Generative AI

Generative Pre-Trained Transformer

Kumar

Generative AI Strategy

Gen AI Strategy

Gen AI Strategy

Generative AI Strategy

Gen AI Strategy

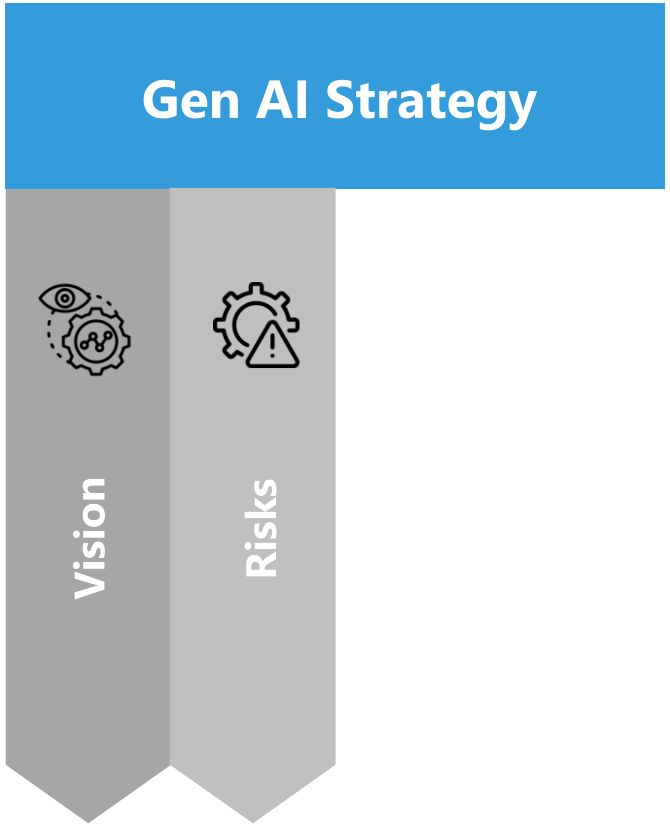
Vision

Gen AI Strategy

Vision: How might the capabilities created by GenAI impact our strategy?

Kumar

Generative AI Strategy



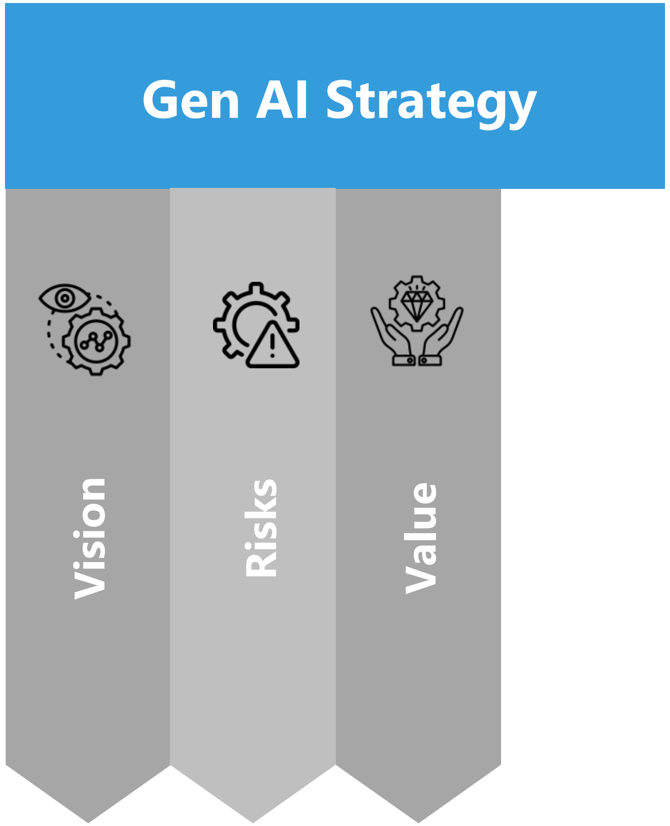
Gen AI Strategy

Vision: How might the capabilities created by GenAI impact our strategy?

Risks: How will we safeguard proprietary and sensitive data? Who will be accountable for GenAI's outputs, and to what degree?

Kumar

Generative AI Strategy



Gen AI Strategy

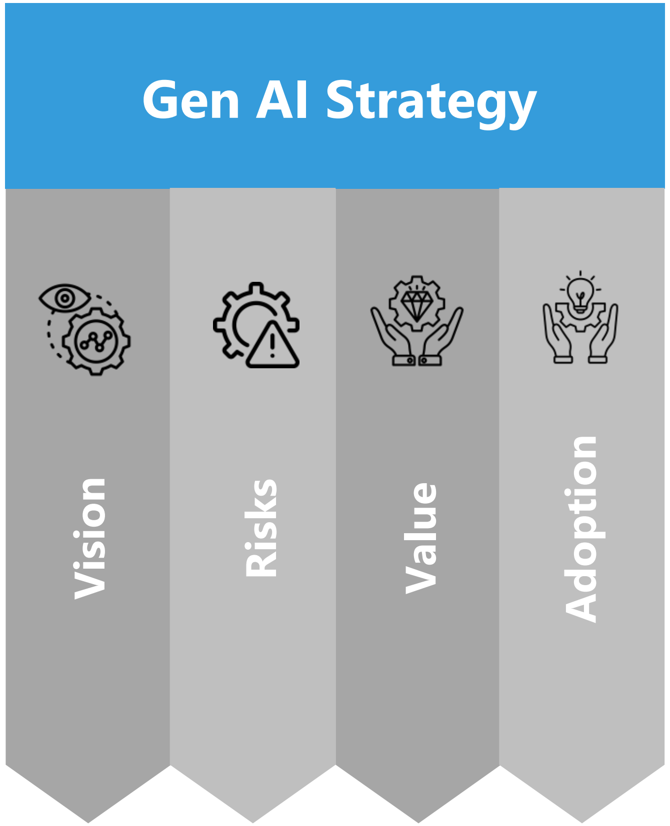
Vision: How might the capabilities created by GenAI impact our strategy?

Risks: How will we safeguard proprietary and sensitive data?
Who will be accountable for GenAI's outputs, and to what degree?

Value: What new opportunities and capabilities will it present?

Kumar

Generative AI Strategy



Gen AI Strategy

Vision: How might the capabilities created by GenAI impact our strategy?

Risks: How will we safeguard proprietary and sensitive data? Who will be accountable for GenAI's outputs, and to what degree?

Value: What new opportunities and capabilities will it present?

Adoption: How will customer or internal staff react to its use? Which ones will we need to transform or create?

Kumar

Generative AI Strategy - Vision

Vision

Kumar

Generative AI Strategy - Vision

Vision

Generative AI business goal, use case and metrics - Example

Business Goal	Use Cases	Metrics
Workforce Productivity	Paragraph comparison	Documents generation, augmentation
Customer Experience	Service/Product bundle recommendation	Products/Customer
Reduce Cost	computer code generation/conversion	Application code generation
Accelerate New Product and R&D	Create SEO list	Documents with SEO score

Generative AI Strategy - Vision

Vision

Generative AI business goal, use case and metrics - Example

Business Goal	Use Cases	Metrics
Workforce Productivity	Paragraph comparison	Documents generation, augmentation
Customer Experience	Service/Product bundle recommendation	Products/Customer
Reduce Cost	computer code generation/conversion	Application code generation
Accelerate New Product and R&D	Create SEO list	Documents with SEO score

- Which of these business goals will make sense for organisation to adopt generative AI?
- Which one of these use case should be consider as a good starting point for Gen AI ?
- What is the business maturity for these use cases ?

Kumar

Generative AI Strategy - Vision

Vision

Generative AI business goal, use case and metrics - Example

Business Goal	Use Cases	Metrics
Workforce Productivity	Paragraph comparison	Documents generation, augmentation
Customer Experience	Service/Product bundle recommendation	Products/Customer
Reduce Cost	computer code generation/conversion	Application code generation
Accelerate New Product and R&D	Create SEO list	Documents with SEO score

- Which of these business goals will make sense for organisation to adopt generative AI?
- Which one of these use case should be consider as a good starting point for Gen AI ?
- What is the business maturity for these use cases ?

Road map towards business goal and generative AI

Short term	Mid term	Long term
• Workforce Productivity • Reduce Cost	• Customer experience Is mid term rather than short term because there is still a lot of unknows when it comes to generative AI. It going to take little bit more maturity and risk management practices in place	• R&D, New products In long run generative AI will have significant impact on research & development in organisation. Like ability to create new products faster. It can also change economics of research & development

Generative AI Strategy - Vision

Vision

Generative AI business goal, use case and metrics - Example

Business Goal	Use Cases	Metrics
Workforce Productivity	Paragraph comparison	Documents generation, augmentation
Customer Experience	Service/Product bundle recommendation	Products/Customer
Reduce Cost	computer code generation/conversion	Application code generation
Accelerate New Product and R&D	Create SEO list	Documents with SEO score

- Which of these business goals will make sense for organisation to adopt generative AI?
- Which one of these use case should be consider as a good starting point for Gen AI ?
- What is the business maturity for these use cases ?

Road map towards business goal and generative AI

Short term	Mid term	Long term
• Workforce Productivity • Reduce Cost	• Customer experience Is mid term rather than short term because there is still a lot of unknows when it comes to generative AI. It going to take little bit more maturity and risk management practices in place	• R&D, New products In long run generative AI will have significant impact on research & development in organisation. Like ability to create new products faster. It can also change economics of research & development

Set AI metrics

- To measure the value of individual use cases, you'll need success metrics that tie into your overarching business goal.
- Choose the metrics as shown in below example that relate to specific key success factors and provide a timeframe in which you expect to demonstrate value.

Business Goal	Success Metrics	Date Completed
Workforce Productivity	time spent on value-added tasks	*DD-MM-YYYY*
Customer Experience	Customer satisfaction index	*DD-MM-YYYY*
Reduce Cost	Reduction in CapEX and OpEx	*DD-MM-YYYY*
Accelerate New Product and R&D	Revenue growth for product lines Number of new business initiatives	*DD-MM-YYYY*

Kumar

Generative AI Strategy - Risk

Generative AI models are significant technology advancement, but the risk they pose are non-trivial. The risks are categorised into three categories

Risk

Generative AI Strategy - Risk

Generative AI models are significant technology advancement, but the risk they pose are non-trivial. The risks are categorised into three categories

Risk

Risks	Type	Details
Regulatory	• Legal risk • Regulatory compliance and privacy laws	• Are you compliance with industry or country specific privacy. • Some of legal risks of these generative AI are not well understood today. A simple example is in particular with image or coding domain you have models like Stable Diffusion, DALL-E and codex, these models are massively trained on internet data and in many cases the datasets they are trained could be problematic their authenticity is not verified and non-permissive license.

Generative AI Strategy - Risk

Generative AI models are significant technology advancement, but the risk they pose are non-trivial. The risks are categorised into three categories

Risk

Risks	Type	Details
Regulatory	• Legal risk • Regulatory compliance and privacy laws	• Are you compliance with industry or country specific privacy. • Some of legal risks of these generative AI are not well understood today. A simple example is in particular with image or coding domain you have models like Stable Diffusion, DALL-E and codex, these models are massively trained on internet data and in many cases the datasets they are trained could be problematic their authenticity is not verified and non-permissive license.
Reputational / Ethical	• Bias and lack of transparency • AI decision risk • AI/ML attacks • Hallucinations	**Bias:** Model could be biased, and this bias could be propagated to downstream. For example, the prompt was about attorney and paralegal the model assumed attorney has a male and paralegal as female this gender bias existed in model because of data it used. **Decision risks** : the models are creating output which could be factual inaccurate this is closely associated with hallucinations. These models always give response, but we don't know if they accurate because they are black box in nature. **AI/ML attacks**: Other type of risk is if you are fine-tunning or prompt engineering these models, the risk of prompt injection is the way to persuasive the model to give inaccurate responses

Generative AI Strategy - Risk

Generative AI models are significant technology advancement, but the risk they pose are non-trivial. The risks are categorised into three categories

Risk

Risks	Type	Details
Regulatory	• Legal risk • Regulatory compliance and privacy laws	• Are you compliance with industry or country specific privacy. • Some of legal risks of these generative AI are not well understood today. A simple example is in particular with image or coding domain you have models like Stable Diffusion, DALL-E and codex, these models are massively trained on internet data and in many cases the datasets they are trained could be problematic their authenticity is not verified and non-permissive license.
Reputational / Ethical	• Bias and lack of transparency • AI decision risk • AI/ML attacks • Hallucinations	**Bias:** Model could be biased, and this bias could be propagated to downstream. For example, the prompt was about attorney and paralegal the model assumed attorney has a male and paralegal as female this gender bias existed in model because of data it used. **Decision risks** : the models are creating output which could be factual inaccurate this is closely associated with hallucinations. These models always give response, but we don't know if they accurate because they are black box in nature. **AI/ML attacks**: Other type of risk is if you are fine-tunning or prompt engineering these models, the risk of prompt injection is the way to persuasive the model to give inaccurate responses
Competencies	• Technology debit • Talent management	Competencies oriented risks are closely related to outdated technology. For example, there are lot tools which allow you to operationalise generative AI with your applications for example they would include prompt engineering tool, they might include responsible AI tool, vector databases to store mathematically representation of the data. These are next generational tools which are non-existent in most of organisation. This requires fair amount of rapid modernisation from technology standpoint And other areas are skill challenges especially in prompt engineering .

Kumar

Generative AI Strategy - Risk

Generative AI models are significant technology advancement, but the risk they pose are non-trivial. The risks are categorised into three categories

Risk

Risks	Type	Details
Regulatory	• Legal risk • Regulatory compliance and privacy laws	• Are you compliance with industry or country specific privacy. • Some of legal risks of these generative AI are not well understood today. A simple example is in particular with image or coding domain you have models like Stable Diffusion, DALL-E and codex, these models are massively trained on internet data and in many cases the datasets they are trained could be problematic their authenticity is not verified and non-permissive license.
Reputational / Ethical	• Bias and lack of transparency • AI decision risk • AI/ML attacks • Hallucinations	**Bias:** Model could be biased, and this bias could be propagated to downstream. For example, the prompt was about attorney and paralegal the model assumed attorney has a male and paralegal as female this gender bias existed in model because of data it used. **Decision risks** : the models are creating output which could be factual inaccurate this is closely associated with hallucinations. These models always give response, but we don't know if they accurate because they are black box in nature. **AI/ML attacks**: Other type of risk is if you are fine-tunning or prompt engineering these models, the risk of prompt injection is the way to persuasive the model to give inaccurate responses
Competencies	• Technology debit • Talent management	Competencies oriented risks are closely related to outdated technology. For example, there are lot tools which allow you to operationalise generative AI with your applications for example they would include prompt engineering tool, they might include responsible AI tool, vector databases to store mathematically representation of the data. These are next generational tools which are non-existent in most of organisation. This requires fair amount of rapid modernisation from technology standpoint And other areas are skill challenges especially in prompt engineering .

Who should assess and mitigate LLMs risks ?

Risks	Risk Category	Responsible	Mitigation	Action Plan	
Regulatory	Adhere to regulations e.g., Failure to identify PII e.g., FOIA analysis	CIO/CTO and CRO	Human in loop to check Document process/results/explain-ability	Understand the evolving regulatory landscape.	Enable collaboration b/w AI practitioners, legal and risk to evaluate use case feasibility and acceptable risks.
Reputational / Ethical	Secure and safe e.g., Improper tone e.g., Classification errors or offenses	CIO/CTO	Spot checks for quality Guard rails to limit error	Acknowledge the threats against AI posed by both malicious and actors within organisation.	Bolster security across enterprise security controls, data integrity and AI model monitoring.
Competencies	Technology debit	CIO/CTO	Apply training Hackathons and mentoring	Align AI strategy with cloud strategy and explore cloud as foundation for AI.	Create a technology roadmap to modernise data and analytics infrastructure and align with AI goals.

Kumar

Generative AI Strategy – Risk *continue*

Risk

Challenges of Responsible Generative AI

Challenges of Responsible Generative AI	What are the risks	How to mitigate
Toxicity	<u>*LLM can return response that can be potentially harmful or discriminatory towards protected groups or protected attributes*</u> • Toxicity at its core implies certain language or content that can be harmful or discriminatory towards certain groups, especially towards marginalized groups or protected groups.	• Careful curation of training data • Train guardrail models to filter out unwanted content • Diverse group of human annotators
Hallucination	<u>*LLM may generates factually incorrect content*</u> • Hallucinations, we think about things that are simply not true, or maybe something that seems like it could be true, but it isn't. It has no basis to it. • A lot of times we don't know what the model is actually learning and sometimes the model will try to fill gaps where it has missing data. And often this can lead to false statements or the hallucinations.	• Educate user about how generative AI works • Add disclaimers • Augment LLMs with independent, verified citation databases • Define intended/unintended use cases
Intellectual property	<u>*Ensure people aren't plagiarizing, make sure there aren't any copyright issues*</u> • It can be plagiarizing someone's previous work OR you can have copyright issues for pieces of work and content that already exists.	• Mixtures of technology, policy and legal mechanism • Machine "unlearning" – still primitive • Filtering and blocking approaches

Risk = potential harm x likelihood to occur

Unfair = (potential harm x likelihood to occur) - benefit

"in the simplest terms... [an AI] practice is unfair if it causes more harm than good."

Kumar

Generative AI Strategy – Value and Adoption *continue*

Value and Adoption

How much will generative AI cost?

Do we build our own or license or customizable proprietary models with foundation model platforms and work vendors and partners?

Generative AI Strategy – Value and Adoption *continue*

Value and Adoption

How much will generative AI cost?

Do we build our own or license or customizable proprietary models with foundation model platforms and work vendors and partners?

Content Type	
Enterprise Control Level	
Solution Type	
Required Technologies	
Cost	

Kumar

Generative AI Strategy – Value and Adoption *continue*

Value and Adoption

How much will generative AI cost?

Do we build our own or license or customizable proprietary models with foundation model platforms and work vendors and partners?

Content Type	**Non-Sensitive Text**
Enterprise Control Level	Non controls required
Solution Type	Chat GPT
Required Technologies	Open AI hosted application
Cost	

Generative AI Strategy – Value and Adoption *continue*

Value and Adoption

How much will generative AI cost?

Do we build our own or license or customizable proprietary models with foundation model platforms and work vendors and partners?

Content Type	Non-Sensitive Text	PII / Enterprise IP included
Enterprise Control Level	Non controls required	LLM with privacy
Solution Type	Chat GPT	LLM API accessed via application
Required Technologies	Open AI hosted application	Cloud instance with LLM APIs
Cost		

Kumar

Generative AI Strategy – Value and Adoption *continue*

Value and Adoption

How much will generative AI cost?

Do we build our own or license or customizable proprietary models with foundation model platforms and work vendors and partners?

Content Type	**Non-Sensitive Text**	**PII / Enterprise IP included**	**Enterprise Data and model instruction needed included**
Enterprise Control Level	Non controls required	LLM with privacy	LLM with privacy, polices and data
Solution Type	Chat GPT	LLM API accessed via application	LLM API with standard privacy policy and incident data injection
Required Technologies	Open AI hosted application	Cloud instance with LLM APIs	Cloud instance with LLM APIs, prompt engineering, custom polices, indexed databases
Cost			

Kumar

Generative AI Strategy – Value and Adoption *continue*

Value and Adoption

How much will generative AI cost?

Do we build our own or license or customizable proprietary models with foundation model platforms and work vendors and partners?

Content Type	Non-Sensitive Text	PII / Enterprise IP included	Enterprise Data and model instruction needed included	Model Fine Tunning to improve use case / performance
Enterprise Control Level	Non controls required	LLM with privacy	LLM with privacy, polices and data	LLM with privacy polices, data and added model layers
Solution Type	Chat GPT	LLM API accessed via application	LLM API with standard privacy policy and incident data injection	Modified LLM with transfer learning / Added layers to adjust the model output
Required Technologies	Open AI hosted application	Cloud instance with LLM APIs	Cloud instance with LLM APIs, prompt engineering, custom polices, indexed databases	Licensed customisable model / Proprietary model, data, ML platform
Cost				

Kumar

Generative AI Strategy – Value and Adoption *continue*

Value and Adoption

How much will generative AI cost?

Do we build our own or license or customizable proprietary models with foundation model platforms and work vendors and partners?

Content Type	**Non-Sensitive Text**	**PII / Enterprise IP included**	**Enterprise Data and model instruction needed included**	**Model Fine Tunning to improve use case / performance**	**Custom Model created for unique use case**
Enterprise Control Level	Non controls required	LLM with privacy	LLM with privacy, polices and data	LLM with privacy polices, data and added model layers	Enterprise hosted costume differentiated model
Solution Type	Chat GPT	LLM API accessed via application	LLM API with standard privacy policy and incident data injection	Modified LLM with transfer learning / Added layers to adjust the model output	Custom build LLM using enterprise data
Required Technologies	Open AI hosted application	Cloud instance with LLM APIs	Cloud instance with LLM APIs, prompt engineering, custom polices, indexed databases	Licensed customisable model / Proprietary model, data, ML platform	Custom build proprietary model, data , ML platform
Cost					

Kumar

Generative AI Strategy – Value and Adoption

Value and Adoption

Kumar

Generative AI Strategy – Value and Adoption

Value and Adoption

Key questions for Gen AI value and adoptions
How can we prioritise Gen AI use case ? What are the criteria's we can use ?
How will we consume Generative AI services (embedded APPS vs Cloud APIs vs. Prompt Engineering vs. Fine tunning vs. Build Models)?
What new Gen AI roles and skills will we require?

Kumar

Generative AI Strategy – Value and Adoption

Value and Adoption

Key questions for Gen AI value and adoptions
How can we prioritise Gen AI use case ? What are the criteria's we can use ?
How will we consume Generative AI services (embedded APPS vs Cloud APIs vs. Prompt Engineering vs. Fine tunning vs. Build Models)?
What new Gen AI roles and skills will we require?

Framework to prioritise Gen AI projects

	Technical Feasibility		
Use Case	Access to Labelled Data	Architecture and Technology Feasibility	Skills availability
Use case-1	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE
Use case-2	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE

Generative AI Strategy – Value and Adoption

Value and Adoption

Key questions for Gen AI value and adoptions
How can we prioritise Gen AI use case ? What are the criteria's we can use ?
How will we consume Generative AI services (embedded APPS vs Cloud APIs vs. Prompt Engineering vs. Fine tunning vs. Build Models)?
What new Gen AI roles and skills will we require?

Framework to prioritise Gen AI projects

	Technical Feasibility			Business Value		
Use Case	Access to Labelled Data	Architecture and Technology Feasibility	Skills availability	Aligns with Business goals	Sponsor Support	Measurable KPIs
Use case-1	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE
Use case-2	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE

Generative AI Strategy – Value and Adoption

Value and Adoption

Key questions for Gen AI value and adoptions
How can we prioritise Gen AI use case ? What are the criteria's we can use ?
How will we consume Generative AI services (embedded APPS vs Cloud APIs vs. Prompt Engineering vs. Fine tunning vs. Build Models)?
What new Gen AI roles and skills will we require?

Framework to prioritise Gen AI projects

	Technical Feasibility			Business Value			Overall Value		
Use Case	Access to Labelled Data	Architecture and Technology Feasibility	Skills availability	Aligns with Business goals	Sponsor Support	Measurable KPIs	Overall Business Value (1 to 10)	Overall Technical Feasibility (1 to 10)	Score/Ranking
Use case-1	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	5	2	
Use case-2	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	3	10	

Kumar

Generative AI Strategy – Value and Adoption

Value and Adoption

Key questions for Gen AI value and adoptions
How can we prioritise Gen AI use case ? What are the criteria's we can use ?
How will we consume Generative AI services (embedded APPS vs Cloud APIs vs. Prompt Engineering vs. Fine tunning vs. Build Models)?
What new Gen AI roles and skills will we require?

Framework to prioritise Gen AI projects

	Technical Feasibility			Business Value			Overall Value		
Use Case	Access to Labelled Data	Architecture and Technology Feasibility	Skills availability	Aligns with Business goals	Sponsor Support	Measurable KPIs	Overall Business Value (1 to 10)	Overall Technical Feasibility (1 to 10)	Score/Ranking
Use case-1	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	5	2	
Use case-2	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	3	10	

Different approaches and comparison to deploy generative AI - example

Kumar

Generative AI Strategy – Value and Adoption

Value and Adoption

Key questions for Gen AI value and adoptions
How can we prioritise Gen AI use case ? What are the criteria's we can use ?
How will we consume Generative AI services (embedded APPS vs Cloud APIs vs. Prompt Engineering vs. Fine tunning vs. Build Models)?
What new Gen AI roles and skills will we require?

Framework to prioritise Gen AI projects

	Technical Feasibility			Business Value			Overall Value		
Use Case	Access to Labelled Data	Architecture and Technology Feasibility	Skills availability	Aligns with Business goals	Sponsor Support	Measurable KPIs	Overall Business Value (1 to 10)	Overall Technical Feasibility (1 to 10)	Score/Ranking
Use case-1	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	5	2	
Use case-2	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	3	10	

Different approaches and comparison to deploy generative AI - example

Approach / Decision Factor	Consumer Generative Models as Embedded ISV APIs (e.g. ChatGPT via O365)	Embed Model APIs into Custom APP	Extent Gen AI Models		Build custom models
			Prompt Engineering	Fine Tunning	

Kumar

Generative AI Strategy – Value and Adoption

Value and Adoption

Key questions for Gen AI value and adoptions
How can we prioritise Gen AI use case ? What are the criteria's we can use ?
How will we consume Generative AI services (embedded APPS vs Cloud APIs vs. Prompt Engineering vs. Fine tunning vs. Build Models)?
What new Gen AI roles and skills will we require?

Framework to prioritise Gen AI projects

	Technical Feasibility			Business Value			Overall Value		
Use Case	Access to Labelled Data	Architecture and Technology Feasibility	Skills availability	Aligns with Business goals	Sponsor Support	Measurable KPIs	Overall Business Value (1 to 10)	Overall Technical Feasibility (1 to 10)	Score/Ranking
Use case-1	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	5	2	
Use case-2	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	YES NO MAYBE	3	10	

Different approaches and comparison to deploy generative AI - example

Approach / Decision Factor	Consumer Generative Models as Embedded ISV APIs (e.g. ChatGPT via O365)	Embed Model APIs into Custom APP	Extent Gen AI Models		Build custom models
			Prompt Engineering	Fine Tunning	
Cost					
Organisation / Domain Knowledge					
Security & Privacy					
Model Quality (Hallucinations , performance)					
Implementation Complexity					

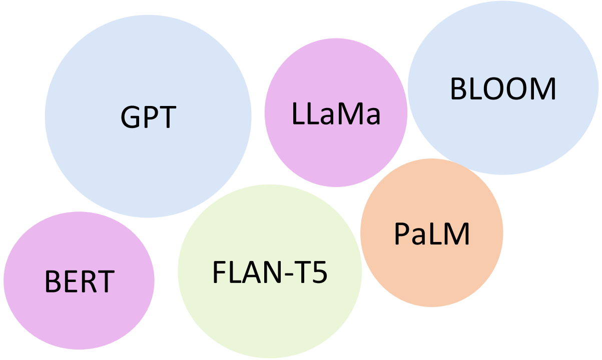
Kumar

Overview of LLMs

Overview of LLMs

What are Large Language models (LLMs)

- LLMs are deep neural network models.
- They are capable of creating content that mimics or approximates human ability.

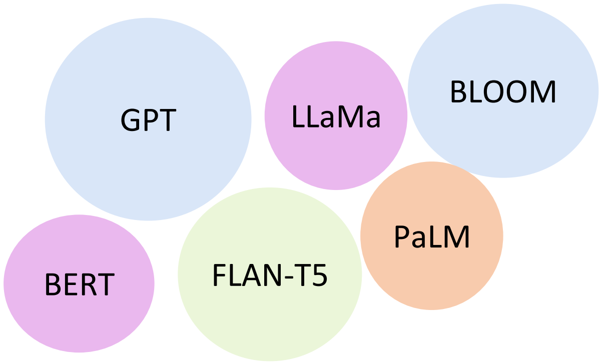


Examples LLM model

Overview of LLMs

What are Large Language models (LLMs)

- LLMs are deep neural network models.
- They are capable of creating content that mimics or approximates human ability.



Examples LLM model

How can you interact with LMM

The way you interact with language models is quite different than other machine learning and programming paradigms. In those cases, you write computer code with formalized syntax to interact with libraries and APIs. In contrast, large language models are able to take natural language or human written instructions and perform tasks much as a human would.

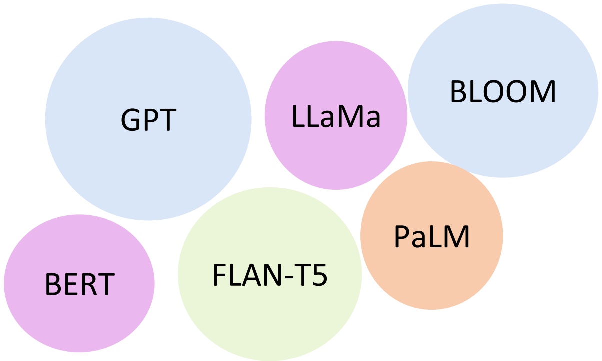
The text that you pass to an LLM is known as a prompt. The space or memory that is available to the prompt is called the context window, and this is typically large enough for a few thousand words but differs from model to model.

Kumar

Overview of LLMs

What are Large Language models (LLMs)

- LLMs are deep neural network models.
- They are capable of creating content that mimics or approximates human ability.



Examples LLM model

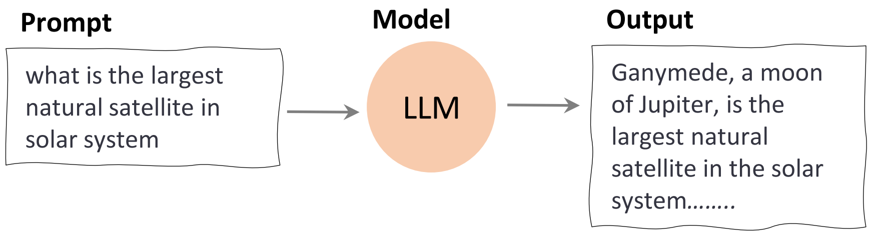
How can you interact with LMM

The way you interact with language models is quite different than other machine learning and programming paradigms. In those cases, you write computer code with formalized syntax to interact with libraries and APIs. In contrast, large language models are able to take natural language or human written instructions and perform tasks much as a human would.

The text that you pass to an LLM is known as a prompt. The space or memory that is available to the prompt is called the context window, and this is typically large enough for a few thousand words but differs from model to model.

Example of interaction with LLM

You ask the model to determine where Ganymede is located in the solar system. The prompt is passed to the model, the model then predicts the next words, and because your prompt contained a question, this model generates an answer. The output of the model is called a completion, and the act of using the model to generate text is known as inference. The completion is the response to original prompt, followed by the generated text.



Kumar

LLM use cases and tasks

LLMs and generative AI are focused on chats tasks. Next word prediction is the base concept behind a number of different capabilities. However, this conceptually simple technique can be used for a variety of other tasks within text generation. Below are few use case examples :

LLM use cases and tasks

LLMs and generative AI are focused on chats tasks. Next word prediction is the base concept behind a number of different capabilities. However, this conceptually simple technique can be used for a variety of other tasks within text generation. Below are few use case examples :

Use cases	**Use case task**
Example 1: You can ask a model to write an essay based on a prompt, to summarize conversations	Essay Writer Input : Write a two short paragraph essay for title “AI Ethics” Generate Output : The development and deployment of AI systems raise profound questions about privacy, bias.... AI ethics necessitates a careful balance between technological progress and moral values.....

LLM use cases and tasks

LLMs and generative AI are focused on chats tasks. Next word prediction is the base concept behind a number of different capabilities. However, this conceptually simple technique can be used for a variety of other tasks within text generation. Below are few use case examples :

Use cases	**Use case task**
Example 1: You can ask a model to write an essay based on a prompt, to summarize conversations	Essay Writer Input : Write a two short paragraph essay for title “AI Ethics” Generate Output : The development and deployment of AI systems raise profound questions about privacy, bias.... AI ethics necessitates a careful balance between technological progress and moral values.....
Example 2: You provide the dialogue as part of your prompt and the model uses this data along with its understanding of natural language to generate a summary.	Summarise Input : Support.txt Generate Output : In the chat session, support efficiently and effectively assists person-1, who was initially unable to trade stocks due insufficient funds. leading to positive customer service experience

Kumar

LLM use cases and tasks

LLMs and generative AI are focused on chats tasks. Next word prediction is the base concept behind a number of different capabilities. However, this conceptually simple technique can be used for a variety of other tasks within text generation. Below are few use case examples :

Use cases	**Use case task**
Example 1: You can ask a model to write an essay based on a prompt, to summarize conversations	**Essay Writer** Input : Write a two short paragraph essay for title “AI Ethics” [Generate] Output : The development and deployment of AI systems raise profound questions about privacy, bias.... AI ethics necessitates a careful balance between technological progress and moral values.....
Example 2: You provide the dialogue as part of your prompt and the model uses this data along with its understanding of natural language to generate a summary.	**Summarise** Input : Support.txt [Generate] Output : In the chat session, support efficiently and effectively assists person-1, who was initially unable to trade stocks due insufficient funds. leading to positive customer service experience
Example 3: You can use models for a variety of translation tasks from traditional translation between two different languages, such as English and Spanish or French and German.	**Translate** Input : translate "time traveller" to Spanish [Generate] Output : "viajero del tiempo."

Kumar

LLM use cases and tasks - *continue*

LLM use cases and tasks - *continue*

Use cases	**Use case task**
Example 3: Translate natural language to machine code. For example, you could ask a model to write some Python code that will return the mean of every column in a data frame and the model will generate code that you can pass to an interpreter.	Code AI Input : write python code that will return the mean of every column in a dataframe Generate Output :

```python
import pandas as pd

# Create a sample DataFrame
data = {'A': [1, 2, 3, 4, 5],
        'B': [6, 7, 8, 9, 10],
        'C': [11, 12, 13, 14, 15]}

df = pd.DataFrame(data)

# Calculate the mean of every column
column_means = df.mean()
```

LLM use cases and tasks - *continue*

Use cases | **Use case task**

Example 3: Translate natural language to machine code. For example, you could ask a model to write some Python code that will return the mean of every column in a data frame and the model will generate code that you can pass to an interpreter.

Code AI

Input :

write python code that will return the mean of every column in a dataframe

Generate

Output :

```
import pandas as pd

# Create a sample DataFrame
data = {'A': [1, 2, 3, 4, 5],
        'B': [6, 7, 8, 9, 10],
        'C': [11, 12, 13, 14, 15]}

df = pd.DataFrame(data)

# Calculate the mean of every column
column_means = df.mean()
```

Example 4: You can use LLMs to carry out smaller, focused tasks like information retrieval. In this example, you ask the model to identify all of the names, dates, places, and people identified in a news article. This is known as named entity recognition, like classification problem.

Entity Extraction

Input :

identify all of the companies name, date and places in "so we started NetSafe to redefine Cloud, Network and Data Security. Since 2019, we have built the market-leading cloud security company and an award-winning culture powered by hundreds of employees spread across offices in Santa Clara, St. Louis, Bangalore, London, Melbourne, and Tokyo."

Generate

Output :

1. **Company Name:** NetSafe
2. **Date:** Since 2019
3. **Places:**
   - Santa Clara
   - St. Louis
   - Bangalore
   - London
   - Melbourne
   - Tokyo

LLM use cases and tasks - *continue*

Use cases	**Use case task**
Example 3: Translate natural language to machine code. For example, you could ask a model to write some Python code that will return the mean of every column in a data frame and the model will generate code that you can pass to an interpreter.	**Code AI** Input : write python code that will return the mean of every column in a dataframe Generate Output : see code below
Example 4: You can use LLMs to carry out smaller, focused tasks like information retrieval. In this example, you ask the model to identify all of the names, dates, places, and people identified in a news article. This is known as named entity recognition, like classification problem.	**Entity Extraction** Input : identify all of the companies name, date and places in "so we started NetSafe to redefine Cloud, Network and Data Security. Since 2019, we have built the market-leading cloud security company and an award-winning culture powered by hundreds of employees spread across offices in Santa Clara, St. Louis, Bangalore, London, Melbourne, and Tokyo." Generate Output : 1.**Company Name:** NetSafe 2.**Date:** Since 2019 3.**Places:** • Santa Clara • St. Louis • Bangalore • London • Melbourne • Tokyo
Example 5: An area of active development is augmenting LLMs by connecting them to external data sources or using them to invoke external APIs. You can use this ability to provide the model with information it doesn't know from its pre-training and to enable your model to power interactions with the real-world.	**External Data Retrieval and Augmentation** Input : Medicare Benefits Schedule(mbs) item number knee replacement and price Generate *Formatting API query.. | Making request.... | Processing response... Done.* Output : see table below

Code AI output:

```python
import pandas as pd

# Create a sample DataFrame
data = {'A': [1, 2, 3, 4, 5],
        'B': [6, 7, 8, 9, 10],
        'C': [11, 12, 13, 14, 15]}

df = pd.DataFrame(data)

# Calculate the mean of every column
column_means = df.mean()
```

External Data Retrieval and Augmentation output:

MBS item number	Surgery type	Medicare rebate (75%)
49518	Total knee arthroplasty (total replacement)	$1,044.90

Kumar

High level working of Transformer

LLMs are boarder class of transformer models. The underlying transformer is a set of neural networks that consist of an encoder, decoder with self-attention and multi-attention (or multi-headed attention) mechanism.

