FBE SEMİNER YAZIMINA BAŞLAMADAN ÖNCE

1. Seminer yazım kuralları hakkında birinci kaynak Tez Yazım Kılavuzudur.
2. Bu şablon, Enstitü internet sayfasındaki "Şablon Açıklaması" dikkate alınarak hazırlandığında sadece şekil yönünden Tez Yazım Kılavuzundaki şartları sağlar. Bölüm içeriklerinin doğru hazırlanması seminer yazarının sorumluluğundadır.
3. Bu şablon MS Word Belge Şablonu olarak hazırlanmıştır. Sadece Enstitü internet sayfasından indireceğiniz şablonlara güveniniz. Başka kaynaklardan edindiğiniz şablonlara güvenmeyiniz. Bu konuda Enstitü sorumluluk kabul etmez.
4. Word dosyaları yazıcı ayarlarına bağlı olarak sayfa kaymalarına neden olabilmektedir. Bu yüzden, yazımı tamamlanan seminerin PDF formatında kaydedilmesi ve bu format üzerinde kontroller yapılıp PDF dosyasından baskısının alınması önerilir. Kağıt ortamındaki seminer kopyasının Tez Yazım Kurallarına uymak zorunda olduğu unutulmamalıdır.
5. Tüm paragraf işaretlerini (boşluklar ve tablar gibi) ve gizli biçimlendirme işaretlerini (sayfa sonu, bölüm sonu gibi) Word programında Giriş sekmesinde ¶ simgesine tıklayıp görebilir veya gizleyebilirsiniz. Bölüm sonu ve sayfa sonu ifadelerini gereksiz şekilde silmeyiniz. Aksi takdirde sayfa numaralandırma yapısı bozulacak ve yeniden yapılandırmak zorunda kalınacaktır.
6. Şablon içindeki tüm açıklama yazıları seminerinizin baskısından önce mutlaka silinmelidir.
7. Bir sonraki sayfada bulunan tabloyu eksiksiz doldurduğunuzda İlk Sayfalar ve Ön Bölümler hazırlanmış olacaktır. Tablo içindeki tüm alanlar klavyeden giriş yapılarak doldurulmalı, kesinlikle stil galerisinden stil değişimi veya başka bir alandan kopyala/yapıştır yapılmamalıdır.
8. Seminer kitapçığının sadece şekil açısından değil aynı zamanda bazı bölümlerinin içeriklerinin de Tez Yazım Kurallarına uymak zorunda olduğu unutulmamalıdır. Önsöz, Özet, Abstract, Listeler ve Giriş Bölümü içeriklerinin nasıl hazırlanması gerektiği konusunda Tez Yazım Kılavuzu ve Şablon Açıklaması mutlaka incelenmelidir.
9. Bu bilgilendirme sayfası ve bundan sonraki tablo sayfası seminer kitapçığında bulunmamalıdır.

Seminer yazımınızda kolaylıklar dileriz.

- **TABLOYU DOLDURURKEN** sadece klavyeden giriş yapınız, başka bir ortamdan kopyala/yapıştır yapmayınız. Ayrıca, stiller galerisinden farklı bir stil uygulamayınız.
- **TABLOYU DOLDURDUĞUNUZDA İLGİLİ TÜM ALANLARA YAZDIĞINIZ İFADE AKTARILACAKTIR.**
- Seminer başlığında zorunlu durumlarda italik, üst-alt simge, Yunan karakterleri oluşturulabilir. İngilizce kelime var ise sadece o kelime işaretlenip "Gözden Geçir" sekmesinden dil İngilizce yapılır. Küçük harf kullanılması gerekiyor ise Pencere Altı ve İç Kapakta seminer Başlığı üzerinde metin kutusu (kırmızı bilgi kutuları gibi) oluşturup gerekli düzeltmeler yapılabilir.
- Bilim Dalı yoksa, Pencere Alt Kapağında bulunan "Bilim Dalı" ifadesini bir boşluk vererek siliniz.
- Mendeley eklentisi ile çalışıldığında hata mesajı alırsanız "End" seçimi yaparak işlemlerinize devam ediniz.
- İkinci Danışman yoksa, Kapak Sayfasındaki "İkinci Danışman" satırını ve Onay Sayfasındaki İkinci Danışman satırlarını siliniz.

Seminer Yazarı:	Mohammad Amin ASLAMI	
	TÜRKÇE	**İNGİLİZCE**
	Seminer Başlığı, Enstitü tarafından kayıt altına alınacak ve transkriptinize işlenecektir.	
Tez Türü:	Yüksek Lisans Semineri	Master's Seminar
Seminer Başlığı:	Almayla Artırılmış Üretim ile Büyük Dil Modellerinin Geliştirilmesi: Teknikler, Zorluklar ve Gelecekteki Yönler	Enhancing Large Language Models with Retrieval-Augmented Generation: Techniques, Challenges, and Future Directions
Anabilim Dalı:	Yazılım Mühendisliği	Software Engineering
Bilim Dalı:	Yazılım Mühendisliği	Software Engineering
Sunum Tarihi:	18.01.2025	
Seminer Ön Sayfa Sayısı:	xxiv	Kısaltmalar Listesinin bulunduğu sayfa numarasıdır
Seminer Sayfa Sayısı:	53	Kaynaklar Bölümünün bitiş sayfa numarasıdır
		Danışman ONAYI
Danışman:	Assoc. Prof. Dr. Ferhat UÇAR	<Onayladım>
Danışman Kurumu:	Fırat University, Fen Fakültesi	

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ALMAYLA ARTIRILMIŞ ÜRETİM İLE BÜYÜK DİL MODELLERİNİN GELİŞTİRİLMESİ: TEKNİKLER, ZORLUKLAR VE GELECEKTEKİ YÖNLER

Mohammad Amin ASLAMI

Yüksek Lisans Semineri

YAZILIM MÜHENDİSLİĞİ ANABİLİM DALI

Yazılım Mühendisliği Bilim Dalı

OCAK 2025

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Yazılım Mühendisliği Anabilim Dalı

Yüksek Lisans Semineri

ENHANCING LARGE LANGUAGE MODELS WITH RETRIEVAL-AUGMENTED GENERATION: TECHNIQUES, CHALLENGES, AND FUTURE DIRECTIONS

Seminer Yazarı
Mohammad Amin ASLAMI

Danışman
Assoc. Prof. Dr. Ferhat UÇAR

OCAK 2025
ELAZIĞ

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Yazılım Mühendisliği Anabilim Dalı

Yüksek Lisans Semineri

Başlığı: Almayla Artırılmış Üretim ile Büyük Dil Modellerinin Geliştirilmesi: Teknikler, Zorluklar ve Gelecekteki Yönler

Yazarı: Mohammad Amin ASLAMI

Sunum Tarihi: 18.01.2025

SEMİNER ONAYI

Fırat Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına göre hazırlanan bu seminer aşağıda imzası bulunan danışman tarafından değerlendirilmiş ve akademik dinleyicilere açık yapılan sunum sonucunda kabul edilmiştir.

İmza

Danışman: Assoc. Prof. Dr. Ferhat UÇAR — Onayladım
Fırat University, Fen Fakültesi

BEYAN

Fırat Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırladığım "Almayla Artırılmış Üretim ile Büyük Dil Modellerinin Geliştirilmesi: Teknikler, Zorluklar ve Gelecekteki Yönler " Başlıklı Yüksek Lisans Seminerinın içindeki bütün bilgilerin doğru olduğunu, bilgilerin üretilmesi ve sunulmasında bilimsel etik kurallarına uygun davrandığımı, kullandığım bütün kaynakları atıf yaparak belirttiğimi, maddi ve manevi desteği olan tüm kurum/kuruluş ve kişileri belirttiğimi, burada sunduğum veri ve bilgileri unvan almak amacıyla daha önce hiçbir şekilde kullanmadığımı beyan ederim.

18.01.2025

Mohammad Amin ASLAMI

ÖNSÖZ

I would like to express my heartfelt gratitude to my valuable advisor Assoc. Prof. Dr. Ferhat UÇAR, who did not withhold his knowledge, experience and help from me despite his intensity during the seminar work I conducted. I also wish him continued success in his academic, professional and educational life.

Mohammad Amin ASLAMI
ELAZIĞ, 2025

CONTENTS

Sayfa

ÖZET

Almayla Artırılmış Üretim ile Büyük Dil Modellerinin Geliştirilmesi: Teknikler, Zorluklar ve Gelecekteki Yönler

Mohammad Amin ASLAMI

Yüksek Lisans Semineri

FIRAT ÜNİVERSİTESİ
Fen Bilimleri Enstitüsü

Yazılım Mühendisliği Anabilim Dalı

Ocak 2025, Sayfa: xxiv + 53

Gerçek zamanlı harici veri kaynaklarına erişim sağlayarak performanslarını artırmak için büyük dil modellerine (BDM'ler) Almayla-artırılmış üretim (AAÜ) teknikleri. Kapsanan temel alanlar arasında belge alma ve yerleştirme tabanlı arama, sinir ağları ve alma tabanlı yaklaşımları birleştiren hibrit modeller ve uzun biçimli üretim için bellekle artırılmış teknikler yer alır.

Zorluklar arasında gecikme, bilgi aşırı yüklenmesi ve alan-özel bilgi ve önyargıların ele alınmasının sağlanması yer alır. Gelecekteki yönler alma verimliliğini, gerçek zamanlı bilgi güncellemelerini ve kullanıcı merkezli özelleştirmeyi iyileştirmeyi içerir. AAÜ entegrasyonunun amacı, BDM 'lere daha geniş bilgi erişimi sağlamak ve çeşitli alanlarda kişiselleştirilmiş bilgi sunumu yapabilen daha çok yönlü ve doğru akıllı sistemlere yol açmaktır.

Anahtar Kelimeler: Almayla-artırılmış üretim (AAÜ), Büyük dil modelleri (BDM), Geliştirme, Üretken Yapay Zeka

ABSTRACT

Enhancing Large Language Models with Retrieval-Augmented Generation: Techniques, Challenges, and Future Directions

Mohammad Amin ASLAMI

Master's Seminar

FIRAT UNIVERSITY
Graduate School of Natural and Applied Sciences

Department of Software Engineering

January 2025, Pages: xxiv + 53

The retrieval-augmented generation (RAG) techniques into large language models (LLMs) to enhance their performance by allowing real-time access to external data sources. The key areas covered include document retrieval and embedding-based search, hybrid models combining neural networks and retrieval-based approaches, and memory-augmented techniques for long-form generation.

Challenges include latency, information overload, and ensuring domain-specific knowledge and biases are addressed. Future directions involve improving retrieval efficiency, real-time knowledge updates, and user-centric customization. The purpose of RAG integration is to provide LLMs with broader knowledge access, leading to more versatile and accurate intelligent systems capable of personalized information delivery across diverse fields.

Keywords: Retrieval-augmented generation (RAG), Large language models (LLMs), Enhancing, Generative artificial intelligence (GAI)

FIGURE LISTS

Page

TABLE LISTS

Page

RESTRICTIONS

AI :Artificial Intelligence
API :Application Programming Interface
AQA :Active Query Answering
CPU :Central Processing Unit
ETL :Extract, Transform, Load
FT :Fine-Tuning
GAI :Generative Artificial Intelligence
GB :Giga Byte
GPU :Graphics Processing Unit
RI :Information Retrieval
LLMs :Large Language Models
LSTM :Long-Short Time Memory
NLP :Natural Language Processing
NLG :Natural Language Generation
PB :Peta Byte
PE :Prompt Engineering
QLoRA :Quantized Low-Rank Adaptation
RAG :Retrieval-Augmented Generation
RL :Reinforcement Learning
ReFT :Reinforcement Fine-Tuning
RAM :Random Access Memory
SSFT :Self-supervised Fine-Tuning
SFT :Supervised Fine-Tuning
TB :Tera Byte

1. INTRODUCTION

Retrieval-Augmented Generation (RAG) is a powerful technique in Large Language Models (LLMs) that reduces errors or "hallucinations" by grounding responses in external data sources. By incorporating relevant context from external databases or knowledge bases, RAG significantly improves the cost-effectiveness of LLM operations. This is primarily due to the ease of updating retrieval indexes, which is far less resource-intensive than continuously fine-tuning pre-trained models. In addition, RAG provides streamlined access to current information, saving both time and money and enabling more efficient handling of current data in LLMs [1].

As an AI framework, RAG optimizes LLM output by leveraging authoritative sources outside of the model's training data. This approach combines the generative capabilities of LLMs with the strengths of traditional information retrieval systems, such as databases, to produce more accurate and contextually relevant answers. LLMs play a critical role in powering intelligent chatbots and other natural language processing (NLP) applications. However, despite their strengths, they face limitations such as reliance on static training data and occasional inaccuracies or unpredictable results. LLMs are also prone to outdated responses when dealing with information-intensive topics or topics that require deep, up-to-date knowledge, and they can introduce response bias due to the perspectives present in their training data. While LLMs are widely used in various domains, these limitations can hinder their effectiveness in retrieving and presenting accurate information. RAG addresses these challenges by guiding LLMs to relevant and verified information from external knowledge sources, thereby improving response accuracy and reliability. As the use of LLMs expands, the applications of RAG continue to grow, making it a critical component of today's advanced AI solutions [2].

2. ARCHITECUTRE OF RAG

Retrieval-Augmented Generation (RAG) extends Large Language Models (LLMs) by integrating external sources of information. The process involves taking external data, vectorizing it using embeddings, converting user queries into vectors, performing similarity searches to retrieve relevant information, and using LLM to produce answers that combine internal and external information. Key techniques for optimizing RAG include query rewriting, segmentation, leveraging external tools, ensuring data quality, and leveraging metadata. Together, these methods improve the accuracy, relevance, and contextuality of AI-generated answers. [2].

The concept of Retrieval Augmented Generation (RAG) models is based on the integration of two core components of NLP: information retrieval (IR) and natural language generation (NLG). The RAG framework, first introduced by Lewis et al, combines dense retrieval methods with large-scale generative models to produce responses that are both contextually relevant and factually accurate. By explicitly retrieving relevant passages from a large corpus and augmenting this information in the generation process, RAG models improve the factual grounding of their output with current knowledge. A generic workflow of a Retrieval Augmented Generation (RAG) system, showing how it fundamentally enhances the capabilities of Large Language Models (LLMs) by grounding their outputs in real-time relevant information, is illustrated in *Figure 2.1.* Unlike static models that generate responses based only on closed-world knowledge [21].

