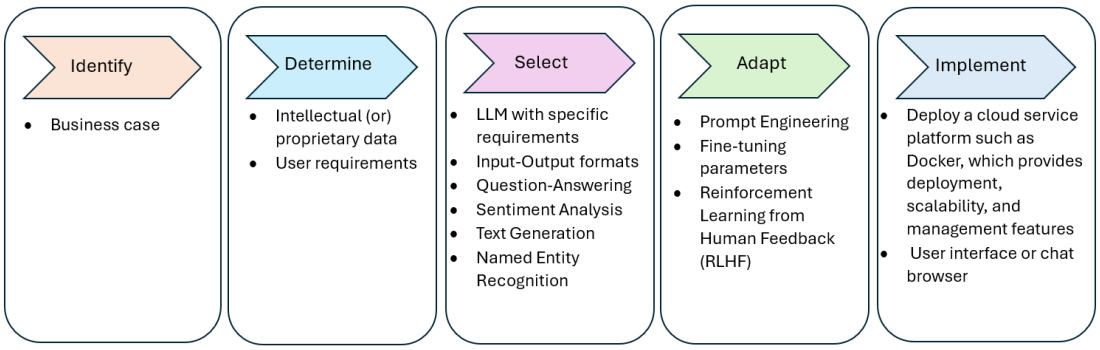

Figure 1: Phases of generative AI project life cycle

users to tailor the output to their preferences, while transparency about the system's operation and data usage fosters trust. Privacy and security are paramount, with clear communication and control over user data [2, 6]. Guidance and assistance should be available, especially for users unfamiliar with generative AI technology, and consistency across interactions and platforms ensures a cohesive experience. Accessibility considerations ensure the system is usable by all users, while performance optimization minimizes latency. Ethical considerations, including addressing biases and ethical implications, are also essential for responsible system design [28].

3 Phases of Generative AI Project Life Cycle

The details and enhancements of the software development life cycle and data science or data analytics life cycle projects are already known. This paper focuses on the generative AI (AIGC) project life cycle. The different phases in this life cycle are built on the work authored by David Baum from Snowflake [7]. Figure 1 shows the different phases. The first phase of AIGC is specifically on identifying the business use case and defining the scope. In this phase, the task is to determine the type of content generation required, such as customized descriptions of costumes, translation between languages, text summarization, synthetic data generation, music creation from lyrics, or providing rapid responses by generating answers for customers.

The second phase involves determining the intellectual data necessary for effectively customizing the model. LLMs are pre-trained on massive amounts of data sourced from various existing repositories, including websites, research articles, and source code repositories [18]. Typically, this dataset spans petabytes in size, encompassing domain knowledge. A clear understanding of the business use case is crucial for defining the data and user requirements.

The third phase involves selecting an appropriate LLM from the available options. Numerous open-source LLMs exist, including Bloom, Bret, Falcon, X Gen-7B, LLAMA, GPT-NeoX, and GPT-J, among others. Projects like LLAMA offer readily accessible resources that can be modified and deployed within our environments [1]. These LLMs boast billions of parameters, enhancing their ability to produce precise and contextually relevant outcomes. However, their extensive parameterization demands significant training resources to tailor them to our specific requirements. The decision between employing large or small models entails striking a balance between cost and performance. Developers must carefully evaluate their needs and resources to determine the most suitable LLM for their particular use case.

The fourth phase is adapting LLMs to the use case. To perform this prompt engineer-

ing, fine-tuning parameters, and human reinforcement learning from human feedback (RLHF) techniques are used to meet specific needs [4]. Prompt Engineering is the process of designing effective prompts or inputs for language models to generate desired outputs, particularly for those based on the Transformer architecture like GPT (Generative Pre-trained Transformer) models [19]. Language models like GPT are trained to generate text based on the input they receive, making the quality and specificity of the input. Fine-tuning allows for the adaptation of an LLM to specific tasks by updating its pre-trained parameters, thereby offering users the flexibility to customize LLMs according to their needs and achieve improved performance across various applications. This process involves selecting a relevant pre-trained LLM, refining it with related datasets, and training the model to generate responses tailored to specific prompts. The fine-tuned LLM is then evaluated to ensure it meets the desired requirements, with adjustments to parameters like learning rate and batch size made as necessary to optimize outcomes. Reinforcement Learning from Human Feedback (RLHF) is a technique used to refine and enhance the performance of artificial intelligence systems, particularly in the domain of natural language processing [4]. RLHF involves training models, such as chatbots, to engage in more natural and contextually relevant conversations by incorporating direct feedback from human interactions. This approach aims to improve the model's understanding of human prompts, refine its ability to generate responses that align with user preferences, and mitigate the risk of generating inappropriate or harmful content. RLHF holds significant potential across various sectors, facilitating the development of personalized assistants for businesses, customized learning plans for educational purposes, individualized treatment strategies in healthcare, and tailored recommendations in entertainment. Furthermore, RLHF serves to enhance model performance while also addressing concerns related to the adoption of internet-trained models, including the propagation of undesirable language patterns [4].