High level working of Transformer

LLMs are boarder class of transformer models. The underlying transformer is a set of neural networks that consist of an encoder, decoder with self-attention and multi-attention (or multi-headed attention) mechanism.

Transformer’s architecture

In the earlier generation of recurrent neural network **(RNN)**s , the algorithms learns to each word next to its neighbour as shown rather than whole sentence. *(optional Next-Word-Prediction demo)*

High level working of Transformer

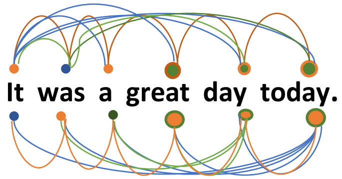
LLMs are boarder class of transformer models. The underlying transformer is a set of neural networks that consist of an encoder, decoder with self-attention and multi-attention (or multi-headed attention) mechanism.

Transformer's architecture

In the earlier generation of recurrent neural network **(RNN)**s , the algorithms learns to each word next to its neighbour as shown rather than whole sentence. *(optional Next-Word-Prediction demo)*

In **transformer architecture** the power lies in its ability to learn the relevance and context of all of the words in a sentence as shown. Generative Pre-Trained Transformers (GPT) attention weights are calculated between each word and every other word.

Transformer applies attention weights to those relationships so that the model learns the relevance of each word to each other words no matter where they are in the input. This gives the algorithm the ability to learn wider context of the document/sentence/prompt.



Attention map weights. In this example you can see the work **today** is strongly connected with or paying attention to the word **great** and **day**

High level working of Transformer

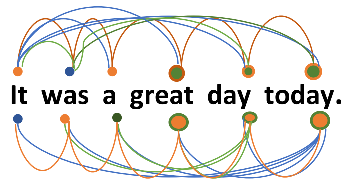
LLMs are boarder class of transformer models. The underlying transformer is a set of neural networks that consist of an encoder, decoder with self-attention and multi-attention (or multi-headed attention) mechanism.

Transformer's architecture

In the earlier generation of recurrent neural network (**RNN)**s , the algorithms learns to each word next to its neighbour as shown rather than whole sentence. *(optional Next-Word-Prediction demo)*

In **transformer architecture** the power lies in its ability to learn the relevance and context of all of the words in a sentence as shown. Generative Pre-Trained Transformers (GPT) attention weights are calculated between each word and every other word.

Transformer applies attention weights to those relationships so that the model learns the relevance of each word to each other words no matter where they are in the input. This gives the algorithm the ability to learn wider context of the document/sentence/prompt.

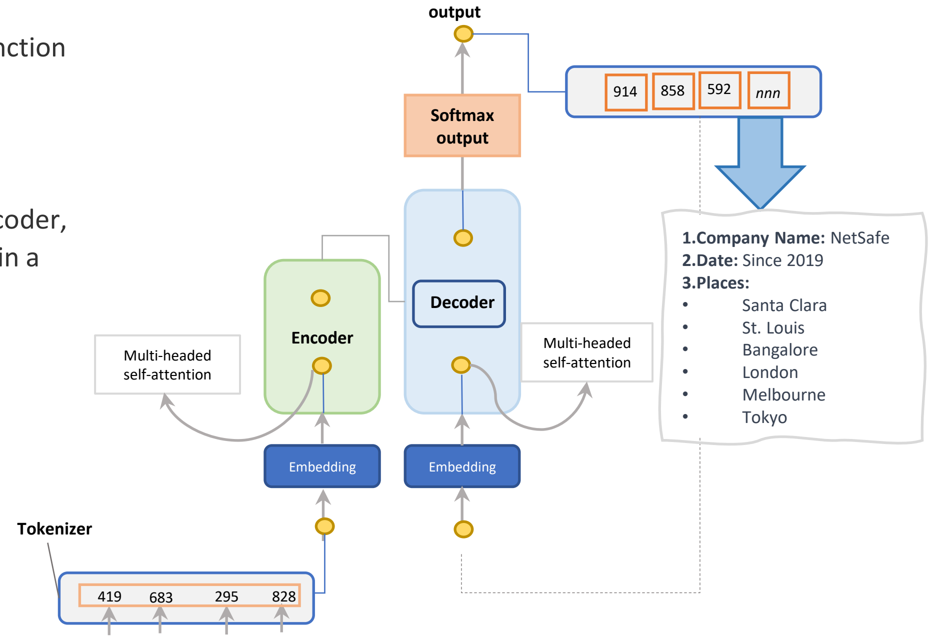


Attention map weights. In this example you can see the work **today** is strongly connected with or paying attention to the word **great** and **day**

High level working of Transformer

The transformer architecture is split into two distinct parts, the **encoder** and **decoder**. These components work in conjunction with each other, and they share a number of similarities.

- ML models are just big statistical calculators, and they work with numbers. So before passing texts into the model to process, you must first tokenize the words.
- The encoder encodes input sequences into a deep representation of the structure and meaning of the input. The decoder, working from input token triggers, uses the encoder's contextual understanding to generate new tokens. It does this in a loop until some stop condition has been reached.



input: identify all of the companies name, date and places in "so we started NetSafe to redefine Cloud, Network and Data Security. Since 2019, we have built the market-leading cloud security company and an award-winning culture powered by hundreds of employees spread across offices in Santa Clara, St. Louis, Bangalore, London, Melbourne, and Tokyo."

High level working of Transformer

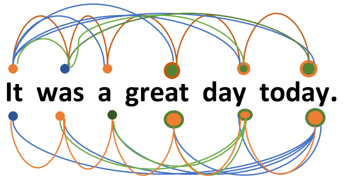
LLMs are boarder class of transformer models. The underlying transformer is a set of neural networks that consist of an encoder, decoder with self-attention and multi-attention (or multi-headed attention) mechanism.

Transformer's architecture

In the earlier generation of recurrent neural network (**RNN)**s , the algorithms learns to each word next to its neighbour as shown rather than whole sentence. *(optional [Next-Word-Prediction demo](#))*

In **transformer architecture** the power lies in its ability to learn the relevance and context of all of the words in a sentence as shown. Generative Pre-Trained Transformers (GPT) attention weights are calculated between each word and every other word.

Transformer applies attention weights to those relationships so that the model learns the relevance of each word to each other words no matter where they are in the input. This gives the algorithm the ability to learn wider context of the document/sentence/prompt.



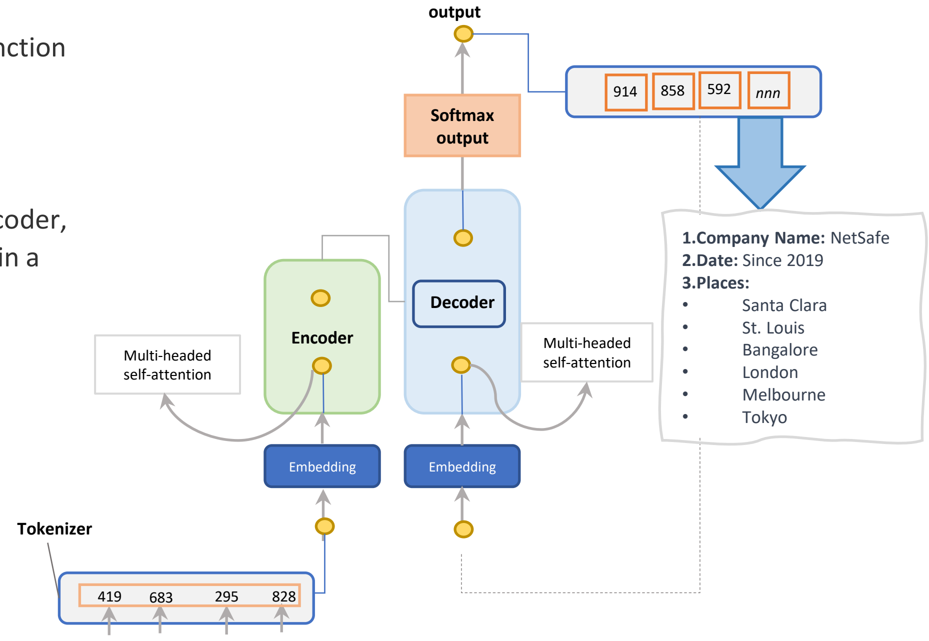
Attention map weights. In this example you can see the work **today** is strongly connected with or paying attention to the word **great** and **day**

High level working of Transformer

The transformer architecture is split into two distinct parts, the **encoder** and **decoder**. These components work in conjunction with each other, and they share a number of similarities.

- ML models are just big statistical calculators, and they work with numbers. So before passing texts into the model to process, you must first tokenize the words.
- The encoder encodes input sequences into a deep representation of the structure and meaning of the input. The decoder, working from input token triggers, uses the encoder's contextual understanding to generate new tokens. It does this in a loop until some stop condition has been reached.

Encoder-only models work as sequence-to-sequence models, the input sequence and the output sequence or the same length. Their use is less common these days, but by adding additional layers to the architecture, you can train encoder-only models to perform classification tasks such as sentiment analysis but is an example of an encoder-only model.



input: identify all of the companies name, date and places in "so we started NetSafe to redefine Cloud, Network and Data Security. Since 2019, we have built the market-leading cloud security company and an award-winning culture powered by hundreds of employees spread across offices in Santa Clara, St. Louis, Bangalore, London, Melbourne, and Tokyo."

[Kumar](#)

High level working of Transformer

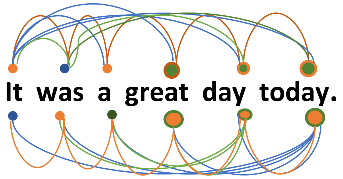
LLMs are boarder class of transformer models. The underlying transformer is a set of neural networks that consist of an encoder, decoder with self-attention and multi-attention (or multi-headed attention) mechanism.

Transformer’s architecture

In the earlier generation of recurrent neural network **(RNN)**s , the algorithms learns to each word next to its neighbour as shown rather than whole sentence. *(optional Next-Word-Prediction demo)*

In **transformer architecture** the power lies in its ability to learn the relevance and context of all of the words in a sentence as shown. Generative Pre-Trained Transformers (GPT) attention weights are calculated between each word and every other word.

Transformer applies attention weights to those relationships so that the model learns the relevance of each word to each other words no matter where they are in the input. This gives the algorithm the ability to learn wider context of the document/sentence/prompt.



Attention map weights. In this example you can see the work **today** is strongly connected with or paying attention to the word **great** and **day**

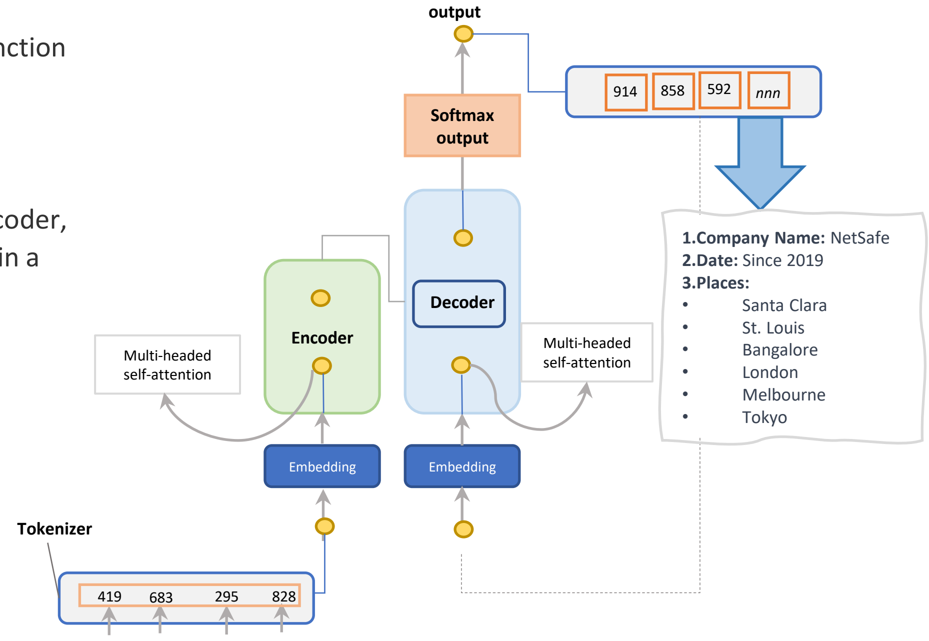
High level working of Transformer

The transformer architecture is split into two distinct parts, the **encoder** and **decoder**. These components work in conjunction with each other, and they share a number of similarities.

- ML models are just big statistical calculators, and they work with numbers. So before passing texts into the model to process, you must first tokenize the words.
- The encoder encodes input sequences into a deep representation of the structure and meaning of the input. The decoder, working from input token triggers, uses the encoder's contextual understanding to generate new tokens. It does this in a loop until some stop condition has been reached.

Encoder-only models work as sequence-to-sequence models, the input sequence and the output sequence or the same length. Their use is less common these days, but by adding additional layers to the architecture, you can train encoder-only models to perform classification tasks such as sentiment analysis but is an example of an encoder-only model.

Encoder-decoder models, perform well on sequence-to-sequence tasks such as translation, where the input sequence and the output sequence can be different lengths. You can also scale and train this type of model to perform general text generation tasks example BART and T5



input: identify all of the companies name, date and places in "so we started NetSafe to redefine Cloud, Network and Data Security. Since 2019, we have built the market-leading cloud security company and an award-winning culture powered by hundreds of employees spread across offices in Santa Clara, St. Louis, Bangalore, London, Melbourne, and Tokyo."

Kumar

High level working of Transformer

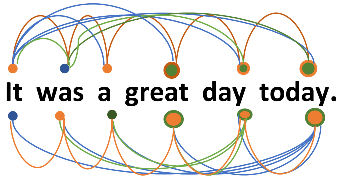
LLMs are boarder class of transformer models. The underlying transformer is a set of neural networks that consist of an encoder, decoder with self-attention and multi-attention (or multi-headed attention) mechanism.

Transformer's architecture

In the earlier generation of recurrent neural network (**RNN**)s , the algorithms learns to each word next to its neighbour as shown rather than whole sentence. *(optional [Next-Word-Prediction demo](#))*

In **transformer architecture** the power lies in its ability to learn the relevance and context of all of the words in a sentence as shown. Generative Pre-Trained Transformers (GPT) attention weights are calculated between each word and every other word.

Transformer applies attention weights to those relationships so that the model learns the relevance of each word to each other words no matter where they are in the input. This gives the algorithm the ability to learn wider context of the document/sentence/prompt.



Attention map weights. In this example you can see the work **today** is strongly connected with or paying attention to the word **great** and **day**

High level working of Transformer

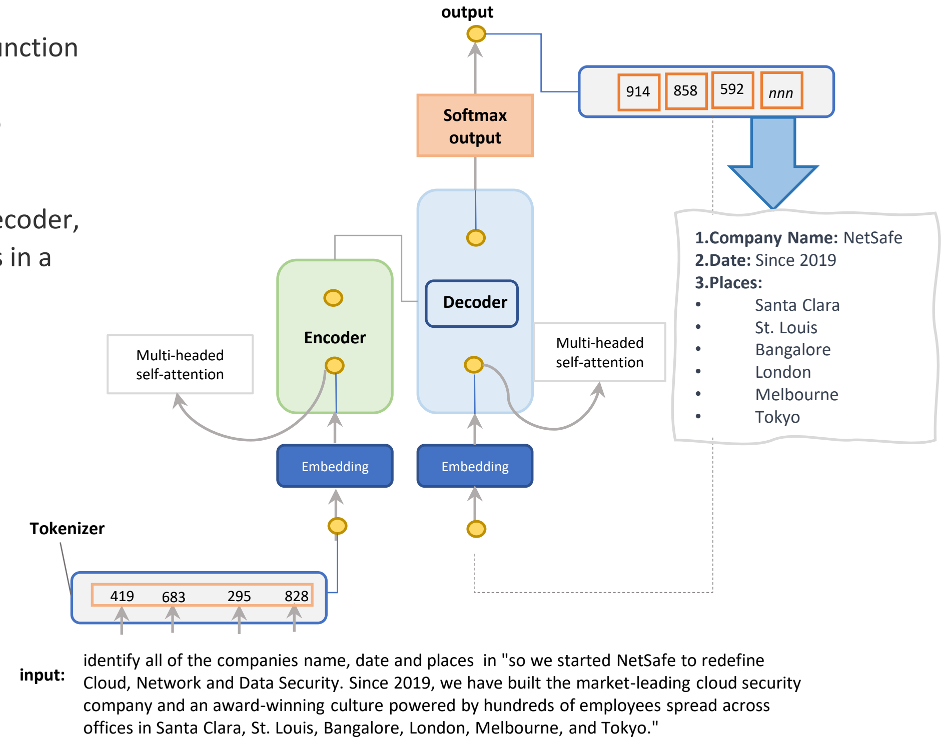
The transformer architecture is split into two distinct parts, the **encoder** and **decoder**. These components work in conjunction with each other, and they share a number of similarities.

- ML models are just big statistical calculators, and they work with numbers. So before passing texts into the model to process, you must first tokenize the words.
- The encoder encodes input sequences into a deep representation of the structure and meaning of the input. The decoder, working from input token triggers, uses the encoder's contextual understanding to generate new tokens. It does this in a loop until some stop condition has been reached.

Encoder-only models work as sequence-to-sequence models, the input sequence and the output sequence or the same length. Their use is less common these days, but by adding additional layers to the architecture, you can train encoder-only models to perform classification tasks such as sentiment analysis but is an example of an encoder-only model.

Encoder-decoder models, perform well on sequence-to-sequence tasks such as translation, where the input sequence and the output sequence can be different lengths. You can also scale and train this type of model to perform general text generation tasks example BART and T5

Decoder-only models are some of the most commonly used today. Their capabilities have grown. These models can now generalize to most tasks. Popular decoder-only models include the GPT family of models, BLOOM, Jurassic, LLaMA, CTRL, Transformer XL and many more.



[Kumar](#)

Generative AI Project Life Cycle

Generative AI Project Life Cycle

Scope

Define the use case

Kumar

Generative AI Project Life Cycle

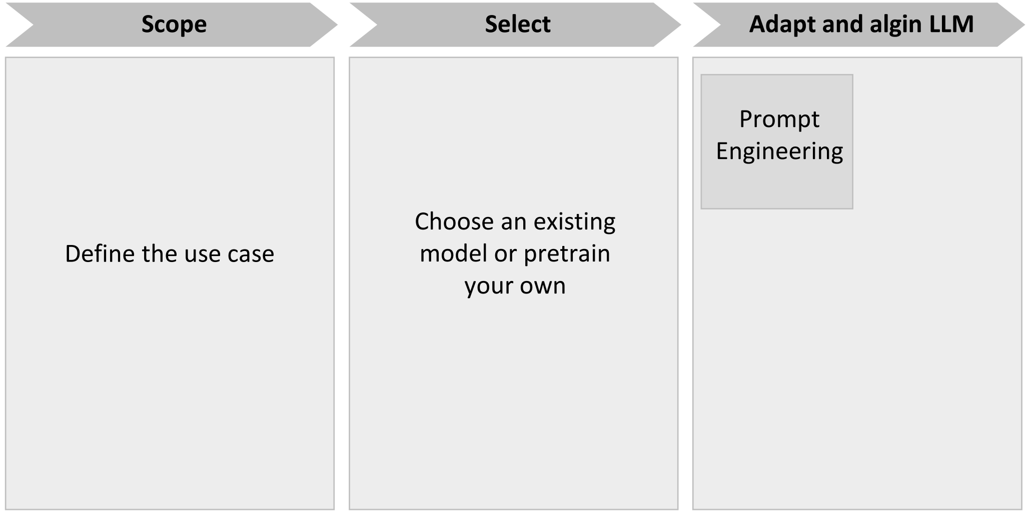
Scope	Select
Define the use case	Choose an existing model or pretrain your own

Kumar

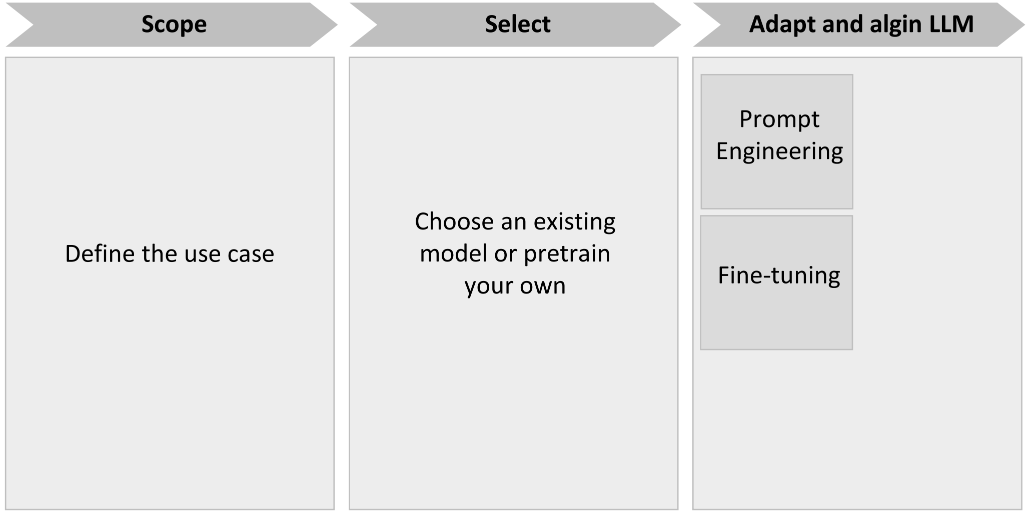
Generative AI Project Life Cycle

Scope	Select	Adapt and algin LLM
Define the use case	Choose an existing model or pretrain your own	

Generative AI Project Life Cycle



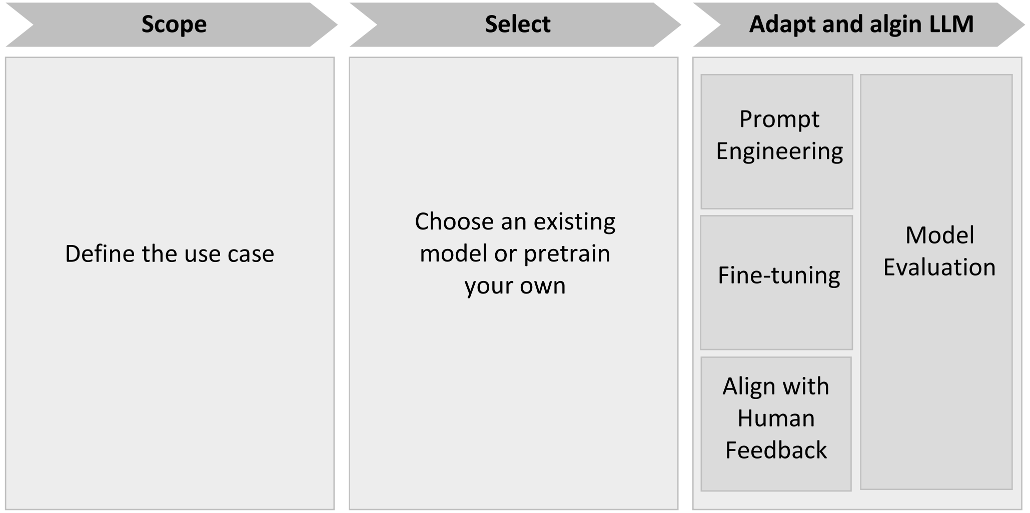
Generative AI Project Life Cycle



Generative AI Project Life Cycle

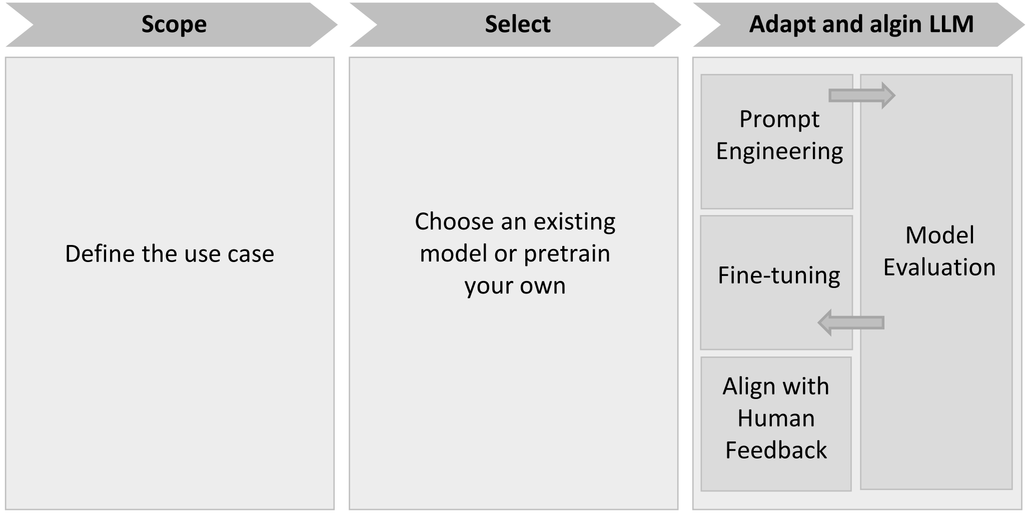
Scope	Select	Adapt and algin LLM
Define the use case	Choose an existing model or pretrain your own	Prompt Engineering Fine-tuning Align with Human Feedback

Generative AI Project Life Cycle



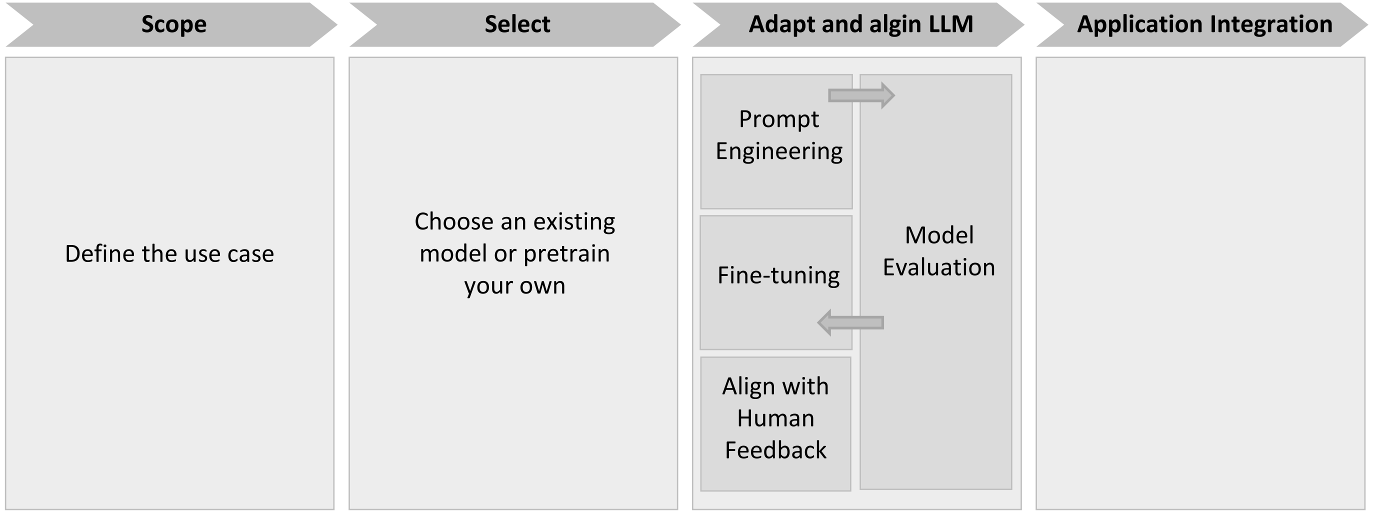
Kumar

Generative AI Project Life Cycle



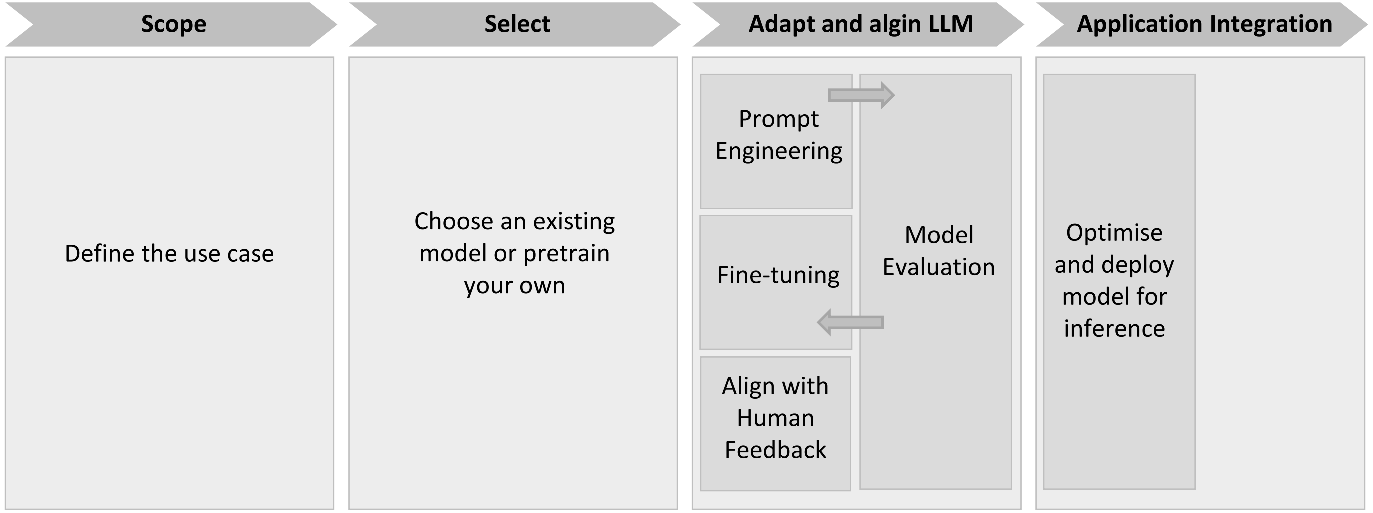
Kumar

Generative AI Project Life Cycle



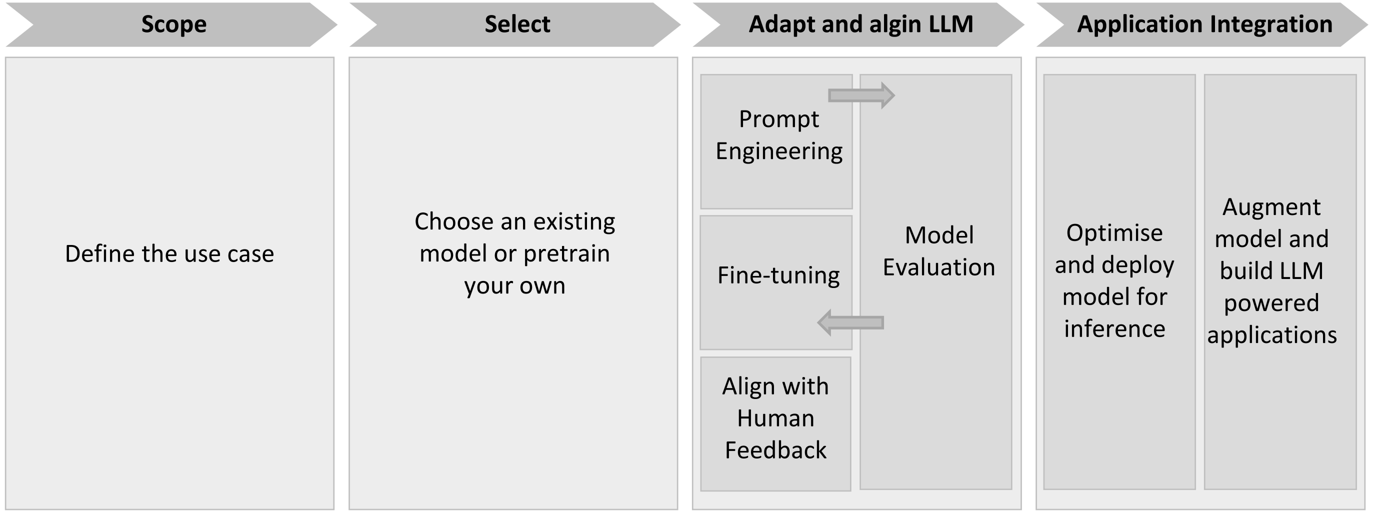
Kumar

Generative AI Project Life Cycle



Kumar

Generative AI Project Life Cycle



Kumar

Generative AI Project Life Cycle - Choose existing model or pre-train

Generative AI Project Life Cycle - Choose existing model or pre-train

Select
Existing models or pre-train

Consideration for choosing LLM model

- Your first choice will be to work with an existing model. *(if it aligns to your governance and privacy polices)*
- There are specific circumstances where training your own model from scratch might be advantageous.

Foundation model

Pretrained LLM

Train your own model

Custom LLM

Select
Existing models or pre-train

Generative AI Project Life Cycle - Choose existing model or pre-train

Consideration for choosing LLM model

- Your first choice will be to work with an existing model. *(if it aligns to your governance and privacy polices)*
- There are specific circumstances where training your own model from scratch might be advantageous.

Foundation model	**Train your own model**
Pretrained LLM	Custom LLM

LLM model hubs

- In general you'll begin the process of developing your application using an existing foundation model.
- Many open-source models are available to use in application.
- The developers of some of the major frameworks for building generative AI applications like Hugging Face and PyTorch , have curated hubs where you can browse these models.
- A really useful feature of these hubs is the inclusion of model cards, that describe important details including the best use cases for each model, how it was trained, and known limitations of LLM e.g. [Model Card for FLAN-T5 base]

Model Card for FLAN-T5 base

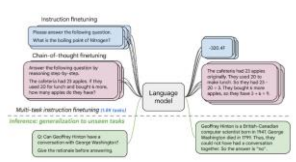


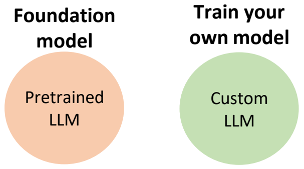
Table of content

1. Model Details
2. Usage
3. Uses
4. Bias, Risks and Limitation
5. Training Details
6. Evaluation
7. Environmental Impact
8. …..

Generative AI Project Life Cycle - Choose existing model or pre-train

Consideration for choosing LLM model

- Your first choice will be to work with an existing model. *(if it aligns to your governance and privacy polices)*
- There are specific circumstances where training your own model from scratch might be advantageous.



LLM model hubs

- In general you'll begin the process of developing your application using an existing foundation model.
- Many open-source models are available to use in application.
- The developers of some of the major frameworks for building generative AI applications like Hugging Face and PyTorch , have curated hubs where you can browse these models.
- A really useful feature of these hubs is the inclusion of model cards, that describe important details including the best use cases for each model, how it was trained, and known limitations of LLM e.g. [Model Card for FLAN-T5 base]

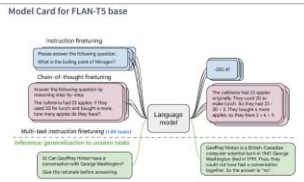
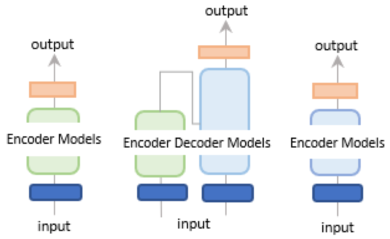


Table of content

1. Model Details
2. Usage
3. Uses
4. Bias, Risks and Limitation
5. Training Details
6. Evaluation
7. Environmental Impact
8. …..

LLM models types

- The exact model that you'd choose will depend on the details of the task, you need to carry out.
- Variance of the transformer model architecture are suited to different language tasks, largely because of differences in how the models are trained.

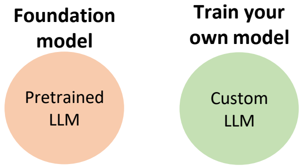


[Kumar]

Generative AI Project Life Cycle - Choose existing model or pre-train

Consideration for choosing LLM model

- Your first choice will be to work with an existing model. *(if it aligns to your governance and privacy polices)*
- There are specific circumstances where training your own model from scratch might be advantageous.



LLM model hubs

- In general you'll begin the process of developing your application using an existing foundation model.
- Many open-source models are available to use in application.
- The developers of some of the major frameworks for building generative AI applications like Hugging Face and PyTorch , have curated hubs where you can browse these models.
- A really useful feature of these hubs is the inclusion of model cards, that describe important details including the best use cases for each model, how it was trained, and known limitations of LLM e.g. [Model Card for FLAN-T5 base]

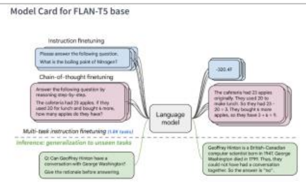
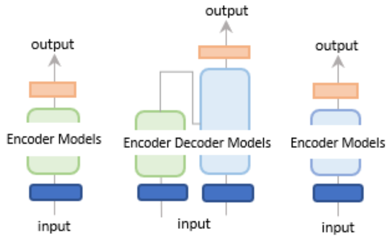


Table of content

1. Model Details
2. Usage
3. Uses
4. Bias, Risks and Limitation
5. Training Details
6. Evaluation
7. Environmental Impact
8. .....