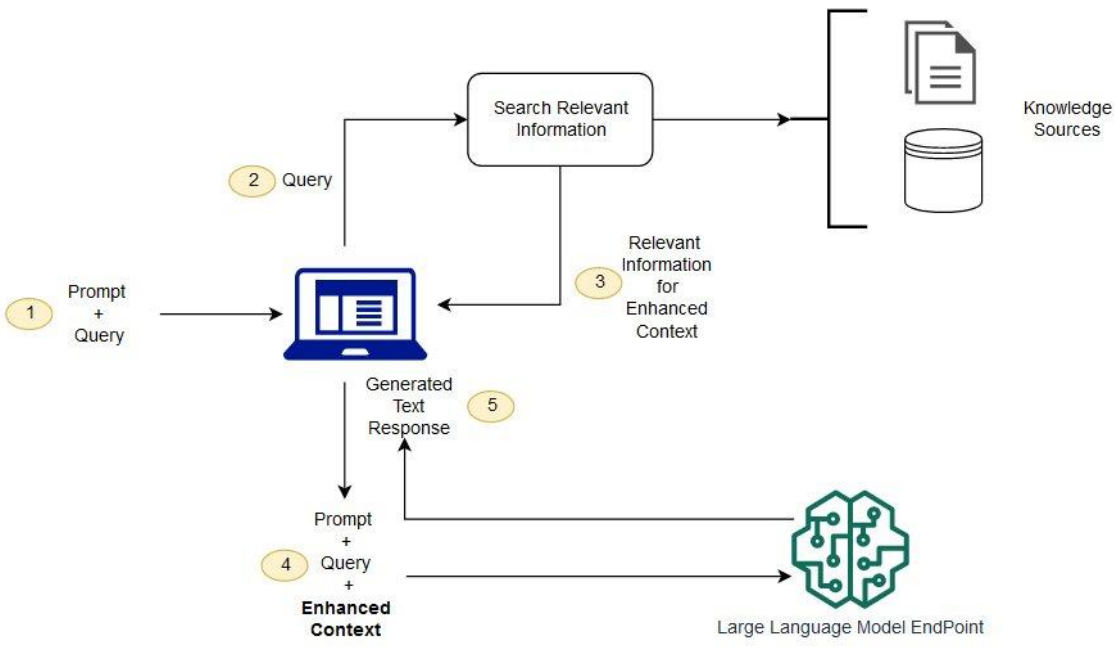


Figure 2.1. Conceptual flow of using RAG with LLMs [34]

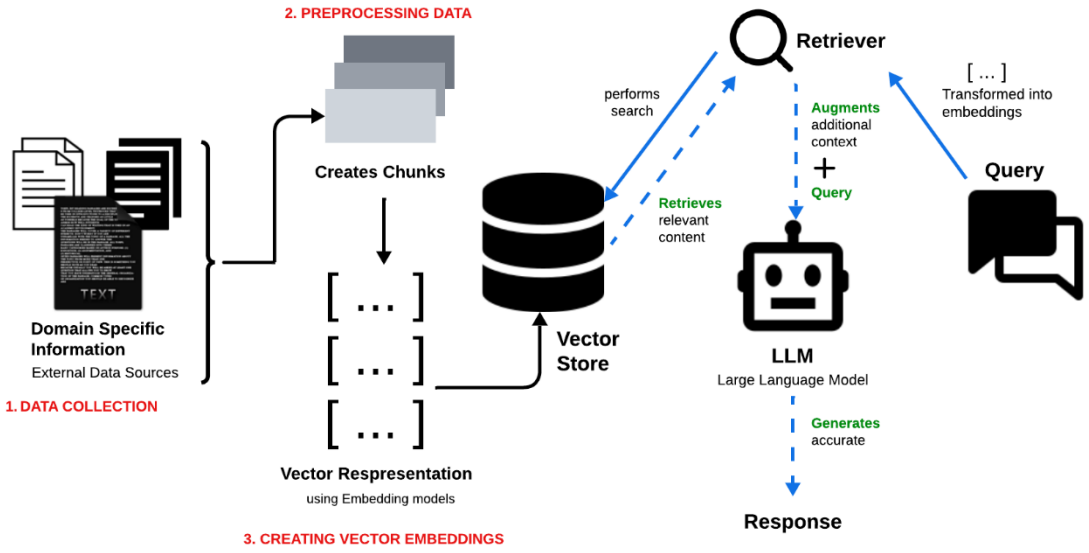


Figure 2.2. Architecture of Retrieval Augmented Generation (RAG) System [21]

2.1. Real-world Use Cases for RAG

Today, RAG programs are widely used in various fields. Some of their common applications include. RAG models improve query answering systems by obtaining accurate information from authoritative sources. One use case for RAG engines is information retrieval in healthcare organizations, where the engine can answer medical questions based on medical literature.

RAGs are very effective at simplifying content creation by generating relevant information. They are also valuable for creating concise summaries of information from multiple sources.

RAG applications also power chatbots, allowing chatbots and virtual assistants to provide accurate and relevant answers. This makes them ideal for use as customer service chatbots and virtual assistants that can provide detailed and informative answers during interactions.

RAG models are also used in knowledge-based search systems, educational tools, and legal research assistants. They can provide briefings, prepare study materials, help draft documents, analyze legal cases, and formulate arguments [2].

2.1.1. General Working Logic of RAG Model

A typical implementation of RAG is shown in *Figure 2.3.* Here, a user asks ChatGPT a question about a recent widely discussed news story. Because ChatGPT relies on pre-trained data, it initially lacks the ability to provide updates on recent developments. RAG fills this information gap by learning from and incorporating information from external databases. In this case, it collects relevant news articles related to the user's query. These articles, combined with the original question, form a comprehensive query that enables LLMs to provide a well-informed response [3].

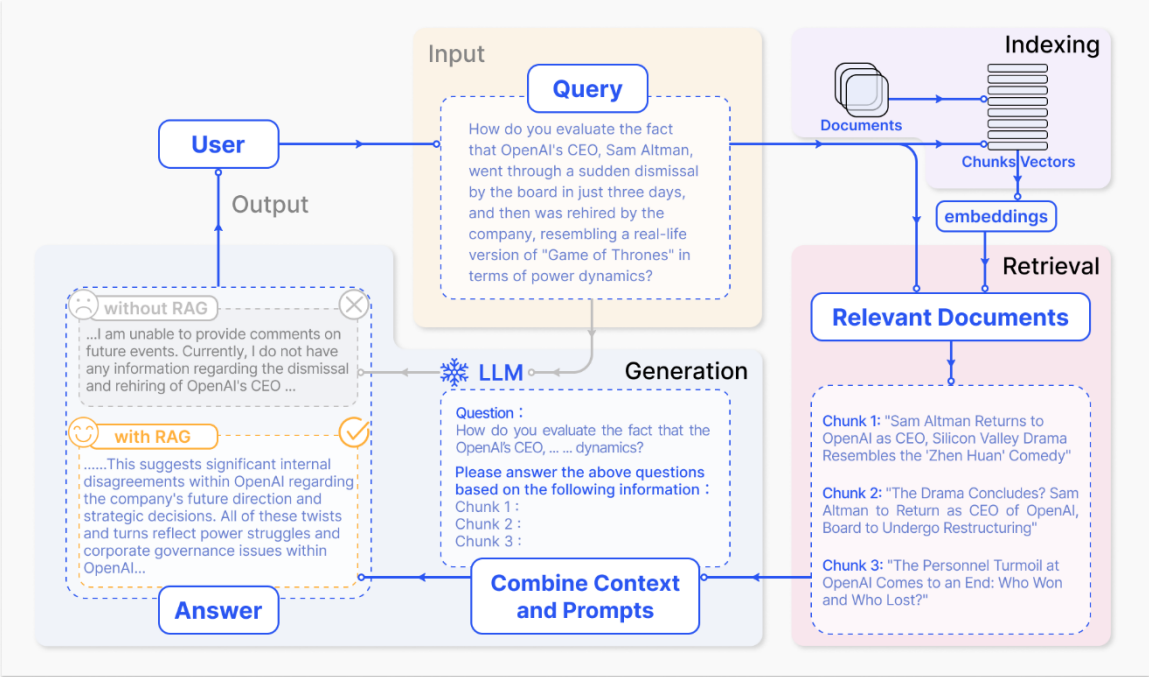


Figure 2.3. Overview of RAG [3]

Figure 2.3: A representative example of applying the RAG process to answering questions. It consists mainly of 3 steps. 1) Indexing. Documents are divided into chunks, encoded into vectors, and stored in a vector database. 2) Retrieval. Retrieve the top k chunks most relevant to the question based on semantic similarity. 3) Generation. Generate the final answer by inputting the original question and the retrieved chunks together into the LLM [3].

2.1.2. Imporving LLMs with RAG

2.1.3. LLMs Comperss World Knowledge

A fundamental LLMs feature of LLMs is the ability to compress world knowledge. The way this works is you take a huge slice of the world's knowledge through more books and doucments than anyone could ever read in their lifetime and you use it to train a LLMs and what happens in this training process is that all the knowledge and concepts theories and events that have happened in the that are represented in the text of the training data they get represented and stored in the models weights so essentially what has happend is we're comressed all that information into single language model well this has led to come of the *biggest AI innovations the world has ever seen there two key limitations* for compressing knowledge this way [19].

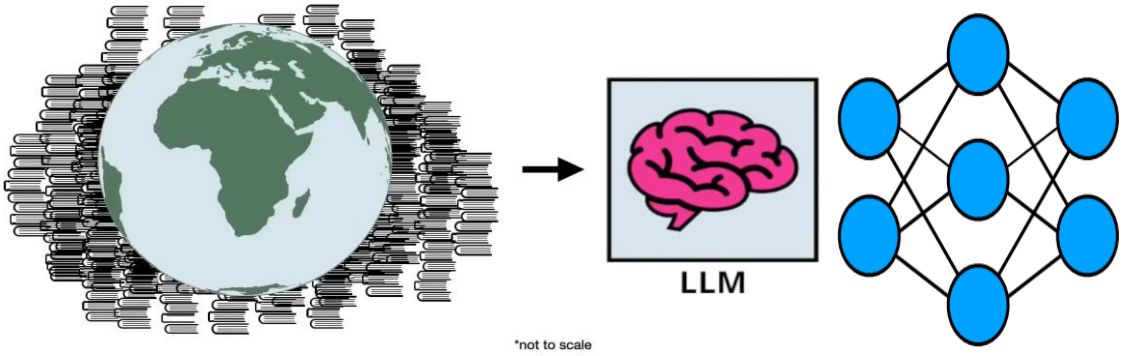


Figure 2.4. LLMs Compress World Knowledge [35]

2.1.4. LLMs 2 Key Limitations

2.1.4.1 1-Static World Knowledge

It means that it does not update as new events happen and new information becomes available, and for anyone who has used Chat GPT, tried to ask it a question about current events has probably seen a message like this: *As of my last update in January 2022, I do not have access to real-time information, so I can't provide specific events from February 2024* [19].

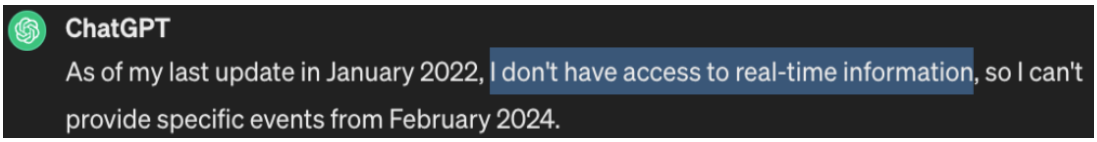


Figure 2.5. Static Knowledge [35]

2.1.4.2 2-Lack of Specialized Information

That these LLMs are trained on a massive *corpus(body)* of text the result of that is that they're really good at general knowledge but they tend to *fall short when comes to more Niche and specialized* information mainly because the wasn't very prominet in their training data so when Shaw TALIBI asked Chat GPT *how I old was* [19].

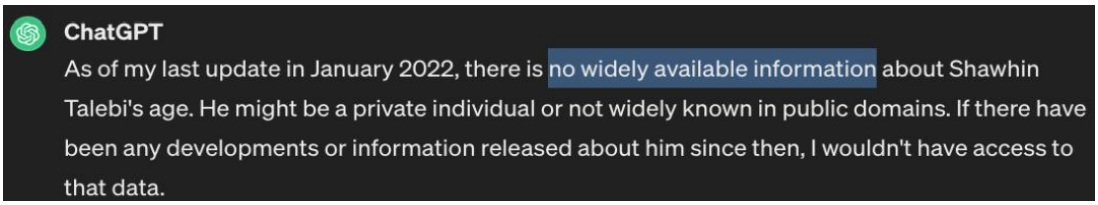


Figure 2.6. Lack of Specialized Information [35]

2.2. What is RAG?

One way we can mitigate both of these limitations is by using Retrieval-Augmented-Generation (RAG). Augmenting (complementing) LLM with specialized and mutable knowledge base. Basically, we can have a knowledge base that contains domain specific information that is updatable where we can add and remove information as needed. The typicalall way we will use *LLMs is we will pass it a prompt and it will split out a response,* the basic usage will rely on the internal knowledge of the model in generating the response based on the prompt..

If we want to add RAG into the mix it would look something like **Figure 2.8.** So instead of starting with a prompt we will start with say a user query which gets passed into RAG module and what the RAG module does is that it connects to a specialized knowledge base and it will grab pieces which are relevant to the user's query and create a prompt that we can pass into LLMs and so notice that we are not fundamentally changing how using the LLMs it's still prompt in and response out the only thing we are doing is augmenting this whole workflow this RAG module which instead of passing in a user query or prompt directly to the model we just have **Figure 2.8.** *pre-processing step to ensure* that the proper context and information is include in the prompt one question you are might have is why do we have to build out this RAG module can not fine-tune the LLMs using specialized knowledge, so that we can jsut use it the standard way and the answer to that question is yes so you can definitely fine-tune large language model (LLMs) specialized knowledge to teach it that information so to speak however empirically fine-tuning a model seems to be less effective way of giving it specialzied knowledge [19].

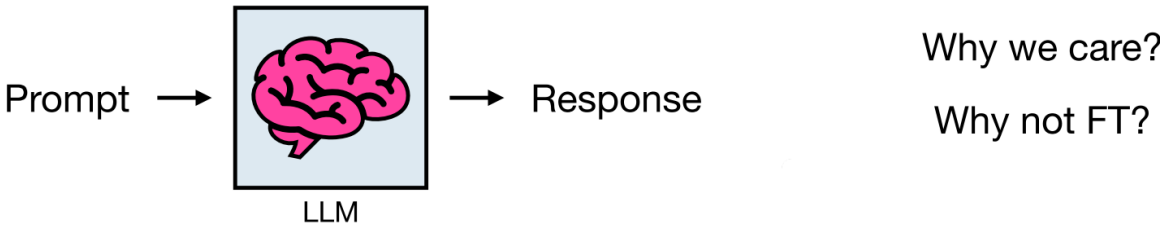


Figure 2.7. LLMs Structure [35]

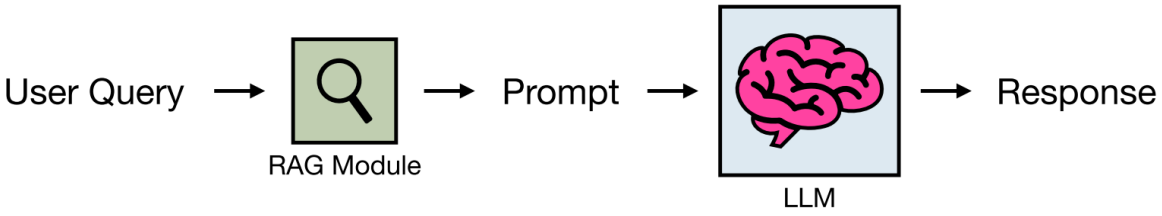


Figure 2.8. RAG and LLMs Togethers [35]

2.3. How it works (RAG)?

The RAG module actually works the RAG module consists of 2 key elements. retriever and knowledge base, 1-first is the **Retriever** and 2-second is the **Knowledge Base**, so the way these two things work together is that a user query comes in it gets passed to the retriever which takes the query and searches the knowledge base for relevant pieces of information it then extracts that relevant information and uses it to output a prompt the way this retrieval step typically works is using so-called text embeddings [19].