The final phase is the implementation of the app by deploying it into containers. DevOps teams often utilize containerization software like Docker to streamline the deployment of LLM applications, ensuring consistency across diverse computing environments [17]. Despite the benefits containers offer for sophisticated AI models with specialized processing needs and access to large datasets, the complexity of managing containerized workloads at scale can divert valuable time and resources from application development. A viable solution involves adopting a cloud data platform that simplifies the deployment, management, and scalability of LLMs and other containerized workloads within a fully managed infrastructure. This approach allows teams to execute LLM jobs in a governed environment, leverage configurable hardware options such as GPUs, and access a scalable pool of compute resources without the burden of infrastructure management [21]. In addition, integrating with third-party providers via marketplace apps further enhances flexibility and accessibility for developers and data scientists, enabling them to focus on solving business challenges rather than managing compute and storage resources.

4 Case Study

This section explains the phases of the generative AI project life cycle for the implemented chatbot. The detailed steps are given below.

1. Identify the business use case: Develop a chatbot for a higher education graduate program to answer questions from students all over the world, with a majority of the student population being from India. The chatbot needs to generate new and contextually relevant answers for the users. The chatbot also needs to understand the cultural vocabulary of the users.

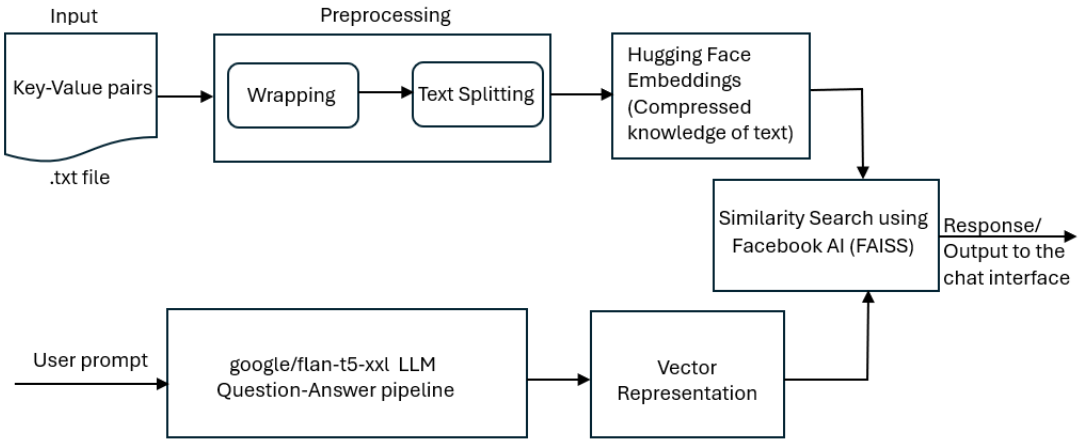

Figure 2: Architecture for implementation of chatbot using LLM

2. Determine the intellectual or proprietary data: The data for the graduate program is obtained from the program coordinator and the website, and it has been saved in a .txt file using key-value pairs. Each piece of information, such as program details, course offerings, faculty information, and application deadlines, is stored with a corresponding key that uniquely identifies it. Information from the .txt file is then extracted and compressed into a numerical representation within a continuous vector space. This numerical representation of words in vector format is referred to as embeddings. The purpose of embeddings is to capture semantic and syntactic relationships between words in the .txt file.