LLM models types

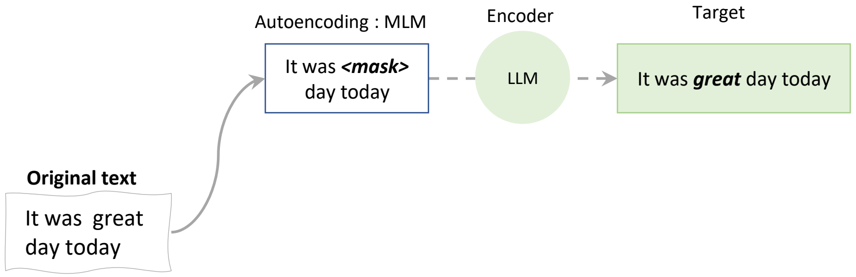
- The exact model that you'd choose will depend on the details of the task, you need to carry out.
- Variance of the transformer model architecture are suited to different language tasks, largely because of differences in how the models are trained.



<u>Pre-train LLM model by transformer type</u>

Encoder LLM models

Encoder-only models are also known as Autoencoding models and are pre-trained using masked language modelling (**MLM**). They correspond to the encoder part of the original transformer architecture and are often used with sentence classification or token classification e.g. BERT and RoBERTa

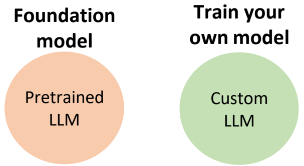


Generative AI Project Life Cycle - Choose existing model or pre-train

Select
Existing models or pre-train

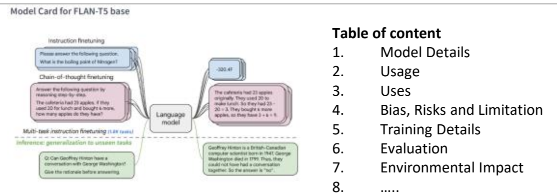
Consideration for choosing LLM model

- Your first choice will be to work with an existing model. *(if it aligns to your governance and privacy polices)*
- There are specific circumstances where training your own model from scratch might be advantageous.



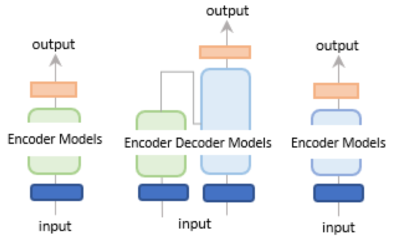
LLM model hubs

- In general you'll begin the process of developing your application using an existing foundation model.
- Many open-source models are available to use in application.
- The developers of some of the major frameworks for building generative AI applications like Hugging Face and PyTorch , have curated hubs where you can browse these models.
- A really useful feature of these hubs is the inclusion of model cards, that describe important details including the best use cases for each model, how it was trained, and known limitations of LLM e.g. [Model Card for FLAN-T5 base]



LLM models types

- The exact model that you'd choose will depend on the details of the task, you need to carry out.
- Variance of the transformer model architecture are suited to different language tasks, largely because of differences in how the models are trained.



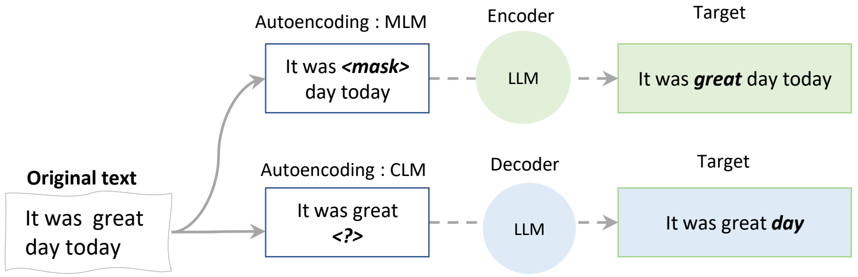
Pre-train LLM model by transformer type

Encoder LLM models

Encoder-only models are also known as Autoencoding models and are pre-trained using masked language modelling (**MLM**). They correspond to the encoder part of the original transformer architecture and are often used with sentence classification or token classification e.g. BERT and RoBERTa

Decoder LLM models

Decoder (Autoregressive) models are pre-trained using causal language modelling (**CLM**). Models of this type make use of the decoder component of the original transformer architecture, and often used for text generation e.g. GBT and BLOOM

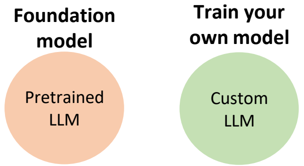


[Kumar]

Generative AI Project Life Cycle - Choose existing model or pre-train

Consideration for choosing LLM model

- Your first choice will be to work with an existing model. *(if it aligns to your governance and privacy polices)*
- There are specific circumstances where training your own model from scratch might be advantageous.



LLM model hubs

- In general you'll begin the process of developing your application using an existing foundation model.
- Many open-source models are available to use in application.
- The developers of some of the major frameworks for building generative AI applications like Hugging Face and PyTorch , have curated hubs where you can browse these models.
- A really useful feature of these hubs is the inclusion of model cards, that describe important details including the best use cases for each model, how it was trained, and known limitations of LLM e.g. [Model Card for FLAN-T5 base]

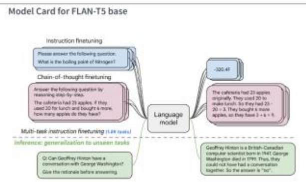
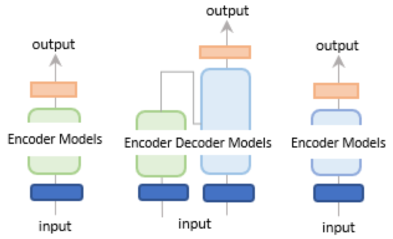


Table of content

1. Model Details
2. Usage
3. Uses
4. Bias, Risks and Limitation
5. Training Details
6. Evaluation
7. Environmental Impact
8. .....

LLM models types

- The exact model that you'd choose will depend on the details of the task, you need to carry out.
- Variance of the transformer model architecture are suited to different language tasks, largely because of differences in how the models are trained.



<u>Pre-train LLM model by transformer type</u>

Encoder LLM models

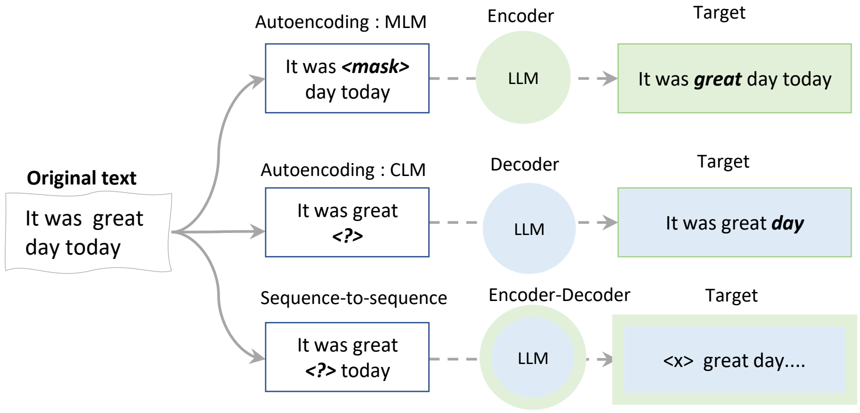
Encoder-only models are also known as Autoencoding models and are pre-trained using masked language modelling (**MLM**). They correspond to the encoder part of the original transformer architecture and are often used with sentence classification or token classification e.g. BERT and RoBERTa

Decoder LLM models

Decoder (Autoregressive) models are pre-trained using causal language modelling (**CLM**). Models of this type make use of the decoder component of the original transformer architecture, and often used for text generation e.g. GBT and BLOOM

Encoder-Decoder LLM models

Sequence-to-sequence models use both the encoder and decoder part off the original transformer architecture. The exact details of the pre-training objective vary from model to model. For e.g., T5 model is pre-trained using span corruption. Sequence-to-sequence models are often used for translation, summarization, and question-answering. e.g. T5 , BART



[Kumar]

Select
Existing models or pre-train

Generative AI Project Life Cycle - Choose existing model or pre-train

In general, you might tend to use existing LLM , this saves you a lot of time and can get you to a working prototype much faster. However, there could be situation where you may find it necessary to pretrain your own model from scratch to achieve good task performance for domain adaptation.

If your domain requires highly accurate results within a particular domain unlike generalised LLM you may need to perform domain specific adaptation to achieve good performance. Example of these domains are Legal, Medical, Finance, Climate, Pharmaceutical and Education

Generative AI Project Life Cycle - Choose existing model or pre-train

In general, you might tend to use existing LLM , this saves you a lot of time and can get you to a working prototype much faster. However, there could be situation where you may find it necessary to pretrain your own model from scratch to achieve good task performance for domain adaptation.

If your domain requires highly accurate results within a particular domain unlike generalised LLM you may need to perform domain specific adaptation to achieve good performance. Example of these domains are Legal, Medical, Finance, Climate, Pharmaceutical and Education

Comparison of general LLMs and domain-specific LLMs

Criteria	General LLMs (e.g. Chat GPT 3.5, PaLM, Chohere, Falcon etc)	Domain-specific LLMs (e.g. FoodUDT-1B)
Purpose	Developed to understand and generate text across a broad range of topics and contexts.	Specifically trained to understand and generate text in a particular niche or domain.

Generative AI Project Life Cycle - Choose existing model or pre-train

In general, you might tend to use existing LLM , this saves you a lot of time and can get you to a working prototype much faster. However, there could be situation where you may find it necessary to pretrain your own model from scratch to achieve good task performance for domain adaptation.

If your domain requires highly accurate results within a particular domain unlike generalised LLM you may need to perform domain specific adaptation to achieve good performance. Example of these domains are Legal, Medical, Finance, Climate, Pharmaceutical and Education

Comparison of general LLMs and domain-specific LLMs

Criteria	General LLMs (e.g. Chat GPT 3.5, PaLM, Chohere, Falcon etc)	Domain-specific LLMs (e.g. FoodUDT-1B)
Purpose	Developed to understand and generate text across a broad range of topics and contexts.	Specifically trained to understand and generate text in a particular niche or domain.
Training data	Trained on diverse internet text, encompassing a wide range of topics	Trained on domain-specific data (~ 500K recipes).

Kumar

Generative AI Project Life Cycle - Choose existing model or pre-train

In general, you might tend to use existing LLM , this saves you a lot of time and can get you to a working prototype much faster. However, there could be situation where you may find it necessary to pretrain your own model from scratch to achieve good task performance for domain adaptation.

If your domain requires highly accurate results within a particular domain unlike generalised LLM you may need to perform domain specific adaptation to achieve good performance. Example of these domains are Legal, Medical, Finance, Climate, Pharmaceutical and Education

Comparison of general LLMs and domain-specific LLMs

Criteria	General LLMs (e.g. Chat GPT 3.5, PaLM, Chohere, Falcon etc)	Domain-specific LLMs (e.g. FoodUDT-1B)
Purpose	Developed to understand and generate text across a broad range of topics and contexts.	Specifically trained to understand and generate text in a particular niche or domain.
Training data	Trained on diverse internet text, encompassing a wide range of topics	Trained on domain-specific data (~ 500K recipes).
Knowledge depth	General understanding of a wide range of topics, including domain-specific topics at a high level.	Deep understanding of specific domain.

Kumar

Generative AI Project Life Cycle - Choose existing model or pre-train

In general, you might tend to use existing LLM , this saves you a lot of time and can get you to a working prototype much faster. However, there could be situation where you may find it necessary to pretrain your own model from scratch to achieve good task performance for domain adaptation.

If your domain requires highly accurate results within a particular domain unlike generalised LLM you may need to perform domain specific adaptation to achieve good performance. Example of these domains are Legal, Medical, Finance, Climate, Pharmaceutical and Education

Comparison of general LLMs and domain-specific LLMs

Criteria	General LLMs (e.g. Chat GPT 3.5, PaLM, Chohere, Falcon etc)	Domain-specific LLMs (e.g. FoodUDT-1B)
Purpose	Developed to understand and generate text across a broad range of topics and contexts.	Specifically trained to understand and generate text in a particular niche or domain.
Training data	Trained on diverse internet text, encompassing a wide range of topics	Trained on domain-specific data (~ 500K recipes).
Knowledge depth	General understanding of a wide range of topics, including domain-specific topics at a high level.	Deep understanding of specific domain.
Representation of context	Represents text and context based on a broad understanding of language and world knowledge. For example, “apple” is closer to “iPhone” than to “apple pie”.	Represents text and context based on deep, specialized knowledge. For example, “apple” is closer to “apple pie” than to “iPhone.”

Kumar

Generative AI Project Life Cycle - Choose existing model or pre-train

In general, you might tend to use existing LLM , this saves you a lot of time and can get you to a working prototype much faster. However, there could be situation where you may find it necessary to pretrain your own model from scratch to achieve good task performance for domain adaptation.

If your domain requires highly accurate results within a particular domain unlike generalised LLM you may need to perform domain specific adaptation to achieve good performance. Example of these domains are Legal, Medical, Finance, Climate, Pharmaceutical and Education

Comparison of general LLMs and domain-specific LLMs

Criteria	General LLMs (e.g. Chat GPT 3.5, PaLM, Chohere, Falcon etc)	Domain-specific LLMs (e.g. FoodUDT-1B)
Purpose	Developed to understand and generate text across a broad range of topics and contexts.	Specifically trained to understand and generate text in a particular niche or domain.
Training data	Trained on diverse internet text, encompassing a wide range of topics	Trained on domain-specific data (~ 500K recipes).
Knowledge depth	General understanding of a wide range of topics, including domain-specific topics at a high level.	Deep understanding of specific domain.
Representa tion of context	Represents text and context based on a broad understanding of language and world knowledge. For example, “apple” is closer to “iPhone” than to “apple pie”.	Represents text and context based on deep, specialized knowledge. For example, “apple” is closer to “apple pie” than to “iPhone.”
Word associations	Forms associations based on generic context. For example “fruit”, “sugar” and “dessert” are equally related.	Forms associations based on domain-specific context. For example, “dessert” and “sugar” are more related than “fruit” and “sugar.”

Kumar

Generative AI Project Life Cycle - Choose existing model or pre-train

In general, you might tend to use existing LLM , this saves you a lot of time and can get you to a working prototype much faster. However, there could be situation where you may find it necessary to pretrain your own model from scratch to achieve good task performance for domain adaptation.

If your domain requires highly accurate results within a particular domain unlike generalised LLM you may need to perform domain specific adaptation to achieve good performance. Example of these domains are Legal, Medical, Finance, Climate, Pharmaceutical and Education

Comparison of general LLMs and domain-specific LLMs

Criteria	**General LLMs (e.g. Chat GPT 3.5, PaLM, Chohere, Falcon etc)**	**Domain-specific LLMs (e.g. FoodUDT-1B)**
Purpose	Developed to understand and generate text across a broad range of topics and contexts.	Specifically trained to understand and generate text in a particular niche or domain.
Training data	Trained on diverse internet text, encompassing a wide range of topics	Trained on domain-specific data (~ 500K recipes).
Knowledge depth	General understanding of a wide range of topics, including domain-specific topics at a high level.	Deep understanding of specific domain.
Representation of context	Represents text and context based on a broad understanding of language and world knowledge. For example, “apple” is closer to “iPhone” than to “apple pie”.	Represents text and context based on deep, specialized knowledge. For example, “apple” is closer to “apple pie” than to “iPhone.”
Word associations	Forms associations based on generic context. For example “fruit”, “sugar” and “dessert” are equally related.	Forms associations based on domain-specific context. For example, “dessert” and “sugar” are more related than “fruit” and “sugar.”
Limitations	May not possess in-depth understanding or generate accurate outputs in specialized domains.	Limited to its specific domain and may not generate accurate outputs outside that domain.

Kumar

Generative AI Project Life Cycle - Choose existing model or pre-train

In general, you might tend to use existing LLM , this saves you a lot of time and can get you to a working prototype much faster. However, there could be situation where you may find it necessary to pretrain your own model from scratch to achieve good task performance for domain adaptation.

If your domain requires highly accurate results within a particular domain unlike generalised LLM you may need to perform domain specific adaptation to achieve good performance. Example of these domains are Legal, Medical, Finance, Climate, Pharmaceutical and Education

Comparison of general LLMs and domain-specific LLMs

Criteria	**General LLMs (e.g. Chat GPT 3.5, PaLM, Chohere, Falcon etc)**	**Domain-specific LLMs (e.g. FoodUDT-1B)**
Purpose	Developed to understand and generate text across a broad range of topics and contexts.	Specifically trained to understand and generate text in a particular niche or domain.
Training data	Trained on diverse internet text, encompassing a wide range of topics	Trained on domain-specific data (~ 500K recipes).
Knowledge depth	General understanding of a wide range of topics, including domain-specific topics at a high level.	Deep understanding of specific domain.
Representation of context	Represents text and context based on a broad understanding of language and world knowledge. For example, “apple” is closer to “iPhone” than to “apple pie”.	Represents text and context based on deep, specialized knowledge. For example, “apple” is closer to “apple pie” than to “iPhone.”
Word associations	Forms associations based on generic context. For example “fruit”, “sugar” and “dessert” are equally related.	Forms associations based on domain-specific context. For example, “dessert” and “sugar” are more related than “fruit” and “sugar.”
Limitations	May not possess in-depth understanding or generate accurate outputs in specialized domains.	Limited to its specific domain and may not generate accurate outputs outside that domain.
Advantages	Versatile in general conversational, summarisation, translation and other generic tasks.	Highly accurate and efficient in its specific domain due to target training (recipe, food and cooking)

Generative AI Project Life Cycle - Choose existing model or pre-train

In general, you might tend to use existing LLM , this saves you a lot of time and can get you to a working prototype much faster. However, there could be situation where you may find it necessary to pretrain your own model from scratch to achieve good task performance for domain adaptation.

If your domain requires highly accurate results within a particular domain unlike generalised LLM you may need to perform domain specific adaptation to achieve good performance. Example of these domains are Legal, Medical, Finance, Climate, Pharmaceutical and Education

Comparison of general LLMs and domain-specific LLMs

Criteria	General LLMs (e.g. Chat GPT 3.5, PaLM, Chohere, Falcon etc)	Domain-specific LLMs (e.g. FoodUDT-1B)
Purpose	Developed to understand and generate text across a broad range of topics and contexts.	Specifically trained to understand and generate text in a particular niche or domain.
Training data	Trained on diverse internet text, encompassing a wide range of topics	Trained on domain-specific data (~ 500K recipes).
Knowledge depth	General understanding of a wide range of topics, including domain-specific topics at a high level.	Deep understanding of specific domain.
Representation of context	Represents text and context based on a broad understanding of language and world knowledge. For example, “apple” is closer to “iPhone” than to “apple pie”.	Represents text and context based on deep, specialized knowledge. For example, “apple” is closer to “apple pie” than to “iPhone.”
Word associations	Forms associations based on generic context. For example “fruit”, “sugar” and “dessert” are equally related.	Forms associations based on domain-specific context. For example, “dessert” and “sugar” are more related than “fruit” and “sugar.”
Limitations	May not possess in-depth understanding or generate accurate outputs in specialized domains.	Limited to its specific domain and may not generate accurate outputs outside that domain.
Advantages	Versatile in general conversational, summarisation, translation and other generic tasks.	Highly accurate and efficient in its specific domain due to target training (recipe, food and cooking)

Few examples of Domain-specific LLMs

LLM	Details
BloombergGPT	is a causal language model designed with decoder-only architecture. The model operated with 50 billion parameters and was trained from scratch with domain specific data in finance. It outperformed similar models on financial tasks by a significant margin while maintaining or bettering the others on general language tasks.
Med-PaLM 2	is a custom language model that Google built by training on curated medical datasets. The model can accurately answer medical questions, putting it on par with medical professionals in some use cases. When put to the test, MedPalm 2 scored an 86.5% mark on the MedQA dataset consisting of US Medical Licensing Examination questions.
KAI-GPT	is a large language model trained to deliver conversational AI in the banking industry. The model enables transparent, safe, and accurate use of generative AI models when servicing banking customers.
FinGPT	is a lightweight language model pre-trained with financial data. It provides a more affordable training option than the proprietary BloombergGPT. It also incorporates reinforcement learning from human feedback to enable further personalization. FinGPT scores well against other models on financial sentiment analysis datasets.

Kumar

Generative AI Project Life Cycle - Choose existing model or pre-train

How to create a Domain-specific LLM

There are two options to develop domain-specific models.

Generative AI Project Life Cycle - Choose existing model or pre-train

Select
Pre-train your own model

How to create a Domain-specific LLM
There are two options to develop domain-specific models.

Option1 - Build an entire domain-specific model from scratch
You can train a foundational model entirely from a blank slate with industry-specific knowledge. This involves getting the model to learn self-supervised with unlabelled data. During training, the model applies next-token prediction and mask-level modelling. The model attempts to predict words sequentially by masking specific tokens in a sentence.

Generative AI Project Life Cycle - Choose existing model or pre-train

How to create a Domain-specific LLM
There are two options to develop domain-specific models.

Option1 - Build an entire domain-specific model from scratch
You can train a foundational model entirely from a blank slate with industry-specific knowledge. This involves getting the model to learn self-supervised with unlabelled data. During training, the model applies next-token prediction and mask-level modelling. The model attempts to predict words sequentially by masking specific tokens in a sentence.

Challenges of building an entire domain-specific model from scratch

Data challenges	The procurement of substantial, niche-specific data could be demanding, especially when dealing with specialized or confidential data.
Technical and Resource challenges	Selection of suitable architecture and parameters necessitates specialized knowledge and comes with considerable cost. Evaluation becomes complex owing to the lack of established benchmarks for niche-specific tasks, and accuracy, safety, and compliance validation of model responses present additional challenges.
Ethical challenges	Robust content moderation mechanisms must be in place to prevent potentially inappropriate or harmful content generated by domain-specific LLMs.

Generative AI Project Life Cycle - Choose existing model or pre-train

How to create a Domain-specific LLM

There are two options to develop domain-specific models.

Option1 - Build an entire domain-specific model from scratch

You can train a foundational model entirely from a blank slate with industry-specific knowledge. This involves getting the model to learn self-supervised with unlabelled data. During training, the model applies next-token prediction and mask-level modelling. The model attempts to predict words sequentially by masking specific tokens in a sentence.

Challenges of building an entire domain-specific model from scratch

Data challenges	The procurement of substantial, niche-specific data could be demanding, especially when dealing with specialized or confidential data.
Technical and Resource challenges	Selection of suitable architecture and parameters necessitates specialized knowledge and comes with considerable cost. Evaluation becomes complex owing to the lack of established benchmarks for niche-specific tasks, and accuracy, safety, and compliance validation of model responses present additional challenges.
Ethical challenges	Robust content moderation mechanisms must be in place to prevent potentially inappropriate or harmful content generated by domain-specific LLMs.

Option2 – Fine-tune an LLM for domain-specific needs

Not all use cases require to train domain-specific models from scratch especial where output of LLM is used as indicative information only and where there is costs and time constrain.

In these use cases, fine-tuning a foundational model is sufficient to perform a specific task with reasonable accuracy. This approach has lesser challenges i.e. requires lesser datasets, computation, and time.

Generative AI Project Life Cycle - Choose existing model or pre-train

How to create a Domain-specific LLM
There are two options to develop domain-specific models.

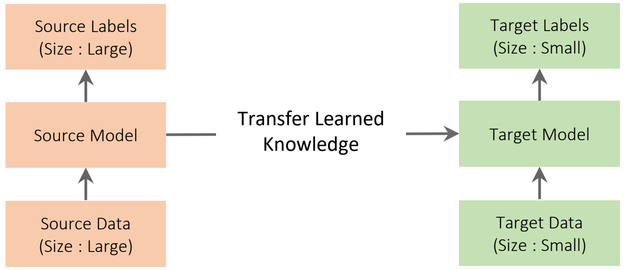
Option1 - Build an entire domain-specific model from scratch
You can train a foundational model entirely from a blank slate with industry-specific knowledge. This involves getting the model to learn self-supervised with unlabelled data. During training, the model applies next-token prediction and mask-level modelling. The model attempts to predict words sequentially by masking specific tokens in a sentence.

Challenges of building an entire domain-specific model from scratch

Data challenges	The procurement of substantial, niche-specific data could be demanding, especially when dealing with specialized or confidential data.
Technical and Resource challenges	Selection of suitable architecture and parameters necessitates specialized knowledge and comes with considerable cost. Evaluation becomes complex owing to the lack of established benchmarks for niche-specific tasks, and accuracy, safety, and compliance validation of model responses present additional challenges.
Ethical challenges	Robust content moderation mechanisms must be in place to prevent potentially inappropriate or harmful content generated by domain-specific LLMs.

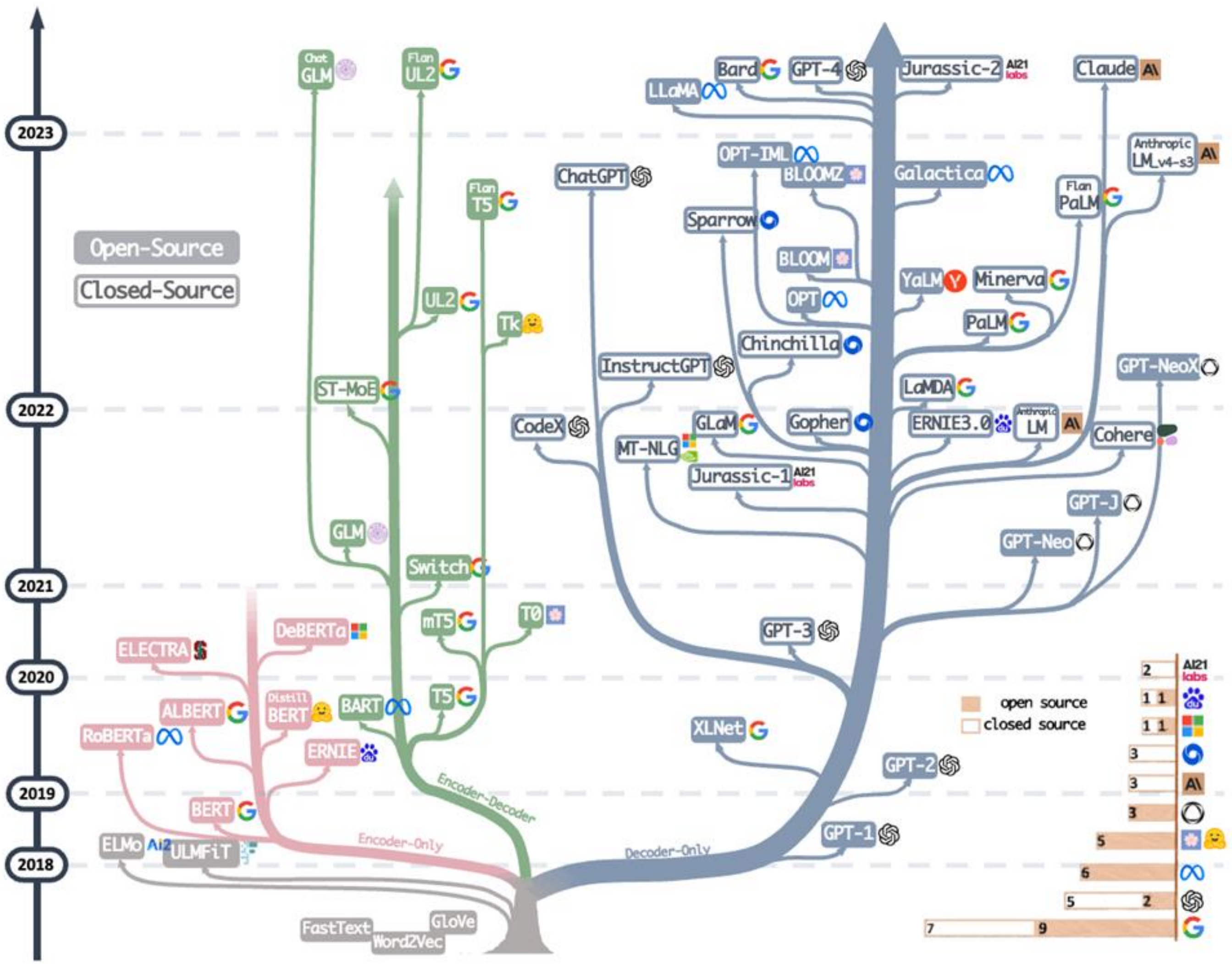
Option2 – Fine-tune an LLM for domain-specific needs
Not all use cases require to train domain-specific models from scratch especial where output of LLM is used as indicative information only and where there is costs and time constrain.
In these use cases, fine-tuning a foundational model is sufficient to perform a specific task with reasonable accuracy. This approach has lesser challenges i.e. requires lesser datasets, computation, and time.

When fine-tuning an LLM, select a pre-trained model like GPT and LLaMa, which already possess exceptional linguistic capability. And refine the model's weight by training it with a small set of annotated data with a slow learning rate. The principle of fine-tuning enables the language model to adopt the knowledge that new data presents while retaining the existing ones it initially learned.
Transfer learning is a unique technique that allows a pre-trained model to apply its knowledge to a new task. It is instrumental when you can't curate sufficient datasets to fine-tune a model. In transfer learning the models existing weights/layers are feezed and appended with new trainable ones to the top.
MedPaLM is an example of a domain-specific model trained with this approach. It is built upon PaLM, a 540 billion parameters language model.



Generative AI Project Life Cycle - LLM Evolution Tree

Select
Existing models or pre-train



The evolution tree of LLM traces the development of language models in recent years. Models on the same branch have closer relationship.

- Transformer-based models are shown in non-grey colour
- Decoder-only models in the blue coloured branch
- Encoder-only models in pink colour
- Encoder-Decoder models in green colour.
- Open-source LLMs are represented by solid squares
- Closed-source LLMs are represented by hallow squares.

Source : *Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond*

Kumar

Generative AI Project Life Cycle - Prompt Engineering (ICL)

Interacting with transformer model through natural language, creating prompts using written words, not code is called prompt engineering.

Generative AI Project Life Cycle - Prompt Engineering (ICL)

Interacting with transformer model through natural language, creating prompts using written words, not code is called prompt engineering.

Terminology	Description
Prompt	The text that you feed into the model is called the prompt.

Prompt

identify all of the companies name, date and places in "so we started NetSafe to redefine Cloud, Network and Data Security. Since 2019, we have built the market-leading cloud security company and an award-winning culture powered by hundreds of employees spread across offices in Santa Clara, St. Louis, Bangalore, London, Melbourne, and Tokyo."

Generative AI Project Life Cycle - Prompt Engineering (ICL)

Adapt and align model
Prompt-engineering

Interacting with transformer model through natural language, creating prompts using written words, not code is called prompt engineering.

Terminology	Description
Prompt	The text that you feed into the model is called the prompt.
Context window	The entire text or the memory that is available to use for the prompt is called the context window.

Prompt

identify all of the companies name, date and places in "so we started NetSafe to redefine Cloud, Network and Data Security. Since 2019, we have built the market-leading cloud security company and an award-winning culture powered by hundreds of employees spread across offices in Santa Clara, St. Louis, Bangalore, London, Melbourne, and Tokyo."

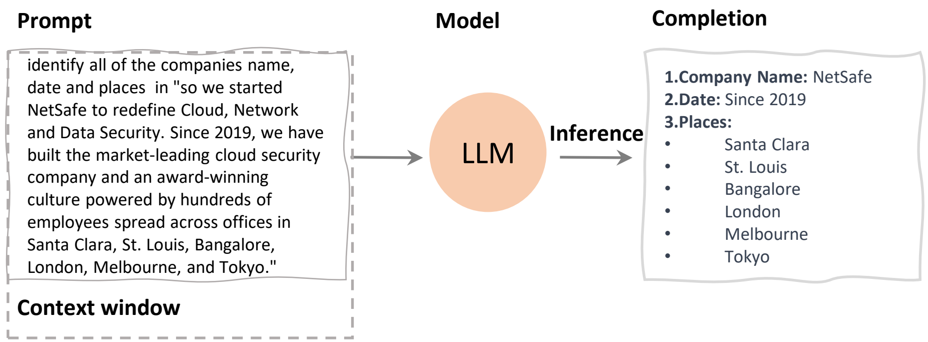
Context window

Kumar

Generative AI Project Life Cycle - Prompt Engineering (ICL)

Interacting with transformer model through natural language, creating prompts using written words, not code is called prompt engineering.

Terminology	Description
Prompt	The text that you feed into the model is called the prompt.
Context window	The entire text or the memory that is available to use for the prompt is called the context window.
Inference	The act of generating text is known as inference.
Completion	The output text is known as the completion.

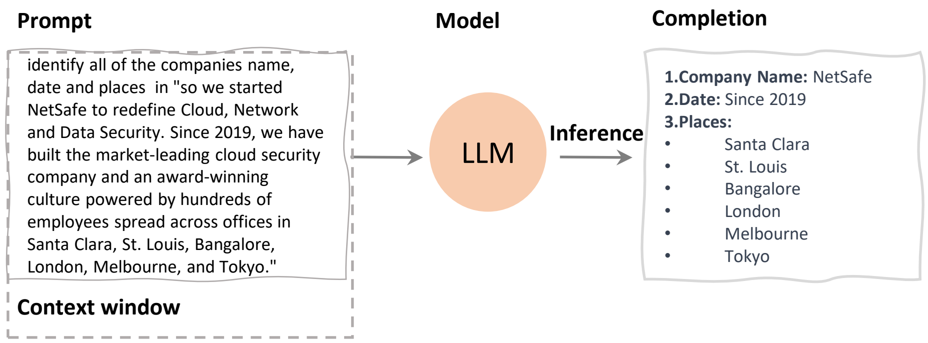


Generative AI Project Life Cycle - Prompt Engineering (ICL)

Adapt and align model Prompt-engineering

Interacting with transformer model through natural language, creating prompts using written words, not code is called prompt engineering.

Terminology	Description
Prompt	The text that you feed into the model is called the prompt.
Context window	The entire text or the memory that is available to use for the prompt is called the context window.
Inference	The act of generating text is known as inference.
Completion	The output text is known as the completion.
Prompt Engineering	Frequently encounter situations where the model doesn't produce the outcome that you want on the first try. You may have to revise the language in your prompt or the way that it's written several times to get the model to behave in the way that you want. The work to develop and improve the prompt is known as prompt engineering.



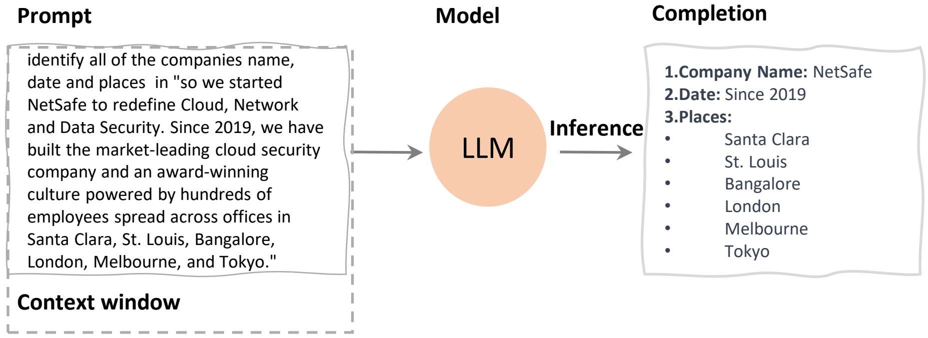
Kumar

Generative AI Project Life Cycle - Prompt Engineering (ICL)

Adapt and align model
Prompt-engineering

Interacting with transformer model through natural language, creating prompts using written words, not code is called prompt engineering.

Terminology	Description
Prompt	The text that you feed into the model is called the prompt.
Context window	The entire text or the memory that is available to use for the prompt is called the context window.
Inference	The act of generating text is known as inference.
Completion	The output text is known as the completion.
Prompt Engineering	Frequently encounter situations where the model doesn't produce the outcome that you want on the first try. You may have to revise the language in your prompt or the way that it's written several times to get the model to behave in the way that you want. The work to develop and improve the prompt is known as prompt engineering.
In-context learning (ICL)	one powerful strategy to get the model to produce better outcomes is to include examples of the task that you want the model to carry out inside the prompt. Providing examples inside the context window is called in-context learning.