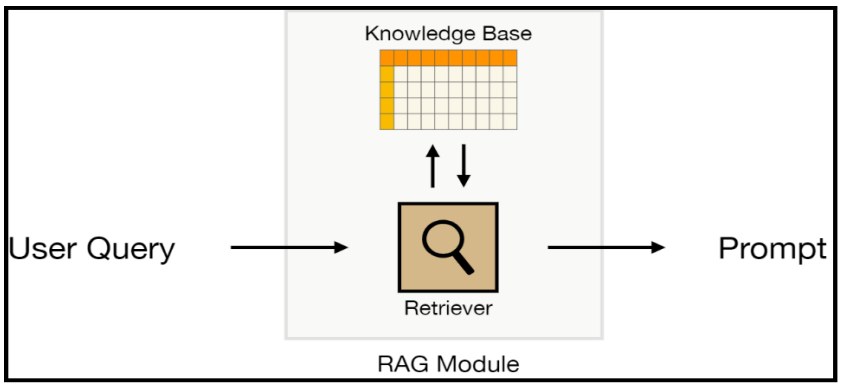


Figure 2.9. How RAG works [35]

2.4. Retriever on RAG Model

Text embeddings are numbers that represent the meaning of some given text, so let's say we have a collection of words like tree, lotus flower, daisy, the sun, saturn, jupyter, basketball, baseball, satellite, spaceship. Text embeddings are a set of numbers associated with each word and concept that we see here, but they are not just any set of numbers that actually capture the meaning of the underlying text, so if we were to plot them on an X axis. Y axis, similar concepts will be close together, While concepts that are very different from each other will be spaced far apart, we see that plants tend to be located close together, celestial bodies tend to be close together, these sports balls tend to be located close together, and things that you typically see in space tend to be located close together, and notice that the balls are closer to celestial bodies than to plants. The way we can use this for search is to say that each of these items is a piece of information in our knowledge base, you know, we have a description of this tree, lotus flower, Jupyter, and so forth. What we can do is represent each item in our knowledge base as a point in this embedding space and then represent the user's query as another point in this embedding space and then to do a search we just look at the items in the knowledge base closest to the query and return them as a search result that's all we're going to say about text embeddings and text embedding bases. Searching for now is actually a pretty rich and deep topic [19].

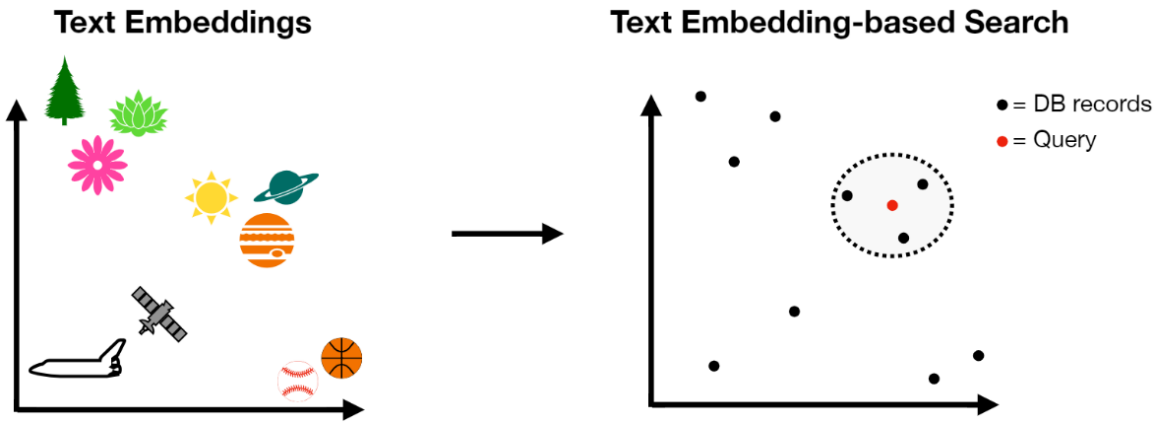


Figure 2.10. Text Embeddings + Retrieval [35]

2.5. Creating Knowledge Base

You have a stack of documents that you want to make available to LLMs. The process of taking these raw files and turning them into a knowledge base can be broken down into four steps. The first step is to load the documents. This consists of taking the collection of documents that you want to include in the knowledge base and converting them into a parsable format. The key here is to make sure that the critical information in your documents is in a text format, because at the end of the day, large language models (LLMs) only understand text. Any information you want to pass to them has to be in that format. Second, you want to chunk the documents. The reason that *step (2) is important is that LLMs have a fixed context window*, which means that you cannot dump all of your documents into the prompt and pass it to the LLMs. You have to break it up into smaller pieces. Even if you have a model with a large context window, chunking documents leads to better performance because you often don't need the whole document-just a sentence or two. Chunking ensures that only relevant information is passed to the prompt. The third step is to take each chunk and translate it into the text embeddings discussed in the previous topic. This translates a chunk of text into a vector or set of numbers representing its meaning. Finally, we take all these vectors and load them into a vector database for text embedding-based search, as shown in the previous topic [19].

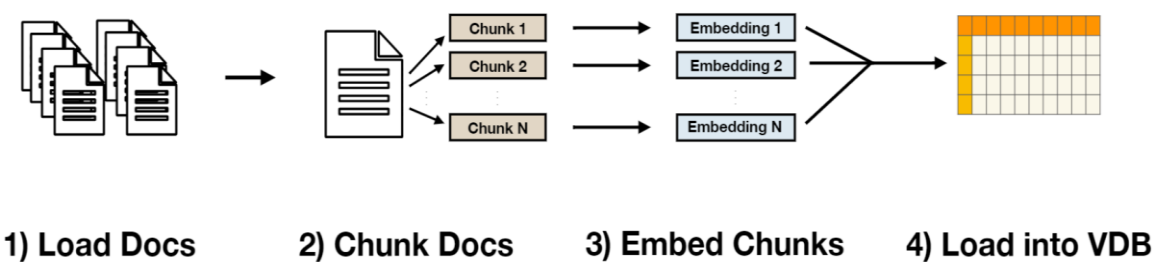


Figure 2.11. Knowledge Base [35]

2.6. Text Embeddings

2.6.1. Problem With Text

Text is not computable.

There's a fundamental challenge in trying to analyze text First of all, text is not computable, which means we can't do math with text the same way we can with numbers For example, let's say you go to a networking event and you're talking to other professionals in the data space If you wanted to summarize the typical height of everyone at the networking event, that's something that would be pretty straightforward All you'd have to do is measure the heights of everyone at the networking event, and then you can summarize those heights using something like an average And that's something that's easy to do using a calculator, Excel, or your favorite language. And that's something that you can easily calculate using a calculator, Excel, or your favorite programming language. However, if you had the job descriptions of everyone at this networking event, summarizing everyone's roles wouldn't be so easy because there's no cell function or mathematical operator that allows you to plug in text and generate a summary [20].

Feet	Inches
5	7
5	9
5	9
6	2
6	0
5	6
5	10

Table 1.2. Summarize Heights (Avg ~ 5’ 9”) [36]

Summarize Heights
Data Analyst in retail, 5 yrs. Specializes in consumer behavior & predictive analytics.
ML Engineer in fintech, 3 yrs. Develops algorithms for stock market predictions.
Data Scientist in healthcare, 10+ yrs. Uses patient data for better care outcomes.
Database Admin for e-commerce, 7 yrs. Focuses on efficient, secure data storage.
BI Analyst in hospitality, early career. Turns data into insights for growth.
Data Architect, 15 yrs, builds big data systems and data lakes for startups.
Freelance Data Visualization Specialist, 4 yrs. Makes complex data engaging.

Table 2.1. Summarize Roles 1 [36]

Summarize Heights
Data Analyst in retail, 5 yrs. Specializes in consumer behavior & predictive analytics.
ML Engineer in fintech, 3 yrs. Develops algorithms for stock market predictions.
Data Scientist in healthcare, 10+ yrs. Uses patient data for better care outcomes.
Database Admin for e-commerce, 7 yrs. Focuses on efficient, secure data storage.
BI Analyst in hospitality, early career. Turns data into insights for growth.
Data Architect, 15 yrs, builds big data systems and data lakes for startups.

Table 2.2. Summarize Roles 🤷‍♂️ 2 [36]

Figure 2.12. About People [36]

2.7. What is Text Embeddings?

Translate words into numbers.

Job Description		Text Embedding
Data Analyst in retail, 5 yrs	→	(3.4, 1.5)
ML Engineer in fintech, 3 yrs	→	(1.8, 3.1)
Data Scientist in healthcare, 10+ yrs.	→	(6.6, 2)
Database Admin for e-commerce, 7 yrs.	→	(4.2, 5)
BI Analyst in hospitality, early career.	→	(0.5, 1)
Data Architect, 15 yrs.	→	(6.5, 5)
Freelance Data Visualization Specialist, 4 yrs.	→	(2.6, 1.6)
Senior Data Engineer in automotive, 8 yrs.	→	(5, 5.5)

Figure 2.13. Text ot Numbers [36]

2.8. Text Embeddings Meaningful

Translate words into *meaningful* numbers. That would look something like this where the numbers that we generated in the text embedding define the location of each person's job description We can see the data analyst in retail with 5 years of experience is somewhere here and they are located next to a freelance data visualization specialist with 4 years of experience However these people are relatively far away from this guy who is a data architect with 15 years of experience And so this is the way that text embeddings capture the meaning of the underlying text namely job descriptions that are similar will be located close together while job descriptions that are very different will be located far away from each other From this view the BI analyst in hospitality early career is very different than the data architect with 15 years of experience [20].

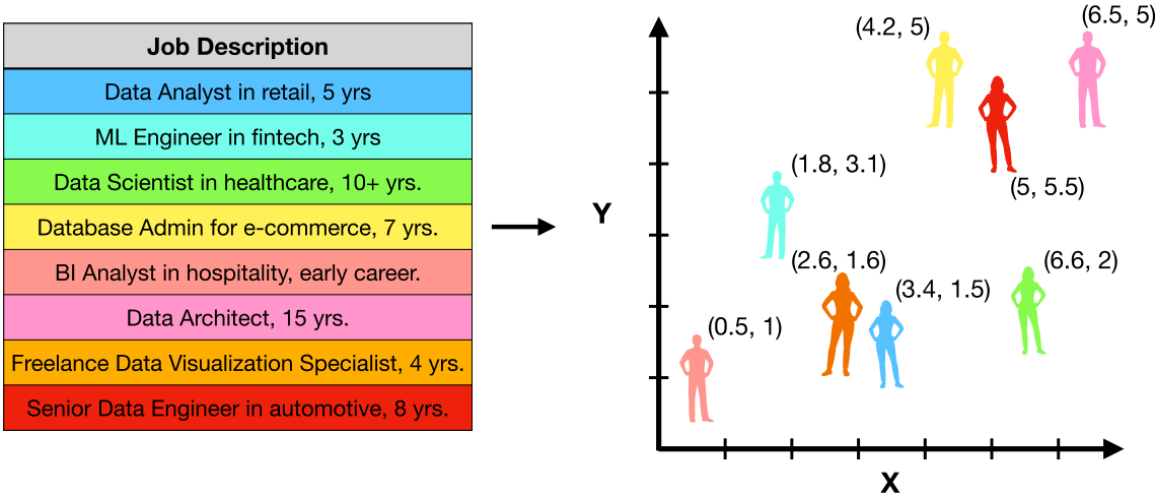


Figure 2.14. Text Embeddings (Meaningful) [36]

2.9. Why We Should Care About Text Embeddings

If you've used ChatGPT in the past, the question that might come to mind is, Shaw, why should I care about these text embeddings and translating text into numbers? Can't I just pass all my text to ChatGPT and let it figure out what I'm trying to do? And the answer is yes, there are many tasks that don't require these text embeddings, and they're better suited for ChatGPT. For example, if we wanted to summarize the 8 job descriptions from the networking event, it would actually be super easy to pass all of them to ChatGPT and generate a summary. However, if you have a use case that's not just some one-off, low-stakes task, but it's a very complex one, ChatGPT will be able to do it. off low-stakes task, but it's something that you're trying to integrate into your product or a website or some broader production-level software system, then you'd probably want to at least consider using text embeddings before you go out and build an AI agent, and the reason is that when we're talking about AI assistants, it's really early days for this kind of technology, and

there's still a lot of things that we don't understand about using these big language models in this way.Another downside to using an AI assistant for your use Another downside of using an AI assistant in your application is that there's a lot of computational cost associated with running these large models Another downside is the inherent security risks that come with building LLM-based applications And so there's a nice list of top ten LLM security risks at reference number 3 that I recommend you check out For example, these are things like prompt injection, which are maliciously crafted prompts that cause the LLM to expose private or confidential data or disrupt some expected decision pipeline Finally, responses from these LLM-based systems can be unpredictable. based systems can be unpredictable and prone to hallucinations, where the system generates fictitious or unhelpful information. On the other hand, text embeddings have been around for decades, and there are several past example applications that we can look to for guidance on future applications and use cases. Another advantage of using text embeddings is that they have a much lower computational and thus financial cost than using an entire large language model Another advantage is that there are far fewer security risks associated with using an embedding model compared to a large language model that users can access through a chat interface Finally, the responses are more predictable because each word or piece of text generates a deterministic set of numbers, so there's no randomness in the process [20].

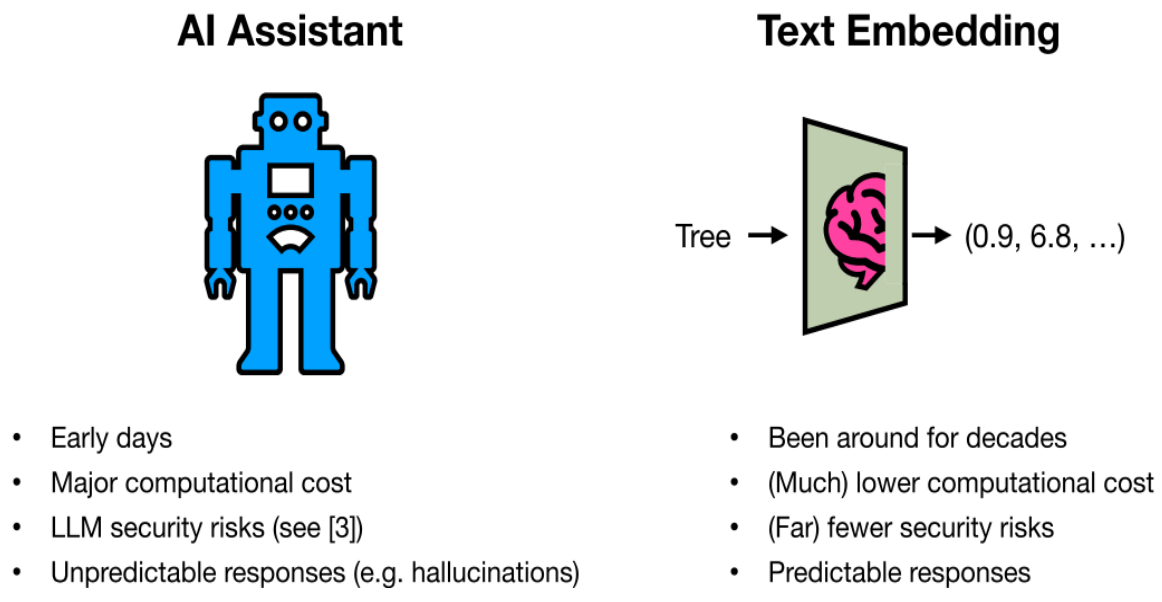


Figure 2.15. AI Assistant and Text Embedding [36]

2.10. Use Case 1: Text Classification

Text classification, which *is the process of assigning a label to a piece of text*. For example, if we were to take all the people from our networking event and their associated job descriptions, a classification task might be to try to determine which people are data analysts and which people are not data analysts based on their respective job descriptions So that might be pretty easy here, because anything to the left of this blue line we could label as a data analyst and anything to the right we could label as not a data analyst. However, text classification is a broadly applicable use case. Some other situations might be classifying emails as phishing attacks versus not phishing attacks, or classifying credit applications as fraudulent versus not fraudulent, and the list goes on and on and on with this high-level understanding [20].