3. Selecting an LLM: LLMs are pre-trained with vast data corpora. In this case study, FLAN-T5 XXL by Google was chosen [14]. This model specializes in Text-to-Text Generation across three different languages and is built using transformers, PyTorch, and TensorFlow frameworks. With a size of 11.3 billion parameters, it offers significant computational power. The datasets used to train this LLM are gsm8k, lambada, aqua_rat, esnli, quasc, qrec, djaym7/wiki_dialog, and qed. The rationale behind choosing this LLM lies in its suitability for question-answering pipelines in chatbots and its prowess in text-to-text generation tasks. To implement the chatbot, Google Colab was utilized to write the source code in Python and import the necessary libraries. Further details on the chatbot implementation are provided in the final phase.

4. Adapting an LLM for use: Adapting the question-answering pipeline of the LLM to develop a chatbot through fine-tuning and prompt engineering. Fine-tuning the LLM involves collecting user feedback, and iteratively collecting and integrating user feedback to enhance the model's ability to generate accurate and contextually relevant responses.

5. Implementing the app by deploying it to a cloud service platform: Implementing the chatbot app and utilizing Streamlit for the chat interface. Utilizing Streamlit, we aim to design a user-friendly chat interface modified to meet diverse user needs and preferences. Our interface will seamlessly accommodate a wide range of user inputs and responses, by providing a realistic and engaging interaction experience.

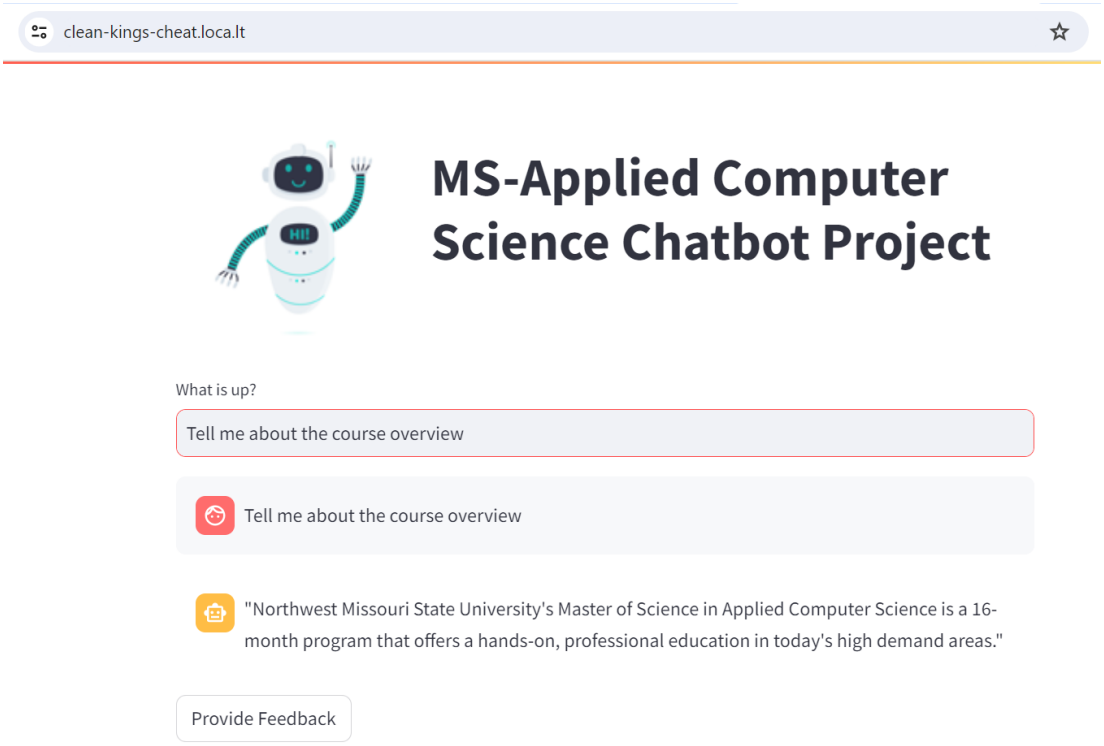

Figure 3: User interface of chatbot

4.1 Implementation of chatbot using LLM

This section presents the detailed steps to implement a chatbot. The source code of the chatbot is given in our GitHub repository[1]. The high-level steps are shown in Figure 2. The screenshot of the chat interface and the sample prompt and its response is shown in Figure 3.