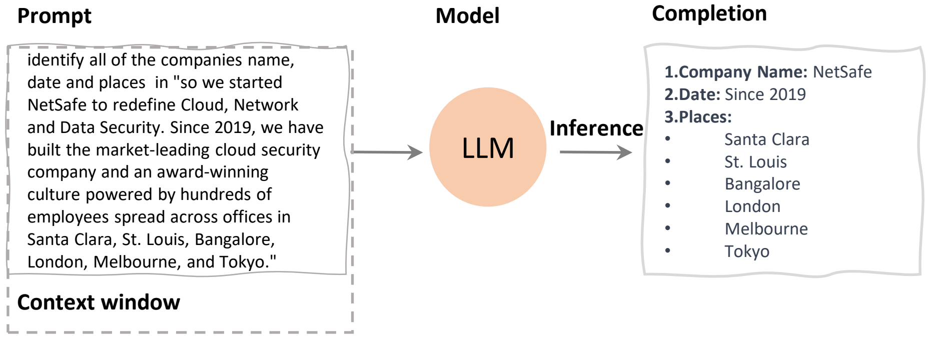


Generative AI Project Life Cycle - Prompt Engineering (ICL)

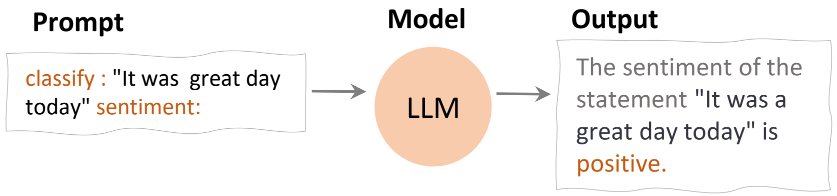
Adapt and align model Prompt-engineering

Interacting with transformer model through natural language, creating prompts using written words, not code is called prompt engineering.

Terminology	Description
Prompt	The text that you feed into the model is called the prompt.
Context window	The entire text or the memory that is available to use for the prompt is called the context window.
Inference	The act of generating text is known as inference.
Completion	The output text is known as the completion.
Prompt Engineering	Frequently encounter situations where the model doesn't produce the outcome that you want on the first try. You may have to revise the language in your prompt or the way that it's written several times to get the model to behave in the way that you want. The work to develop and improve the prompt is known as prompt engineering.
In-context learning (ICL)	one powerful strategy to get the model to produce better outcomes is to include examples of the task that you want the model to carry out inside the prompt. Providing examples inside the context window is called in-context learning.



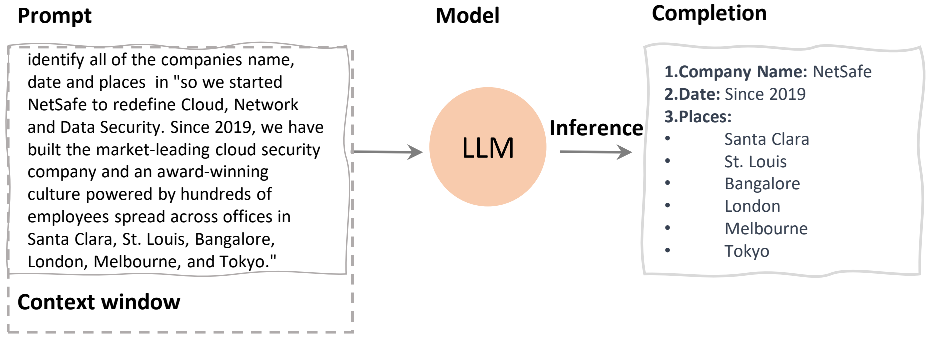
Zero-shot inference	In-context learning (ICL) zero shot interface in-context learning, you can help LLMs learn more about the task being asked by including examples or additional data in the prompt. The prompt consists of the instruction, "Classify this review," followed by some context, which in this case is the review text itself, and an instruction to produce the sentiment at the end.

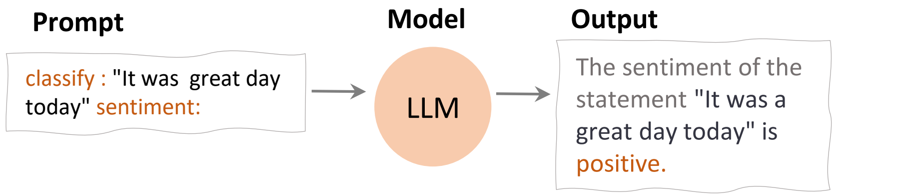


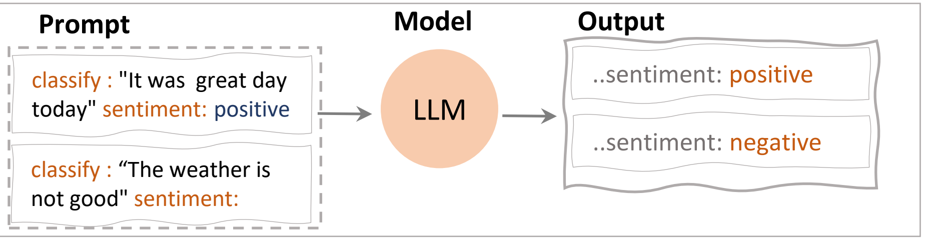
Generative AI Project Life Cycle - Prompt Engineering (ICL)

Interacting with transformer model through natural language, creating prompts using written words, not code is called prompt engineering.

Terminology	Description
Prompt	The text that you feed into the model is called the prompt.
Context window	The entire text or the memory that is available to use for the prompt is called the context window.
Inference	The act of generating text is known as inference.
Completion	The output text is known as the completion.
Prompt Engineering	Frequently encounter situations where the model doesn't produce the outcome that you want on the first try. You may have to revise the language in your prompt or the way that it's written several times to get the model to behave in the way that you want. The work to develop and improve the prompt is known as prompt engineering.
In-context learning (ICL)	one powerful strategy to get the model to produce better outcomes is to include examples of the task that you want the model to carry out inside the prompt. Providing examples inside the context window is called in-context learning.
Zero-shot inference	In-context learning (ICL) zero shot interface in-context learning, you can help LLMs learn more about the task being asked by including examples or additional data in the prompt. The prompt consists of the instruction, "Classify this review," followed by some context, which in this case is the review text itself, and an instruction to produce the sentiment at the end.
One-shot inference	Specifying that the model should classify the review, the prompt text includes a sample review, followed by a completed sentiment analysis. In this case, the review is positive. Next, prompt states the instruction again and includes the actual input review that we want the model to analyse. The inclusion of a single example is known as one-shot inference.





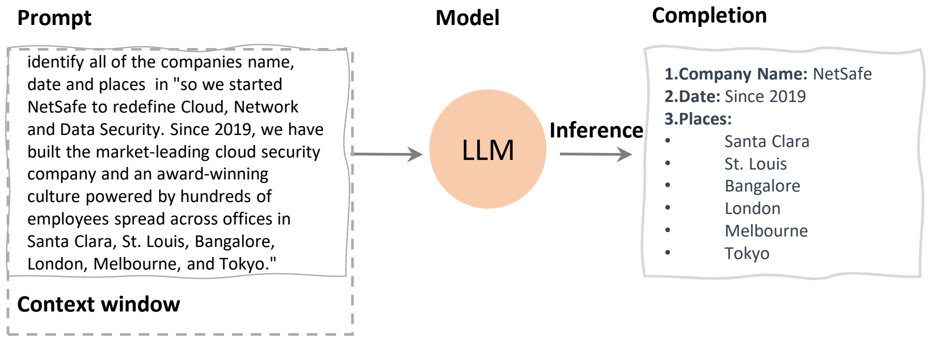


Generative AI Project Life Cycle - Prompt Engineering (ICL)

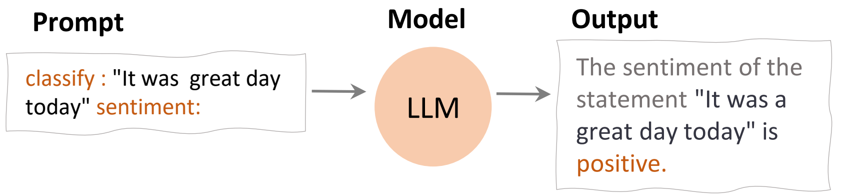
Adapt and align model
Prompt-engineering

Interacting with transformer model through natural language, creating prompts using written words, not code is called prompt engineering.

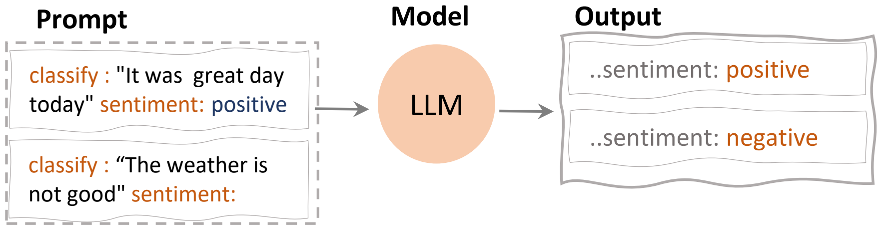
Terminology	Description
Prompt	The text that you feed into the model is called the prompt.
Context window	The entire text or the memory that is available to use for the prompt is called the context window.
Inference	The act of generating text is known as inference.
Completion	The output text is known as the completion.
Prompt Engineering	Frequently encounter situations where the model doesn't produce the outcome that you want on the first try. You may have to revise the language in your prompt or the way that it's written several times to get the model to behave in the way that you want. The work to develop and improve the prompt is known as prompt engineering.
In-context learning (ICL)	one powerful strategy to get the model to produce better outcomes is to include examples of the task that you want the model to carry out inside the prompt. Providing examples inside the context window is called in-context learning.



Zero-shot inference	In-context learning (ICL) zero shot interface in-context learning, you can help LLMs learn more about the task being asked by including examples or additional data in the prompt. The prompt consists of the instruction, "Classify this review," followed by some context, which in this case is the review text itself, and an instruction to produce the sentiment at the end.

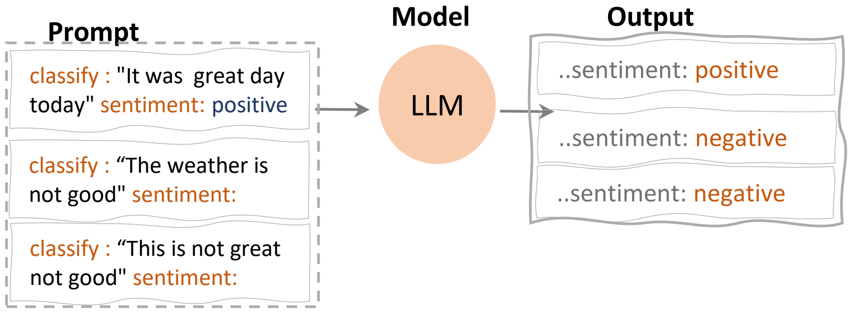


One-shot inference	Specifying that the model should classify the review, the prompt text includes a sample review, followed by a completed sentiment analysis. In this case, the review is positive. Next, prompt states the instruction again and includes the actual input review that we want the model to analyse. The inclusion of a single example is known as one-shot inference.



Few-shot inference	Sometimes a single example won't be enough for the model to learn what you want it to do. So, you can extend the idea of giving a single example to include multiple examples.

- Large models are good at zero-shot inference
- Small models can benefit from one-shot or few-shot inference
- The context window have a limit on the amount of in-context learning that you can pass into the model



Kumar

Generative AI Project Life Cycle - Prompt Engineering (ICL)

Adapt and align model
Prompt-engineering

Interacting with transformer model through natural language, creating prompts using written words, not code is called prompt engineering.

Terminology	Description
Prompt	The text that you feed into the model is called the prompt.
Context window	The entire text or the memory that is available to use for the prompt is called the context window.
Inference	The act of generating text is known as inference.
Completion	The output text is known as the completion.
Prompt Engineering	Frequently encounter situations where the model doesn't produce the outcome that you want on the first try. You may have to revise the language in your prompt or the way that it's written several times to get the model to behave in the way that you want. The work to develop and improve the prompt is known as prompt engineering.
In-context learning (ICL)	one powerful strategy to get the model to produce better outcomes is to include examples of the task that you want the model to carry out inside the prompt. Providing examples inside the context window is called in-context learning.

Prompt | **Model** | **Completion**

identify all of the companies name, date and places in "so we started NetSafe to redefine Cloud, Network and Data Security. Since 2019, we have built the market-leading cloud security company and an award-winning culture powered by hundreds of employees spread across offices in Santa Clara, St. Louis, Bangalore, London, Melbourne, and Tokyo."

Context window

LLM → **Inference** →

1. **Company Name:** NetSafe
2. **Date:** Since 2019
3. **Places:**
- Santa Clara
- St. Louis
- Bangalore
- London
- Melbourne
- Tokyo

Zero-shot inference

In-context learning (ICL) zero shot interface in-context learning, you can help LLMs learn more about the task being asked by including examples or additional data in the prompt.
The prompt consists of the instruction, "Classify this review," followed by some context, which in this case is the review text itself, and an instruction to produce the sentiment at the end.

Prompt | **Model** | **Output**

classify : "It was great day today" sentiment:

LLM

The sentiment of the statement "It was a great day today" is positive.

One-shot inference

Specifying that the model should classify the review, the prompt text includes a sample review, followed by a completed sentiment analysis. In this case, the review is positive.
Next, prompt states the instruction again and includes the actual input review that we want the model to analyse.
The inclusion of a single example is known as one-shot inference.

Prompt | **Model** | **Output**

classify : "It was great day today" sentiment: positive

classify : "The weather is not good" sentiment:

LLM

..sentiment: positive

..sentiment: negative

Few-shot inference

Sometimes a single example won't be enough for the model to learn what you want it to do. So, you can extend the idea of giving a single example to include multiple examples.

- Large models are good at zero-shot inference
- Small models can benefit from one-shot or few-shot inference
- The context window have a limit on the amount of in-context learning that you can pass into the model

Generally, if you find that your model isn't performing even after including 5 or 6 six examples, you should try **fine-tuning** your model instead. Fine-tuning performs additional training on the model using new data to make it more capable of the task you want it to perform.

Prompt | **Model** | **Output**

classify : "It was great day today" sentiment: positive

classify : "The weather is not good" sentiment:

classify : "This is not great not good" sentiment:

LLM

..sentiment: positive

..sentiment: negative

..sentiment: negative

Kumar

Generative AI Project Life Cycle - Generative AI Configuration Settings

As larger and larger models have been trained, it's become clear that the ability of models to perform multiple tasks and how well they perform those tasks depends strongly on the scale of the model.
Models with more parameters are able to capture more understanding of language.
You may have to try out a few models to find the right one for your use case. Once you've found the model that is working for you, there are a few settings that you can experiment with to influence the structure and style of the completions that the model generates this is called **Generative configuration settings**.

Generative AI Project Life Cycle - Generative AI Configuration Settings

As larger and larger models have been trained, it's become clear that the ability of models to perform multiple tasks and how well they perform those tasks depends strongly on the scale of the model.
Models with more parameters are able to capture more understanding of language.
You may have to try out a few models to find the right one for your use case. Once you've found the model that is working for you, there are a few settings that you can experiment with to influence the structure and style of the completions that the model generates this is called **Generative configuration settings**.

Generative AI configuration settings
Some of the methods and associated configuration parameters that you can use to influence the way that the model makes the final decision.

- These are different than the training parameters which are learned during training time.
- These configuration parameters are invoked at inference time and give you control over things like the maximum number of tokens in the completion, and how creative the output.

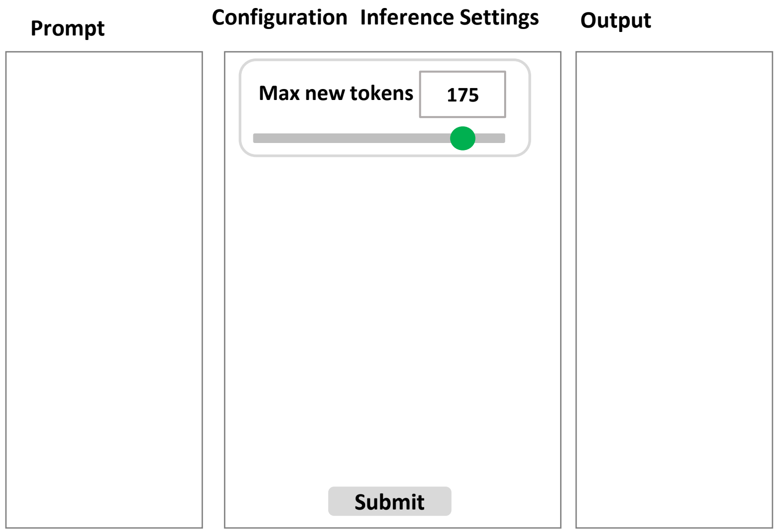
Generative AI Project Life Cycle - Generative AI Configuration Settings

As larger and larger models have been trained, it's become clear that the ability of models to perform multiple tasks and how well they perform those tasks depends strongly on the scale of the model.
Models with more parameters are able to capture more understanding of language.
You may have to try out a few models to find the right one for your use case. Once you've found the model that is working for you, there are a few settings that you can experiment with to influence the structure and style of the completions that the model generates this is called **Generative configuration settings**.

Generative AI configuration settings
Some of the methods and associated configuration parameters that you can use to influence the way that the model makes the final decision.

- These are different than the training parameters which are learned during training time.
- These configuration parameters are invoked at inference time and give you control over things like the maximum number of tokens in the completion, and how creative the output.

Inference	Description
Max new tokens	You can use it to limit the number of tokens that the model will generate (limit on the number of times the model will go through the selection process.)



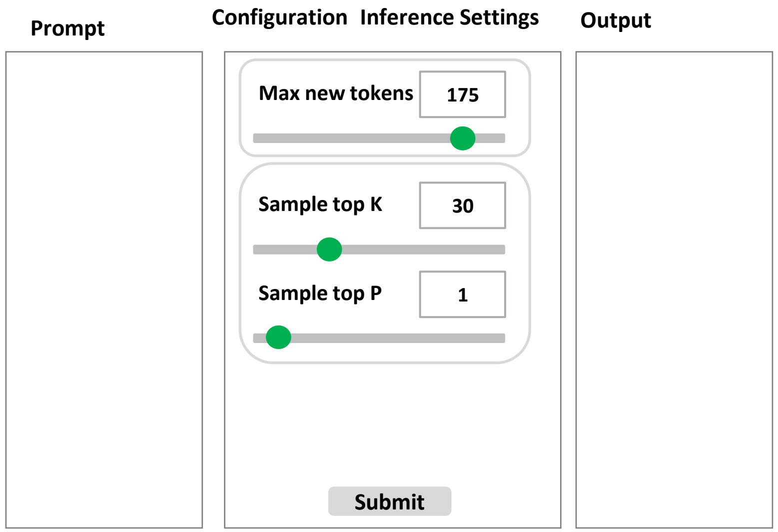
Generative AI Project Life Cycle - Generative AI Configuration Settings

As larger and larger models have been trained, it's become clear that the ability of models to perform multiple tasks and how well they perform those tasks depends strongly on the scale of the model.
Models with more parameters are able to capture more understanding of language.
You may have to try out a few models to find the right one for your use case. Once you've found the model that is working for you, there are a few settings that you can experiment with to influence the structure and style of the completions that the model generates this is called **Generative configuration settings**.

Generative AI configuration settings
Some of the methods and associated configuration parameters that you can use to influence the way that the model makes the final decision.

- These are different than the training parameters which are learned during training time.
- These configuration parameters are invoked at inference time and give you control over things like the maximum number of tokens in the completion, and how creative the output.

Inference	Description
Max new tokens	You can use it to limit the number of tokens that the model will generate (limit on the number of times the model will go through the selection process.)
Sample top K	top k sampling techniques to help limit the random sampling and increase the chance that the output will be sensible. You can specify a top k value which instructs the model to choose from only the k tokens with the highest probability.
Sample top P	top p setting to limit the random sampling to the predictions whose combined probabilities do not exceed p.



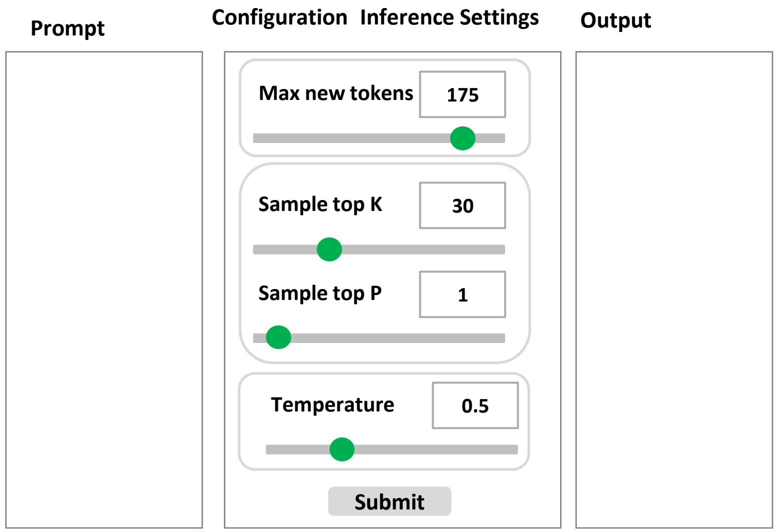
Generative AI Project Life Cycle - Generative AI Configuration Settings

As larger and larger models have been trained, it's become clear that the ability of models to perform multiple tasks and how well they perform those tasks depends strongly on the scale of the model.
Models with more parameters are able to capture more understanding of language.
You may have to try out a few models to find the right one for your use case. Once you've found the model that is working for you, there are a few settings that you can experiment with to influence the structure and style of the completions that the model generates this is called **Generative configuration settings**.

Generative AI configuration settings
Some of the methods and associated configuration parameters that you can use to influence the way that the model makes the final decision.

- These are different than the training parameters which are learned during training time.
- These configuration parameters are invoked at inference time and give you control over things like the maximum number of tokens in the completion, and how creative the output.

Inference	Description
Max new tokens	You can use it to limit the number of tokens that the model will generate (limit on the number of times the model will go through the selection process.)
Sample top K	top k sampling techniques to help limit the random sampling and increase the chance that the output will be sensible. You can specify a top k value which instructs the model to choose from only the k tokens with the highest probability.
Sample top P	top p setting to limit the random sampling to the predictions whose combined probabilities do not exceed p.
Temperature	Another parameter that you can use to control the randomness of the model output is known as temperature. • This parameter influences the shape of the probability distribution that the model calculates for the next token. • Broadly, the higher the temperature, the higher the randomness, and the lower the temperature, the lower the randomness.



Generative AI Project Life Cycle - Fine-tunning

By performing prompt engineering i.e. including one or more examples of what you want the model to do, known as one shot or few shot inference, can be enough to help the model identify the task and generate a good completion.

However, this strategy has a couple of drawbacks.

- for smaller models, it doesn't always work, even when five or six examples are included
- any examples you include in your prompt take up valuable space in the context window, reducing the amount of room you have to include other useful information. To overcome this another solution exists that is **fine-tuning**. This is process further trains a base model.

Generative AI Project Life Cycle - Fine-tunning

By performing prompt engineering i.e. including one or more examples of what you want the model to do, known as one shot or few shot inference, can be enough to help the model identify the task and generate a good completion.

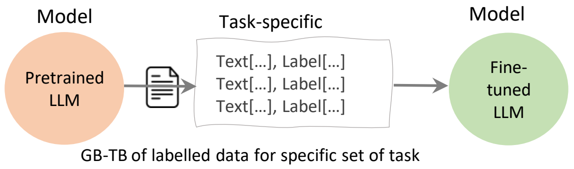
However, this strategy has a couple of drawbacks.

- for smaller models, it doesn't always work, even when five or six examples are included
- any examples you include in your prompt take up valuable space in the context window, reducing the amount of room you have to include other useful information. To overcome this another solution exists that is **fine-tuning**. This is process further trains a base model.

LLM fine-tunning overview

Fine-tunning

- Fine-tuning is a supervised learning process where you use a data set of labelled examples to update the weights of the LLM.



Generative AI Project Life Cycle - Fine-tunning

By performing prompt engineering i.e. including one or more examples of what you want the model to do, known as one shot or few shot inference, can be enough to help the model identify the task and generate a good completion.

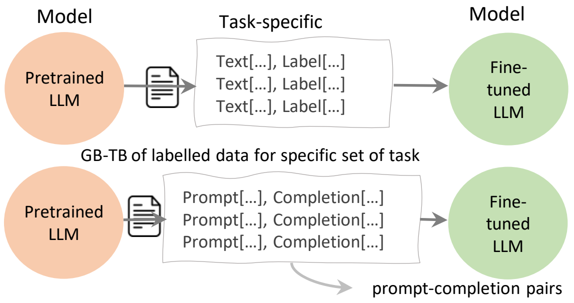
However, this strategy has a couple of drawbacks.

- for smaller models, it doesn't always work, even when five or six examples are included
- any examples you include in your prompt take up valuable space in the context window, reducing the amount of room you have to include other useful information. To overcome this another solution exists that is **fine-tuning**. This is process further trains a base model.

LLM fine-tunning overview

Fine-tunning

- Fine-tuning is a supervised learning process where you use a data set of labelled examples to update the weights of the LLM.
- The labelled examples are prompt completion pairs, the fine-tuning process extends the training of the model to improve its ability to generate good completions for a specific task.



Generative AI Project Life Cycle - Fine-tunning

By performing prompt engineering i.e. including one or more examples of what you want the model to do, known as one shot or few shot inference, can be enough to help the model identify the task and generate a good completion.

However, this strategy has a couple of drawbacks.

- for smaller models, it doesn't always work, even when five or six examples are included
- any examples you include in your prompt take up valuable space in the context window, reducing the amount of room you have to include other useful information. To overcome this another solution exists that is **fine-tuning**. This is process further trains a base model.

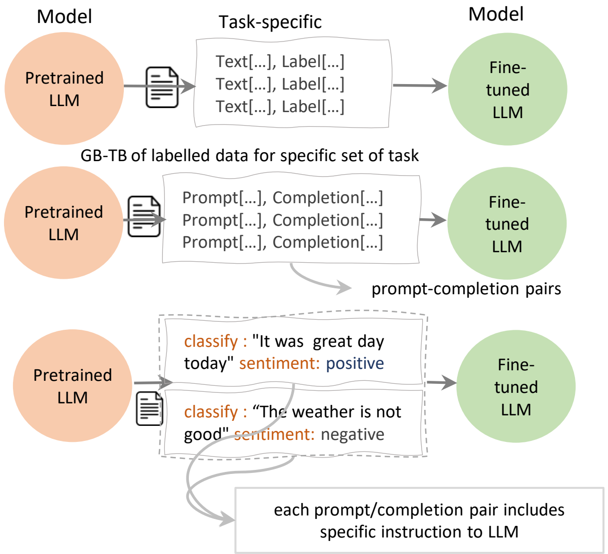
LLM fine-tunning overview

Fine-tunning

- Fine-tuning is a supervised learning process where you use a data set of labelled examples to update the weights of the LLM.
- The labelled examples are prompt completion pairs, the fine-tuning process extends the training of the model to improve its ability to generate good completions for a specific task.

Instruction-fine-tunning

The fine-tunning with instruction prompts is most common way to fine-tune LLMs. It trains the model using examples that demonstrate how it should respond to a specific instruction.
Instruction-fine-tuning, where all of model's weights are updated and results in new version of model with updated weights.
The potential downside to fine-tuning on a single task may lead to a phenomenon called catastrophic forgetting.



Generative AI Project Life Cycle - Fine-tunning

By performing prompt engineering i.e. including one or more examples of what you want the model to do, known as one shot or few shot inference, can be enough to help the model identify the task and generate a good completion.

However, this strategy has a couple of drawbacks.

- for smaller models, it doesn't always work, even when five or six examples are included
- any examples you include in your prompt take up valuable space in the context window, reducing the amount of room you have to include other useful information. To overcome this another solution exists that is **fine-tuning**. This is process further trains a base model.

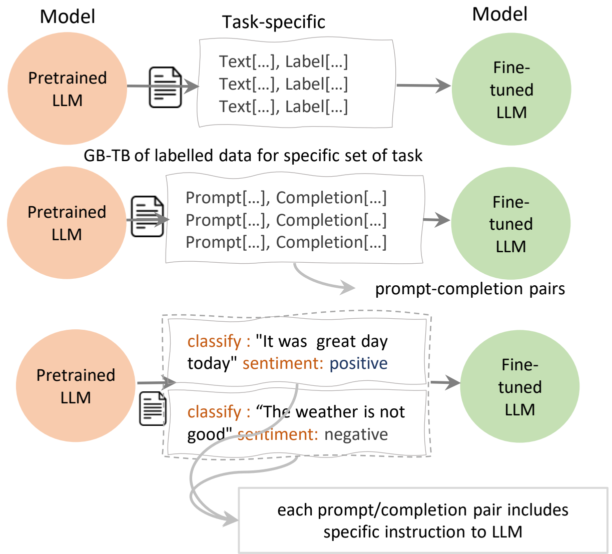
LLM fine-tunning overview

Fine-tunning

- Fine-tuning is a supervised learning process where you use a data set of labelled examples to update the weights of the LLM.
- The labelled examples are prompt completion pairs, the fine-tuning process extends the training of the model to improve its ability to generate good completions for a specific task.

Instruction-fine-tunning

The fine-tunning with instruction prompts is most common way to fine-tune LLMs. It trains the model using examples that demonstrate how it should respond to a specific instruction.
Instruction-fine-tuning, where all of model's weights are updated and results in new version of model with updated weights.
The potential downside to fine-tuning on a single task may lead to a phenomenon called catastrophic forgetting.



How to avoid catastrophic forgetting

Option1

- It's important to decide whether catastrophic forgetting actually impacts your use case.?. If all you need is reliable performance on the single task you fine-tuned on, it may not be an issue that the model can't generalize to other tasks.
- If you need the model to maintain its multitask generalized capabilities you can perform fine-tuning on multiple tasks. **Multi-task** fine-tuning may require 50-100,000 examples across many tasks, and so will require more data and compute to train e.g. FLAN → FLAN-T5 and FLAN-PALM is the flattening struct version of PALM foundation model.

Kumar

Generative AI Project Life Cycle - Fine-tunning

By performing prompt engineering i.e. including one or more examples of what you want the model to do, known as one shot or few shot inference, can be enough to help the model identify the task and generate a good completion.

However, this strategy has a couple of drawbacks.
- for smaller models, it doesn't always work, even when five or six examples are included
- any examples you include in your prompt take up valuable space in the context window, reducing the amount of room you have to include other useful information. To overcome this another solution exists that is **fine-tuning**. This is process further trains a base model.

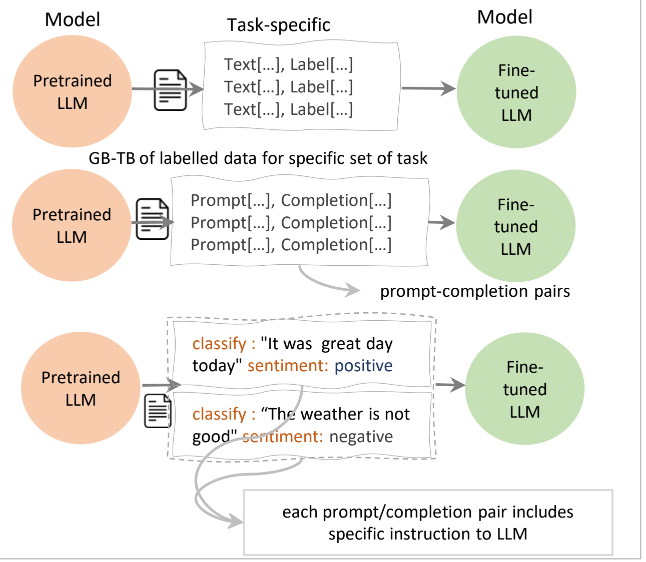
LLM fine-tunning overview

Fine-tunning

- Fine-tuning is a supervised learning process where you use a data set of labelled examples to update the weights of the LLM.
- The labelled examples are prompt completion pairs, the fine-tuning process extends the training of the model to improve its ability to generate good completions for a specific task.

Instruction-fine-tunning

The fine-tunning with instruction prompts is most common way to fine-tune LLMs. It trains the model using examples that demonstrate how it should respond to a specific instruction.
Instruction-fine-tuning, where all of model's weights are updated and results in new version of model with updated weights.
The potential downside to fine-tuning on a single task may lead to a phenomenon called catastrophic forgetting.



How to avoid catastrophic forgetting

Option1
- It's important to decide whether catastrophic forgetting actually impacts your use case.?. If all you need is reliable performance on the single task you fine-tuned on, it may not be an issue that the model can't generalize to other tasks.
- If you need the model to maintain its multitask generalized capabilities you can perform fine-tuning on multiple tasks. **Multi-task** fine-tuning may require 50-100,000 examples across many tasks, and so will require more data and compute to train e.g. FLAN → FLAN-T5 and FLAN-PALM is the flattening struct version of PALM foundation model.

Option2
- Perform **P**arameter **E**fficient **F**ine-**t**uning (**PEFT**). PEFT is a set of techniques that preserves the weights of the original LLM and trains only a small number of task-specific adapter layers and parameters.
- PEFT shows greater robustness to catastrophic forgetting since most of the pre-trained weights are left unchanged.

Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

Adapt and align model
PEFT

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

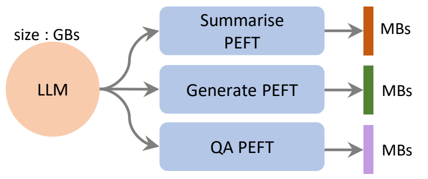
Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

Overview of PEFT

Train small number of parameters

With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task.

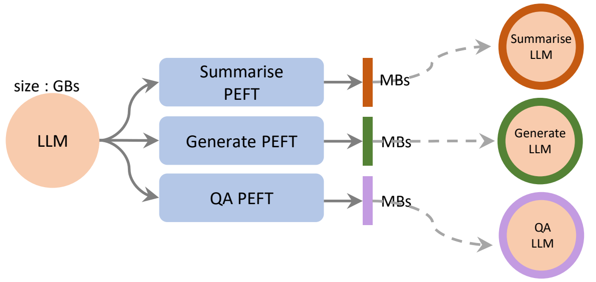


Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

Overview of PEFT

Train small number of parameters	With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task.
New parameters combined with original LLM weights	The new parameters are combined with the original LLM weights for inference, allowing efficient adaptation of the original model to multiple tasks.



Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

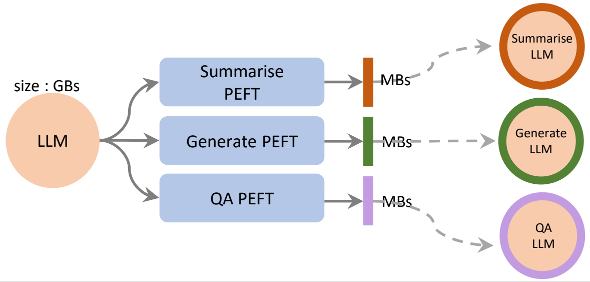
Overview of PEFT

Train small number of parameters

With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task.

New parameters combined with original LLM weights

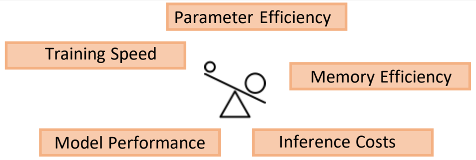
The new parameters are combined with the original LLM weights for inference, allowing efficient adaptation of the original model to multiple tasks.



PEFT Trade-off

There are several methods you can use for parameter efficient fine-tuning, each with trade-offs on: parameter efficiency, memory efficiency, training speed, model quality and inference costs.

- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.
- plateau in validation loss for rank greater than 16 i.e. using larger LoRA matrices didn’t improve the performance/accuracy score.
- ranks in the range of 4 to 32 can provide a good trade-off between reducing trainable parameters and preserving performance.



Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

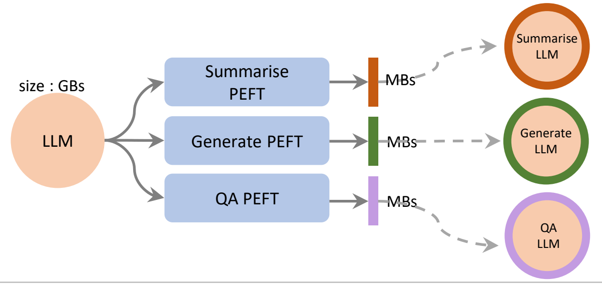
Overview of PEFT

Train small number of parameters

With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task.

New parameters combined with original LLM weights

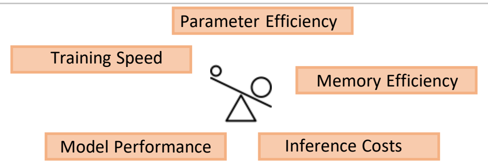
The new parameters are combined with the original LLM weights for inference, allowing efficient adaptation of the original model to multiple tasks.



PEFT Trade-off

There are several methods you can use for parameter efficient fine-tuning, each with trade-offs on: parameter efficiency, memory efficiency, training speed, model quality and inference costs.

- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.
- plateau in validation loss for rank greater than 16 i.e. using larger LoRA matrices didn’t improve the performance/accuracy score.
- ranks in the range of 4 to 32 can provide a good trade-off between reducing trainable parameters and preserving performance.