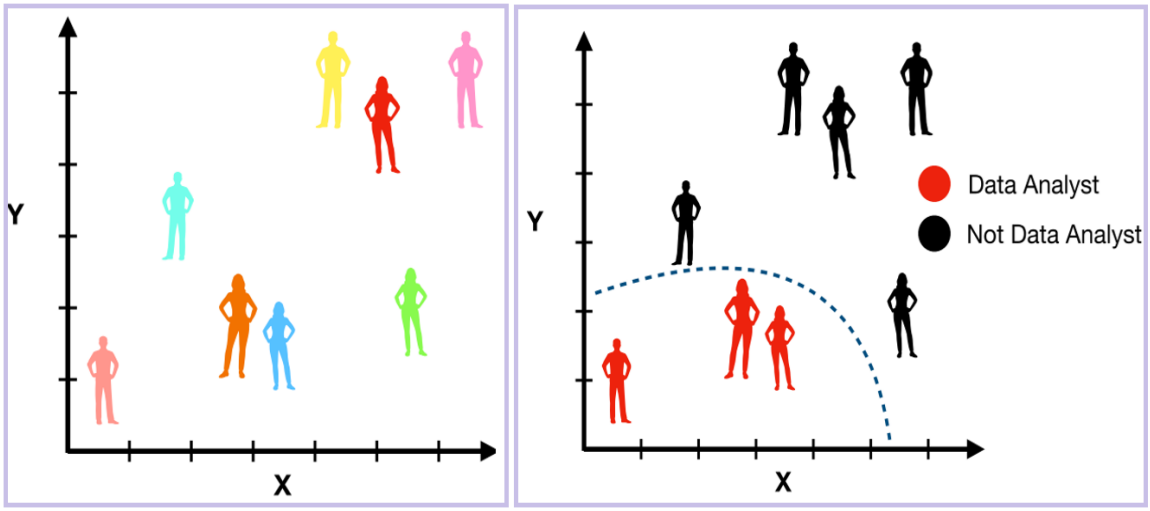


Figure 2.16. Text Classification - Data Analyst [36]

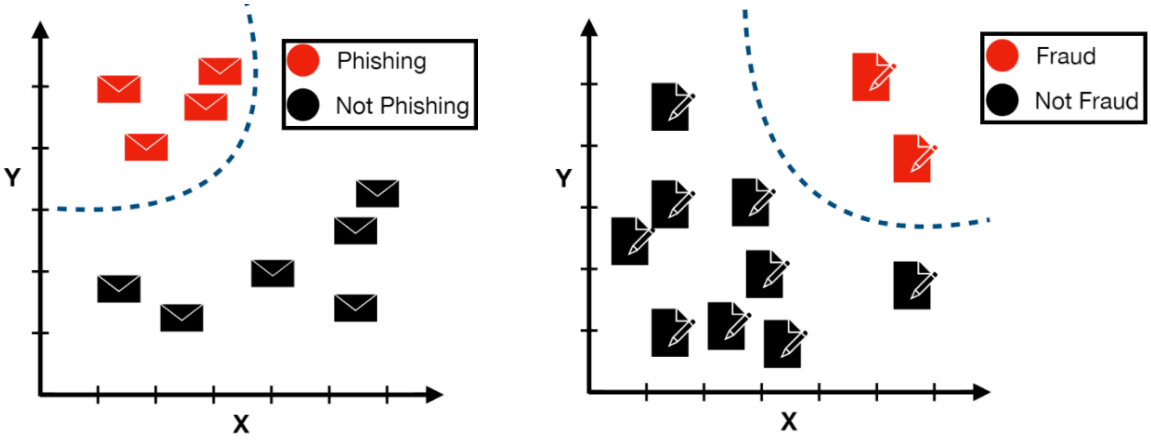


Figure 2.17. Phishing & Not Phishing - Fraud & Not Fraud [36]

2.11. Use Case 2: Semantic Search

Semantic search returns results based on the meaning of a query rather than matching specific keywords. Using text embeddings, a query like "I need someone to build my data infrastructure" can be embedded in a concept space alongside job descriptions. Semantic search identifies the closest matches, such as "data engineer" or "building data pipelines," even if the wording differs. Unlike keyword search, which might fail if specific terms aren't mentioned, semantic search connects related skills like ETL processes or data modeling, capturing the query's intent more effectively and delivering meaningful, context-aware results. [20].

Semantic search aims to understand the meaning behind words and phrases, allowing search engines to deliver more contextually relevant results. Unlike traditional keyword-based searches that rely on exact matches, semantic search takes into account the relationship between words, their contextual meaning, and even the intent behind the query. This is achieved through the use of embeddings and vector databases [22].

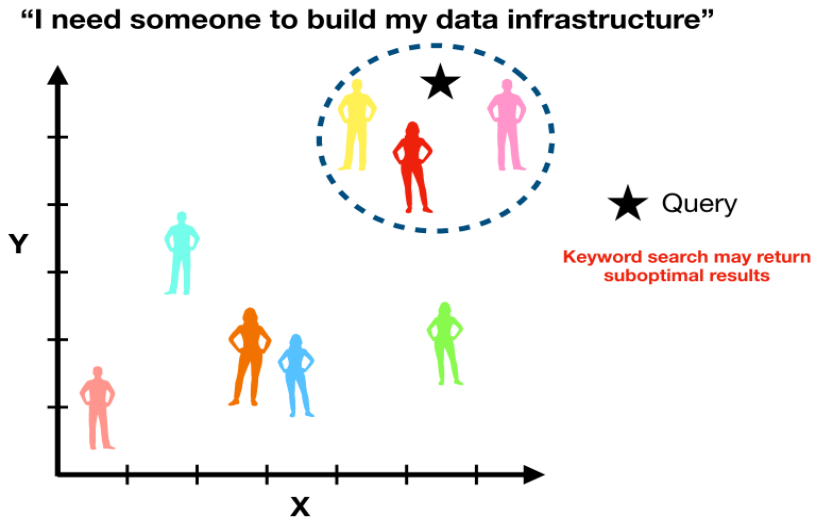


Figure 2.18. Semantic Search (Query) [36]

3. WHAT IS LARGE LANGUAGE MODELS (LLMS)

The large language models (LLMs) that can generate human-like text, and I've been using GPT in its various forms for years.

LLMs are an instance of something called a foundation model. Foundation models are pre-trained on large amounts of unlabeled and self-supervised data, meaning that the model learns from patterns in the data in a way that produces generalizable and adaptive output. And LLMs are instances of basic models applied specifically to text and text-like things, in this case things like

code. LLMs are trained on large datasets of text, such as books, articles, and conversations. When we say "large", these models can be tens of gigabytes (10 GB) in size and are trained on enormous amounts of text data, we're talking potentially petabytes of data here, a text file that is one gigabyte in size can store about 178 million words, a lot of words just in 1GB and 1PB it's about 1 million. LLMs are also among the largest models when it comes to the number of parameters, a parameter is a value that the model can change independently as it learns and the more parameters a model has the more complex it can be, GPT-3 for example is pre-trained on a corpus of actually 45TB of data and it uses 175 billion ML parameters [4].

A key characteristic of LLMs is their ability to respond to unpredictable requests. A traditional computer program receives commands in its accepted syntax or from a particular set of user inputs. A video game has a finite set of buttons, an application has a finite set of things a user can click or type, and a programming language consists of precise if/then statements [5].

In contrast, an LLM can respond to natural human language and use data analysis to answer an unstructured question or prompt in a way that makes sense. While a typical computer program would not recognize a prompt such as "What are the four greatest funk bands in history?", an LLM might respond with a list of four such bands and a reasonably cogent defense of why they are the best [5].

In terms of the information they provide, however, LLMs can only be as reliable as the data they ingest. If they are fed false information, they will give false information in response to user queries. LLMs also sometimes "hallucinate": they create false information when they are unable to provide an accurate answer. For example, in 2022, the news outlet Fast Company asked ChatGPT about Tesla's last fiscal quarter; while ChatGPT provided a coherent news article in response, much of the information in it was made up [5].

In terms of security, user-facing applications based on LLMs are as vulnerable to bugs as any other application. LLMs can also be manipulated by malicious input to provide certain types of responses over others-including responses that are dangerous or unethical. Finally, one of the security problems with LLMs is that users may upload secure, confidential data into them to increase their own productivity. But LLMs use the input they receive to train their models, and they are not designed to be secure repositories; they can expose confidential data in response to queries from other users [5].

3.1. How Do Work Large Language Models (LLMs)

LLMs are three things: data, architecture and finally we can think of it as education. These three things are really the components of an LLM. We have already discussed in the previous paragraph the enormous amount of text data that goes into these things. As for the architecture, it is a neural network and for GPT it is a transformer. Transformer architecture allows the model to

handle sequences of data such as sentences or lines of code. Transformers are designed to understand the context of each word in a sentence by considering it in relation to every other word. This allows the model to build a comprehensive understanding of the sentence structure and the meaning of the words within it. This architecture is then trained on all of this large amount of data, and during training, the model learns to predict the next word in a sentence. So "the sky is..." it starts with a random guess, "the sky is bug". But with each iteration, the model adjusts its internal parameters to reduce the difference between its predictions and the actual outcomes; the model keeps doing this, gradually improving its word predictions until it can reliably generate coherent sentences. Forget "bug", it can figure out that it's "blue". Now the model can be fine-tuned on a smaller, more specific data set. Here, the model refines its understanding to perform that specific task more accurately. Fine-tuning is what allows a general language model to become an expert at a specific task. [4]

LLMs - Models		
Data	Architecutre	Training

Table 3.1. How do llms works

3.2. Next-Word Prediction Paradigm

We have *Figure 3.1.* text listen to your and want predict what the next word would be but clearly there’s not just one word that can go after the string of words there are actually many word we can put after this text and it would make sense in the next word prediction paradigm what langauge model is tyring to do is to predict the probability distribution of the next word give the previous words, what is might look is listen your heart might be the most probable next word but another likley word could be gut or listen to your body or listen to your parents and listen to your grandma and so this is essentially the core task that these LLMs are trained to do and the way the LLMs will learn these probabilities is that it will see so many examples in this massive corpus of text that is trained on and it has a massive number of internal parameters so it can efficiently represent all the different statistical associations with different words. An important point here is that context matters if we simply added the word don’t to the front this string here and it changed it to don’t listen to your then this probability distribution could look entirely different because just by adding one word before this sentence we completely change the meaning of the sentence and so to put this a bit more mathematically we say this is the most technical thing in this part of works. This is an example of a Auto-regression task. So auto meaning self, regression meaning you’re tyring to predict something so what is the notation means, what is the probability of the end text or more technically the end token given the preceding M token s N, the formula is core task most

LLMs are doing and somehow through this very simple task of predict the next word we get this incredible performance from tools like Chat GPT and other LLMs [23] [24].

The Large Language Models (LLMs), such as OpenAI's GPT, Google's Bard, etc., are essentially advanced next-word prediction systems. This is the foundation for their much broader capabilities, such as text generation and complex problem solving. Although their applications seem limitless, at their core they are simply predicting the word that is most likely to appear next, based on patterns they have observed in large data sets. This paper discusses next-word prediction, how it works in LLMs, and why this simple mechanism produces remarkable capabilities [25].

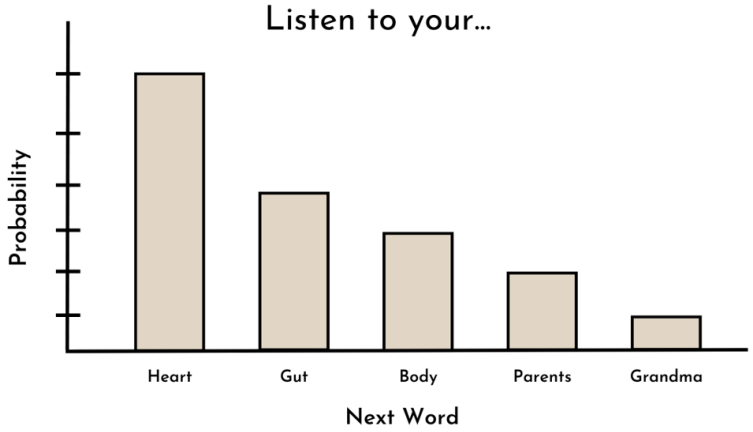


Figure 3.1. Next-word prediction [37]

$$P(t_n|t_{n-1}, \dots, t_{n-m}) \tag{1.1}$$

3.3. Business Applications

For customer service applications, companies can use LLMs to create intelligent chatbots that can that can efficiently and effectively handle a variety of customer queries, freeing up human agents for more complex issues. Another good area, content creation, can benefit from LLMs, which can help generate articles, emails, social media posts, and even YouTube video scripts. LLMs can even contribute to software development. They can help generate and review code with remarkable accuracy, and that's just scratching the surface. As big language models continue to evolve, we're sure to discover more innovative applications. That's why we're so excited about big language models [4].

3.3.1. What Is Quantitatively On LLMs

Quantitatively large language models are large, they have many more parameters than previous language models, and so these days it's anywhere from tens to hundreds of billions of parameters, the model parameters are numbers that define how the model will take an input and

generate the output, so it's essentially the numbers that define the model itself. That's a quantitative perspective of what distinguishes big language models from not big language models. [6].

Quantitatvely

Number of model parameters (2.1)

i.e. ~10-100 Billion

3.3.2. What Is Qualitatively On LLMs

Qualitative perspective and these so-called emergent properties that start to show up. Large language models become large and so emergent properties is the language used in this paper, *Survey of Large Language Models*, but essentially what this term means is that there are properties in LLMs that don't appear in smaller LLMs and so an example of this is zero learning. [6].

Qualitatively

Emergent properties (2.2)

i.e. Zero-shot learnig

3.3.3. Zero-Shot Learning

One, definition of zero shot learning is the capability of a ML model to complete a task it was not explicitly trained to do.