LangChain is a framework for developing applications powered by LLMs. It enables applications that connect a language model to sources of context. This integrates various components such as embedding, vector databases, LLMs, etc. Using these components, we can provide external documents to the significant language models and build AI applications such as chatbots seamlessly. Faiss-cpu, Facebook AI Similarity Search, is a Python library, used for efficient similarity search, embeddings of multimedia documents that are similar to each other, and clustering of dense vectors. The sentence-transformers package enables the utilization of pre-trained models for sentence embeddings in Python projects. Sentence embeddings are numerical representations of sentences that capture their semantic meanings. These embeddings can be useful for NLP tasks, such as semantic similarity, clustering, classification, and information retrieval.

The `TextLoader` class is used to import the input .txt file. This file consists of the data in the form of key and value pairs. These key-value pairs are the information about the graduate program. The key is the sample prompt and the value is the relevant response to the prompt. The `textwrap` module is used for wrapping and formatting of plain text in documents. The `os` module is used for interacting with the operating system to access and manipu-

[1]https://github.com/bandiajay/GenAIChatbot

Table 1: Sample prompts and responses from chatbot

Input prompt	**Chatbot Response**	**Relevant response**
1. tell me about the course overview	Northwest Missouri State University's Master of Science in Applied Computer Science is a 16-month program that offers a hands-on, professional education in today's high demand areas.	Yes
2. How many credit hours are required to graduate from ACS Course?	Thirty credit hours of graduate credits are required for completion of the M.S. in Applied Computer Science.	Yes
3. Provide the list of all courses that were taught by Dr. Ajay Bandi	Generative AI, Mobile computing iOS, Patterns and frameworks, Advance Databases, Java, Big Data	Yes
4. What are the documents required for getting admission in masters in ACS course?	Official English proficiency scores (TOEFL, IELTS, PTE, DUOLINGO) Bank statement and Statement of support. Copy of your passport Official transcripts from all colleges and universities you have attended or are now attending for undergraduate and graduate coursework	Yes
5. Can you tell all the available scholarships for ACS students?	Any student who gets admitted to our program will receive the $500 scholarship for the first semester. In any semester, if you get a full graduate assistantship or 3/4th graduate assistantship, or 1/2 graduate assistantship, your 100% or 75% or 50% of the tuition fee will be waived off, respectively.	Yes
6. What is the tuition fee for a student in MS ACS course?	Total Graduate Cost for ACS is $16,802.20	Yes
7. What is the Last date to apply for spring semester ?	November 1	Yes

late (read, write, open) the input file. The text is then split into a smaller chunk of data, using a `CharacterTextSplitter` object, which is provided by `langchain.text_splitter` module. The size of each chunk is set to 100 characters, with no overlap between consecutive text chunks. This is to control how the text will be split into smaller units. Then the langchain framework that integrates with popular embedding models, such as "HuggingFaceEmbeddings" allows to generate embeddings for text document. These embeddings capture semantic meaning and contextual information, enabling more advanced NLP tasks such as similarity search.

The LLM used in this case study is `google/flan-t5-xxl` [14, 22], and it is adapted for the question-answering pipeline by importing the `load_qa_chain` function from the module called `langchain.chains.question_answering`. The `HuggingFaceHub` class from the LangChain module is used to obtain access to the Flan LLM without the need for explicit downloading.

The user prompt is then converted into vector embeddings using the LLM. The randomness and diversity of the output has been adjusted by setting the temperature parameter to 0.8, and the maximum length parameter is set to 512, which defines the maximum length of the input prompt that the LLM processes. Finally, a similarity search is performed on both the embedding spaces of the input file and the user prompt. A similarity metric is used to identify the maximum similarity score between both vectors. Once similar vectors are identified, the output can be generated based on the application context. An interactive web application is developed using Streamlit library for the user to send a prompt and receive a response. A feedback form is also provided for the user in the chat application.

4.2 Results

Figure 3 shows the user interface of the chatbot, where the user can enter prompts and receive responses. Table 1 displays sample results of the Applied Computer Science (ACS) chatbot's prompts and responses. The chatbot's responses are descriptive and contextually relevant to the prompts. However, for prompt #5, the expected answer is 'Yes', but the response provides detailed information about all available scholarships for students. The LLM is adapting to the business case by adjusting the prompts used to elicit responses. Through experimentation, it has been observed that providing more refined or optimized prompts leads to improved outcomes. As a result, the LLM's performance is enhanced as it learns to generate more accurate and relevant responses tailored to the specific requirements of the business case. This iterative process of refining prompts allows the LLM to better understand the context leading to better results.