PEFT methods

Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

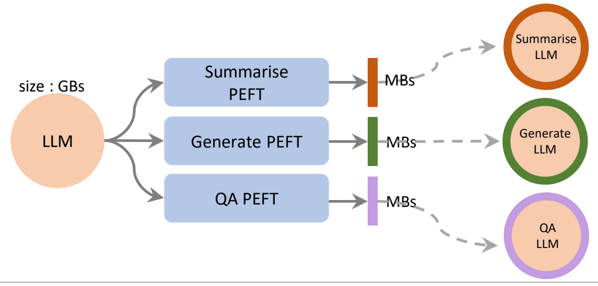
Overview of PEFT

Train small number of parameters

With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task.

New parameters combined with original LLM weights

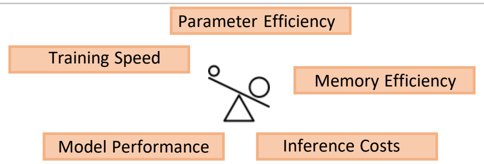
The new parameters are combined with the original LLM weights for inference, allowing efficient adaptation of the original model to multiple tasks.



PEFT Trade-off

There are several methods you can use for parameter efficient fine-tuning, each with trade-offs on: parameter efficiency, memory efficiency, training speed, model quality and inference costs.

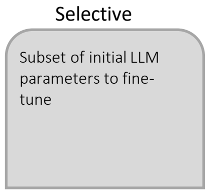
- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.
- plateau in validation loss for rank greater than 16 i.e. using larger LoRA matrices didn't improve the performance/accuracy score.
- ranks in the range of 4 to 32 can provide a good trade-off between reducing trainable parameters and preserving performance.



PEFT methods

Selective

Selective methods are those that fine-tune only a subset of the original LLM parameters. Some of the approaches are: train only certain components of the model, specific layers, individual parameter



Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

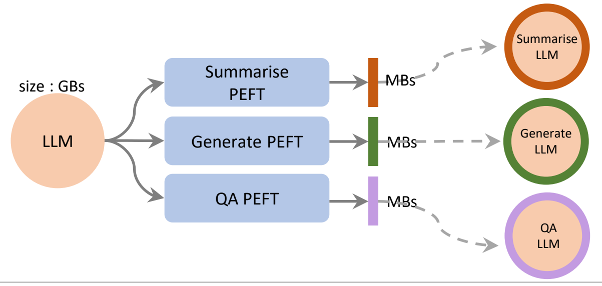
Overview of PEFT

Train small number of parameters

With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task.

New parameters combined with original LLM weights

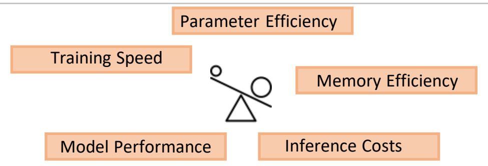
The new parameters are combined with the original LLM weights for inference, allowing efficient adaptation of the original model to multiple tasks.



PEFT Trade-off

There are several methods you can use for parameter efficient fine-tuning, each with trade-offs on: parameter efficiency, memory efficiency, training speed, model quality and inference costs.

- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.
- plateau in validation loss for rank greater than 16 i.e. using larger LoRA matrices didn't improve the performance/accuracy score.
- ranks in the range of 4 to 32 can provide a good trade-off between reducing trainable parameters and preserving performance.



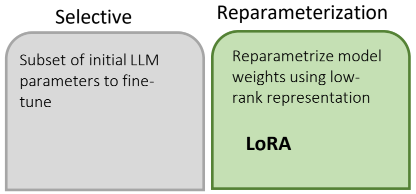
PEFT methods

Selective

Selective methods are those that fine-tune only a subset of the original LLM parameters. Some of the approaches are: train only certain components of the model, specific layers, individual parameter

Reparameterization

This methods also works with the original LLM parameters but reduce the number of parameters to train by creating new low rank transformations of the original weights. A commonly used technique of this type is **LoRA** (Low-rank Adaptation). LoRA is a strategy that reduces the number of parameters to be trained during fine-tuning by freezing all of the original model parameters.

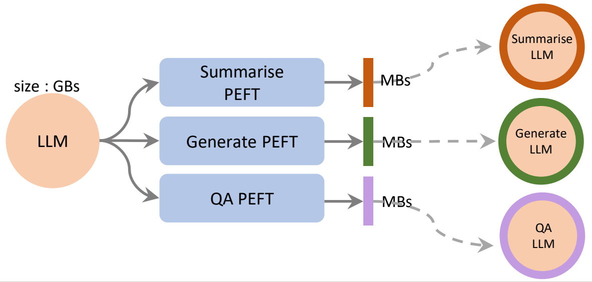


Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

Overview of PEFT

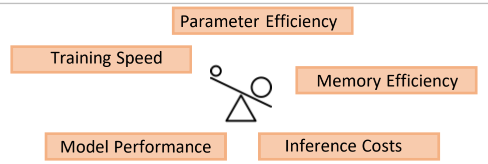
Train small number of parameters	With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task.
New parameters combined with original LLM weights	The new parameters are combined with the original LLM weights for inference, allowing efficient adaptation of the original model to multiple tasks.



PEFT Trade-off

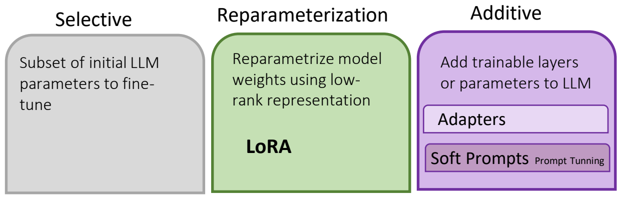
There are several methods you can use for parameter efficient fine-tuning, each with trade-offs on: parameter efficiency, memory efficiency, training speed, model quality and inference costs.

- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.
- plateau in validation loss for rank greater than 16 i.e. using larger LoRA matrices didn't improve the performance/accuracy score.
- ranks in the range of 4 to 32 can provide a good trade-off between reducing trainable parameters and preserving performance.



PEFT methods

Selective	Selective methods are those that fine-tune only a subset of the original LLM parameters. Some of the approaches are: train only certain components of the model, specific layers, individual parameter
Reparameterization	This methods also works with the original LLM parameters but reduce the number of parameters to train by creating new low rank transformations of the original weights. A commonly used technique of this type is **LoRA** (Low-rank Adaptation). LoRA is a strategy that reduces the number of parameters to be trained during fine-tuning by freezing all of the original model parameters.
Additive	Additive methods carry out fine-tuning by keeping all of the original LLM weights frozen and introducing new trainable components. Two main approaches are : Adapter and Soft prompt.



Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

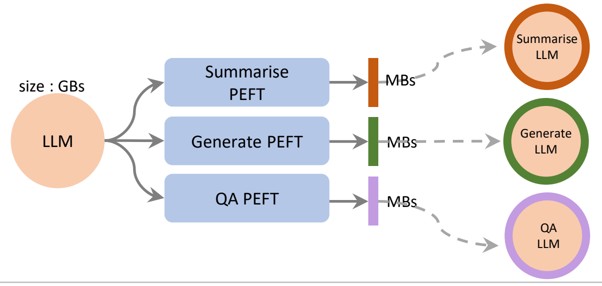
Overview of PEFT

Train small number of parameters

With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task.

New parameters combined with original LLM weights

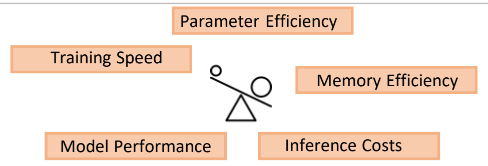
The new parameters are combined with the original LLM weights for inference, allowing efficient adaptation of the original model to multiple tasks.



PEFT Trade-off

There are several methods you can use for parameter efficient fine-tuning, each with trade-offs on: parameter efficiency, memory efficiency, training speed, model quality and inference costs.

- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.
- plateau in validation loss for rank greater than 16 i.e. using larger LoRA matrices didn't improve the performance/accuracy score.
- ranks in the range of 4 to 32 can provide a good trade-off between reducing trainable parameters and preserving performance.



PEFT methods

Selective

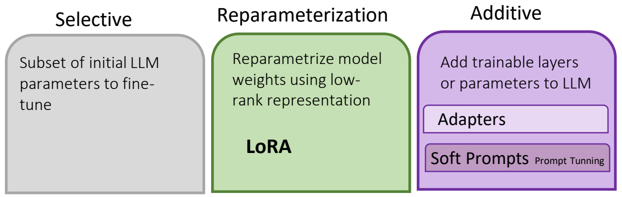
Selective methods are those that fine-tune only a subset of the original LLM parameters. Some of the approaches are: train only certain components of the model, specific layers, individual parameter

Reparameterization

This methods also works with the original LLM parameters but reduce the number of parameters to train by creating new low rank transformations of the original weights. A commonly used technique of this type is **LoRA** (Low-rank Adaptation). LoRA is a strategy that reduces the number of parameters to be trained during fine-tuning by freezing all of the original model parameters.

Additive

Additive methods carry out fine-tuning by keeping all of the original LLM weights frozen and introducing new trainable components. Two main approaches are : Adapter and Soft prompt.



How to choose LoRA rank and how does it impact LLM evaluation metrics ?

Kumar

Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

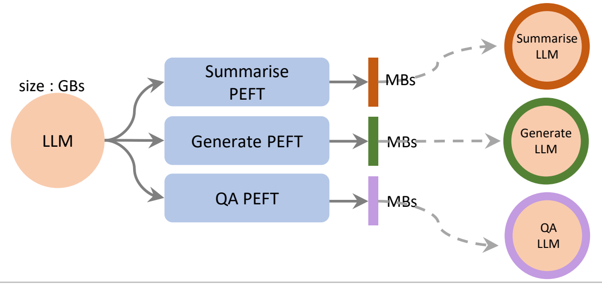
Overview of PEFT

Train small number of parameters

With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task.

New parameters combined with original LLM weights

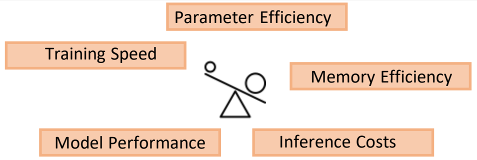
The new parameters are combined with the original LLM weights for inference, allowing efficient adaptation of the original model to multiple tasks.



PEFT Trade-off

There are several methods you can use for parameter efficient fine-tuning, each with trade-offs on: parameter efficiency, memory efficiency, training speed, model quality and inference costs.

- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.
- plateau in validation loss for rank greater than 16 i.e. using larger LoRA matrices didn't improve the performance/accuracy score.
- ranks in the range of 4 to 32 can provide a good trade-off between reducing trainable parameters and preserving performance.



PEFT methods

Selective

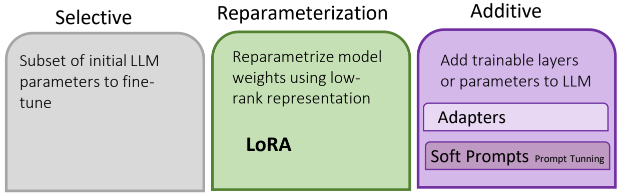
Selective methods are those that fine-tune only a subset of the original LLM parameters. Some of the approaches are: train only certain components of the model, specific layers, individual parameter

Reparameterization

This methods also works with the original LLM parameters but reduce the number of parameters to train by creating new low rank transformations of the original weights. A commonly used technique of this type is **LoRA** (Low-rank Adaptation). LoRA is a strategy that reduces the number of parameters to be trained during fine-tuning by freezing all of the original model parameters.

Additive

Additive methods carry out fine-tuning by keeping all of the original LLM weights frozen and introducing new trainable components. Two main approaches are : Adapter and Soft prompt.



How to choose LoRA rank and how does it impact LLM evaluation metrics ?

In principle, the smaller the rank, the smaller the number of trainable parameters, and the bigger the savings on compute.

The paper LoRA published by researchers at Microsoft explored how different choices of rank impact the LLM model performance. The summary of results in this table are:

LoRA-rank (r)	val_loss (validation loss)	BLEU	ROUGE_L
1	1.23	68.72	0.7052
2	1.21	69.17	0.7052
4	1.18	**70.38**	0.7186
8	1.17	69.57	**0.7196**
16	**1.16**	69.61	0.7177
32	**1.16**	69.33	0.7105
64	**1.16**	69.24	0.7180
128	**1.16**	68.73	0.7127
256	**1.16**	68.92	0.7128
512	**1.16**	68.78	0.7128
1024	1.17	69.37	0.7149

Source : LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS - by Microsoft Corporation

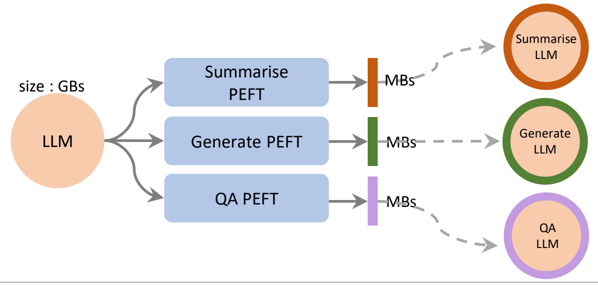
Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

Overview of PEFT

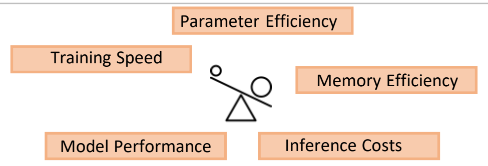
Train small number of parameters — With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task.

New parameters combined with original LLM weights — The new parameters are combined with the original LLM weights for inference, allowing efficient adaptation of the original model to multiple tasks.



PEFT Trade-off — There are several methods you can use for parameter efficient fine-tuning, each with trade-offs on: parameter efficiency, memory efficiency, training speed, model quality and inference costs.

- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.
- plateau in validation loss for rank greater than 16 i.e. using larger LoRA matrices didn't improve the performance/accuracy score.
- ranks in the range of 4 to 32 can provide a good trade-off between reducing trainable parameters and preserving performance.

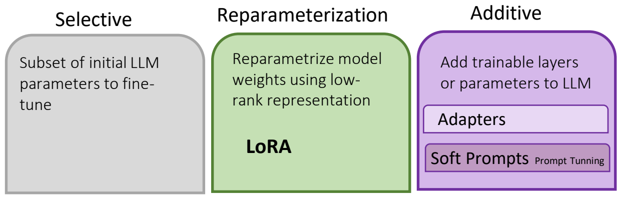


PEFT methods

Selective — Selective methods are those that fine-tune only a subset of the original LLM parameters. Some of the approaches are: train only certain components of the model, specific layers, individual parameter

Reparameterization — This methods also works with the original LLM parameters but reduce the number of parameters to train by creating new low rank transformations of the original weights. A commonly used technique of this type is **LoRA** (Low-rank Adaptation). LoRA is a strategy that reduces the number of parameters to be trained during fine-tuning by freezing all of the original model parameters.

Additive — Additive methods carry out fine-tuning by keeping all of the original LLM weights frozen and introducing new trainable components. Two main approaches are : Adapter and Soft prompt.



How to choose LoRA rank and how does it impact LLM evaluation metrics ?

In principle, the smaller the rank, the smaller the number of trainable parameters, and the bigger the savings on compute.

The paper LoRA published by researchers at Microsoft explored how different choices of rank impact the LLM model performance. The summary of results in this table are:

- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.

LoRA-rank (r)	val_loss (validation loss)	BLEU	ROUGE_L
1	1.23	68.72	0.7052
2	1.21	69.17	0.7052
4	1.18	**70.38**	0.7186
8	1.17	69.57	**0.7196**
16	**1.16**	69.61	0.7177
32	**1.16**	69.33	0.7105
64	**1.16**	69.24	0.7180
128	**1.16**	68.73	0.7127
256	**1.16**	68.92	0.7128
512	**1.16**	68.78	0.7128
1024	1.17	69.37	0.7149

Source : LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS - by Microsoft Corporation

Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

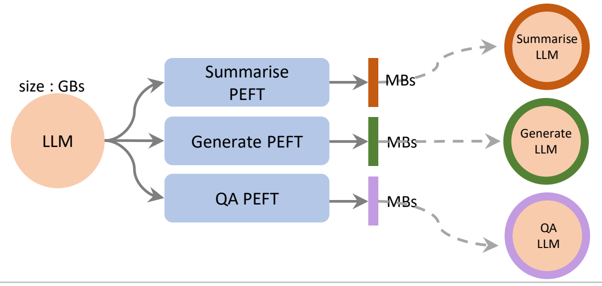
Overview of PEFT

Train small number of parameters

With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task.

New parameters combined with original LLM weights

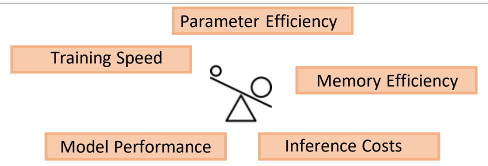
The new parameters are combined with the original LLM weights for inference, allowing efficient adaptation of the original model to multiple tasks.



PEFT Trade-off

There are several methods you can use for parameter efficient fine-tuning, each with trade-offs on: parameter efficiency, memory efficiency, training speed, model quality and inference costs.

- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.
- plateau in validation loss for rank greater than 16 i.e. using larger LoRA matrices didn't improve the performance/accuracy score.
- ranks in the range of 4 to 32 can provide a good trade-off between reducing trainable parameters and preserving performance.



PEFT methods

Selective

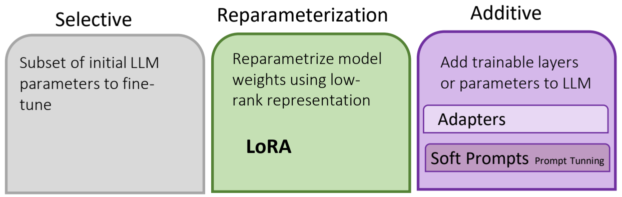
Selective methods are those that fine-tune only a subset of the original LLM parameters. Some of the approaches are: train only certain components of the model, specific layers, individual parameter

Reparameterization

This methods also works with the original LLM parameters but reduce the number of parameters to train by creating new low rank transformations of the original weights. A commonly used technique of this type is **LoRA** (Low-rank Adaptation). LoRA is a strategy that reduces the number of parameters to be trained during fine-tuning by freezing all of the original model parameters.

Additive

Additive methods carry out fine-tuning by keeping all of the original LLM weights frozen and introducing new trainable components. Two main approaches are : Adapter and Soft prompt.



How to choose LoRA rank and how does it impact LLM evaluation metrics ?

In principle, the smaller the rank, the smaller the number of trainable parameters, and the bigger the savings on compute.

The paper LoRA published by researchers at Microsoft explored how different choices of rank impact the LLM model performance. The summary of results in this table are:

- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.
- plateau in validation loss for rank greater than 16 i.e. using larger LoRA matrices didn't improve the performance/accuracy score.

LoRA-rank (r)	val_loss (validation loss)	BLEU	ROUGE_L
1	1.23	68.72	0.7052
2	1.21	69.17	0.7052
4	1.18	**70.38**	0.7186
8	1.17	69.57	**0.7196**
16	**1.16**	69.61	0.7177
32	**1.16**	69.33	0.7105
64	**1.16**	69.24	0.7180
128	**1.16**	68.73	0.7127
256	**1.16**	68.92	0.7128
512	**1.16**	68.78	0.7128
1024	1.17	69.37	0.7149

Source : LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS - by Microsoft Corporation

Generative AI Project Life Cycle - PEFT (Parameter Efficient Fine-tunning)

In full fine-tuning every model weight is updated during supervised learning, whereas in PEFT technique only updates a small subset of parameters.

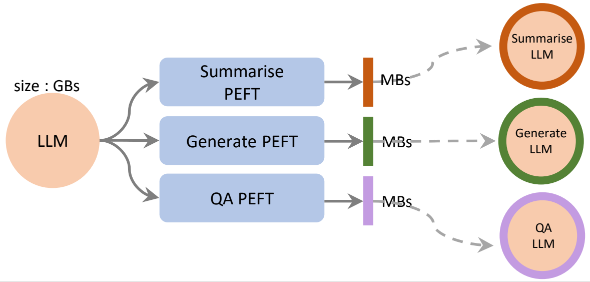
Overview of PEFT

Train small number of parameters

With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task.

New parameters combined with original LLM weights

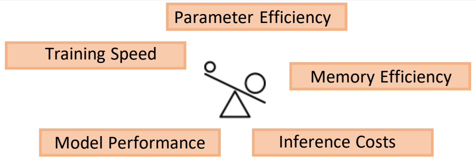
The new parameters are combined with the original LLM weights for inference, allowing efficient adaptation of the original model to multiple tasks.



PEFT Trade-off

There are several methods you can use for parameter efficient fine-tuning, each with trade-offs on: parameter efficiency, memory efficiency, training speed, model quality and inference costs.

- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.
- plateau in validation loss for rank greater than 16 i.e. using larger LoRA matrices didn't improve the performance/accuracy score.
- ranks in the range of 4 to 32 can provide a good trade-off between reducing trainable parameters and preserving performance.



PEFT methods

Selective

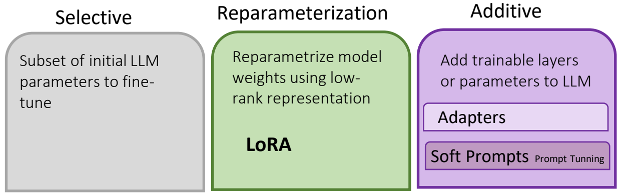
Selective methods are those that fine-tune only a subset of the original LLM parameters. Some of the approaches are: train only certain components of the model, specific layers, individual parameter

Reparameterization

This methods also works with the original LLM parameters but reduce the number of parameters to train by creating new low rank transformations of the original weights. A commonly used technique of this type is **LoRA** (Low-rank Adaptation). LoRA is a strategy that reduces the number of parameters to be trained during fine-tuning by freezing all of the original model parameters.

Additive

Additive methods carry out fine-tuning by keeping all of the original LLM weights frozen and introducing new trainable components. Two main approaches are : Adapter and Soft prompt.



How to choose LoRA rank and how does it impact LLM evaluation metrics ?

In principle, the smaller the rank, the smaller the number of trainable parameters, and the bigger the savings on compute.

The paper LoRA published by researchers at Microsoft explored how different choices of rank impact the LLM model performance. The summary of results in this table are:

- values highlighted in bold indicate the best scores that were achieved for each evaluation metric.
- plateau in validation loss for rank greater than 16 i.e. using larger LoRA matrices didn't improve the performance/accuracy score.
- ranks in the range of 4 to 32 can provide a good trade-off between reducing trainable parameters and preserving performance.

LoRA-rank (r)	val_loss (validation loss)	BLEU	ROUGE_L
1	1.23	68.72	0.7052
2	1.21	69.17	0.7052
4	1.18	**70.38**	0.7186
8	1.17	69.57	**0.7196**
16	**1.16**	69.61	0.7177
32	**1.16**	69.33	0.7105
64	**1.16**	69.24	0.7180
128	**1.16**	68.73	0.7127
256	**1.16**	68.92	0.7128
512	**1.16**	68.78	0.7128
1024	1.17	69.37	0.7149

Source : *LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS - by Microsoft Corporation*

Kumar

Adapt and align model
Evaluate

Generative AI Project Life Cycle - LLM Evaluation

LLM model leader boards publish statements like the “*model demonstrated good performance on this task*” or “*this fine-tuned model showed a large improvement in performance over the base model*”.

- What do statements like this mean ?
- How can you formalize the improvement in performance of your fine-tuned model over the pre-trained model you started with?

Generative AI Project Life Cycle - LLM Evaluation

LLM model leader boards publish statements like the "*model demonstrated good performance on this task*" or "*this fine-tuned model showed a large improvement in performance over the base model*".

- What do statements like this mean ?
- How can you formalize the improvement in performance of your fine-tuned model over the pre-trained model you started with?

Traditional ML

- In traditional machine learning, you can assess how well a model is doing by looking at its accuracy on training and validation data sets where the output is already known.
- You're able to calculate simple metrics such as accuracy, which states the fraction of all predictions that are correct because the models are deterministic.

$$\text{Accuracy} = \frac{\text{Correct Predications}}{\text{Total Predictions}}$$

Generative AI Project Life Cycle - LLM Evaluation

LLM model leader boards publish statements like the "*model demonstrated good performance on this task*" or "*this fine-tuned model showed a large improvement in performance over the base model*".

- What do statements like this mean ?
- How can you formalize the improvement in performance of your fine-tuned model over the pre-trained model you started with?

Traditional ML

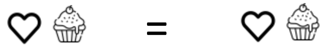
- In traditional machine learning, you can assess how well a model is doing by looking at its accuracy on training and validation data sets where the output is already known.
- You're able to calculate simple metrics such as accuracy, which states the fraction of all predictions that are correct because the models are deterministic.

$$\text{Accuracy} = \frac{\text{Correct Predications}}{\text{Total Predictions}}$$

LLM Evaluation challenges

In LLM models where the output is non-deterministic and language-based evaluation is much more challenging to assess the accuracy.
For e.g. "kids love cupcakes", this is quiet similar to "kids enjoy cupcakes"

"kids love cupcakes" "kids enjoy cupcakes"



Adapt and align model
Evaluate

Generative AI Project Life Cycle - LLM Evaluation

LLM model leader boards publish statements like the "*model demonstrated good performance on this task*" or "*this fine-tuned model showed a large improvement in performance over the base model*".

- What do statements like this mean ?
- How can you formalize the improvement in performance of your fine-tuned model over the pre-trained model you started with?

Traditional ML

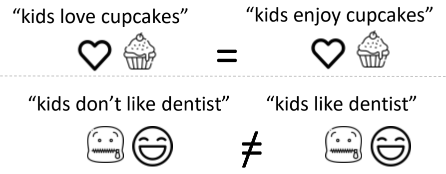
- In traditional machine learning, you can assess how well a model is doing by looking at its accuracy on training and validation data sets where the output is already known.
- You're able to calculate simple metrics such as accuracy, which states the fraction of all predictions that are correct because the models are deterministic.

$$\text{Accuracy} = \frac{\text{Correct Predications}}{\text{Total Predictions}}$$

LLM Evaluation challenges

In LLM models where the output is non-deterministic and language-based evaluation is much more challenging to assess the accuracy. For e.g. "kids love cupcakes", this is quiet similar to "kids enjoy cupcakes"

How do you measure the similarity ? For e.g. "kids don't like dentist" and "kids like dentist". There is only one word difference b/w two sentence however the meaning is completely difference when compared to above example.



Generative AI Project Life Cycle - LLM Evaluation

LLM model leader boards publish statements like the "*model demonstrated good performance on this task*" or "*this fine-tuned model showed a large improvement in performance over the base model*".

- What do statements like this mean ?
- How can you formalize the improvement in performance of your fine-tuned model over the pre-trained model you started with?

Traditional ML

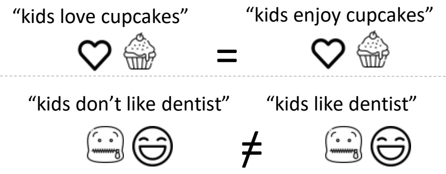
- In traditional machine learning, you can assess how well a model is doing by looking at its accuracy on training and validation data sets where the output is already known.
- You're able to calculate simple metrics such as accuracy, which states the fraction of all predictions that are correct because the models are deterministic.

$$\text{Accuracy} = \frac{\text{Correct Predications}}{\text{Total Predictions}}$$

LLM Evaluation challenges

In LLM models where the output is non-deterministic and language-based evaluation is much more challenging to assess the accuracy. For e.g. "kids love cupcakes", this is quiet similar to "kids enjoy cupcakes"

How do you measure the similarity ? For e.g. "kids don't like dentist" and "kids like dentist". There is only one word difference b/w two sentence however the meaning is completely difference when compared to above example.



LLM Evaluation – Metrics

For humans' brains, we can see the similarities and differences. But when you train a model on millions of sentences, you need an automated, structured way to make measurements
ROUGE and BLEU are two widely used evaluation metrics for different tasks

Generative AI Project Life Cycle - LLM Evaluation

Adapt and align model
Evaluate

LLM model leader boards publish statements like the "*model demonstrated good performance on this task*" or "*this fine-tuned model showed a large improvement in performance over the base model*".

- What do statements like this mean ?
- How can you formalize the improvement in performance of your fine-tuned model over the pre-trained model you started with?

Traditional ML

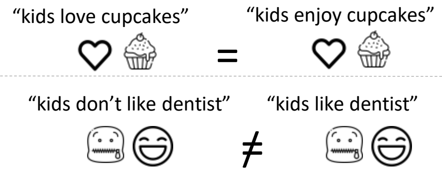
- In traditional machine learning, you can assess how well a model is doing by looking at its accuracy on training and validation data sets where the output is already known.
- You're able to calculate simple metrics such as accuracy, which states the fraction of all predictions that are correct because the models are deterministic.

$$\text{Accuracy} = \frac{\text{Correct Predications}}{\text{Total Predictions}}$$

LLM Evaluation challenges

In LLM models where the output is non-deterministic and language-based evaluation is much more challenging to assess the accuracy. For e.g. "kids love cupcakes", this is quiet similar to "kids enjoy cupcakes"

How do you measure the similarity ? For e.g. "kids don't like dentist" and "kids like dentist". There is only one word difference b/w two sentence however the meaning is completely difference when compared to above example.



LLM Evaluation – Metrics

For humans' brains, we can see the similarities and differences. But when you train a model on millions of sentences, you need an automated, structured way to make measurements

ROUGE and BLEU are two widely used evaluation metrics for different tasks

ROUGE

ROUGE or recall is primarily employed to assess the quality of automatically generated summaries by comparing them to human-generated reference summaries.

ROGUE

- Used for text summarization
- Compares a summary to one or more reference summaries

Generative AI Project Life Cycle - LLM Evaluation

LLM model leader boards publish statements like the "*model demonstrated good performance on this task*" or "*this fine-tuned model showed a large improvement in performance over the base model*".

- What do statements like this mean ?
- How can you formalize the improvement in performance of your fine-tuned model over the pre-trained model you started with?

Traditional ML

- In traditional machine learning, you can assess how well a model is doing by looking at its accuracy on training and validation data sets where the output is already known.
- You're able to calculate simple metrics such as accuracy, which states the fraction of all predictions that are correct because the models are deterministic.

$$\text{Accuracy} = \frac{\text{Correct Predications}}{\text{Total Predictions}}$$

LLM Evaluation challenges

In LLM models where the output is non-deterministic and language-based evaluation is much more challenging to assess the accuracy. For e.g. "kids love cupcakes", this is quiet similar to "kids enjoy cupcakes"

"kids love cupcakes" = "kids enjoy cupcakes"

How do you measure the similarity ? For e.g. "kids don't like dentist" and "kids like dentist". There is only one word difference b/w two sentence however the meaning is completely difference when compared to above example.

"kids don't like dentist" ≠ "kids like dentist"

LLM Evaluation – Metrics

For humans' brains, we can see the similarities and differences. But when you train a model on millions of sentences, you need an automated, structured way to make measurements
ROUGE and BLEU are two widely used evaluation metrics for different tasks

ROUGE

ROUGE or recall is primarily employed to assess the quality of automatically generated summaries by comparing them to human-generated reference summaries.

BLEU

BLEU, or bilingual evaluation understudy is an algorithm designed to evaluate the quality of machine-translated text, again, by comparing it to human-generated translations.

ROGUE
- Used for text summarization
- Compares a summary to one or more reference summaries

BLEU
- Used for text translation
- Compares to human-generated translations

Generative AI Project Life Cycle - LLM Evaluation *continue…..*

LLM Evaluation Metrics Terminology	Before we start calculating metrics. Let's review some terminology. In the anatomy of language: • A unigram is equivalent to a single word • A bigram is two words • n-gram is a group of n-words

$$\textbf{It } \overbrace{\textbf{is } \underbrace{\textbf{very humidity}}_{\text{bigram}}}^{\text{n-gram}} \; \underbrace{\textbf{today.}}_{\text{unigram}}$$

Generative AI Project Life Cycle - LLM Evaluation *continue…..*

LLM Evaluation Metrics Terminology

Before we start calculating metrics. Let's review some terminology.
In the anatomy of language:

- A unigram is equivalent to a single word
- A bigram is two words
- n-gram is a group of n-words

n-gram

It is very humidity today.

bigram unigram

ROUG-1 calculation

You can perform simple metric calculations similar to other machine-learning tasks using recall, precision, and F1.
e.g. a human-generated reference sentence is “It is humidity today.” and LLM generated is “It is very humidity today”

Adapt and align model
Evaluate

Generative AI Project Life Cycle - LLM Evaluation *continue…..*

LLM Evaluation Metrics Terminology	Before we start calculating metrics. Let's review some terminology. In the anatomy of language: • A unigram is equivalent to a single word • A bigram is two words • n-gram is a group of n-words

n-gram

It is very humidity today.

bigram unigram

ROUG-1 calculation	You can perform simple metric calculations similar to other machine-learning tasks using recall, precision, and F1. e.g. a human-generated reference sentence is “It is humidity today.” and LLM generated is “It is very humidity today”
ROUG-1 Recall	The recall metric measures the number of words or unigrams that are matched between the reference and the generated output divided by the number of words or unigrams in the reference.

Reference(human) : It is humidity today
Generated : It is very humidity today

$$\text{ROUGE-1 Recall} = \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$$

Generative AI Project Life Cycle - LLM Evaluation *continue…..*

Adapt and align model
Evaluate

LLM Evaluation Metrics Terminology	Before we start calculating metrics. Let's review some terminology. In the anatomy of language: • A unigram is equivalent to a single word • A bigram is two words • n-gram is a group of n-words	$\textbf{It is } \overbrace{\underbrace{\textbf{very humidity}}_{\text{bigram}}}^{\text{n-gram}} \underbrace{\textbf{today}}_{\text{unigram}}\textbf{.}$
ROUG-1 calculation	You can perform simple metric calculations similar to other machine-learning tasks using recall, precision, and F1. e.g. a human-generated reference sentence is “It is humidity today.” and LLM generated is “It is very humidity today”	**Reference(human)** : It is humidity today **Generated** : It is very humidity today
ROUG-1 Recall	The recall metric measures the number of words or unigrams that are matched between the reference and the generated output divided by the number of words or unigrams in the reference.	$\text{ROUGE-1 Recall} = \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$
ROUG-1 Precision	Precision measures the unigram matches divided by the output size.	$\text{ROUGE-1 Precision:} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$

Generative AI Project Life Cycle - LLM Evaluation *continue…..*

Adapt and align model
Evaluate

LLM Evaluation Metrics Terminology	Before we start calculating metrics. Let's review some terminology. In the anatomy of language: • A unigram is equivalent to a single word • A bigram is two words • n-gram is a group of n-words

$$\overbrace{\textbf{It is } \underbrace{\textbf{very humidity}}_{\text{bigram}}}^{\text{n-gram}} \ \underbrace{\textbf{today.}}_{\text{unigram}}$$

ROUG-1 calculation	You can perform simple metric calculations similar to other machine-learning tasks using recall, precision, and F1. e.g. a human-generated reference sentence is “It is humidity today.” and LLM generated is “It is very humidity today”
ROUG-1 Recall	The recall metric measures the number of words or unigrams that are matched between the reference and the generated output divided by the number of words or unigrams in the reference.
ROUG-1 Precision	Precision measures the unigram matches divided by the output size.
ROUG-1 F1	F1 score is the harmonic mean of both values.

Reference(human) : It is humidity today
Generated : It is very humidity today

$$\text{ROUGE-1 Recall} = \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$$

$$\text{ROUGE-1 Precision:} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$$

$$\text{ROUGE-1 F1:} = 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.8}{1.8} = 0.89$$

Generative AI Project Life Cycle - LLM Evaluation *continue…..*

Adapt and align model
Evaluate

LLM Evaluation Metrics Terminology	Before we start calculating metrics. Let's review some terminology. In the anatomy of language: • A unigram is equivalent to a single word • A bigram is two words • n-gram is a group of n-words	$\textbf{It}\ \overbrace{\textbf{is}\ \underbrace{\textbf{very humidity}}_{\text{bigram}}}^{\text{n-gram}}\ \underbrace{\textbf{today.}}_{\text{unigram}}$
ROUG-1 calculation	You can perform simple metric calculations similar to other machine-learning tasks using recall, precision, and F1. e.g. a human-generated reference sentence is "It is humidity today." and LLM generated is "It is very humidity today"	**Reference(human)** : It is humidity today **Generated** : It is very humidity today
ROUG-1 Recall	The recall metric measures the number of words or unigrams that are matched between the reference and the generated output divided by the number of words or unigrams in the reference.	ROUGE-1 Recall $= \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$
ROUG-1 Precision	Precision measures the unigram matches divided by the output size.	ROUGE-1 Precision: $= \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$
ROUG-1 F1	F1 score is the harmonic mean of both values.	ROUGE-1 F1: $= 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.8}{1.8} = 0.89$
ROUG-2 calculation	You can get a slightly better score by taking into account bigrams(collections of two words). By working with pairs of words you're acknowledging in a very simple way, the ordering of the words in the sentence.	**Reference(human)** : It is / is humidity / humidity today **Generated** : It is / is very / very humidity / humidty today

Generative AI Project Life Cycle - LLM Evaluation *continue…..*

LLM Evaluation Metrics Terminology

Before we start calculating metrics. Let's review some terminology.
In the anatomy of language:

- A unigram is equivalent to a single word
- A bigram is two words
- n-gram is a group of n-words

n-gram

It is very humidity today.

bigram unigram

ROUG-1 calculation

You can perform simple metric calculations similar to other machine-learning tasks using recall, precision, and F1.
e.g. a human-generated reference sentence is “It is humidity today.” and LLM generated is “It is very humidity today”

ROUG-1 Recall

The recall metric measures the number of words or unigrams that are matched between the reference and the generated output divided by the number of words or unigrams in the reference.