So while this may not sound super impressive to us very smart and sophisticated humans this is actually a major innovation in how these state-of-the-art ML models are developed so to see this we can compare the old state-of-the-art paradigm to this new state-of-the-art paradigm the old way and not to long away, we can say like five or ten years ago the way the high performing best ML models were developed was strictly through *supervised* learning. What this would typically look on *Figure 3.2.* you would train a model on thousands if not millions of labeled examples and so it's might have looked like is we have some input text like *Hello* like *Hola* you take all the below examples and you manually assign a label to each example, here we're lebeling the language so English Spansih, so you can imagine that this would take a tremendous amount of human effort to get 1000 if not 1000000 (millions) of high quality examples,

Now we compare this the more recent innovation with LLMs who use a different paradigm, the use so-called *self-supervised learning*. It's looks on *Figure 3.2.* like in the context of large language models is you train a very large model on a very large Corpus of data, this can look like is if you're trying to build a model that can do language classification instead of painstakingly generating this labeled dataset you can just take a corpus of English text and corpus of Spanish text

and train a model in a self-supervised way. In contrast to *supervised learning*, self-supervised learning does not require manual labellig of each example in your dataset the so-called lables or targets for the model are actually defined from the inherent structure of the data or in this context of the text.

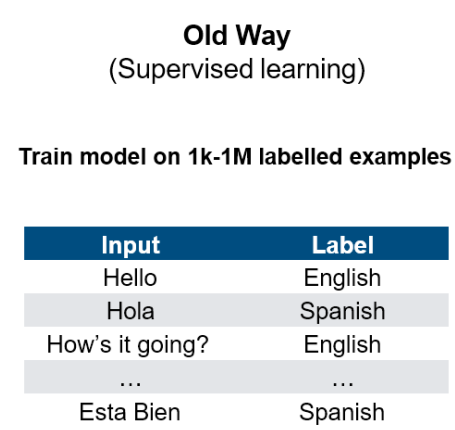


Figure 3.2. Old way (Supervised learning) [38]

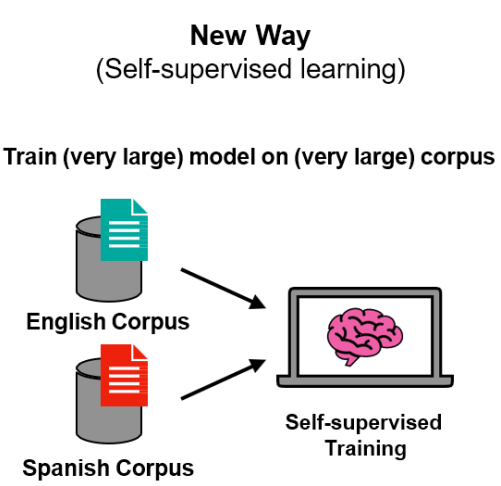


Figure 3.3. New way (Self-supervised learning) [38]

3.3.4. Levels of Using LLMs

In practice, how we can use LLMs, we can talk about three levels in which we can use LLMs. These three levels are ordered by the technical expertise and computational resources required: the most accessible way to use LLMs is prompt engineering, then we have model fine-tuning and finally we have build your own LLMs.

3.3.5. Level 1: Prompt Engineering (PE)

Using an LLM out-of-the-box (i.e. not changing any model parameters).

In a pretty broad definition of PE, define it as just using an LLM out of the box, more specifically not touching and the model parameters, so of those tens of billions or hundreds of billions of paramters that define the model we're not going to touch any of them we're just going to leave them as they are here I'm going to talk about two ways we can do this, One is the easy way, and we know most people in the world have interacted with LLMs, which is using things like ChatGPT, these are like intuitive user interfaces, they don't require any code and they're completely free, anybody can just go to ChatGPT, go to the website, type in a prompt and it will split out an answer, we know ChatGPT is the easiest way.

For applications where the easy way doesn't cut it there's the less obvious way which is using things like the openai api or the hugging face transformers library and these tools provide a

way to interact with LLMs programmatically so essentially using Python in the case of the openai api instead of typing your request into the chat gpt user interface you can send it over to openai using Python and their api and then you get a response back of course their api is not free so you have to pay per api call.

Other way we can do this is via open source solutions, one of which is the Hugging Face Transformers library, which gives you easy access to open source LLMs, because it's free, that's a big advantage, we can run the models locally, no need to send your potentially proprietary or confidential information to a third party like OpenAI.

Prompt engineering is the most accessible way to work with LLMs, just working with a model out of the box may give you sub-optimal performance on a specific task or use case, or the model has really good performance but it's massive it's got like a hundred billion parameters so the question might be is there a way we can use a smaller model but somehow weaken it in a way to have good performance on our very narrow and specific use case [6].

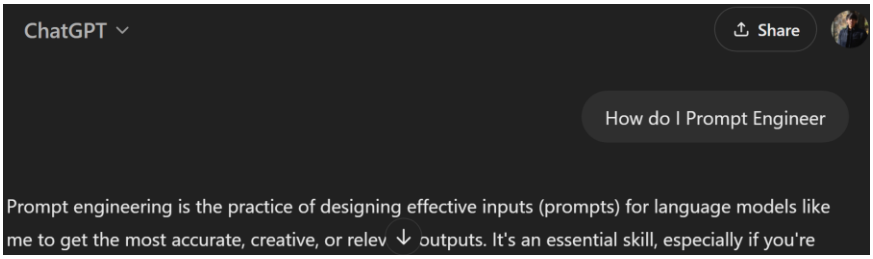


Figure 3.4. Esay way (ChatGPT)

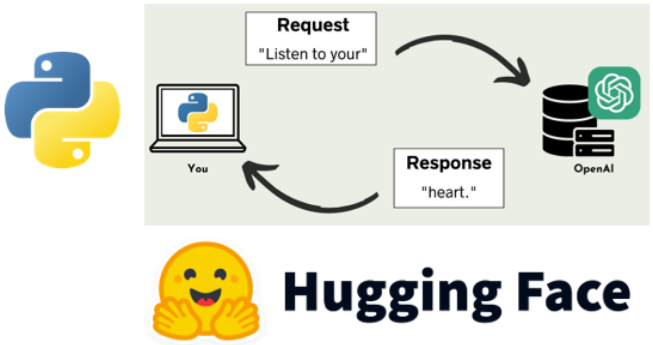


Figure 3.5. Less esay way (OpenAI API, Hugging Face) [38]

3.3.6. Level 2: Model Fine-tuning (FT)

Adjust at least 1 (internal) model parameter (but not all) for a particular task.

In FT we generally have three steps, one we get a pre-trained LLMs maybe from OpenAI or maybe an open source model from the Hugging Face Transformers library and then you update the model parameters given task specific examples, now going back to supervised learning and

self-supervised learning, the pre-trained model is going to be a self-supervised model so it will be trained on this simple word prediction task, but in step2, we are going to do supervised learning or even reinforcement learning to weaken the mode paramters for a specific use case and so this turns out to work very well models like ChatGPT you are not working with the raw pre-trained model, the model that we are interacting with in ChatGPT is actually a fine-tuned model developed using reinforcement learning, one reason why this might work is that in doing this self-supervised task and doing the word prediction the base model this pre-trained LLMs learns useful representations for a wide variety of tasks.

Step3 is you will deploy your fine-tuned LLMs to do some kind of service or you will use it in your daily life and so over sense is between prompt engineering and model fine tuning you can probably handle 99% of LLMs use cases and applications. However, if you are a large organisation or a large enterprise and security is a big concern so you don't want to use open source models or you don't want to send data to a third party via an API and maybe you want your LLMs to be very good at a relatively specific set of tasks, you want to customise the training data in a very specific way and you want to own all the travels, have it for commercial use, all that sort of thing, then it may make sense to go one step further [6].

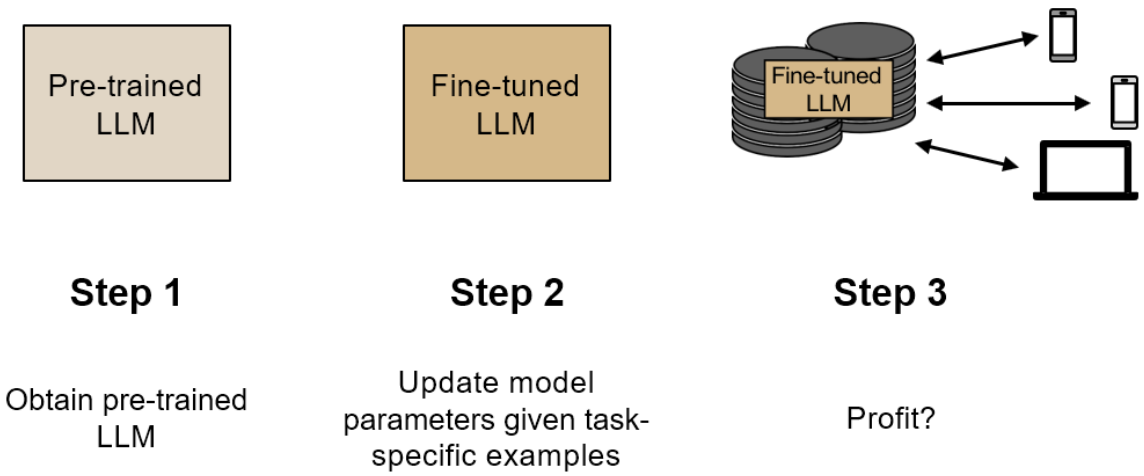


Figure 3.6. Fine-tuning model [38]

3.3.7. Level 3: Build your own LLM

Come with all model parameters.

We can define coming up with all the model parameters, we will just talk about how to do this at a very high level. First we need to get our data and so what this might look like is you will get a book corpus a Wikipedia corpus and a Python corpus and this is billions of tokens of text and then you will take that and pre-process it refine it into your training data set and then you can take

the training data set and do the model training through self-supervised learning and then out of that comes the pre-trained LLM. We can take this as your starting point for level two and go from there [6].

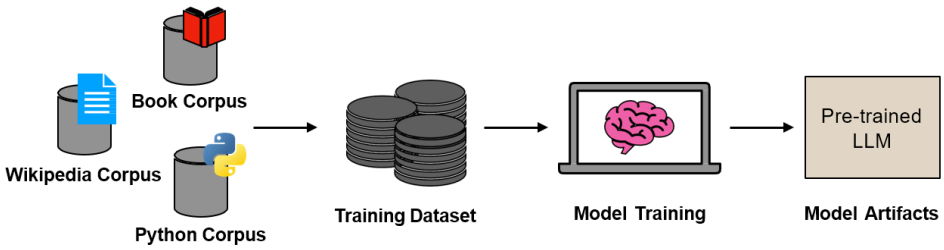


Figure 3.7. How to build your own model (LLM) [38]

3.3.8. What's an API (Application Programming Interface)

API people are often familiar with the term but not so familiar with what it actually means, API stands for Application Programming Interface and here we will define it as a way to interact with a remote application programmatically. This may sound very technical and very scary it's notlet's just consider the following analogy suppose this is you and you have a craving for some Salvadorian pupusas that you had during your Latin America trip from last summer however you're back home and you don't know where you can find pupusas in your hometown but lucky for you you have a super foodie friend who knows every single restaurant in town and so when you were trying to figure out where you can go to find great pupusas. [7]

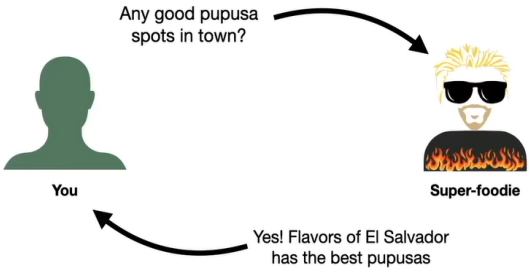


Figure 3.8. API Structure [38]

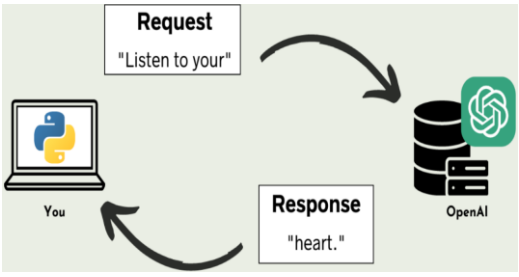


Figure 3.9. OpenAI API Sturcture [38]

3.3.9. OpenAI's (Python) API

It's like ChatGPT but with Python (and more features)

1) Customizable System Message

"I am ChatGPT, a large language model trained by OpenAI, based on the GPT-3.5

architecture. My knowledge is based on information available up until September 2021. Today's date is July 13, 2023."

2) Adjust Input Parameters

Max response length, number of responses, and temperature

3) Process images and other file types

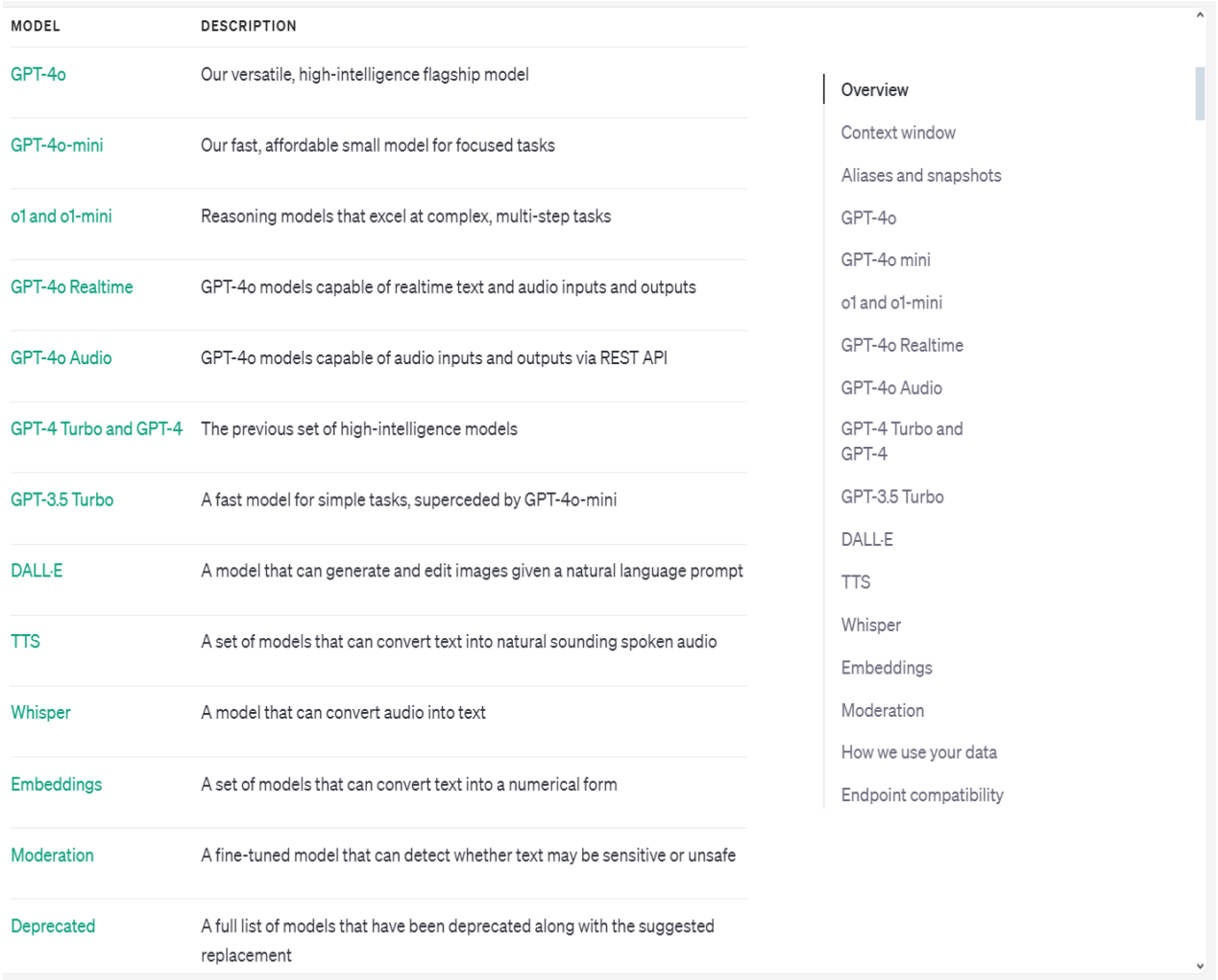
4) Extract helpful word embeddings for down stream tasks

5) Input audio for transcription and translation

6) Model fine-tuning functionality [8].