The chatbot responses exhibit a high degree of relevance and accuracy across all input prompts. For instance, in response to Prompt #1 regarding the course overview, the chatbot provides a concise yet informative overview of the Master of Science program in ACS, demonstrating both relevance and accuracy in addressing the prompt. Similarly, for Prompt #2 concerning the required credit hours for graduation, the chatbot response accurately states the requisite thirty credit hours, ensuring both relevance and accuracy. Moreover, in response to Prompt #3 querying about courses taught by Dr. Ajay Bandi, the chatbot lists relevant courses, aligning closely with the prompt's requirements and displaying accuracy in its response. Additionally, for Prompt #4 regarding admission documents, the chatbot furnishes a comprehensive list of required documents, effectively addressing the prompt with both relevance and accuracy. Furthermore, in response to Prompt #5 regarding available scholarships, the chatbot provides relevant information about tuition fee waivers based on assistantships, demonstrating both relevance and accuracy. Similarly, for Prompt #6 concerning tuition fees, the chatbot response accurately provides the total graduate cost for the ACS program, ensuring both relevance and accuracy. Finally, for Prompt #7 regarding the application deadline, the chatbot specifies the last date to apply for the spring semester, exhibiting both pertinence and accuracy in its response. Overall, the chatbot's responses consistently align with the respective prompts, displaying both relevance and accuracy in addressing various inquiries related to the Master of Science program in ACS.

5 Conclusion

In conclusion, this paper highlights the transformative impact of LLMs on reshaping the landscape of artificial intelligence. Through this study, we have presented the various stages of the generative AI life cycle and illustrated the development of a chatbot tailored to address

inquiries from prospective students. Leveraging tools such as Google Flan LLM and the HuggingFace library, we have successfully processed user prompts and extracted domain knowledge into compact vector embeddings. A similarity check between the input file embedding and the prompt embeddings is performed to generate responses for the user. Additionally, the integration of Streamlit has facilitated seamless user interaction through the chat interface. Our findings demonstrate that the responses generated by the chatbot are not only descriptive but also contextually relevant, with their efficacy improving in response to more detailed prompts. However, a significant challenge lies in the limitation imposed by the size constraint of input files, particularly considering the computational constraints of CPUs. In the future, addressing these constraints by utilizing GPUs and further refining the implementation process will be essential for unlocking the full potential of generative AI systems in real-world applications across various domains.

References

[1] Meysam Alizadeh, Maël Kubli, Zeynab Samei, Shirin Dehghani, Juan Diego Bermeo, Maria Korobeynikova, and Fabrizio Gilardi. Open-source large language models outperform crowd workers and approach chatgpt in text-annotation tasks. *arXiv preprint arXiv:2307.02179*, 2023.

[2] Danielle Allen and E Glen Weyl. The real dangers of generative ai. *Journal of Democracy*, 35(1):147–162, 2024.

[3] Chetan Arora, John Grundy, and Mohamed Abdelrazek. Advancing requirements engineering through generative ai: Assessing the role of llms. *arXiv preprint arXiv:2310.13976*, 2023.

[4] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

[5] Ajay Bandi, Pydi Venkata Satya Ramesh Adapa, and Yudu Eswar Vinay Pratap Kumar Kuchi. The power of generative ai: A review of requirements, models, input–output formats, evaluation metrics, and challenges. *Future Internet*, 15(8):260, 2023.

[6] Ajay Bandi, Abdelaziz Fellah, and Harish Bondalapati. Embedding security concepts in introductory programming courses. *Journal of Computing Sciences in Colleges*, 34(4):78–89, 2019.

[7] David Baum. *Generative AI and LLMs: Snowflake Special Edition*. John Wiley Sons, 2024.

[8] Ankan Bera, Rik Das, Sayantani Ghosh, Raktim Chakraborty, Indranil Mitra, and Prasun Nandy. Harnessing transformers for detecting adverse drug reaction and customized causality explanation using generative ai. In *2023 7th International Conference On Computing, Communication, Control And Automation (ICCUBEA)*, pages 1–6, 2023.

[9] Desirée Bill and Theodor Eriksson. Fine-tuning a llm using reinforcement learning from human feedback for a therapy chatbot application, 2023.

[10] Nghi DQ Bui, Hung Le, Yue Wang, Junnan Li, Akhilesh Deepak Gotmare, and Steven CH Hoi. Codetf: One-stop transformer library for state-of-the-art code llm. *arXiv preprint arXiv:2306.00029*, 2023.