ROUG-1 Precision

Precision measures the unigram matches divided by the output size.

ROUG-1 F1

F1 score is the harmonic mean of both values.

Reference(human) : It is humidity today
Generated : It is very humidity today

$$\text{ROUGE-1 Recall} = \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$$

$$\text{ROUGE-1 Precision:} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$$

$$\text{ROUGE-1 F1:} = 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.8}{1.8} = 0.89$$

ROUG-2 calculation

You can get a slightly better score by taking into account bigrams(collections of two words). By working with pairs of words you're acknowledging in a very simple way, the ordering of the words in the sentence.

ROUG-2 Recall, Precision, F1

You can get a slightly better score by taking into account bigrams(collections of two words). By working with pairs of words you're acknowledging in a very simple way, the ordering of the words in the sentence.
In this example the scores are lower than the ROUGE-1 scores. With longer sentences, they're a greater chance that bigrams don't match, and the scores may be even lower.

Reference(human) : It is is humidity humidity today
Generated : It is is very very humidity humidty today

$$\text{ROUGE-2 Recall:} = \frac{\text{bigram matches}}{\text{bigrams in reference}} = \frac{2}{3} = 0.67$$

$$\text{ROUGE-2 Precision:} = \frac{\text{bigram matches}}{\text{bigrams in output}} = \frac{2}{4} = 0.5$$

$$\text{ROUGE-2 F1:} = 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.335}{1.17} = 0.57$$

Generative AI Project Life Cycle - LLM Evaluation *continue…..*

Adapt and align model
Evaluate

LLM Evaluation Metrics Terminology

Before we start calculating metrics. Let's review some terminology.
In the anatomy of language:

- A unigram is equivalent to a single word
- A bigram is two words
- n-gram is a group of n-words

n-gram

It is very humidity today.

bigram unigram

ROUG-1 calculation

You can perform simple metric calculations similar to other machine-learning tasks using recall, precision, and F1.
e.g. a human-generated reference sentence is "It is humidity today." and LLM generated is "It is very humidity today"

Reference(human) : It is humidity today
Generated : It is very humidity today

ROUG-1 Recall

The recall metric measures the number of words or unigrams that are matched between the reference and the generated output divided by the number of words or unigrams in the reference.

$$\text{ROUGE-1 Recall} = \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$$

ROUG-1 Precision

Precision measures the unigram matches divided by the output size.

$$\text{ROUGE-1 Precision:} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$$

ROUG-1 F1

F1 score is the harmonic mean of both values.

$$\text{ROUGE-1 F1:} = 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.8}{1.8} = 0.89$$

ROUG-2 calculation

You can get a slightly better score by taking into account bigrams(collections of two words). By working with pairs of words you're acknowledging in a very simple way, the ordering of the words in the sentence.

Reference(human) : It is is humidity humidity today
Generated : It is is very very humidity humidty today

ROUG-2 Recall, Precision, F1

You can get a slightly better score by taking into account bigrams(collections of two words). By working with pairs of words you're acknowledging in a very simple way, the ordering of the words in the sentence.
In this example the scores are lower than the ROUGE-1 scores. With longer sentences, they're a greater chance that bigrams don't match, and the scores may be even lower.

$$\text{ROUGE-2 Recall:} = \frac{\text{bigram matches}}{\text{bigrams in reference}} = \frac{2}{3} = 0.67$$

$$\text{ROUGE-2 Precision:} = \frac{\text{bigram matches}}{\text{bigrams in output}} = \frac{2}{4} = 0.5$$

$$\text{ROUGE-2 F1:} = 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.335}{1.17} = 0.57$$

ROUG-L

Another approach is, look for the longest common subsequence (LCS) present in both the generated output and the reference output.

Reference(human) : It is humidity today
Generated : It is very humidity today

$$\text{ROUGE-L Recall:} = \frac{\text{LCS(Gen, Ref)}}{\text{unigrams in reference}} = \frac{2}{4} = 0.5$$

$$\text{ROUGE-L Precision:} = \frac{\text{LCS(Gen, Ref)}}{\text{unigrams in output}} = \frac{2}{5} = 0.4$$

$$\text{ROUGE-L F1:} = 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.2}{0.9} = 0.44$$

Generative AI Project Life Cycle - LLM Evaluation *continue…..*

LLM Evaluation Metrics Terminology

Before we start calculating metrics. Let's review some terminology.
In the anatomy of language:

- A unigram is equivalent to a single word
- A bigram is two words
- n-gram is a group of n-words

$$\overbrace{\text{It is very humidity}}^{\text{n-gram}}\ \underbrace{\text{today.}}_{\text{unigram}}$$

$$\text{It}\ \underbrace{\text{is very humidity}}_{\text{bigram}}\ \text{today.}$$

ROUG-1 calculation

You can perform simple metric calculations similar to other machine-learning tasks using recall, precision, and F1.
e.g. a human-generated reference sentence is "It is humidity today." and LLM generated is "It is very humidity today"

Reference(human) : It is humidity today
Generated : It is very humidity today

ROUG-1 Recall

The recall metric measures the number of words or unigrams that are matched between the reference and the generated output divided by the number of words or unigrams in the reference.

$$\text{ROUGE-1 Recall} = \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$$

ROUG-1 Precision

Precision measures the unigram matches divided by the output size.

$$\text{ROUGE-1 Precision:} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$$

ROUG-1 F1

F1 score is the harmonic mean of both values.

$$\text{ROUGE-1 F1:} = 2\ \frac{\text{precision x recall}}{\text{precision + recall}} = 2\ \frac{0.8}{1.8} = 0.89$$

ROUG-2 calculation

You can get a slightly better score by taking into account bigrams(collections of two words). By working with pairs of words you're acknowledging in a very simple way, the ordering of the words in the sentence.

Reference(human) : It is | is humidity | humidity today
Generated : It is | is very | very humidity | humidty today

ROUG-2 Recall, Precision, F1

You can get a slightly better score by taking into account bigrams(collections of two words). By working with pairs of words you're acknowledging in a very simple way, the ordering of the words in the sentence.
In this example the scores are lower than the ROUGE-1 scores. With longer sentences, they're a greater chance that bigrams don't match, and the scores may be even lower.

$$\text{ROUGE-2 Recall:} = \frac{\text{bigram matches}}{\text{bigrams in reference}} = \frac{2}{3} = 0.67$$

$$\text{ROUGE-2 Precision:} = \frac{\text{bigram matches}}{\text{bigrams in output}} = \frac{2}{4} = 0.5$$

$$\text{ROUGE-2 F1:} = 2\ \frac{\text{precision x recall}}{\text{precision + recall}} = 2\ \frac{0.335}{1.17} = 0.57$$

ROUG-L

Another approach is, look for the longest common subsequence (LCS) present in both the generated output and the reference output.

ROUG-clipping

If generated output contains same repeated words you may have to use clipping function to limit number of unigrams.

$$\text{Modified precision} = \frac{\text{clip(unigram matches)}}{\text{unigrams in output}}$$

Reference(human) : It is humidity today
Generated : It is very humidity today

$$\text{ROUGE-L Recall:} = \frac{\text{LCS(Gen, Ref)}}{\text{unigrams in reference}} = \frac{2}{4} = 0.5$$

$$\text{ROUGE-L Precision:} = \frac{\text{LCS(Gen, Ref)}}{\text{unigrams in output}} = \frac{2}{5} = 0.4$$

$$\text{ROUGE-L F1:} = 2\ \frac{\text{precision x recall}}{\text{precision + recall}} = 2\ \frac{0.2}{0.9} = 0.44$$

Kumar

Generative AI Project Life Cycle - LLM Evaluation *continue…..*

Adapt and align model
Evaluate

LLM Evaluation Metrics Terminology	Before we start calculating metrics. Let's review some terminology. In the anatomy of language: • A unigram is equivalent to a single word • A bigram is two words • n-gram is a group of n-words	n-gram **It is very humidity today.** bigram unigram
ROUG-1 calculation	You can perform simple metric calculations similar to other machine-learning tasks using recall, precision, and F1. e.g. a human-generated reference sentence is “It is humidity today.” and LLM generated is “It is very humidity today”	**Reference(human)** : It is humidity today **Generated** : It is very humidity today
ROUG-1 Recall	The recall metric measures the number of words or unigrams that are matched between the reference and the generated output divided by the number of words or unigrams in the reference.	ROUGE-1 Recall $= \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$
ROUG-1 Precision	Precision measures the unigram matches divided by the output size.	ROUGE-1 Precision: $= \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$
ROUG-1 F1	F1 score is the harmonic mean of both values.	ROUGE-1 F1: $= 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.8}{1.8} = 0.89$
ROUG-2 calculation	You can get a slightly better score by taking into account bigrams(collections of two words). By working with pairs of words you're acknowledging in a very simple way, the ordering of the words in the sentence.	**Reference(human)** : It is is humidity humidity today **Generated** : It is is very very humidity humidty today
ROUG-2 Recall, Precision, F1	You can get a slightly better score by taking into account bigrams(collections of two words). By working with pairs of words you're acknowledging in a very simple way, the ordering of the words in the sentence. In this example the scores are lower than the ROUGE-1 scores. With longer sentences, they're a greater chance that bigrams don't match, and the scores may be even lower.	ROUGE-2 Recall: $= \frac{\text{bigram matches}}{\text{bigrams in reference}} = \frac{2}{3} = 0.67$ ROUGE-2 Precision: $= \frac{\text{bigram matches}}{\text{bigrams in output}} = \frac{2}{4} = 0.5$ ROUGE-2 F1: $= 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.335}{1.17} = 0.57$
ROUG-L **ROUG-clipping**	Another approach is, look for the longest common subsequence (LCS) present in both the generated output and the reference output. If generated output contains same repeated words you may have to use clipping function to limit number of unigrams. Modified precision $= \frac{\text{clip(unigram matches)}}{\text{unigrams in output}}$	**Reference(human)** : It is humidity today **Generated** : It is very humidity today ROUGE-L Recall: $= \frac{\text{LCS(Gen, Ref)}}{\text{unigrams in reference}} = \frac{2}{4} = 0.5$ ROUGE-L Precision: $= \frac{\text{LCS(Gen, Ref)}}{\text{unigrams in output}} = \frac{2}{5} = 0.4$ ROUGE-L F1: $= 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.2}{0.9} = 0.44$
BLEU score	BLEU (bilingual evaluation under study) score is useful for evaluating the quality of machine-translated text. The BLEU score quantifies the quality of a translation by checking how many n-grams in the machine-generated translation match those in the reference translation. To calculate the score, you average precision across a range of different n-gram sizes. Calculating the BLEU score is easy with pre-written libraries from providers like Hugging Face.	BLEU metric = Avg(precision across range of n-gram sizes)

Kumar

Generative AI Project Life Cycle - Reinforcement learning for human feedback

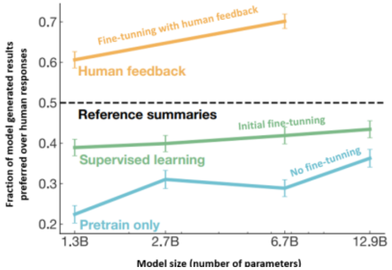
Use the reward model in the reinforcement learning process to update the LLM weights and produce a human aligned model.
✓ It's important you want to start with a model that already has a good performance on your task of interests.

Generative AI Project Life Cycle - Reinforcement learning for human feedback

Use the reward model in the reinforcement learning process to update the LLM weights and produce a human aligned model.
✓ It's important you want to start with a model that already has a good performance on your task of interests.

Why RLHF

- In 2020, researchers at OpenAI published a paper that explored the use of fine-tuning with human feedback to train a model. The article demonstrates model fine-tuned on human feedback produced better responses than a pretrained model, an instruct fine-tuned model, and even the reference human baseline. (source: OpenAI paper Learning to summarize from human feedback)
- A popular technique to finetune large language models with human feedback is called reinforcement learning from human feedback (RLHF).

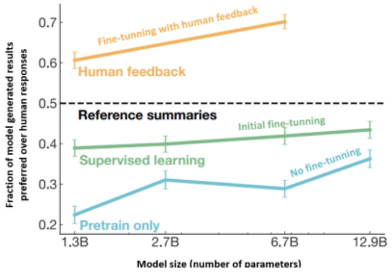


Generative AI Project Life Cycle - Reinforcement learning for human feedback

Use the reward model in the reinforcement learning process to update the LLM weights and produce a human aligned model.
✓ It's important you want to start with a model that already has a good performance on your task of interests.

Why RLHF

- In 2020, researchers at OpenAI published a paper that explored the use of fine-tuning with human feedback to train a model. The article demonstrates model fine-tuned on human feedback produced better responses than a pretrained model, an instruct fine-tuned model, and even the reference human baseline. (source: [OpenAI paper Learning to summarize from human feedback])
- A popular technique to finetune large language models with human feedback is called reinforcement learning from human feedback (RLHF).



Steps to obtain Feedback from humans for RLHF

1
- The first step in fine-tuning an LLM with RLHF is to select a model to work with and use it to prepare a data set for human feedback.
- In general, you may find it easier to start with an instruct model that has already been fine-tuned across many tasks and has some general capabilities.

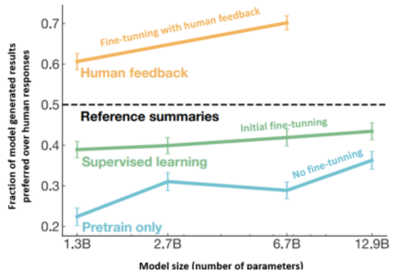
Kumar

Generative AI Project Life Cycle - Reinforcement learning for human feedback

Use the reward model in the reinforcement learning process to update the LLM weights and produce a human aligned model.
✓ It's important you want to start with a model that already has a good performance on your task of interests.

Why RLHF

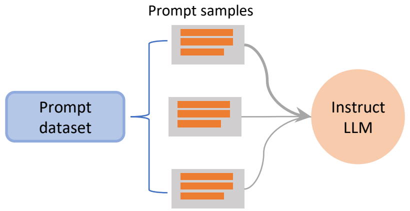
- In 2020, researchers at OpenAI published a paper that explored the use of fine-tuning with human feedback to train a model. The article demonstrates model fine-tuned on human feedback produced better responses than a pretrained model, an instruct fine-tuned model, and even the reference human baseline. (source: OpenAI paper Learning to summarize from human feedback)
- A popular technique to finetune large language models with human feedback is called reinforcement learning from human feedback (RLHF).



Steps to obtain Feedback from humans for RLHF

1
- The first step in fine-tuning an LLM with RLHF is to select a model to work with and use it to prepare a data set for human feedback.
- In general, you may find it easier to start with an instruct model that has already been fine-tuned across many tasks and has some general capabilities.

2
- Then use this LLM along with a prompt data set to generate a number of different responses for each prompt.

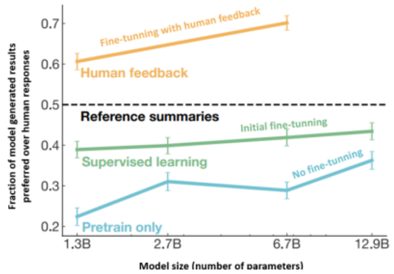


Generative AI Project Life Cycle - Reinforcement learning for human feedback

Use the reward model in the reinforcement learning process to update the LLM weights and produce a human aligned model.
✓ It's important you want to start with a model that already has a good performance on your task of interests.

Why RLHF

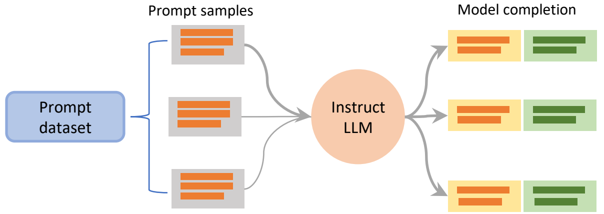
- In 2020, researchers at OpenAI published a paper that explored the use of fine-tuning with human feedback to train a model. The article demonstrates model fine-tuned on human feedback produced better responses than a pretrained model, an instruct fine-tuned model, and even the reference human baseline. (source: OpenAI paper Learning to summarize from human feedback)
- A popular technique to finetune large language models with human feedback is called reinforcement learning from human feedback (RLHF).



Steps to obtain Feedback from humans for RLHF

1
- The first step in fine-tuning an LLM with RLHF is to select a model to work with and use it to prepare a data set for human feedback.
- In general, you may find it easier to start with an instruct model that has already been fine-tuned across many tasks and has some general capabilities.

2
- Then use this LLM along with a prompt data set to generate a number of different responses for each prompt.
- The prompt dataset consists of multiple prompts, each of which gets processed by the LLM to produce a set of completions.



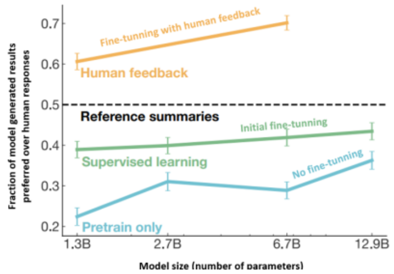
Adapt and align model
Align with human feedback

Generative AI Project Life Cycle - Reinforcement learning for human feedback

Use the reward model in the reinforcement learning process to update the LLM weights and produce a human aligned model.
✓ It's important you want to start with a model that already has a good performance on your task of interests.

Why RLHF

- In 2020, researchers at OpenAI published a paper that explored the use of fine-tuning with human feedback to train a model. The article demonstrates model fine-tuned on human feedback produced better responses than a pretrained model, an instruct fine-tuned model, and even the reference human baseline. (source: [OpenAI paper Learning to summarize from human feedback]())
- A popular technique to finetune large language models with human feedback is called reinforcement learning from human feedback (RLHF).

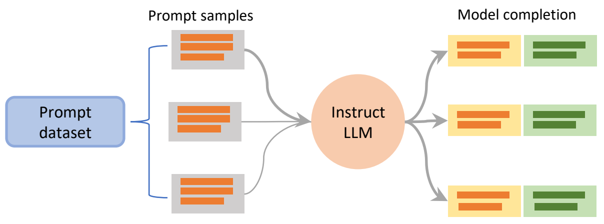


<u>Steps to obtain Feedback from humans for RLHF</u>

1
- The first step in fine-tuning an LLM with RLHF is to select a model to work with and use it to prepare a data set for human feedback.
- In general, you may find it easier to start with an instruct model that has already been fine-tuned across many tasks and has some general capabilities.

2
- Then use this LLM along with a prompt data set to generate a number of different responses for each prompt.
- The prompt dataset consists of multiple prompts, each of which gets processed by the LLM to produce a set of completions.

3
- The next step is to <u>collect feedback from human labellers</u> on the completions generated by the LLM. This is the human feedback portion of reinforcement.
- First, you must decide what criterion you want the humans to assess the completions on. This could be any of the issues like helpfulness or toxicity.



[Kumar]()

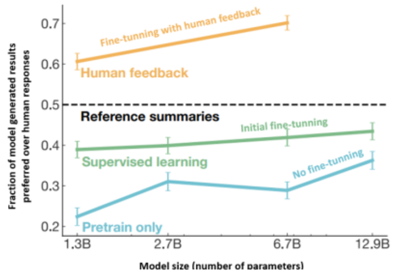
Generative AI Project Life Cycle - Reinforcement learning for human feedback

Adapt and align model
Align with human feedback

Use the reward model in the reinforcement learning process to update the LLM weights and produce a human aligned model.
✓ It's important you want to start with a model that already has a good performance on your task of interests.

Why RLHF

- In 2020, researchers at OpenAI published a paper that explored the use of fine-tuning with human feedback to train a model. The article demonstrates model fine-tuned on human feedback produced better responses than a pretrained model, an instruct fine-tuned model, and even the reference human baseline. (source: OpenAI paper Learning to summarize from human feedback)
- A popular technique to finetune large language models with human feedback is called reinforcement learning from human feedback (RLHF).



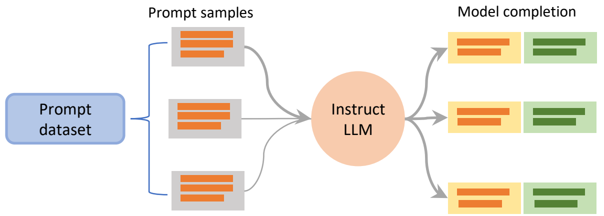
Steps to obtain Feedback from humans for RLHF

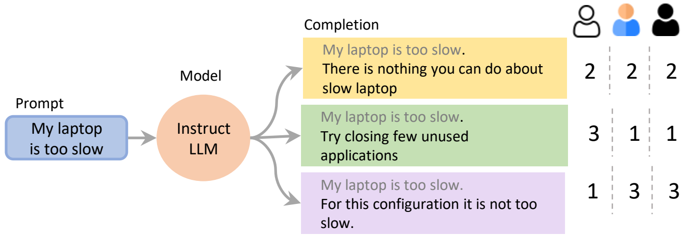
1
- The first step in fine-tuning an LLM with RLHF is to select a model to work with and use it to prepare a data set for human feedback.
- In general, you may find it easier to start with an instruct model that has already been fine-tuned across many tasks and has some general capabilities.

2
- Then use this LLM along with a prompt data set to generate a number of different responses for each prompt.
- The prompt dataset consists of multiple prompts, each of which gets processed by the LLM to produce a set of completions.

3
- The next step is to collect feedback from human labellers on the completions generated by the LLM. This is the human feedback portion of reinforcement.
- First, you must decide what criterion you want the humans to assess the completions on. This could be any of the issues like helpfulness or toxicity.

4
- The task for labellers in this example is to rank the three completions in order of helpfulness from the most helpful to least helpful.
- Once human labellers have completed their assessments of prompt completion sets, you have the data to train the reward model. Which will use instead of humans to classify model completions during the reinforcement learning finetuning process.



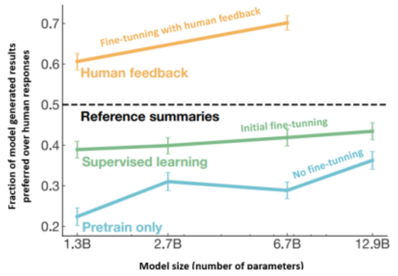


Generative AI Project Life Cycle - Reinforcement learning for human feedback

Use the reward model in the reinforcement learning process to update the LLM weights and produce a human aligned model.
✓ It's important you want to start with a model that already has a good performance on your task of interests.

Why RLHF

- In 2020, researchers at OpenAI published a paper that explored the use of fine-tuning with human feedback to train a model. The article demonstrates model fine-tuned on human feedback produced better responses than a pretrained model, an instruct fine-tuned model, and even the reference human baseline. (source: [OpenAI paper Learning to summarize from human feedback]())
- A popular technique to finetune large language models with human feedback is called reinforcement learning from human feedback (RLHF).



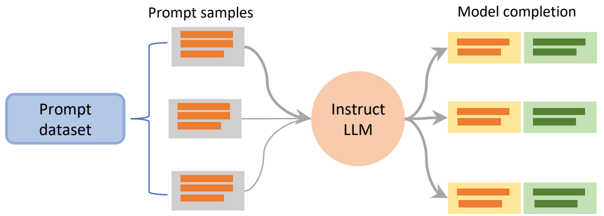
Steps to obtain Feedback from humans for RLHF

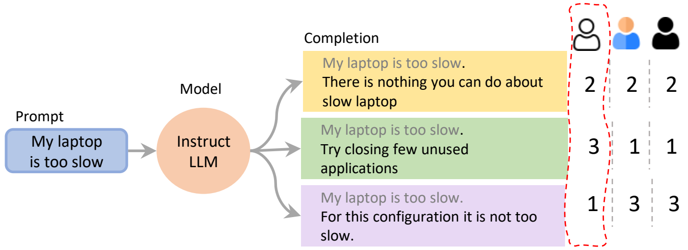
1
- The first step in fine-tuning an LLM with RLHF is to select a model to work with and use it to prepare a data set for human feedback.
- In general, you may find it easier to start with an instruct model that has already been fine-tuned across many tasks and has some general capabilities.

2
- Then use this LLM along with a prompt data set to generate a number of different responses for each prompt.
- The prompt dataset consists of multiple prompts, each of which gets processed by the LLM to produce a set of completions.

3
- The next step is to collect feedback from human labellers on the completions generated by the LLM. This is the human feedback portion of reinforcement.
- First, you must decide what criterion you want the humans to assess the completions on. This could be any of the issues like helpfulness or toxicity.

4
- The task for labellers in this example is to rank the three completions in order of helpfulness from the most helpful to least helpful.
- Once human labellers have completed their assessments of prompt completion sets, you have the data to train the reward model. Which will use instead of humans to classify model completions during the reinforcement learning finetuning process.
- In this e.g. first labeller responses disagree with the others and may indicate that they misunderstood the instructions.



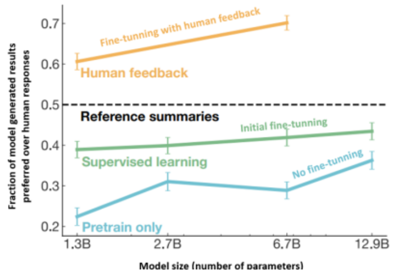


Generative AI Project Life Cycle - Reinforcement learning for human feedback

Use the reward model in the reinforcement learning process to update the LLM weights and produce a human aligned model.
✓ It's important you want to start with a model that already has a good performance on your task of interests.

Why RLHF

- In 2020, researchers at OpenAI published a paper that explored the use of fine-tuning with human feedback to train a model. The article demonstrates model fine-tuned on human feedback produced better responses than a pretrained model, an instruct fine-tuned model, and even the reference human baseline. (source: OpenAI paper Learning to summarize from human feedback)
- A popular technique to finetune large language models with human feedback is called reinforcement learning from human feedback (RLHF).



Steps to obtain Feedback from humans for RLHF

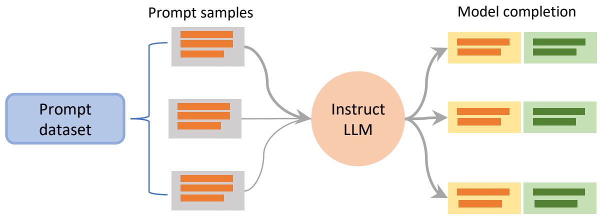
1
- The first step in fine-tuning an LLM with RLHF is to select a model to work with and use it to prepare a data set for human feedback.
- In general, you may find it easier to start with an instruct model that has already been fine-tuned across many tasks and has some general capabilities.

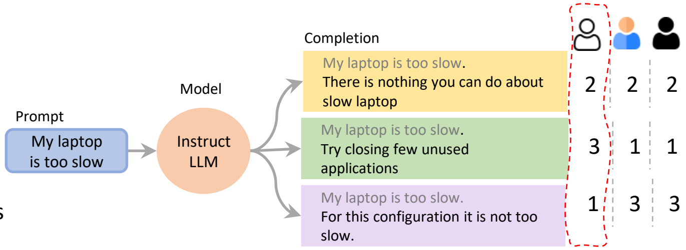
2
- Then use this LLM along with a prompt data set to generate a number of different responses for each prompt.
- The prompt dataset consists of multiple prompts, each of which gets processed by the LLM to produce a set of completions.

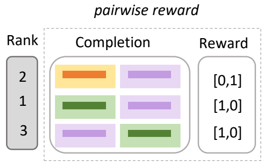
3
- The next step is to collect feedback from human labellers on the completions generated by the LLM. This is the human feedback portion of reinforcement.
- First, you must decide what criterion you want the humans to assess the completions on. This could be any of the issues like helpfulness or toxicity.

4
- The task for labellers in this example is to rank the three completions in order of helpfulness from the most helpful to least helpful.
- Once human labellers have completed their assessments of prompt completion sets, you have the data to train the reward model. Which will use instead of humans to classify model completions during the reinforcement learning finetuning process.
- In this e.g. first labeller responses disagree with the others and may indicate that they misunderstood the instructions.

5
- Before training the reward model, you need to convert the ranking data into a pairwise comparison of completions i.e. all possible pairs of completions should be classified as 0 or 1 score.
- For each pair, you will assign a reward of 1 for the preferred response and a reward of 0 for the less preferred response.







Kumar

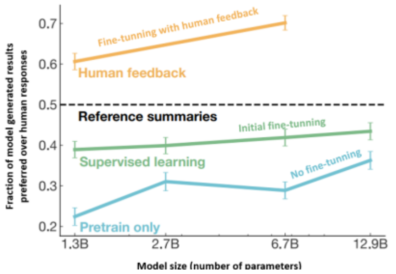
Generative AI Project Life Cycle - Reinforcement learning for human feedback

Adapt and align model
Align with human feedback

Use the reward model in the reinforcement learning process to update the LLM weights and produce a human aligned model.
✓ It's important you want to start with a model that already has a good performance on your task of interests.

Why RLHF

- In 2020, researchers at OpenAI published a paper that explored the use of fine-tuning with human feedback to train a model. The article demonstrates model fine-tuned on human feedback produced better responses than a pretrained model, an instruct fine-tuned model, and even the reference human baseline. (source: [OpenAI paper Learning to summarize from human feedback](#))
- A popular technique to finetune large language models with human feedback is called reinforcement learning from human feedback (RLHF).



Steps to obtain Feedback from humans for RLHF

1
- The first step in fine-tuning an LLM with RLHF is to select a model to work with and use it to prepare a data set for human feedback.
- In general, you may find it easier to start with an instruct model that has already been fine-tuned across many tasks and has some general capabilities.

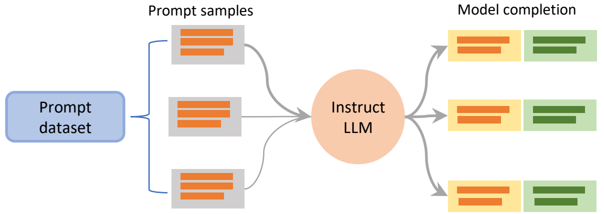
2
- Then use this LLM along with a prompt data set to generate a number of different responses for each prompt.
- The prompt dataset consists of multiple prompts, each of which gets processed by the LLM to produce a set of completions.

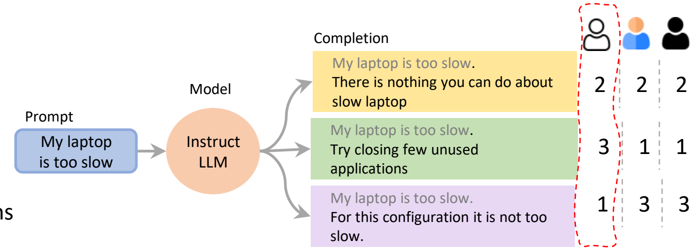
3
- The next step is to collect feedback from human labellers on the completions generated by the LLM. This is the human feedback portion of reinforcement.
- First, you must decide what criterion you want the humans to assess the completions on. This could be any of the issues like helpfulness or toxicity.

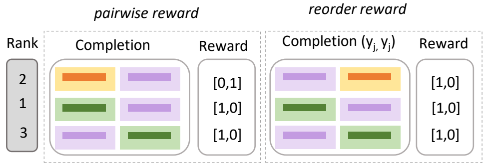
4
- The task for labellers in this example is to rank the three completions in order of helpfulness from the most helpful to least helpful.
- Once human labellers have completed their assessments of prompt completion sets, you have the data to train the reward model. Which will use instead of humans to classify model completions during the reinforcement learning finetuning process.
- In this e.g. first labeller responses disagree with the others and may indicate that they misunderstood the instructions.

5
- Before training the reward model, you need to convert the ranking data into a pairwise comparison of completions i.e. all possible pairs of completions should be classified as 0 or 1 score.
- For each pair, you will assign a reward of 1 for the preferred response and a reward of 0 for the less preferred response.

6
- Then you'll reorder the prompts so that the preferred option comes first. The reward model expects the preferred completion, which is referred to as **Y_j first.**







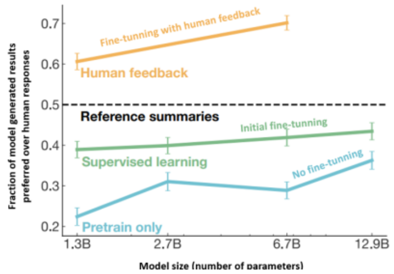
Generative AI Project Life Cycle - Reinforcement learning for human feedback

Adapt and align model
Align with human feedback

Use the reward model in the reinforcement learning process to update the LLM weights and produce a human aligned model.
✓ It's important you want to start with a model that already has a good performance on your task of interests.

Why RLHF

- In 2020, researchers at OpenAI published a paper that explored the use of fine-tuning with human feedback to train a model. The article demonstrates model fine-tuned on human feedback produced better responses than a pretrained model, an instruct fine-tuned model, and even the reference human baseline. (source: [OpenAI paper Learning to summarize from human feedback]())
- A popular technique to finetune large language models with human feedback is called reinforcement learning from human feedback (RLHF).



Steps to obtain Feedback from humans for RLHF

1
- The first step in fine-tuning an LLM with RLHF is to select a model to work with and use it to prepare a data set for human feedback.
- In general, you may find it easier to start with an instruct model that has already been fine-tuned across many tasks and has some general capabilities.

2
- Then use this LLM along with a prompt data set to generate a number of different responses for each prompt.
- The prompt dataset consists of multiple prompts, each of which gets processed by the LLM to produce a set of completions.

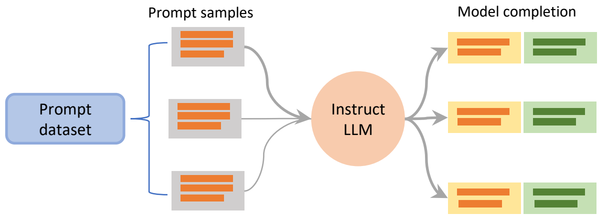
3
- The next step is to collect feedback from human labellers on the completions generated by the LLM. This is the human feedback portion of reinforcement.
- First, you must decide what criterion you want the humans to assess the completions on. This could be any of the issues like helpfulness or toxicity.

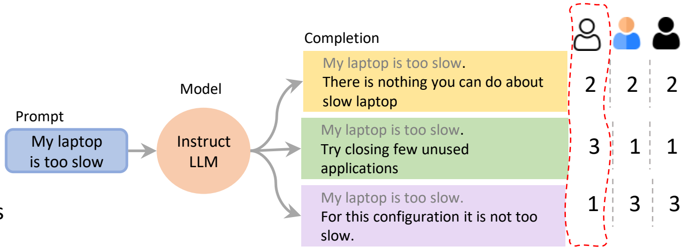
4
- The task for labellers in this example is to rank the three completions in order of helpfulness from the most helpful to least helpful.
- Once human labellers have completed their assessments of prompt completion sets, you have the data to train the reward model. Which will use instead of humans to classify model completions during the reinforcement learning finetuning process.
- In this e.g. first labeller responses disagree with the others and may indicate that they misunderstood the instructions.

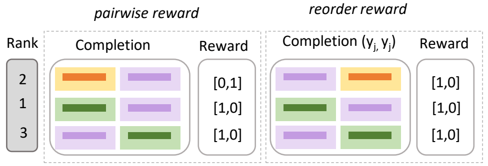
5
- Before training the reward model, you need to convert the ranking data into a pairwise comparison of completions i.e. all possible pairs of completions should be classified as 0 or 1 score.
- For each pair, you will assign a reward of 1 for the preferred response and a reward of 0 for the less preferred response.

6
- Then you'll reorder the prompts so that the preferred option comes first. The reward model expects the preferred completion, which is referred to as **Y_j first.**

7
- Once the model has been trained on the human rank prompt-completion pairs, you can use the reward model as a binary classifier to provide reward value for each prompt-completion pair *[e.g. +ve Logits(not hate) and -ve Logits(hate)]*.
- For a given prompt X, the reward model learns to favour the human-preferred completion **Y_j**,







[Kumar]()

Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

Overview of Fine-tunning with reinforcement learning for RLHF

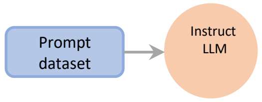
Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

Overview of Fine-tunning with reinforcement learning for RLHF

1 First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.



Kumar

Generative AI Project Life Cycle - Reinforcement learning for human feedback

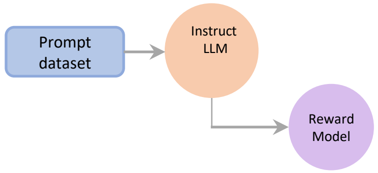
How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

Overview of Fine-tunning with reinforcement learning for RLHF

1 First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.

2 Next, you send this completion, and the original prompt to the reward model as the prompt completion pair to reward model.