3.3.10. OpenAI API Models

Language models and more



MODEL	DESCRIPTION
GPT-4o	Our versatile, high-intelligence flagship model
GPT-4o-mini	Our fast, affordable small model for focused tasks
o1 and o1-mini	Reasoning models that excel at complex, multi-step tasks
GPT-4o Realtime	GPT-4o models capable of realtime text and audio inputs and outputs
GPT-4o Audio	GPT-4o models capable of audio inputs and outputs via REST API
GPT-4 Turbo and GPT-4	The previous set of high-intelligence models
GPT-3.5 Turbo	A fast model for simple tasks, superceded by GPT-4o-mini
DALL·E	A model that can generate and edit images given a natural language prompt
TTS	A set of models that can convert text into natural sounding spoken audio
Whisper	A model that can convert audio into text
Embeddings	A set of models that can convert text into a numerical form
Moderation	A fine-tuned model that can detect whether text may be sensitive or unsafe
Deprecated	A full list of models that have been deprecated along with the suggested replacement

Figure 3.10. OpenAI API Models [39]

3.4. What is Prompt Engineering?

Any us of an LLM out-of-the-box. Prompt engineering is the means by which LLMs areprogrammed via prompts. To demonstrate the power of prompt engineering, we provide the following prompt:

Prompt: "*From now on, I would like you to ask me questions to deploy a Python application to AWS. When you have enough information to deploy the application, create a Python script to automate the deployment*." This example prompt causes ChatGPT to begin asking the user questions about their software application. ChatGPT will drive the question-asking process until it reaches a point where it has sufficient information to generate a Python script that automates deployment. This example demonstrates the programming potential of prompts beyond conventional "generate a method that does X" style prompts or "answer this quiz question" [9].

3.4.1 Two Levels of Prompt Engineering

1) The Easy Way: ChatGPT (or something similar)

Figure 3.11. Esay way (ChatGPT, Barod and Bing) [10]

1) The Less Easy Way: Programmatically

Figure 3.12. The Esay way (Python and Javascript) [10]

3.4.2 Building AI Apps with Prompt Engineering

The Less Easy Way unlocks a new paradigm of software development.

Use Case: Automatic Grader for high school history class

Question: "*Who was the 35th president of the United States of America?*"

Potential Correct Answers:

- *John F. Kennedy*
- *JFK*

- *Jack Kennedy (a common nickname)*
- *John Fitzgerald Kennedy (probably trying to get extra credit)*
- *John F. Kenedy (misspelled last name) [10]*

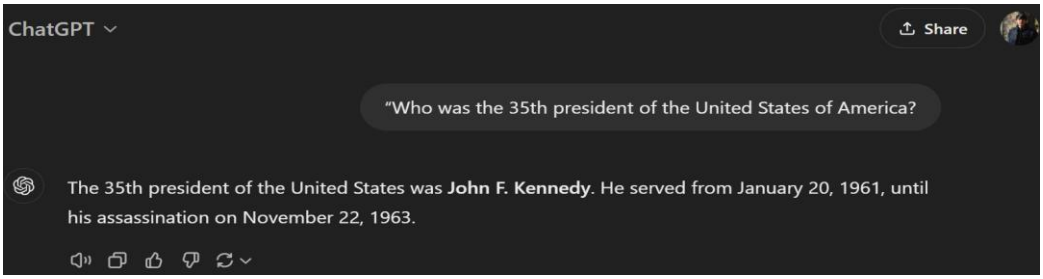


Figure 3.13. AI App - Prompt Engineering

3.4.2 What is Fine-Tuning?

Fine-tuning uses a pre-trained model, such as OpenAI's GPT series, as a foundation. The process involves further training on a smaller, domain-specific dataset. This approach builds upon the model's pre-existing knowledge, enhancing performance on specific tasks with reduced data and computational requirements. FT transfers the pre-trained model's learned patterns and features to new tasks, improving performance and reducing training data needs. It has become popular in NLP for tasks like text classification, sentiment analysis, and question-answering [11].

Figure 3.14. Mind map depicting various dimensions of (LLMs) [11]

3.4.3 Self-supervised Fine-Tuning (SSFT)

Self-supervised learning is another ML approach that does not require labeled data. Nevertheless, differently from the unsupervised scenario, here the model is trained using automatically extracted labels. This technique may be seen as the concatenation of two stages.

1. *the first one takes an unlabeled set of data and produces a set of labels for it;*

2. *The second stage is usually represented by a supervised learning task that makes use of the previously extracted labels.*

The advantage of self-supervised learning over unsupervised learning is given by the possibility to leverage supervised algorithms for the extraction of knowledge from the data. In recent years self-supervision gained a lot of success in NLP. It is often involved in the pretraining of large models . This makes it possible to extract knowledge from large collections of unlabeled data [13].

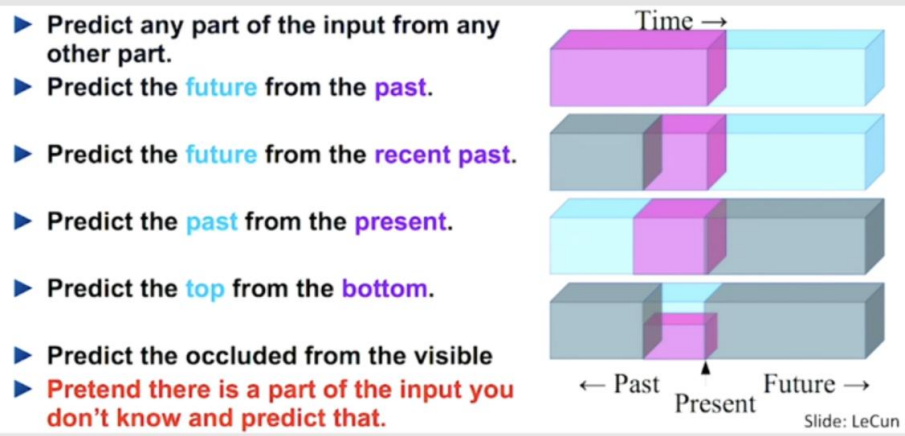


Figure 3.15. Self Supervised Learning [40]

3.4.4 Supervised Fine-Tuning (SFT)

SFT involves providing the LLM with labelled data tailored to the target task. For example, fine-tuning an LLM for text classification in a business context uses a dataset of text snippets with class labels. While effective, this method requires substantial labelled data, which can be costly and time-consuming to obtain [11].

Supervised fine-tuning, involves adapting a pre-trained Language Model (LLM) to a specific downstream task using labeled data. In supervised fine-tuning, the finetuning data is collected from a set of responses validated before hand. That's the main difference to the unsupervised techniques, where data is not validated before hand. While LLM training is (usually) unsupervised, Finetuning is (usually) supervised. During supervised fine-tuning, the pre-trained LLM is fine-tuned on this labeled dataset using supervised learning techniques. The model's

weights are adjusted based on the gradients derived from the task-specific loss, which measures the difference between the LLM's predictions and the ground truth labels [14].

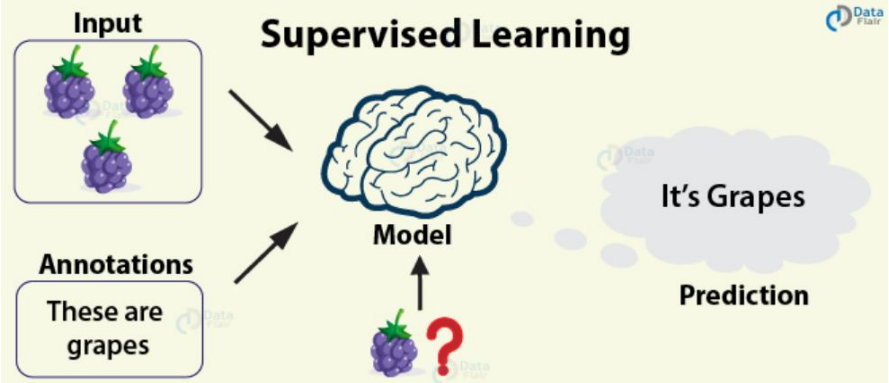


Figure 3.16. Supervised Learning (ML) [40]

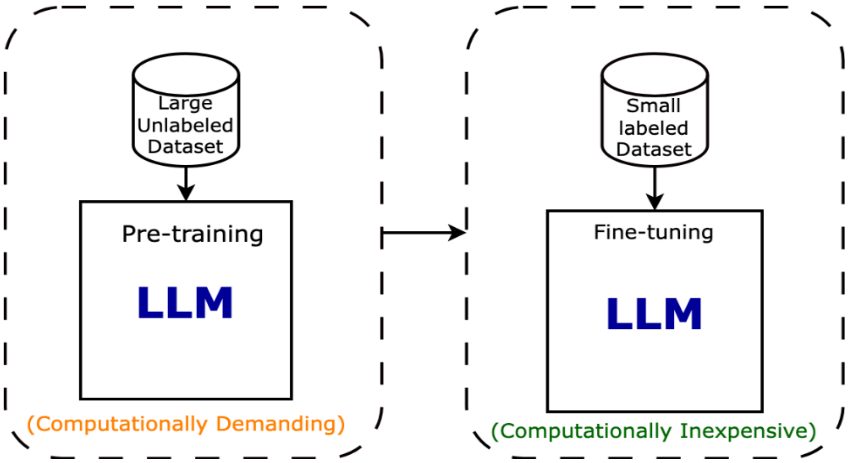


Figure 3.17. Supervised Learning (FT) [41]

3.4.6 Reinforcement fine-tuning (ReFT)

To address this issue and offer a more versatile way to solve advanced reasoning problems, reinforced fine-tuning (ReFT) has come into play. Reinforced fine-tuning (ReFT) starts with supervised fine-tuning (SFT), typically lasting one or two cycles. During this phase, the model gains an essential capability to solve mathematical problems correctly. Following this, ReFT takes the model's training to the next level by employing a reinforcement learning (RL) algorithm using methods like proximal policy optimization (PPO). This advanced stage allows the model to explore and learn from various correct solutions and reasoning methods [12].

Google's Active Query Answering (AQA) system makes use of reinforcement learning.It reformulates the questions asked by the user. For example, if you ask the AQA bot the question – "*What is the birth date of Nikola Tesla*" then the bot would reformulate it into different questions like "*What is the birth year of Nikola Tesla*", "*When was Tesla born*?" and "*When is Tesla's*

birthday". This process of reformulation utilized the traditional sequence2sequence model, but Google has integrated reinforcement Learning into its system to better interact with the query based environment system [15].

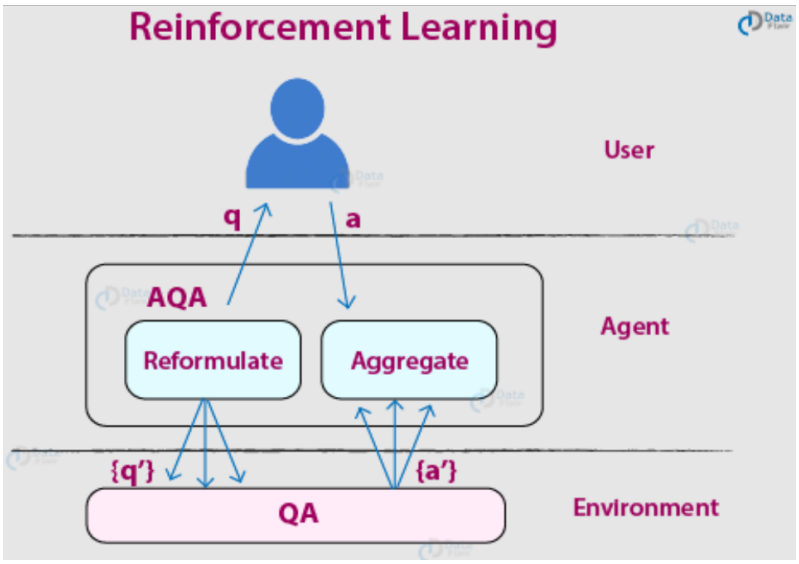


Figure 3.18. Reinforcement Learning (ReFT) [15]

3.4.7 3 Ways to Fine-tune

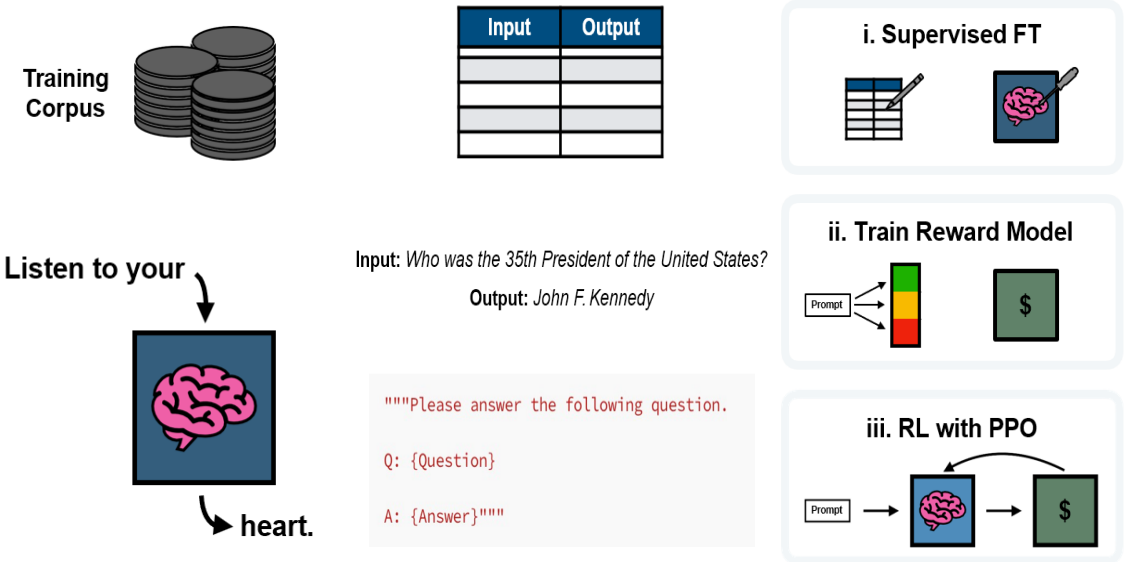


Figure 3.19. Self-supervised [33] **Figure 3.20.** Supervised [33] **Figure 3.21.** Reinforcement Learning [33]

3.4.8 Pre-training vs Fine-tuning

Aspect	Pre-training	Fine-tuning
Definition	Training on a vast amount of unlabelled text data	Adapting a pre-trained model to specific tasks
Data Requirement	Extensive and diverse unlabelled text data	Smaller, task-specific labelled data
Objective	Build general linguistic knowledge	Specialise model for specific tasks
Process	Data collection, training on large dataset, predict next word/sequence	Task-specific data collection, modify last layer for task, train on new dataset, generate output based on tasks
Model Modification	Entire model trained	Last layers adapted for new task
Computational Cost	High (large dataset, complex model)	Lower (smaller dataset, finetuning layers)
Training Duration	Weeks to months	Days to weeks
Purpose	General language understanding	Task-specific performance improvement
Examples	GPT, LLaMA 3	Fine-tuning LLaMA 3 for summarisation

Table 4.1. A Comparative Overview of Pre-training and Fine-tuning in (LLMs) [11].