[11] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 2023.

[12] Boyang Chen, Zongxiao Wu, and Ruoran Zhao. From fiction to fact: the growing role of generative ai in business and finance. *Journal of Chinese Economic and Business Studies*, 21(4):471–496, 2023.

[13] Wonchan Choi, Yan Zhang, and Besiki Stvilia. Exploring applications and user experience with generative ai tools: A content analysis of reddit posts on chatgpt. *Proceedings of the Association*

for Information Science and Technology, 60(1):543–546, 2023.

[14] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022.

[15] Pudi Dhilleswararao, Srinivas Boppu, M Sabarimalai Manikandan, and Linga Reddy Cenkeramaddi. Efficient hardware architectures for accelerating deep neural networks: Survey. *IEEE Access*, 2022.

[16] Quang Do, Dan Roth, Mark Sammons, Yuancheng Tu, and V Vydiswaran. Robust, light-weight approaches to compute lexical similarity. *Computer Science Research and Technical Reports, University of Illinois*, 9, 2009.

[17] Muhammad Usman Hadi, Rizwan Qureshi, Abbas Shah, Muhammad Irfan, Anas Zafar, Muhammad Bilal Shaikh, Naveed Akhtar, Jia Wu, Seyedali Mirjalili, et al. A survey on large language models: Applications, challenges, limitations, and practical usage. *Authorea Preprints*, 2023.

[18] Harald Hammarström and Wilco van den Heuvel. Introduction to the llm special issue 2012 on the history, contact and classification of papuan languages. *Language & Linguistics in Melanesia*, 2012(Special Issue, Part 1):i–v, 2012.

[19] Thomas F Heston and Charya Khun. Prompt engineering in medical education. *International Medical Education*, 2(3):198–205, 2023.

[20] Jungseok Hong, Michael Fulton, and Junaed Sattar. A generative approach towards improved robotic detection of marine litter. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 10525–10531. IEEE, 2020.

[21] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[22] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

[23] Jie Li, Hancheng Cao, Laura Lin, Youyang Hou, Ruihao Zhu, and Abdallah El Ali. User experience design professionals' perceptions of generative artificial intelligence. *arXiv preprint arXiv:2309.15237*, 2023.

[24] Andrew M McNutt, Chenglong Wang, Robert A Deline, and Steven M Drucker. On the design of ai-powered code assistants for notebooks. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2023.

[25] Bertalan Meskó and Eric J Topol. The imperative for regulatory oversight of large language models (or generative ai) in healthcare. *NPJ digital medicine*, 6(1):120, 2023.

[26] Nicolas Morizet. *Introduction to Generative Adversarial Networks*. PhD thesis, Advestis, 2020.

[27] Usman Naseem, Ajay Bandi, Shaina Raza, Junaid Rashid, and Bharathi Raja Chakravarthi. Incorporating medical knowledge to transformer-based language models for medical dialogue generation. In *Proceedings of the 21st Workshop on Biomedical Language Processing*, pages 110–115, 2022.

[28] Tea Osmëni and Maaruf Ali. Generative ai: Impactful considerations to responsible data practices in business execution. In *2023 International Conference on Computing, Networking, Telecommunications Engineering Sciences Applications (CoNTESA)*, pages 75–82, 2023.

[29] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Re, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. 2023.

[30] Aman Singh and Tokunbo Ogunfunmi. An overview of variational autoencoders for source separation, finance, and bio-signal applications. *Entropy*, 24(1):55, 2021.

[31] Nina Singh, Katharine Lawrence, Safiya Richardson, and Devin M Mann. Centering health equity in large language model deployment. *PLOS Digital Health*, 2(10):e0000367, 2023.

[32] Ruixiang Tang, Yu-Neng Chuang, and Xia Hu. The science of detecting llm-generated texts. *arXiv preprint arXiv:2303.07205*, 2023.

[33] Jin Wang, Zishan Huang, Hengli Liu, Nianyi Yang, and Yinhao Xiao. Defecthunter: A novel llm-driven boosted-conformer-based code vulnerability detection mechanism. *arXiv preprint arXiv:2309.15324*, 2023.

[34] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Eric Sun, and Yue Zhang. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *arXiv preprint arXiv:2312.02003*, 2023.