Kumar

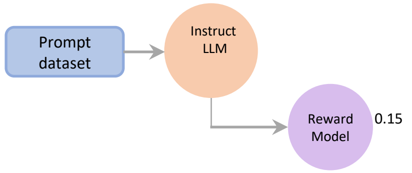
Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

Overview of Fine-tunning with reinforcement learning for RLHF

1. First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.
2. Next, you send this completion, and the original prompt to the reward model as the prompt completion pair to reward model.
3. The reward model evaluates the pair based on the human feedback it was trained on and returns a reward value. A higher reward value for e.g., 0.15 as shown here represents a more aligned response. A less aligned response would receive a lower value, such as negative -0.43 as an e.g.



Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

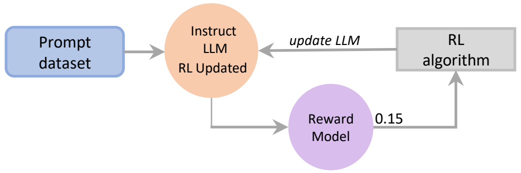
Overview of Fine-tunning with reinforcement learning for RLHF

1 First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.

2 Next, you send this completion, and the original prompt to the reward model as the prompt completion pair to reward model.

3 The reward model evaluates the pair based on the human feedback it was trained on and returns a reward value. A higher reward value for e.g., 0.15 as shown here represents a more aligned response. A less aligned response would receive a lower value, such as negative -0.43 as an e.g.

4
- Then pass this reward value for the prompt completion pair to the reinforcement learning algorithm to update the weights of the LLM, and move it towards generating more aligned, higher reward responses.
- The reinforcement learning (RL) algorithm updates weights of the model. The Instruct LLM now has updated weights.



Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

<u>Overview of Fine-tunning with reinforcement learning for RLHF</u>

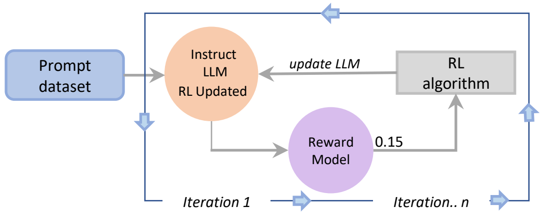
1 First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.

2 Next, you send this completion, and the original prompt to the reward model as the prompt completion pair to reward model.

3 The reward model evaluates the pair based on the human feedback it was trained on and returns a reward value. A higher reward value for e.g., 0.15 as shown here represents a more aligned response. A less aligned response would receive a lower value, such as negative -0.43 as an e.g.

4
- Then pass this reward value for the prompt completion pair to the reinforcement learning algorithm to update the weights of the LLM, and move it towards generating more aligned, higher reward responses.
- The reinforcement learning (RL) algorithm updates weights of the model. The Instruct LLM now has updated weights.

5 These iterations continue for a given number of epochs, similar to other types of fine tuning.



Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

Overview of Fine-tunning with reinforcement learning for RLHF

1 First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.

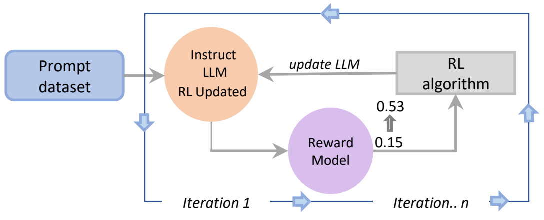
2 Next, you send this completion, and the original prompt to the reward model as the prompt completion pair to reward model.

3 The reward model evaluates the pair based on the human feedback it was trained on and returns a reward value. A higher reward value for e.g., 0.15 as shown here represents a more aligned response. A less aligned response would receive a lower value, such as negative -0.43 as an e.g.

4
- Then pass this reward value for the prompt completion pair to the reinforcement learning algorithm to update the weights of the LLM, and move it towards generating more aligned, higher reward responses.
- The reinforcement learning (RL) algorithm updates weights of the model. The Instruct LLM now has updated weights.

5 These iterations continue for a given number of epochs, similar to other types of fine tuning.

6 If the process is working well, you'll see the reward improving i.e., higher score after each iteration as the model produces text that is increasingly aligned with human preferences.



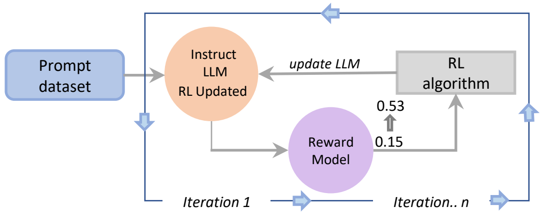
Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

Overview of Fine-tunning with reinforcement learning for RLHF

1. First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.
2. Next, you send this completion, and the original prompt to the reward model as the prompt completion pair to reward model.
3. The reward model evaluates the pair based on the human feedback it was trained on and returns a reward value. A higher reward value for e.g., 0.15 as shown here represents a more aligned response. A less aligned response would receive a lower value, such as negative -0.43 as an e.g.
4.
   - Then pass this reward value for the prompt completion pair to the reinforcement learning algorithm to update the weights of the LLM, and move it towards generating more aligned, higher reward responses.
   - The reinforcement learning (RL) algorithm updates weights of the model. The Instruct LLM now has updated weights.
5. These iterations continue for a given number of epochs, similar to other types of fine tuning.
6. If the process is working well, you'll see the reward improving i.e., higher score after each iteration as the model produces text that is increasingly aligned with human preferences.
7. You will continue this iterative process until your model is aligned based on some evaluation criteria. Once it satisfies evaluation metrics you can refer the fine-tuned model as **human aligned LLM**.



Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

Overview of Fine-tunning with reinforcement learning for RLHF

1 First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.

2 Next, you send this completion, and the original prompt to the reward model as the prompt completion pair to reward model.

3 The reward model evaluates the pair based on the human feedback it was trained on and returns a reward value. A higher reward value for e.g., 0.15 as shown here represents a more aligned response. A less aligned response would receive a lower value, such as negative -0.43 as an e.g.

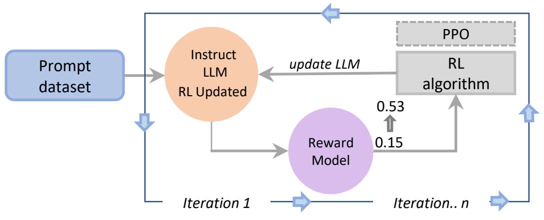
4
- Then pass this reward value for the prompt completion pair to the reinforcement learning algorithm to update the weights of the LLM, and move it towards generating more aligned, higher reward responses.
- The reinforcement learning (RL) algorithm updates weights of the model. The Instruct LLM now has updated weights.

5 These iterations continue for a given number of epochs, similar to other types of fine tuning.

6 If the process is working well, you'll see the reward improving i.e., higher score after each iteration as the model produces text that is increasingly aligned with human preferences.

7 You will continue this iterative process until your model is aligned based on some evaluation criteria. Once it satisfies evaluation metrics you can refer the fine-tuned model as **human aligned LLM**.

8 There are several algorithms that takes the output of the reward model and uses it to update the LLM model weights so that the reward score increases over time. A popular choice is **P**roximal **P**olicy **O**ptimization (**PPO**) .



Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.
✓ Start with a model that already has good performance on your task of interests.

Overview of Fine-tunning with reinforcement learning for RLHF

1 First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.

2 Next, you send this completion, and the original prompt to the reward model as the prompt completion pair to reward model.

3 The reward model evaluates the pair based on the human feedback it was trained on and returns a reward value. A higher reward value for e.g., 0.15 as shown here represents a more aligned response. A less aligned response would receive a lower value, such as negative -0.43 as an e.g.

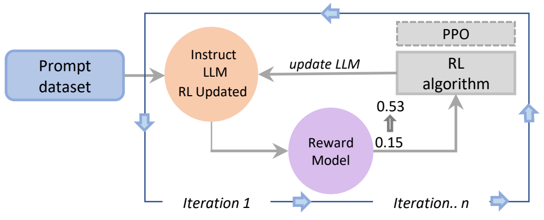
4
- Then pass this reward value for the prompt completion pair to the reinforcement learning algorithm to update the weights of the LLM, and move it towards generating more aligned, higher reward responses.
- The reinforcement learning (RL) algorithm updates weights of the model. The Instruct LLM now has updated weights.

5 These iterations continue for a given number of epochs, similar to other types of fine tuning.

6 If the process is working well, you'll see the reward improving i.e., higher score after each iteration as the model produces text that is increasingly aligned with human preferences.

7 You will continue this iterative process until your model is aligned based on some evaluation criteria. Once it satisfies evaluation metrics you can refer the fine-tuned model as **human aligned LLM**.

8 There are several algorithms that takes the output of the reward model and uses it to update the LLM model weights so that the reward score increases over time. A popular choice is **P**roximal **P**olicy **O**ptimization (**PPO**) .



Reward Hacking and How to avoid reward hacking

Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

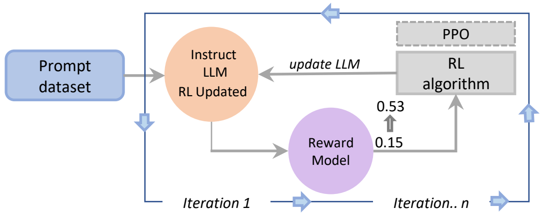
Overview of Fine-tunning with reinforcement learning for RLHF

1 First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.

2 Next, you send this completion, and the original prompt to the reward model as the prompt completion pair to reward model.

3 The reward model evaluates the pair based on the human feedback it was trained on and returns a reward value. A higher reward value for e.g., 0.15 as shown here represents a more aligned response. A less aligned response would receive a lower value, such as negative -0.43 as an e.g.

4
- Then pass this reward value for the prompt completion pair to the reinforcement learning algorithm to update the weights of the LLM, and move it towards generating more aligned, higher reward responses.
- The reinforcement learning (RL) algorithm updates weights of the model. The Instruct LLM now has updated weights.



5 These iterations continue for a given number of epochs, similar to other types of fine tuning.

6 If the process is working well, you'll see the reward improving i.e., higher score after each iteration as the model produces text that is increasingly aligned with human preferences.

7 You will continue this iterative process until your model is aligned based on some evaluation criteria. Once it satisfies evaluation metrics you can refer the fine-tuned model as **human aligned LLM**.

8 There are several algorithms that takes the output of the reward model and uses it to update the LLM model weights so that the reward score increases over time. A popular choice is **P**roximal **P**olicy **O**ptimization (**PPO**) .

Reward Hacking and How to avoid reward hacking

9 However, in reinforcement learning where the algorithm can potential learn to cheat the system by favouring actions that maximize the reward received even if those actions don't align well with the original objective.
This problem is called **Reward Hacking**. For example, the model can start generating low toxicity scores by including phrases like “*most awesome*”, “incredible” or “reliable” these phrase may sound exaggerated for a specific task.

Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

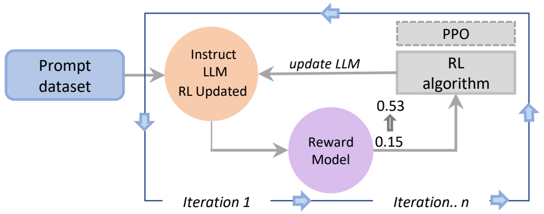
<u>Overview of Fine-tunning with reinforcement learning for RLHF</u>

1 First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.

2 Next, you send this completion, and the original prompt to the reward model as the prompt completion pair to reward model.

3 The reward model evaluates the pair based on the human feedback it was trained on and returns a reward value. A higher reward value for e.g., 0.15 as shown here represents a more aligned response. A less aligned response would receive a lower value, such as negative -0.43 as an e.g.

4
- Then pass this reward value for the prompt completion pair to the reinforcement learning algorithm to update the weights of the LLM, and move it towards generating more aligned, higher reward responses.
- The reinforcement learning (RL) algorithm updates weights of the model. The Instruct LLM now has updated weights.



5 These iterations continue for a given number of epochs, similar to other types of fine tuning.

6 If the process is working well, you'll see the reward improving i.e., higher score after each iteration as the model produces text that is increasingly aligned with human preferences.

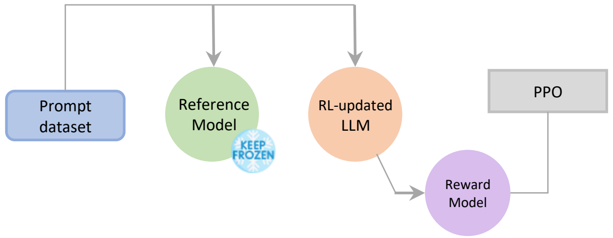
7 You will continue this iterative process until your model is aligned based on some evaluation criteria. Once it satisfies evaluation metrics you can refer the fine-tuned model as **human aligned LLM**.

8 There are several algorithms that takes the output of the reward model and uses it to update the LLM model weights so that the reward score increases over time. A popular choice is **P**roximal **P**olicy **O**ptimization (**PPO**) .

<u>Reward Hacking and How to avoid reward hacking</u>

9 However, in reinforcement learning where the algorithm can potential learn to cheat the system by favouring actions that maximize the reward received even if those actions don't align well with the original objective.
This problem is called **Reward Hacking**. For example, the model can start generating low toxicity scores by including phrases like "*most awesome*", "incredible" or "reliable" these phrase may sound exaggerated for a specific task.

10
- To prevent reward hacking from happening, you can use the initial instruct LLM as performance reference (reference model). The weights of the reference model are frozen and are not updated during iterations of reinforcement learning (RL).
- This way, you always maintain a single reference model to compare to during training, each prompt is passed to both models, generating a completion by the reference LLM and the intermediate LLM model is updated.



Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

Overview of Fine-tunning with reinforcement learning for RLHF

1 First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.

2 Next, you send this completion, and the original prompt to the reward model as the prompt completion pair to reward model.

3 The reward model evaluates the pair based on the human feedback it was trained on and returns a reward value. A higher reward value for e.g., 0.15 as shown here represents a more aligned response. A less aligned response would receive a lower value, such as negative -0.43 as an e.g.

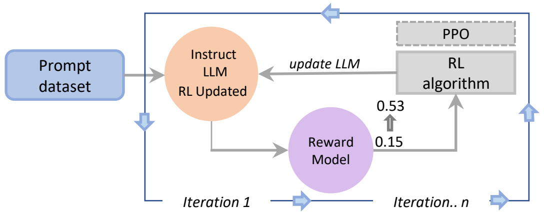
4
- Then pass this reward value for the prompt completion pair to the reinforcement learning algorithm to update the weights of the LLM, and move it towards generating more aligned, higher reward responses.
- The reinforcement learning (RL) algorithm updates weights of the model. The Instruct LLM now has updated weights.

5 These iterations continue for a given number of epochs, similar to other types of fine tuning.

6 If the process is working well, you'll see the reward improving i.e., higher score after each iteration as the model produces text that is increasingly aligned with human preferences.

7 You will continue this iterative process until your model is aligned based on some evaluation criteria. Once it satisfies evaluation metrics you can refer the fine-tuned model as **human aligned LLM**.

8 There are several algorithms that takes the output of the reward model and uses it to update the LLM model weights so that the reward score increases over time. A popular choice is **P**roximal **P**olicy **O**ptimization (**PPO**) .

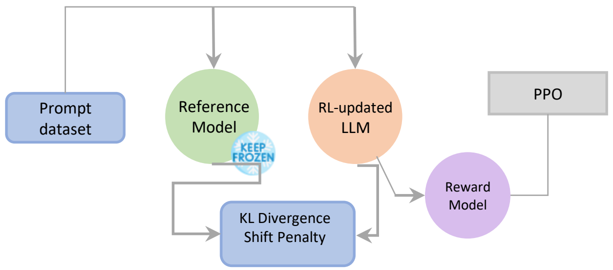


Reward Hacking and How to avoid reward hacking

9 However, in reinforcement learning where the algorithm can potential learn to cheat the system by favouring actions that maximize the reward received even if those actions don't align well with the original objective.
This problem is called **Reward Hacking**. For example, the model can start generating low toxicity scores by including phrases like "*most awesome*", "incredible" or "reliable" these phrase may sound exaggerated for a specific task.

10
- To prevent reward hacking from happening, you can use the initial instruct LLM as performance reference (reference model). The weights of the reference model are frozen and are not updated during iterations of reinforcement learning (RL).
- This way, you always maintain a single reference model to compare to during training, each prompt is passed to both models, generating a completion by the reference LLM and the intermediate LLM model is updated.

11
- At this point, you can compare the two completions and calculate a value called the Kullback-Leibler divergence (**KL divergence**)



Generative AI Project Life Cycle - Reinforcement learning for human feedback

How the reward model gets used in the reinforcement learning process to train your human aligned LLM. How does the reward model in the reinforcement learning process updates the LLM weights and produce a human aligned model.

✓ Start with a model that already has good performance on your task of interests.

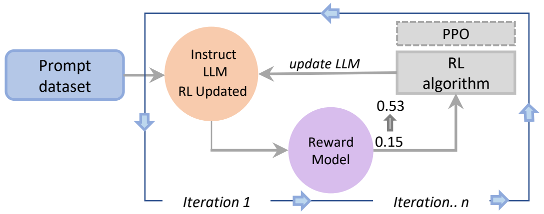
Overview of Fine-tunning with reinforcement learning for RLHF

1 First you pass a prompt from your prompt dataset to the instruct LLM, which then generates a completion.

2 Next, you send this completion, and the original prompt to the reward model as the prompt completion pair to reward model.

3 The reward model evaluates the pair based on the human feedback it was trained on and returns a reward value. A higher reward value for e.g., 0.15 as shown here represents a more aligned response. A less aligned response would receive a lower value, such as negative -0.43 as an e.g.

4
- Then pass this reward value for the prompt completion pair to the reinforcement learning algorithm to update the weights of the LLM, and move it towards generating more aligned, higher reward responses.
- The reinforcement learning (RL) algorithm updates weights of the model. The Instruct LLM now has updated weights.



5 These iterations continue for a given number of epochs, similar to other types of fine tuning.

6 If the process is working well, you'll see the reward improving i.e., higher score after each iteration as the model produces text that is increasingly aligned with human preferences.

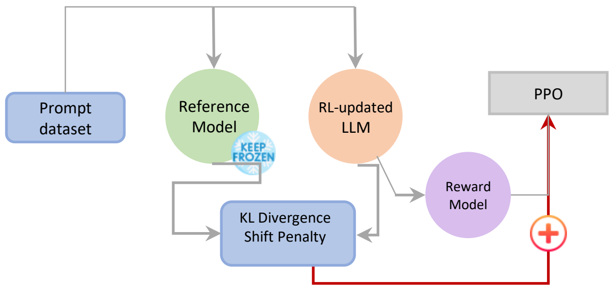
7 You will continue this iterative process until your model is aligned based on some evaluation criteria. Once it satisfies evaluation metrics you can refer the fine-tuned model as **human aligned LLM**.

8 There are several algorithms that takes the output of the reward model and uses it to update the LLM model weights so that the reward score increases over time. A popular choice is **P**roximal **P**olicy **O**ptimization (**PPO**) .

Reward Hacking and How to avoid reward hacking

9 However, in reinforcement learning where the algorithm can potential learn to cheat the system by favouring actions that maximize the reward received even if those actions don't align well with the original objective.
This problem is called **Reward Hacking**. For example, the model can start generating low toxicity scores by including phrases like "*most awesome*", "incredible" or "reliable" these phrase may sound exaggerated for a specific task.

10
- To prevent reward hacking from happening, you can use the initial instruct LLM as performance reference (reference model). The weights of the reference model are frozen and are not updated during iterations of reinforcement learning (RL).
- This way, you always maintain a single reference model to compare to during training, each prompt is passed to both models, generating a completion by the reference LLM and the intermediate LLM model is updated.



11
- At this point, you can compare the two completions and calculate a value called the Kullback-Leibler divergence (**KL divergence**)

12
- Once KL divergence is calculated between the two models, you can added them to the reward calculation.
- This will penalize the RL updated model if it shifts too far from the reference LLM and generates completions that are two different.

Kumar

Generative AI Project Life Cycle - Optimize and deploy model for inference

The things that we have to consider to integrate the LLM model into applications.

- How your LLM will function in deployment ?
- How fast do you need your model to generate completions?
- What compute budget do you have available?
- Are you willing to trade off model performance for improved inference speed or lower storage?

Generative AI Project Life Cycle - Optimize and deploy model for inference

The things that we have to consider to integrate the LLM model into applications.

- How your LLM will function in deployment ?
- How fast do you need your model to generate completions?
- What compute budget do you have available?
- Are you willing to trade off model performance for improved inference speed or lower storage?

Overview of Optimisation techniques for LLM

Generative AI Project Life Cycle - Optimize and deploy model for inference

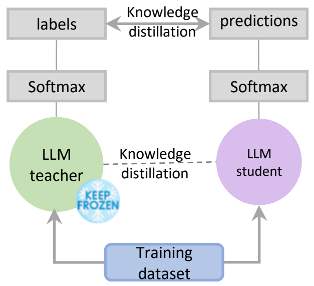
The things that we have to consider to integrate the LLM model into applications.

- How your LLM will function in deployment ?
- How fast do you need your model to generate completions?
- What compute budget do you have available?
- Are you willing to trade off model performance for improved inference speed or lower storage?

Overview of Optimisation techniques for LLM

Techniques	Details
Distillation	Model Distillation is a technique that focuses on using a larger (teacher) model to train a smaller model (*student*). You then use the smaller model for inference to lower your storage and compute budget. • You freeze the teacher model's weights • and use it to generate completions for your training data • At the same time, you generate completions for the training data using your student model • The knowledge distillation between teacher and student model is achieved by minimizing a loss function called the distillation loss

Distillation



Generative AI Project Life Cycle - Optimize and deploy model for inference

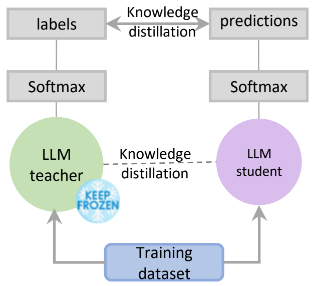
The things that we have to consider to integrate the LLM model into applications.

- How your LLM will function in deployment ?
- How fast do you need your model to generate completions?
- What compute budget do you have available?
- Are you willing to trade off model performance for improved inference speed or lower storage?

Overview of Optimisation techniques for LLM

Techniques	Details
Distillation	Model Distillation is a technique that focuses on using a larger (teacher) model to train a smaller model (*student*). You then use the smaller model for inference to lower your storage and compute budget. • You freeze the teacher model's weights • and use it to generate completions for your training data • At the same time, you generate completions for the training data using your student model • The knowledge distillation between teacher and student model is achieved by minimizing a loss function called the distillation loss
Quantisation	In this technique after model is trained you can perform **P**ost-**T**raining **Q**uantization (**PTQ**) for deployment. • PTQ transforms a model's weights to a lower precision representation, such as 16-bit floating point or 8-bit integer. This reduces the model size and memory footprint, as well as the compute resources needed for model serving • There are trade-offs because sometimes quantization results in a small percentage reduction in model evaluation metrics. However, that reduction can often be worth the cost savings and performance gains.

Distillation



Quantisation

	Bits	Exponents	Fraction	Memory (to store 1 value)	Store value of PI (example)
FP32	32	8	23	4 bytes	3.1415920257568359375
FP16	16	5	10	2 bytes	3.140625
BFLOAT16	16	8	7	2 bytes	3.140625
INT8	8	-/-	7	1 byte	

Kumar

Generative AI Project Life Cycle - Optimize and deploy model for inference

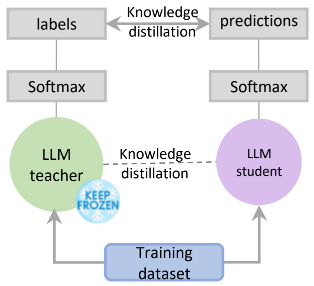
The things that we have to consider to integrate the LLM model into applications.

- How your LLM will function in deployment ?
- How fast do you need your model to generate completions?
- What compute budget do you have available?
- Are you willing to trade off model performance for improved inference speed or lower storage?

Overview of Optimisation techniques for LLM

Techniques	Details
Distillation	Model Distillation is a technique that focuses on using a larger (teacher) model to train a smaller model (*student*). You then use the smaller model for inference to lower your storage and compute budget. • You freeze the teacher model's weights • and use it to generate completions for your training data • At the same time, you generate completions for the training data using your student model • The knowledge distillation between teacher and student model is achieved by minimizing a loss function called the distillation loss
Quantisation	In this technique after model is trained you can perform **P**ost-**T**raining **Q**uantization (**PTQ**) for deployment. • PTQ transforms a model's weights to a lower precision representation, such as 16-bit floating point or 8-bit integer. This reduces the model size and memory footprint, as well as the compute resources needed for model serving • There are trade-offs because sometimes quantization results in a small percentage reduction in model evaluation metrics. However, that reduction can often be worth the cost savings and performance gains.
Model Pruning	Model Pruning, removes redundant model parameters that contribute little to the model's performance i.e. removes model weights with values close or equal to zero • The goal of model pruning is to reduce model size for inference by eliminating weights that are not contributing much to overall model performance. Pruning methods ○ Full model re-training ○ Parameter Efficient Fine tuning (PEFT) such as LoRA ○ Post-training ▪ In theory pruning reduces model size and improves performance ▪ In practice, only small percentage (%) in LLM are zero-weights

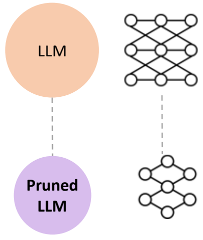
Distillation



Quantisation

	Bits	Exponents	Fraction	Memory (to store 1 value)	Store value of PI (example)
FP32	32	8	23	4 bytes	3.1415920257568359375
FP16	16	5	10	2 bytes	3.140625
BFLOAT16	16	8	7	2 bytes	3.140625
INT8	8	-/-	7	1 byte	

Model Pruning



Kumar

Generative AI Project Lifecycle – Planning Cheat sheet

To plan out stages of the generative AI project life cycle, below cheat sheet provides high-level indication of the time and effort required for each phase.

Generative AI Project Lifecycle – Planning Cheat sheet

To plan out stages of the generative AI project life cycle, below cheat sheet provides high-level indication of the time and effort required for each phase.

	Pre-training
Training duration	Days to Weeks to Months
Customization	Determine model architecture, size and tokenizer. Choose vocabulary size and number of tokens for input/context. Large amount of domain training data.
Objective	Next-token prediction
Expertise	High

Generative AI Project Lifecycle – Planning Cheat sheet

To plan out stages of the generative AI project life cycle, below cheat sheet provides high-level indication of the time and effort required for each phase.

	Pre-training	**Prompt engineering**
Training duration	Days to Weeks to Months	Not required
Customization	Determine model architecture, size and tokenizer. Choose vocabulary size and number of tokens for input/context. Large amount of domain training data.	No need to tune model weights. Only prompt customisation
Objective	Next-token prediction	To increase task performance
Expertise	High	Low

Generative AI Project Lifecycle – Planning Cheat sheet

To plan out stages of the generative AI project life cycle, below cheat sheet provides high-level indication of the time and effort required for each phase.

	Pre-training	**Prompt engineering**	**Prompt-tuning and Fine-tuning**
Training duration	Days to Weeks to Months	Not required	Hours to Days
Customization	Determine model architecture, size and tokenizer. Choose vocabulary size and number of tokens for input/context. Large amount of domain training data.	No need to tune model weights. Only prompt customisation	Tune for specific tasks. Add domain-specific data Update LLM model OR adapter weights.
Objective	Next-token prediction	To increase task performance	To increase task performance
Expertise	High	Low	Low

Generative AI Project Lifecycle – Planning Cheat sheet

To plan out stages of the generative AI project life cycle, below cheat sheet provides high-level indication of the time and effort required for each phase.

	Pre-training	Prompt engineering	Prompt-tuning and Fine-tuning	Reinforcement learning/Human feedback
Training duration	Days to Weeks to Months	Not required	Hours to Days	Hours to Days
Customization	Determine model architecture, size and tokenizer. Choose vocabulary size and number of tokens for input/context. Large amount of domain training data.	No need to tune model weights. Only prompt customisation	Tune for specific tasks. Add domain-specific data Update LLM model OR adapter weights.	Need a separated reward model to align with human goals (helpful, honest & harmless). Update LLM model OR adapter weights.
Objective	Next-token prediction	To increase task performance	To increase task performance	Increase alignment with human preferences.
Expertise	High	Low	Low	Medium to High

Generative AI Project Lifecycle – Planning Cheat sheet

To plan out stages of the generative AI project life cycle, below cheat sheet provides high-level indication of the time and effort required for each phase.

	Pre-training	**Prompt engineering**	**Prompt-tuning and Fine-tuning**	**Reinforcement learning/Human feedback**	**Optimization deployment**
Training duration	Days to Weeks to Months	Not required	Hours to Days	Hours to Days	Hours to Days
Customization	Determine model architecture, size and tokenizer. Choose vocabulary size and number of tokens for input/context. Large amount of domain training data.	No need to tune model weights. Only prompt customisation	Tune for specific tasks. Add domain-specific data Update LLM model OR adapter weights.	Need a separated reward model to align with human goals (helpful, honest & harmless). Update LLM model OR adapter weights.	Reduce model size through model pruning, weight quantization, distillation. Smaller size, faster inference
Objective	Next-token prediction	To increase task performance	To increase task performance	Increase alignment with human preferences.	Increase inference performance
Expertise	High	Low	Low	Medium to High	Medium

Generative AI Project Lifecycle – LLM powered application

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don't know the answer to a problem. This is often called as hallucination.

There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Generative AI Project Lifecycle – LLM powered application

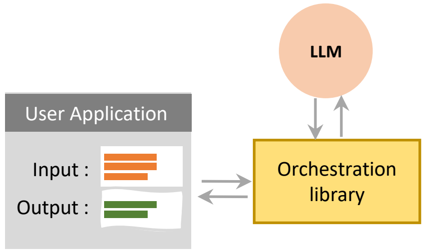
Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don’t know the answer to a problem. This is often called as hallucination.

There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Orchestration Library

LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library



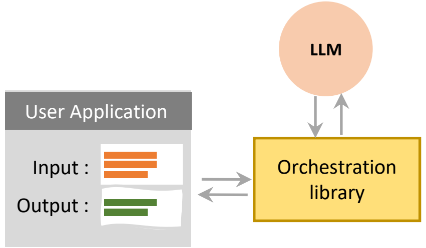
Generative AI Project Lifecycle – LLM powered application

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don't know the answer to a problem. This is often called as hallucination.

There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Orchestration Library	LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library
LangChain	Is a framework that enables technologies to augment and enhance the performance of LLM at runtime by providing access to external data sources or connection to existing APIs of other application.



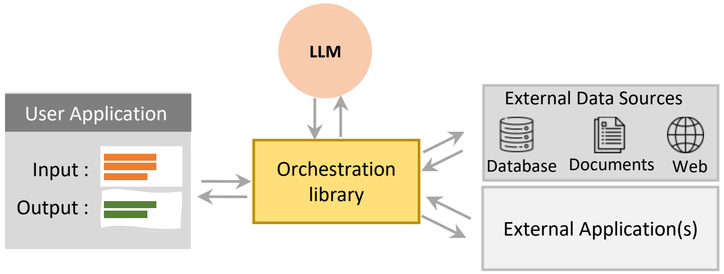
Generative AI Project Lifecycle – LLM powered application

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don't know the answer to a problem. This is often called as hallucination.

There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Orchestration Library	LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library
LangChain	Is a framework that enables technologies to augment and enhance the performance of LLM at runtime by providing access to external data sources or connection to existing APIs of other application.
RAG	Retrieval Augmented Generation(**RAG**) is a framework for building LLM powered systems that make use of external data sources to overcome some of the limitations of these models. • RAG is a great way to overcome the knowledge cut-off issue and help the model update its understanding of the world. • RAG is useful in a case where you want the language model to have access to data that it may not have seen.



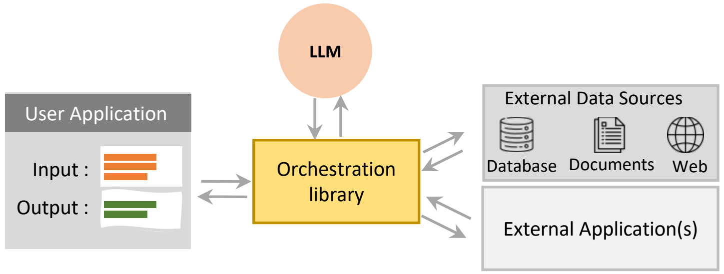
Generative AI Project Lifecycle – LLM powered application

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don’t know the answer to a problem. This is often called as hallucination.

There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Orchestration Library	LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library
LangChain	Is a framework that enables technologies to augment and enhance the performance of LLM at runtime by providing access to external data sources or connection to existing APIs of other application.
RAG	Retrieval Augmented Generation(**RAG**) is a framework for building LLM powered systems that make use of external data sources to overcome some of the limitations of these models. • RAG is a great way to overcome the knowledge cut-off issue and help the model update its understanding of the world. • RAG is useful in a case where you want the language model to have access to data that it may not have seen.



<u>Overview of LLM Powered Application using RAG</u>

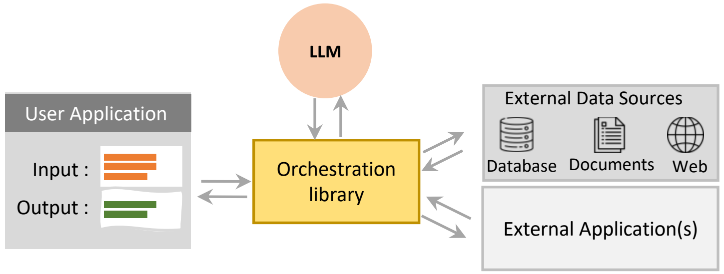
Generative AI Project Lifecycle – LLM powered application

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don't know the answer to a problem. This is often called as hallucination.

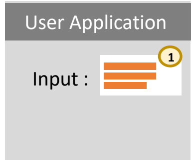
There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Orchestration Library	LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library
LangChain	Is a framework that enables technologies to augment and enhance the performance of LLM at runtime by providing access to external data sources or connection to existing APIs of other application.
RAG	Retrieval Augmented Generation(**RAG**) is a framework for building LLM powered systems that make use of external data sources to overcome some of the limitations of these models. • RAG is a great way to overcome the knowledge cut-off issue and help the model update its understanding of the world. • RAG is useful in a case where you want the language model to have access to data that it may not have seen.



Overview of LLM Powered Application using RAG

1 Let's consider an example of "Claim Assistant Application" which is made available to all members via self-service secured portal. The user wants to know about recent knee surgery and its details requests via LLM application, input is : "I need to know details about my recent claim"



Generative AI Project Lifecycle – LLM powered application

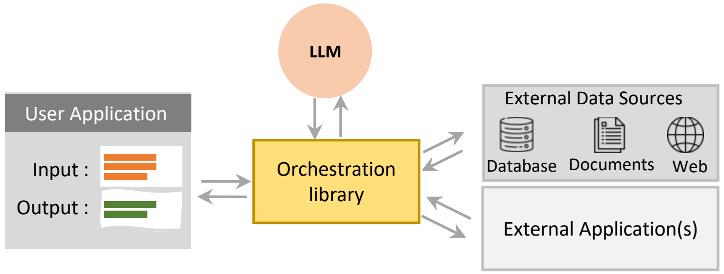
Application Integration
LLM powered applications

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don't know the answer to a problem. This is often called as hallucination.

There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

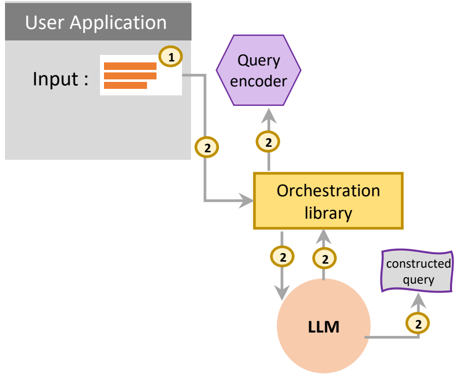
Orchestration Library	LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library
LangChain	Is a framework that enables technologies to augment and enhance the performance of LLM at runtime by providing access to external data sources or connection to existing APIs of other application.
RAG	Retrieval Augmented Generation(**RAG**) is a framework for building LLM powered systems that make use of external data sources to overcome some of the limitations of these models. • RAG is a great way to overcome the knowledge cut-off issue and help the model update its understanding of the world. • RAG is useful in a case where you want the language model to have access to data that it may not have seen.



Overview of LLM Powered Application using RAG

1 Let's consider an example of "Claim Assistant Application" which is made available to all members via self-service secured portal. The user wants to know about recent knee surgery and its details requests via LLM application, input is : "I need to know details about my recent claim"

2 The prompt is passed to query encoders, which encodes it into a form that can be used to query the data source. The external data could be SQL DB or vector store or other data storage format.



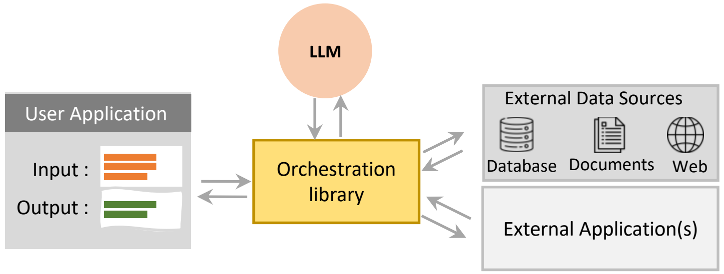
Generative AI Project Lifecycle – LLM powered application

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don't know the answer to a problem. This is often called as hallucination.