3.5.1 QLoRA (Quantized Low-Rank Adaptation)

Fine-tuning large language models (LLMs) is a highly effective way to improve their performance, add desirable behavior, or remove undesirable behavior. However, fine-tuning very large models is prohibitively expensive; regular 16-bit fine-tuning of an LLaMA 65B parameter model requires more than 780 GB of GPU memory. While recent quantization methods can reduce the memory footprint of LLMs, such techniques only work for inference and break down during training. We demonstrate for the first time that it is possible to fine-tune a quantized 4-bit model without performance degradation. Our method, QLORA, uses a novel high-precision technique to quantize a pre-trained model to 4 bits, and then adds a small set of learnable low-rank adapter weights that are tuned by backpropagating gradients through the quantized weights. [16].

Fine-tuning is when we take an existing model and to weak it for a particular use case although this is a simple idea to applying it to large language models isn't always straightforward *the key challange is that LLMs are very computationally expensive which means fine-tuning them in a standard way is not something you can do on a typical computer or laptop*.

3.5.2 Fine-Tuning (Recap-Summary)

Tweaking an existing model for a particular use case. FT is like taking a raw diamond, refining it, and distilling it into something more practical and usable, like a diamond you might put on a diamond ring. In this analogy, the raw diamond is your base model, so that would be something like gpt3, while the final diamond you come away with is your fine-tuned model, which is something like Chat GPT [17].

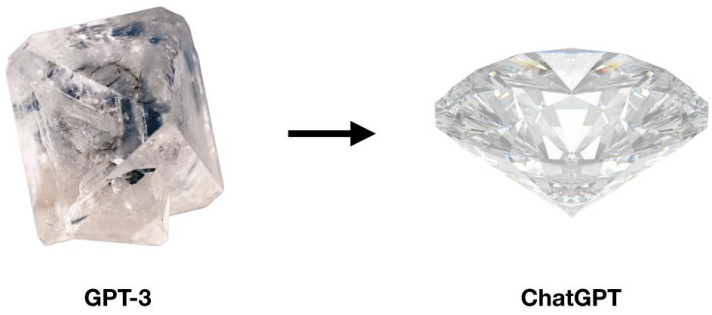


Figure 3.22. Fine-tuning (Raw Diamond to Diamond) [17]

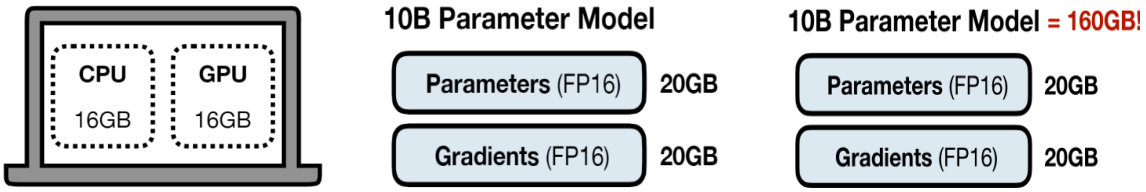
3.5.2 The Problem (LLMs are computationally expensive)

The core problem with fine-tuning LLMs is that they are computationally expensive to get a sense of, say you have a pretty powerful laptop and it comes with a CPU and a GPU where the CPU has 16 GB, the CPU has 16 GB of RAM and your GPU has 16 GB of RAM. We want to fine-tune a model with 10 billion parameters, each of these parameters corresponds to a number that we need to represent on our machine, the standard way to do this is to use the FP16 number format, which requires about two bytes of memory per parameter, so just doing some simple math here, 10 billion parameters times 2 bytes per parameter comes to 20 GB of memory just to store the model parameters, so one problem here is that this 20 GB model won't fit on the CPU or the GPU, but maybe we can get clever in how we distribute the memory so that the load of the model is split between the CPU and the GPU, But when we talk about fine-tuning, we're talking about re-training the model per parameter, which is going to require more than just storing the parameters of the models. Another thing we need is the gradients, these are numbers that we use to update the model parameters in the training process, we have a gradient that's just going to be a number for each parameter in the model, so that adds another 20 GB of memory, so we've gone from 20 to 40, and now even if we get super clever with how we distribute it across our CPU and GPU, it's still not going to fit, so we actually need to add another GPU. To even make that work, but of course that's not the whole story, you also need room for the optimizer states, so if you're using an optimizer like 'Adam' which is very widely used, this is going to take up the memory footprint of model training, where this is coming from is an optimizer like 'Adam' is going to store the memory value and the

variance value for each parameter in your model, so we have 2 numbers per parameter In addition, these values have to be encoded with higher precision, so instead of the FP16 format, they're going to be encoded in the FootPrint FP32 format, and so when it's all said and done, there's about a 12x multiplier for the memory footprint of these optimizer states. That means we're going to need a lot more GPUs to actually do the fine tuning. So we come to a grand total of 160 GB of memory required to train a 10 billion parameter model of course these enormous memory requirements are not going to fit on your laptop and it's going to require some heavy duty hardware to run so 160 GB if you get like an 80 GB GPU like the 100 you're going to need at least two of those and and those are about $20000 dollars. We are probably talking about like $50000 dollar just for the hardware to fine tune a 10 billion parameter model in the standard way.

That the reason QLoRA comes, QLoRA technique that makes this whole fine-tuning process much more efficient so much so that you can just run it on your laptop here without the need for all these extra GPUs, *before QLoRA we wil talk about Quantization.* [17].

The cost of an LLM typically depends on the number of input and output tokens, where a word is about 1.3 tokens [26]. Typically, output tokens cost more than input tokens. For example, output tokens cost 5 × as much as input tokens for Claude 3 models [27]. The cost also varies from model to model. Even as new models are released that are cheaper than their predecessors, the state-of-the-art models remain costly. For example, Claude Opus output tokens are 375 × as expensive as Amazon Titan Text Lite on AWS bedrock [28]. Table 1 lists example use case costs. While individually these costs are small, in the aggregate they can add up to a large bill. Low-income users are likely to find this price prohibitive [29]. To better support these users, we explore three strategies to reduce costs [30].



CPU & GPU | 10 Billion | 160GB = $20000 Dollar

Figure 3.23. Fine The Problem (Fine-Tuning) LLMs Cost - 1 [17]

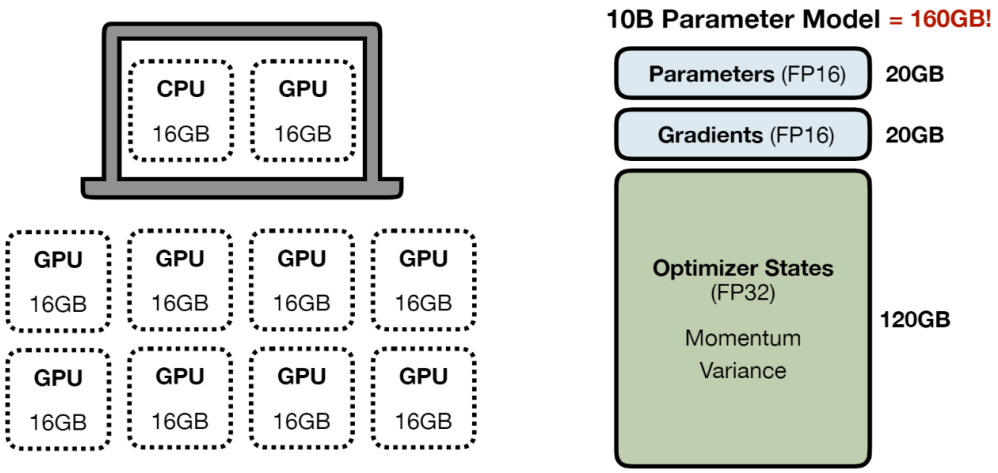


Figure 3.24. Fine The Problem (Fine-Tuning) LLMs Cost - 2 [17]

4. QUANTIZATION METHOD

4.1. How is Quantization Works?

Quantization = splitting range into buckets

Quantization, in mathematics and digital signal processing, is the process of mapping input values from a large (often continuous) set to output values in a smaller (countable) set, often with a finite number of elements. Rounding and truncation are typical examples of quantization. Quantization is involved to some degree in almost all digital signal processing, since the process of representing a signal in digital form usually involves rounding. Quantization is also at the heart of virtually all lossy compression algorithms. The simplest way to quantize a signal is to choose the digital amplitude value that is closest to the original analog amplitude. This example shows the original analog signal (green), the quantized signal (black dots), the signal reconstructed from the quantized signal (yellow), and the difference between the original and reconstructed signals (red). The difference between the original and reconstructed signals is the quantization error and, in this simple quantization scheme, is a deterministic function of the input signal [18].

In the example below on *Figure 4.1*. Range of numbers into bins, so as an example, now we're looking at any number between 0 - 100, obviously there are infinite numbers that can fit into this range, you know, there's like 27, 55.3, 83, and so on and so forth. 7823 and so on and so forth what quantization consists of is taking this infinite range of numbers and dividing it into discrete bins one way to do this is to quantify this infinite range using integers so what that would look like for our three numbers here is that 27 would go into this 27th bucket, 55.3 would go into this 55 bucket and then 83.7823 would go into this 83 bucket. But we can go one step further if we want to quantize by 10's instead for integers it would look like 27 goes with 20, 55 goes with 50, and 83 goes with 80, so that's the basic idea and the reason is important quantization is required every time

you want to represent numbers in a computer, the reason is it that *if you want to code single number at list infinite range of possibility this will require infinite bytes of memory it just can't be done* at some point when you are talking about physically consiered system like a computer to have to make some apporximation. So we go from the infinite range to the range quantized by integers, which requires 0.875 bytes per number, and then we go further for 10 buckets which requires half a byte 0. 5 bytes per number, one thing there is a natural tradeoff, we have a lot of buckets which gives us a lot of precision but it's going to increase the memory footprint of our model, however you could have very few buckets for quantization which would minimize the memory footprint but that would be a pretty crude approximation of the model you're working with. *So balancing this tradeoff is a key contribution of QLoRA*. [17].

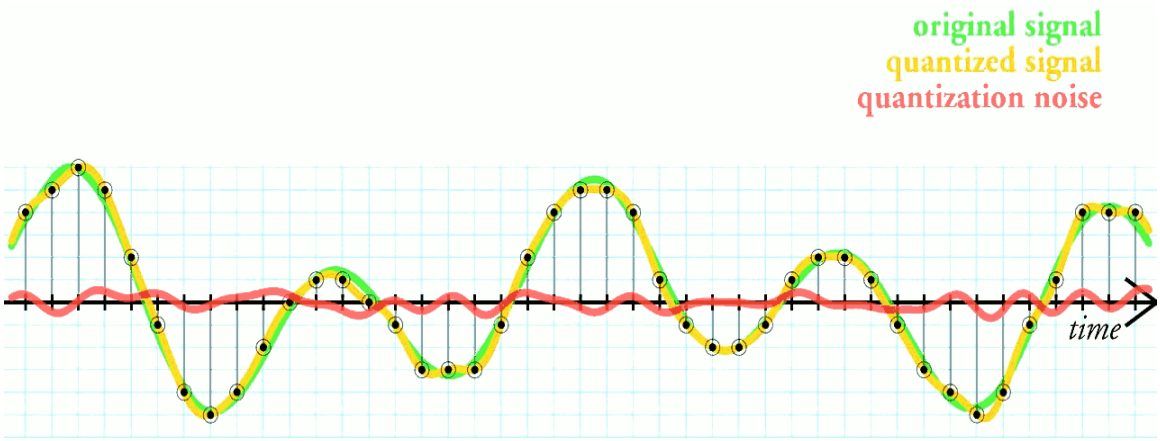


Figure 4.1. Quantization (Signal) [18]

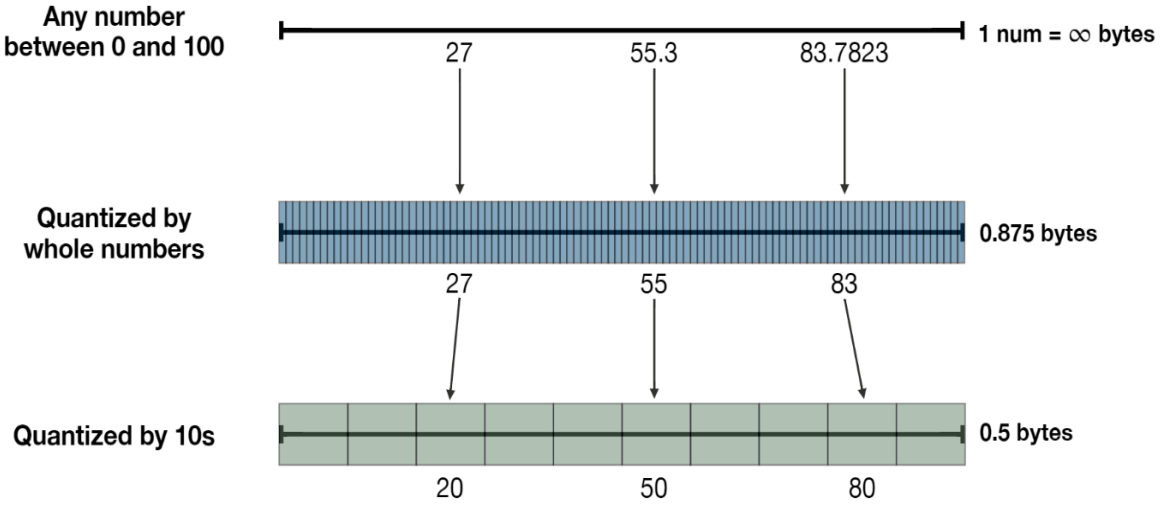


Figure 4.2. Quantization (QLoRA) [17]

4.2. Full Finetuning, LoRA and QLoRA

QLoRA introduces several innovations to reduce memory consumption without sacrificing performance: (1) 4-bit NormalFloat, an information-theoretically optimal quantization data type for normally distributed data that empirically performs better than 4-bit integers and 4-bit floats. (2) Double Quantization, a method that quantizes the quantization constants, saving on average about 0.37 bits per parameter (about 3 GB for a 65B model). (3) Paged optimizers, which use NVIDIA unified memory to avoid the gradient checkpointing memory spikes that occur when processing a mini-batch with a long sequence length. We combine these contributions into a better tuned LoRA approach that includes adapters at each network layer, thereby avoiding almost all of the accuracy tradeoffs seen in previous work. QLoRA's efficiency allows us to conduct in-depth study of instruction fine-tuning and chatbot performance at model scale, which would be impossible with regular fine-tuning due to memory overhead. [18].

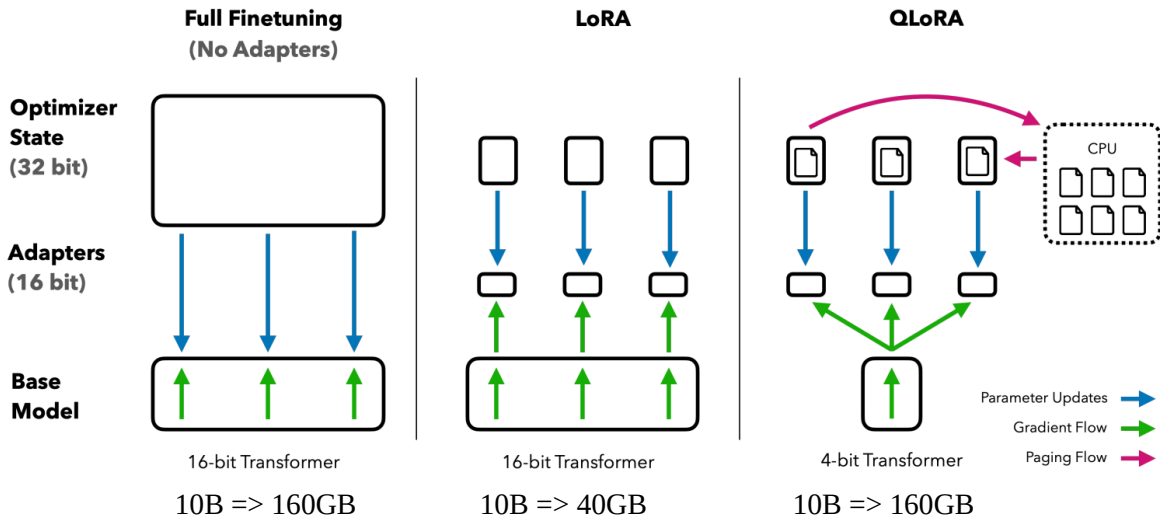


Figure 4.3. Different finetuning methods and their memory requirements. QLORA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes [16].

4.3. Behavioral Differences Between LoRA and Full Fine-Tuning

We have identified structural differences in the solutions of **LoRA** and full fine-tuning. Here, we investigate whether LoRA and full fine-tuning produce measurable differences in fine-tuned model behavior. While we have already seen that they perform nearly identically on their in-distribution test set, we evaluate whether these behavioral similarities hold under other distributions [31].

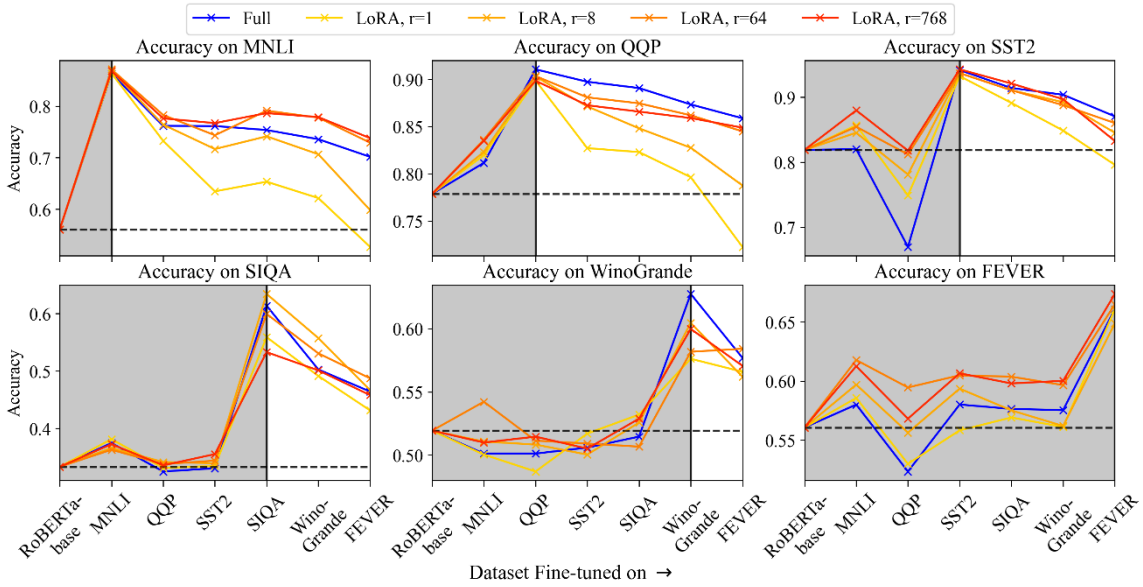


Figure 4.4. Continual Learning performance of RoBERTa for full fine-tuning and LoRA. We sequentially train on six tasks, in order from left to right. Horizontal dotted line indicates baseline pre-trained performance. Vertical solid line indicates when a specific dataset is fine-tuned on. Gray region represents performance before the model has been trained on that task. We are interested in the differences in accuracies of these methods both right after training (at the vertical black line) and later (in the white region). We see that low ranks of LoRA forget previously learned tasks more [32].

4.4. QLORA Finetuning

QLORA achieves high-fidelity 4-bit fine-tuning through two techniques we propose 4-bit NormalFloat (NF4) quantization and double quantization. In addition, we introduce paged optimizers to prevent memory spikes during gradient checkpointing from causing out-of-memory errors, which have traditionally made finetuning on a single machine difficult for large models.QLoRA has a low-precision storage datatype, in our case usually 4-bit, and a computation datatype, which is usually BFloat16. In practice, this means that whenever a QLoRA weight tensor is used, we quantize the tensor to BFloat16 and then perform a matrix multiplication in 16-bit [32].

5. RESULT

The seminar focused on the integration of Retrieval-Augmented Generation (RAG) techniques with Large Language Models (LLMs), emphasizing their transformative impact on generative AI systems. By grounding LLM output in external data sources, RAG reduces inaccuracies and increases the reliability of model responses. This approach bridges the gap between the vast but static training knowledge of LLMs and the dynamic, real-time information needs of diverse applications.

The RAG framework was explored in detail, highlighting its core components: the retriever, which searches external knowledge bases, and the generator, which generates responses based on the retrieved data. This architecture allows models to access up-to-date and domain-specific information, significantly improving their adaptability and relevance. For example, in scenarios such as healthcare, legal research, and customer support, RAG enables systems to provide contextually rich and accurate answers that often outperform traditional generative models.

The seminar also addressed technical challenges, including computational latency, the risk of overloading models with excessive or irrelevant information, and ethical concerns related to bias and data security. Strategies for overcoming these challenges were discussed, such as optimizing retrieval methods, implementing quality controls, and refining system scalability to support real-time updates and user customization.

Looking to the future, the seminar identified promising directions for RAG-enabled LLMs. These include improving retrieval efficiency, integrating multimodal inputs, and expanding the application of RAG systems across industries. Particular emphasis was placed on enabling fine-grained, personalized user interactions and facilitating real-time updates to knowledge bases to ensure that AI systems remain relevant and responsive.

In summary, RAG represents a critical evolution in generative AI, enabling LLMs to deliver accurate, versatile, and context-aware responses across multiple domains. The seminar underscored the importance of combining retrieval-based techniques with LLMs to overcome current limitations and pave the way for more robust, scalable, and impactful AI solution.

SOURCES

[1] Medium,-https://medium.com/@juanc.olamendy/rag-best-practices-enhancing-large-language-models-with-retrieval-augmented-generation-6961c8b834ff, Access: 14-11-2024

[2] Marktechpost, https://www.marktechpost.com/2024/09/29/enhancing-language-models-with-retrieval-augmented-generation-a-comprehensive-guide/, Access: 14-11-2024

[3] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, H. Wang "Retrieval-Augmented Generation for Large Language Models: A Survey" *arXiv:2312.10997v5 [cs.CL] 27 Mar 2024*

[4] Youtube, https://www.youtube.com/watch?v=5sLYAQS9sWQ, Access: 18-11-2024

[5] Cloudflare, https://www.cloudflare.com/learning/ai/what-is-large-language-model/, Access: 18-11-2024.

[6] S.Talebi, https://www.youtube.com/watch?v=tFHeUSJAYbE&list=PLz-ep5RbHosU2hnz5ejezwaYpdMutMVB0&index=1, Access: 29-11-2024

[7] S.Talebi, https://www.youtube.com/watch?v=czvVibB2lRA&list=PLz-ep5RbHosU2hnz5ejezwaYpdMutMVB0&index=8, Access: 18-12-2024

[8] Github, https://github.com/ShawhinT/YouTube-Blog/blob/main/LLMs/_slides/openai-api.pdf, Access: 20-12-2024

[9] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, D. C. Schmidt, "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT" *arXiv:2302.11382 [cs.SE]*, 20 December 2024

[10] Github, https://github.com/ShawhinT/YouTube-Blog/blob/main/LLMs/_slides/prompt-engineering.pdf, Access: 21-12-2024

[11] V.B Parthasarathy, A. Zafar, A. Khan, and A. Shahid, "The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities" *arXiv:2408.13296 [cs.LG]*, 22 December 2024

[12] SuperAnnotate, https://www.superannotate.com/blog/reinforced-fine-tuning, Access: 22-12-2024

[13] B. Moell, N. Afsarmanesh, F. Olsson, P.G. Examiner, J. Beskow, "Self-Supervised Fine-Tuning of sentence embedding models using a Smooth Inverse Frequency model", *https://kth.diva-portal.org/smash/get/diva2:1813068/FULLTEXT01.pdf*, 23 December 2024

[14] Medium, https://medium.com/mantisnlp/supervised-fine-tuning-customizing-llms-a2c1edbf22c3, Access: 23-12-2024

[15] Data Flair, https://data-flair.training/blogs/types-of-machine-learning-algorithms/, Access: 23-12-2024

[16] T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs" *arXiv:2305.14314 [cs.LG]*, 27 December 2024

[17] YouTube, https://www.youtube.com/watch?v=XpoKB3usmKc&list=PLz-ep5RbHosU2hnz5ejezwaYpdMutMVB0&index=10, Access: 27-12-2024

[18] WikipediA, https://en.wikipedia.org/wiki/Quantization_(signal_processing), Access: 27-12-2024

[19] YouTube, https://youtu.be/Ylz779Op9Pw?si=btCyBEkcRWSgrnC-, Access: 29-12-2024

[20] YouTube, https://youtu.be/sNa_uiqSlJo?si=FrF-JEIJjcHoK9Im, Access: 29-12-2024

[21] A. Khan, M. Hasan, K. Kemell, J. Rasku, P. Abrahamsson “Developing Retrieval Augmented Generation (RAG) based LLM Systems from PDFs: An Experience Report” *arXiv:2410.15944v1 [cs.SE]* , 06 January 2025

[22] Medium, https://medium.com/@pankaj_pandey/exploring-semantic-search-using-embeddings-and-vector-databases-with-some-popular-use-cases-2543a79d3ba6, Access: 07-01-2025

[23] Youtube, https://youtu.be/tFHeUSJAYbE?si=sswH0bZ5xIQBKY5s, Access: 07-01-2025

[24] W.X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.Y. Nie, J.R. Wen “A Survey of Large Language Models” *arXiv:2303.18223 [cs.CL]* , 07 January 2025

[25] Majumdar, Partha “Large Language Models (LLMs) are Next Word Predictors” *10.5281/zenodo.14290909* , 07 January 2025

[26] Platfrom OpenAI, https://platform.openai.com/docs/introduction, Access: 08-01-2025

[27] Platfrom OpenAI, https://platform.openai.com/docs/guides/optimizing-llm-accuracy, Access: 08-01-2025

[28] AWS Amazon, https://aws.amazon.com/bedrock/pricing/, Access: 08-01-2025

[29] F. R. Dogar, I. AQazi, A. R. Tariq, G. Murtaza, A. Ahmad, N. Stocking “MissIt: Using Missed Calls for Free, Extremely Low Bit-Rate Communication in Developing Regions” *https://dblp.org/rec/conf/chi/DogarQTMAS20* , 08 January 2025

[30] N. Martin, A. B. Faisal, H. Eltigani, R. Haroon, S. Lamelas, F. Dogar “LLMProxy: Reducing Cost to Access Large Language Models” *arXiv:2410.11857v1 [cs.DC]*, 08 January 2025

[31] R. Shuttleworth, J. Andreas, A. Torralba, P. Sharma “LoRA vs Full Fine-tuning: An Illusion of Equivalence” *arXiv:2410.21228v1 [cs.LG]*, 08 January 2025

[32] T. Dettmers, A. Pagnoni, A. Holtzman, “QLoRA: Efficient Finetuning of Quantized LLMs” *arXiv:2305.14314v1 [cs.LG]*, 08 January 2025

[33] Github, https://github.com/ShawhinT/YouTube-Blog/blob/main/LLMs/_slides/fine-tuning.pdf, Access: 20-12-2024

[34] AWS Amazon, https://aws.amazon.com/what-is/retrieval-augmented-generation/, Access: 14-11-2024

[35] Github, https://github.com/ShawhinT/YouTube-Blog/blob/main/LLMs/_slides/rag.pdf, Access: 29-12-2024

[36] Github, https://github.com/ShawhinT/YouTube-Blog/blob/main/LLMs/_slides/text-embeddings.pdf, Access: 29-12-2024

[37] Github, https://github.com/ShawhinT/YouTube-Blog/blob/main/LLMs/_slides/intro-to-llms.pdf, Access: 07-01-2025

[38] Github, https://github.com/ShawhinT/YouTube-Blog/blob/main/LLMs/_slides/intro-to-llms.pdf, Access: 07-01-2025

[39] OpenAI Platform, https://platform.openai.com/docs/models/gp, Access: 20-12-2024

[40] Neptune.ai (Neptune Blog), https://neptune.ai/blog/self-supervised-learning, Access: 22-12-2024

[41] Intuitive Tutorials, https://intuitivetutorial.com/2023/06/18/large-language-models-in-deep-learning/, Access: 23-12-2024

RESUME

Mohammad Amin ASLAMI

PERSONAL INFORMATION

Place of Brith : DAYKUNDI
Date of Brith : 08-12-1999
Nationality : AFGHAN
Adress : YAKUPLU MAH. 59 SK. DISKAPI: 27 KAT 3. DAIRE 4 /BEYLIKDUZU.IST
Email : aminaslamiaf@gmail.com
Languages : ENGLISH(Level: B2), TURKISH(Level: C2 - Second Lang), PERSION(Native)

RESEARCHER INFORMATION

Student Orcid ID : **XXXX-XXXX-XXXX-XXXX**
Advisor Orcid ID : **XXXX-XXXX-XXXX-XXXX**

EDUCATION INFORMATION

Master Degree : **"**Enhancing Large Language Models with Retrieval-Augmented Generation: Techniques, Challenges, and Future Directions"
Firat University, Graduate School of Natural and Applied Sciences, Software Engineering Department, 2025-2025
Advisor: Assoc. Prof. Dr. Ferhat UÇAR

Bachelor Degree : Firat University, Faculty of Tecnology, Software Engineering, 2023

High School : Meraj Private High School, Herat, 2015

RESEARCH EXPERIENCE

- ✓ Python, Rag & LLMs
- ✓ Java, Java Forms
- ✓ C#
- ✓ Html, Css and Javasscript,
- ✓ React-Native

JOB EXPERIENCE

2023 - Continuing : Salesman, A3 Altın Yapı Const Inds & Trade Ltd Com, ISTANBUL