There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Orchestration Library	LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library
LangChain	Is a framework that enables technologies to augment and enhance the performance of LLM at runtime by providing access to external data sources or connection to existing APIs of other application.
RAG	Retrieval Augmented Generation(**RAG**) is a framework for building LLM powered systems that make use of external data sources to overcome some of the limitations of these models. • RAG is a great way to overcome the knowledge cut-off issue and help the model update its understanding of the world. • RAG is useful in a case where you want the language model to have access to data that it may not have seen.

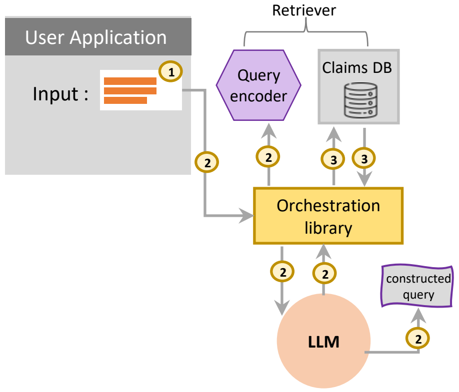


Overview of LLM Powered Application using RAG

1 Let's consider an example of "Claim Assistant Application" which is made available to all members via self-service secured portal. The user wants to know about recent knee surgery and its details requests via LLM application, input is : "I need to know details about my recent claim"

2 The prompt is passed to query encoders, which encodes it into a form that can be used to query the data source. The external data could be SQL DB or vector store or other data storage format.

3 The constructed query is executed against the data store to search for claims for a member-ID. The Retriever returns the best single or group of records from the data source and combines the new information with the original user prompt.



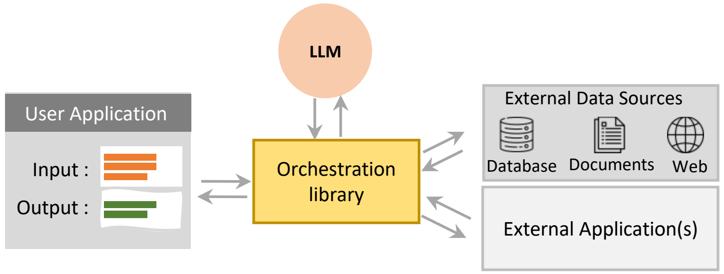
Generative AI Project Lifecycle – LLM powered application

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don't know the answer to a problem. This is often called as hallucination.

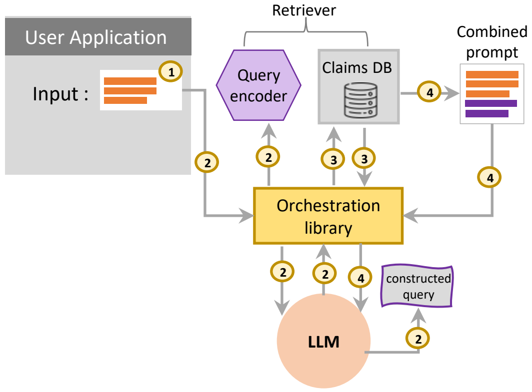
There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Orchestration Library	LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library
LangChain	Is a framework that enables technologies to augment and enhance the performance of LLM at runtime by providing access to external data sources or connection to existing APIs of other application.
RAG	Retrieval Augmented Generation(**RAG**) is a framework for building LLM powered systems that make use of external data sources to overcome some of the limitations of these models. • RAG is a great way to overcome the knowledge cut-off issue and help the model update its understanding of the world. • RAG is useful in a case where you want the language model to have access to data that it may not have seen.



Overview of LLM Powered Application using RAG

1. Let's consider an example of "Claim Assistant Application" which is made available to all members via self-service secured portal. The user wants to know about recent knee surgery and its details requests via LLM application, input is : "I need to know details about my recent claim"
2. The prompt is passed to query encoders, which encodes it into a form that can be used to query the data source. The external data could be SQL DB or vector store or other data storage format.
3. The constructed query is executed against the data store to search for claims for a member-ID. The Retriever returns the best single or group of records from the data source and combines the new information with the original user prompt.
4. The new expanded prompt that now contains information about the specific claim of interest is than passed to the LLM, which generates a completion that makes use of the combined data.



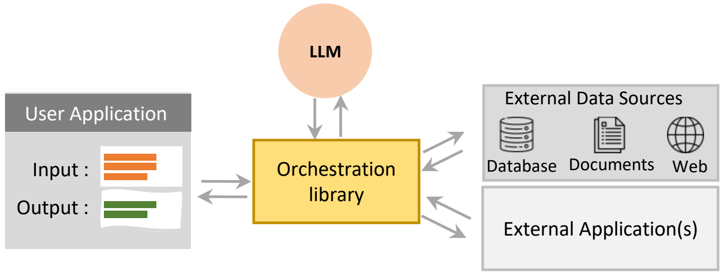
Generative AI Project Lifecycle – LLM powered application

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don't know the answer to a problem. This is often called as hallucination.

There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Orchestration Library	LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library
LangChain	Is a framework that enables technologies to augment and enhance the performance of LLM at runtime by providing access to external data sources or connection to existing APIs of other application.
RAG	Retrieval Augmented Generation(**RAG**) is a framework for building LLM powered systems that make use of external data sources to overcome some of the limitations of these models. • RAG is a great way to overcome the knowledge cut-off issue and help the model update its understanding of the world. • RAG is useful in a case where you want the language model to have access to data that it may not have seen.



Overview of LLM Powered Application using RAG

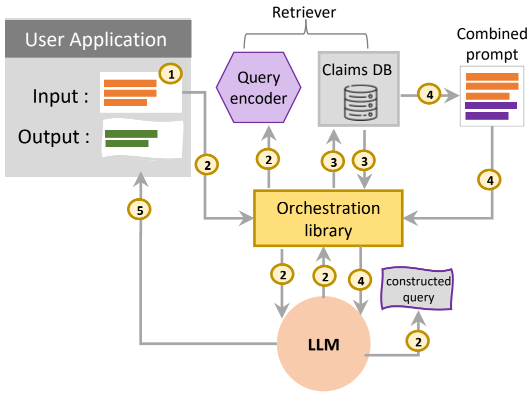
1 Let's consider an example of "Claim Assistant Application" which is made available to all members via self-service secured portal. The user wants to know about recent knee surgery and its details requests via LLM application, input is : "I need to know details about my recent claim"

2 The prompt is passed to query encoders, which encodes it into a form that can be used to query the data source. The external data could be SQL DB or vector store or other data storage format.

3 The constructed query is executed against the data store to search for claims for a member-ID. The Retriever returns the best single or group of records from the data source and combines the new information with the original user prompt.

4 The new expanded prompt that now contains information about the specific claim of interest is than passed to the LLM, which generates a completion that makes use of the combined data.

5 The model uses the information in the context of the prompt to generate a completion that contains the more accurate details for specific type of claim in this e.g. for knee surgery.



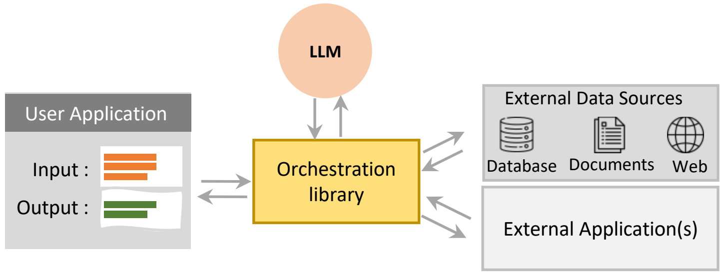
Generative AI Project Lifecycle – LLM powered application

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don't know the answer to a problem. This is often called as hallucination.

There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Orchestration Library	LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library
LangChain	Is a framework that enables technologies to augment and enhance the performance of LLM at runtime by providing access to external data sources or connection to existing APIs of other application.
RAG	Retrieval Augmented Generation(**RAG**) is a framework for building LLM powered systems that make use of external data sources to overcome some of the limitations of these models. • RAG is a great way to overcome the knowledge cut-off issue and help the model update its understanding of the world. • RAG is useful in a case where you want the language model to have access to data that it may not have seen.



Overview of LLM Powered Application using RAG

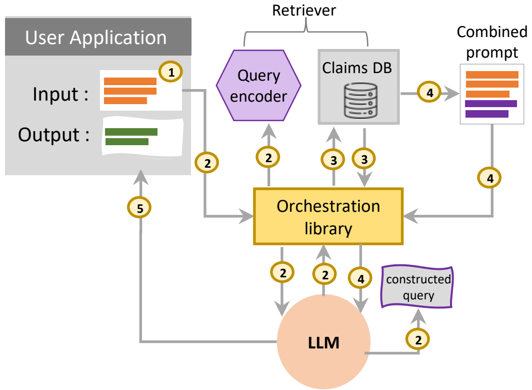
1 Let's consider an example of "Claim Assistant Application" which is made available to all members via self-service secured portal. The user wants to know about recent knee surgery and its details requests via LLM application, input is : "I need to know details about my recent claim"

2 The prompt is passed to query encoders, which encodes it into a form that can be used to query the data source. The external data could be SQL DB or vector store or other data storage format.

3 The constructed query is executed against the data store to search for claims for a member-ID. The Retriever returns the best single or group of records from the data source and combines the new information with the original user prompt.

4 The new expanded prompt that now contains information about the specific claim of interest is than passed to the LLM, which generates a completion that makes use of the combined data.

5 The model uses the information in the context of the prompt to generate a completion that contains the more accurate details for specific type of claim in this e.g. for knee surgery.



Data preparation for RAG

Kumar

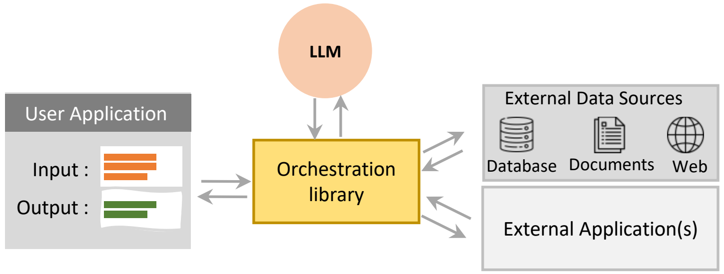
Generative AI Project Lifecycle – LLM powered application

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don't know the answer to a problem. This is often called as hallucination.

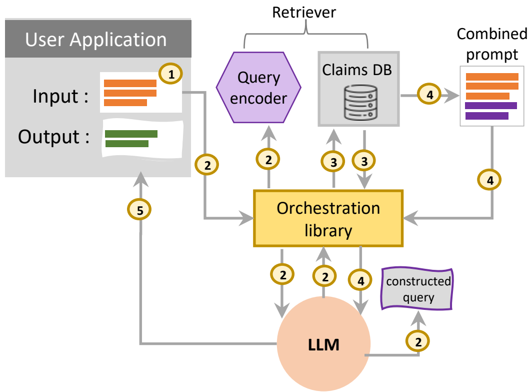
There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Orchestration Library	LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library
LangChain	Is a framework that enables technologies to augment and enhance the performance of LLM at runtime by providing access to external data sources or connection to existing APIs of other application.
RAG	Retrieval Augmented Generation(**RAG**) is a framework for building LLM powered systems that make use of external data sources to overcome some of the limitations of these models. • RAG is a great way to overcome the knowledge cut-off issue and help the model update its understanding of the world. • RAG is useful in a case where you want the language model to have access to data that it may not have seen.



Overview of LLM Powered Application using RAG

1. Let's consider an example of "Claim Assistant Application" which is made available to all members via self-service secured portal. The user wants to know about recent knee surgery and its details requests via LLM application, input is : "I need to know details about my recent claim"
2. The prompt is passed to query encoders, which encodes it into a form that can be used to query the data source. The external data could be SQL DB or vector store or other data storage format.
3. The constructed query is executed against the data store to search for claims for a member-ID. The Retriever returns the best single or group of records from the data source and combines the new information with the original user prompt.
4. The new expanded prompt that now contains information about the specific claim of interest is than passed to the LLM, which generates a completion that makes use of the combined data.
5. The model uses the information in the context of the prompt to generate a completion that contains the more accurate details for specific type of claim in this e.g. for knee surgery.



Data preparation for RAG

Implementing RAG involves design process its not just simply adding text into LLM. The two key consideration for using external data in RAG are :

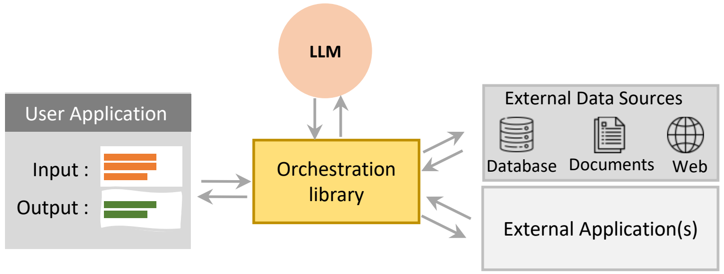
Generative AI Project Lifecycle – LLM powered application

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don't know the answer to a problem. This is often called as hallucination.

There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Orchestration Library	LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library
LangChain	Is a framework that enables technologies to augment and enhance the performance of LLM at runtime by providing access to external data sources or connection to existing APIs of other application.
RAG	Retrieval Augmented Generation(**RAG**) is a framework for building LLM powered systems that make use of external data sources to overcome some of the limitations of these models. • RAG is a great way to overcome the knowledge cut-off issue and help the model update its understanding of the world. • RAG is useful in a case where you want the language model to have access to data that it may not have seen.



Overview of LLM Powered Application using RAG

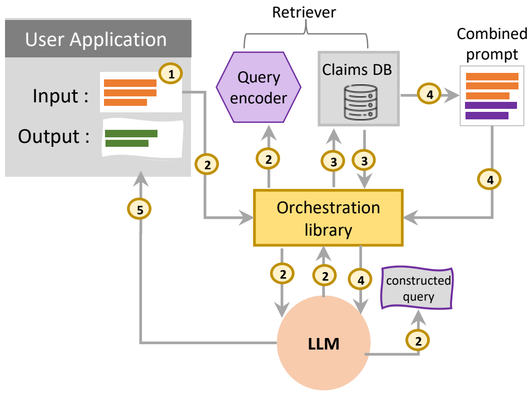
1 Let's consider an example of "Claim Assistant Application" which is made available to all members via self-service secured portal. The user wants to know about recent knee surgery and its details requests via LLM application, input is : "I need to know details about my recent claim"

2 The prompt is passed to query encoders, which encodes it into a form that can be used to query the data source. The external data could be SQL DB or vector store or other data storage format.

3 The constructed query is executed against the data store to search for claims for a member-ID. The Retriever returns the best single or group of records from the data source and combines the new information with the original user prompt.

4 The new expanded prompt that now contains information about the specific claim of interest is than passed to the LLM, which generates a completion that makes use of the combined data.

5 The model uses the information in the context of the prompt to generate a completion that contains the more accurate details for specific type of claim in this e.g. for knee surgery.



Data preparation for RAG

Implementing RAG involves design process its not just simply adding text into LLM. The two key consideration for using external data in RAG are :

1 The data must fit inside the context window.

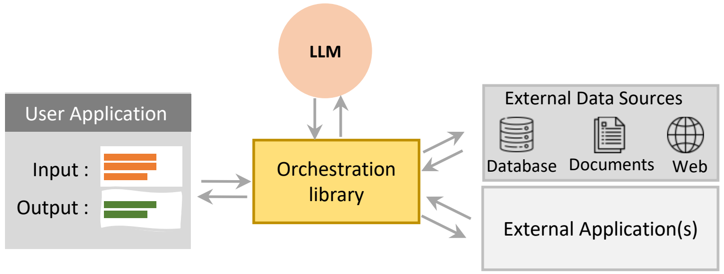
Generative AI Project Lifecycle – LLM powered application

Although all the training, tuning, and aligning techniques can help you build a great model for your application. But there are some broader challenges with large language models that can't be solved by training alone.

- One issue is that the internal knowledge held by a model cuts (knowledge cut-off) off at the moment of pretraining.
- Models may struggle with complex math.
- LLMs have tendency to generate output even when they don't know the answer to a problem. This is often called as hallucination.

There are some techniques that you can use to help your LLM overcome these issues by connecting to external data sources and applications.
This involves connecting your LLM to these external components and fully integrate everything for deployment within your application.

Orchestration Library	LLM application must manage the passing of user input to the large language model. This is often done through some type of orchestration library
LangChain	Is a framework that enables technologies to augment and enhance the performance of LLM at runtime by providing access to external data sources or connection to existing APIs of other application.
RAG	Retrieval Augmented Generation(**RAG**) is a framework for building LLM powered systems that make use of external data sources to overcome some of the limitations of these models. • RAG is a great way to overcome the knowledge cut-off issue and help the model update its understanding of the world. • RAG is useful in a case where you want the language model to have access to data that it may not have seen.



Overview of LLM Powered Application using RAG

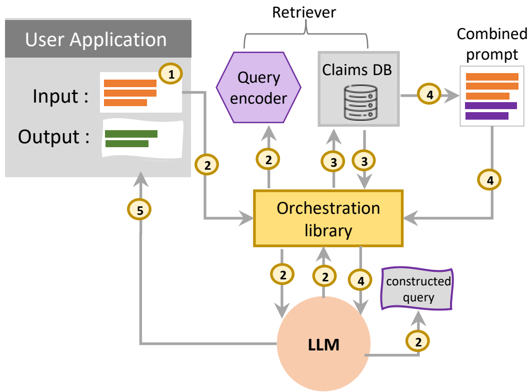
1 Let's consider an example of "Claim Assistant Application" which is made available to all members via self-service secured portal. The user wants to know about recent knee surgery and its details requests via LLM application, input is : "I need to know details about my recent claim"

2 The prompt is passed to query encoders, which encodes it into a form that can be used to query the data source. The external data could be SQL DB or vector store or other data storage format.

3 The constructed query is executed against the data store to search for claims for a member-ID. The Retriever returns the best single or group of records from the data source and combines the new information with the original user prompt.

4 The new expanded prompt that now contains information about the specific claim of interest is than passed to the LLM, which generates a completion that makes use of the combined data.

5 The model uses the information in the context of the prompt to generate a completion that contains the more accurate details for specific type of claim in this e.g. for knee surgery.



Data preparation for RAG

Implementing RAG involves design process its not just simply adding text into LLM. The two key consideration for using external data in RAG are :

1 The data must fit inside the context window.

2 Data must be in a format that allows its relevance to be assessed at inference time i.e. **Embedding vectors**.

Kumar

Generative AI Project Lifecycle – Interacting with external applications

In the previous section, we saw how LLMs can interact with external data sources. Now let's look at how LLM can interact with external applications.

- In general, connecting LLMs to external applications allows the model to interact with the broader world, extending their utility beyond language tasks.
- As the "Claim Assistant Application" example, LLMs can be used to trigger actions when given the ability to interact with APIs.
- LLMs can also connect to other programming resources. For example, a Python interpreter that can enable models to incorporate accurate calculations into their outputs.

Generative AI Project Lifecycle – Interacting with external applications

In the previous section, we saw how LLMs can interact with external data sources. Now let's look at how LLM can interact with external applications.

- In general, connecting LLMs to external applications allows the model to interact with the broader world, extending their utility beyond language tasks.
- As the "Claim Assistant Application" example, LLMs can be used to trigger actions when given the ability to interact with APIs.
- LLMs can also connect to other programming resources. For example, a Python interpreter that can enable models to incorporate accurate calculations into their outputs.

Let's extended with pervious example "Claim Assistant Application" where member wants to know the details about recent knee surgery claim and details needs to be emailed.

- It's important to note that prompt's structure and completion are curial for these workflows
- To trigger actions, the completions generated by the LLM must contain these important information plan actions, format output and validate actions.

Generative AI Project Lifecycle – Interacting with external applications

In the previous section, we saw how LLMs can interact with external data sources. Now let's look at how LLM can interact with external applications.

- In general, connecting LLMs to external applications allows the model to interact with the broader world, extending their utility beyond language tasks.
- As the “Claim Assistant Application” example, LLMs can be used to trigger actions when given the ability to interact with APIs.
- LLMs can also connect to other programming resources. For example, a Python interpreter that can enable models to incorporate accurate calculations into their outputs.

Let’s extended with pervious example “Claim Assistant Application” where member wants to know the details about recent knee surgery claim and details needs to be emailed.

- It's important to note that prompt's structure and completion are curial for these workflows
- To trigger actions, the completions generated by the LLM must contain these important information plan actions, format output and validate actions.

Plan actions

Plan actions : The model needs to be able to generate a set of instructions so that the application knows what actions to take.

- These instructions need to be understandable and correspond to allowed actions.
- In the Claim Assistant Application example for instance, the important steps were checking the claim, obtaining member-ID from portal, verifying user email, and emailing the claim details.

Plan actions
Steps to process return:
Step1: Check claim
Step2: Request member-ID
Step3: Verify member email
Step4: Email claim details

Generative AI Project Lifecycle – Interacting with external applications

In the previous section, we saw how LLMs can interact with external data sources. Now let's look at how LLM can interact with external applications.

- In general, connecting LLMs to external applications allows the model to interact with the broader world, extending their utility beyond language tasks.
- As the "Claim Assistant Application" example, LLMs can be used to trigger actions when given the ability to interact with APIs.
- LLMs can also connect to other programming resources. For example, a Python interpreter that can enable models to incorporate accurate calculations into their outputs.

Let's extended with pervious example "Claim Assistant Application" where member wants to know the details about recent knee surgery claim and details needs to be emailed.

- It's important to note that prompt's structure and completion are curial for these workflows
- To trigger actions, the completions generated by the LLM must contain these important information plan actions, format output and validate actions.

Plan actions

Plan actions : The model needs to be able to generate a set of instructions so that the application knows what actions to take.

- These instructions need to be understandable and correspond to allowed actions.
- In the Claim Assistant Application example for instance, the important steps were checking the claim, obtaining member-ID from portal, verifying user email, and emailing the claim details.

Format output

Format output: The completion needs to be formatted in a way that the broader application can understand. This could be as simple as a specific sentence structure or as complex as writing a script in Python or generating a SQL command.

- For example, here is a SQL query that would determine whether an claim is present in the claims database.

Plan actions
Steps to process return:
Step1: Check claim
Step2: Request member-ID
Step3: Verify member email
Step4: Email claim details

Format outputs
SQL:
SELECT co1, col2, FROM claim WHERE memberID = '*LLM2525*' AND claimDate <= (currDate – 90)

Generative AI Project Lifecycle – Interacting with external applications

In the previous section, we saw how LLMs can interact with external data sources. Now let's look at how LLM can interact with external applications.

- In general, connecting LLMs to external applications allows the model to interact with the broader world, extending their utility beyond language tasks.
- As the "Claim Assistant Application" example, LLMs can be used to trigger actions when given the ability to interact with APIs.
- LLMs can also connect to other programming resources. For example, a Python interpreter that can enable models to incorporate accurate calculations into their outputs.

Let's extended with pervious example "Claim Assistant Application" where member wants to know the details about recent knee surgery claim and details needs to be emailed.

- It's important to note that prompt's structure and completion are curial for these workflows
- To trigger actions, the completions generated by the LLM must contain these important information plan actions, format output and validate actions.

Plan actions

Plan actions : The model needs to be able to generate a set of instructions so that the application knows what actions to take.

- These instructions need to be understandable and correspond to allowed actions.
- In the Claim Assistant Application example for instance, the important steps were checking the claim, obtaining member-ID from portal, verifying user email, and emailing the claim details.

Format output

Format output: The completion needs to be formatted in a way that the broader application can understand. This could be as simple as a specific sentence structure or as complex as writing a script in Python or generating a SQL command.

- For example, here is a SQL query that would determine whether an claim is present in the claims database.

Validate actions

Lastly the model may need to collect information that allows it to validate an action.

- For example, in the Claim Assistant Application, the application needed to verify the email address of member. Any information that is required for validation needs to be obtained from the member/user and contained in the completion so it can be passed through to the application.

Plan actions	Format outputs	Validate actions
Steps to process return: **Step1:** Check claim **Step2:** Request member-ID **Step3:**Verify member email **Step4:** Email claim details	SQL: **SELECT co1, col2, FROM claim WHERE memberID = '*LLM2525*' AND claimDate <= (currDate – 90)**	Collect required member information and make sure its is in the completion Member email: **mailme@email.com**

Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don’t want LLM to predict most probable token instead we want accurate answer.

Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don’t want LLM to predict most probable token instead we want accurate answer.

- One strategy that has demonstrated some success is prompting the model to think more like a human is by breaking the problem down into steps.
  - And allowing the LLM to interact with external application that are good at math for e.g. Python interpreter, this helps to overcome the LLM limitation.
  - One framework for augmenting LLMs in this way is called **P**rogram-**A**ided **L**anguage (**PAL**) models.
  - The method makes use of chain of thought prompting to generate executable Python scripts. The scripts that the model generates are passed to an interpreter to execute.

Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don't want LLM to predict most probable token instead we want accurate answer.

- One strategy that has demonstrated some success is prompting the model to think more like a human is by breaking the problem down into steps.
  - And allowing the LLM to interact with external application that are good at math for e.g. Python interpreter, this helps to overcome the LLM limitation.
  - One framework for augmenting LLMs in this way is called **P**rogram-**A**ided **L**anguage (**PAL**) models.
  - The method makes use of chain of thought prompting to generate executable Python scripts. The scripts that the model generates are passed to an interpreter to execute.

Overview of PAL	The strategy behind PAL is to have the LLM generate completions where reasoning steps are accompanied by computer code. This code is then passed to an interpreter to carry out the calculations necessary to solve the problem.

Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don't want LLM to predict most probable token instead we want accurate answer.

- One strategy that has demonstrated some success is prompting the model to think more like a human is by breaking the problem down into steps.
  - And allowing the LLM to interact with external application that are good at math for e.g. Python interpreter, this helps to overcome the LLM limitation.
  - One framework for augmenting LLMs in this way is called **P**rogram-**A**ided **L**anguage (**PAL**) models.
  - The method makes use of chain of thought prompting to generate executable Python scripts. The scripts that the model generates are passed to an interpreter to execute.

Overview of PAL

The strategy behind PAL is to have the LLM generate completions where reasoning steps are accompanied by computer code.
This code is then passed to an interpreter to carry out the calculations necessary to solve the problem.

PAL LLM one-shot inference example

Input : Patrick already bowled 3 overs. He needs to bowl 2 more overs. In cricket each over consists of 6 balls to be bowled. How many balls has he bowled in total?

Generate

Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don't want LLM to predict most probable token instead we want accurate answer.

- One strategy that has demonstrated some success is prompting the model to think more like a human is by breaking the problem down into steps.
  - And allowing the LLM to interact with external application that are good at math for e.g. Python interpreter, this helps to overcome the LLM limitation.
  - One framework for augmenting LLMs in this way is called **P**rogram-**A**ided **L**anguage (**PAL**) models.
  - The method makes use of chain of thought prompting to generate executable Python scripts. The scripts that the model generates are passed to an interpreter to execute.

Overview of PAL

The strategy behind PAL is to have the LLM generate completions where reasoning steps are accompanied by computer code.
This code is then passed to an interpreter to carry out the calculations necessary to solve the problem.

PAL Example

- LLM has processed our request by creating a reasoning steps highlighted yellow and passing them as script to interpreter highlighted in gold.

PAL LLM one-shot inference example

Input : Patrick already bowled 3 overs. He needs to bowl 2 more overs. In cricket each over consists of 6 balls to be bowled. How many balls has he bowled in total?

Generate

Output : Patrick has bowled 3 overs, and each over consists of 6 balls. So, he has bowled 3 overs * 6 balls per over = 18 balls already.

If he needs to bowl 2 more overs, that's an additional 2 overs * 6 balls per over = 12 balls.

Therefore, in total, Patrick has bowled 18 + 12 = 30 balls.

Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don't want LLM to predict most probable token instead we want accurate answer.

- One strategy that has demonstrated some success is prompting the model to think more like a human is by breaking the problem down into steps.
  - And allowing the LLM to interact with external application that are good at math for e.g. Python interpreter, this helps to overcome the LLM limitation.
  - One framework for augmenting LLMs in this way is called **P**rogram-**A**ided **L**anguage (**PAL**) models.
  - The method makes use of chain of thought prompting to generate executable Python scripts. The scripts that the model generates are passed to an interpreter to execute.

Overview of PAL

The strategy behind PAL is to have the LLM generate completions where reasoning steps are accompanied by computer code.
This code is then passed to an interpreter to carry out the calculations necessary to solve the problem.

PAL Example

- LLM has processed our request by creating a reasoning steps highlighted yellow and passing them as script to interpreter highlighted in gold.

- Variables are declared based on the text in each reasoning step
- The values of variables are assigned either directly, as #3 and #6 in the first line of test in this example, or as a calculations using numbers present in the reasoning text as you see in next sentence and in the second line.
- The model can also work with variables it creates in pervious steps, as you see in the third line.

PAL LLM one-shot inference example

Input : Patrick already bowled 3 overs. He needs to bowl 2 more overs. In cricket each over consists of 6 balls to be bowled. How many balls has he bowled in total?

Generate

Output : Patrick has bowled 3 overs, and each over consists of 6 balls. So, he has bowled 3 overs * 6 balls per over = 18 balls already.

If he needs to bowl 2 more overs, that's an additional 2 overs * 6 balls per over = 12 balls.

Therefore, in total, Patrick has bowled 18 + 12 = 30 balls.

Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don't want LLM to predict most probable token instead we want accurate answer.

- One strategy that has demonstrated some success is prompting the model to think more like a human is by breaking the problem down into steps.
    - And allowing the LLM to interact with external application that are good at math for e.g. Python interpreter, this helps to overcome the LLM limitation.
    - One framework for augmenting LLMs in this way is called **P**rogram-**A**ided **L**anguage (**PAL**) models.
    - The method makes use of chain of thought prompting to generate executable Python scripts. The scripts that the model generates are passed to an interpreter to execute.

Overview of PAL

The strategy behind PAL is to have the LLM generate completions where reasoning steps are accompanied by computer code.
This code is then passed to an interpreter to carry out the calculations necessary to solve the problem.

PAL Example

- ❑ LLM has processed our request by creating a reasoning steps highlighted yellow and passing them as script to interpreter highlighted in gold.

- Variables are declared based on the text in each reasoning step
- The values of variables are assigned either directly, as #3 and #6 in the first line of test in this example, or as a calculations using numbers present in the reasoning text as you see in next sentence and in the second line.
- The model can also work with variables it creates in pervious steps, as you see in the third line.

PAL LLM one-shot inference example

Input : Patrick already bowled 3 overs. He needs to bowl 2 more overs. In cricket each over consists of 6 balls to be bowled. How many balls has he bowled in total?

Generate

Output : Patrick has bowled 3 overs, and each over consists of 6 balls. So, he has bowled 3 overs * 6 balls per over = 18 balls already.

If he needs to bowl 2 more overs, that's an additional 2 overs * 6 balls per over = 12 balls.

Therefore, in total, Patrick has bowled 18 + 12 = 30 balls.

Overview of LangChain

Kumar

Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don't want LLM to predict most probable token instead we want accurate answer.

- One strategy that has demonstrated some success is prompting the model to think more like a human is by breaking the problem down into steps.
  - And allowing the LLM to interact with external application that are good at math for e.g. Python interpreter, this helps to overcome the LLM limitation.
  - One framework for augmenting LLMs in this way is called **P**rogram-**A**ided **L**anguage (**PAL**) models.
  - The method makes use of chain of thought prompting to generate executable Python scripts. The scripts that the model generates are passed to an interpreter to execute.

Overview of PAL

The strategy behind PAL is to have the LLM generate completions where reasoning steps are accompanied by computer code.
This code is then passed to an interpreter to carry out the calculations necessary to solve the problem.

PAL Example

- ❑ LLM has processed our request by creating a reasoning steps highlighted yellow and passing them as script to interpreter highlighted in gold.

- Variables are declared based on the text in each reasoning step
- The values of variables are assigned either directly, as #3 and #6 in the first line of test in this example, or as a calculations using numbers present in the reasoning text as you see in next sentence and in the second line.
- The model can also work with variables it creates in pervious steps, as you see in the third line.

PAL LLM one-shot inference example

Input : Patrick already bowled 3 overs. He needs to bowl 2 more overs. In cricket each over consists of 6 balls to be bowled. How many balls has he bowled in total?

Generate

Output : Patrick has bowled 3 overs, and each over consists of 6 balls. So, he has bowled 3 overs * 6 balls per over = 18 balls already.

If he needs to bowl 2 more overs, that's an additional 2 overs * 6 balls per over = 12 balls.

Therefore, in total, Patrick has bowled 18 + 12 = 30 balls.

Overview of LangChain

Depending on your LLM use case you may need to manage multiple decision points, validations actional and calls to external application.
One framework that can orchestrate these actions is LangChain.
LangChain framework provides you with modular pieces that contain the components necessary to work with LLMs.

Kumar

Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don't want LLM to predict most probable token instead we want accurate answer.

- One strategy that has demonstrated some success is prompting the model to think more like a human is by breaking the problem down into steps.
    - And allowing the LLM to interact with external application that are good at math for e.g. Python interpreter, this helps to overcome the LLM limitation.
    - One framework for augmenting LLMs in this way is called **P**rogram-**A**ided **L**anguage (**PAL**) models.
    - The method makes use of chain of thought prompting to generate executable Python scripts. The scripts that the model generates are passed to an interpreter to execute.

Overview of PAL

The strategy behind PAL is to have the LLM generate completions where reasoning steps are accompanied by computer code.
This code is then passed to an interpreter to carry out the calculations necessary to solve the problem.

PAL Example

- ❑ LLM has processed our request by creating a reasoning steps highlighted yellow and passing them as script to interpreter highlighted in gold.

- Variables are declared based on the text in each reasoning step
- The values of variables are assigned either directly, as #3 and #6 in the first line of test in this example, or as a calculations using numbers present in the reasoning text as you see in next sentence and in the second line.
- The model can also work with variables it creates in pervious steps, as you see in the third line.

PAL LLM one-shot inference example

Input : Patrick already bowled 3 overs. He needs to bowl 2 more overs. In cricket each over consists of 6 balls to be bowled. How many balls has he bowled in total?

Generate

Output : Patrick has bowled 3 overs, and each over consists of 6 balls. So, he has bowled 3 overs * 6 balls per over = 18 balls already.

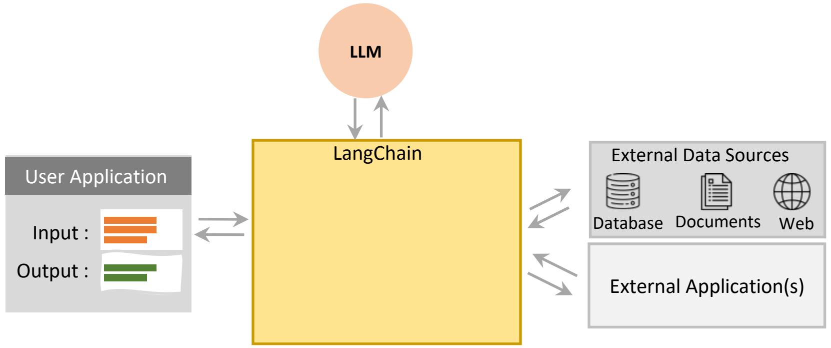
If he needs to bowl 2 more overs, that's an additional 2 overs * 6 balls per over = 12 balls.

Therefore, in total, Patrick has bowled 18 + 12 = 30 balls.

Overview of LangChain

Depending on your LLM use case you may need to manage multiple decision points, validations actional and calls to external application.
One framework that can orchestrate these actions is LangChain.
LangChain framework provides you with modular pieces that contain the components necessary to work with LLMs.

Components of LangChain framework :



Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don't want LLM to predict most probable token instead we want accurate answer.

- One strategy that has demonstrated some success is prompting the model to think more like a human is by breaking the problem down into steps.
  - And allowing the LLM to interact with external application that are good at math for e.g. Python interpreter, this helps to overcome the LLM limitation.
  - One framework for augmenting LLMs in this way is called **P**rogram-**A**ided **L**anguage (**PAL**) models.
  - The method makes use of chain of thought prompting to generate executable Python scripts. The scripts that the model generates are passed to an interpreter to execute.

Overview of PAL

The strategy behind PAL is to have the LLM generate completions where reasoning steps are accompanied by computer code.
This code is then passed to an interpreter to carry out the calculations necessary to solve the problem.

PAL Example

- ❑ LLM has processed our request by creating a reasoning steps highlighted yellow and passing them as script to interpreter highlighted in gold.

- Variables are declared based on the text in each reasoning step
- The values of variables are assigned either directly, as #3 and #6 in the first line of test in this example, or as a calculations using numbers present in the reasoning text as you see in next sentence and in the second line.
- The model can also work with variables it creates in pervious steps, as you see in the third line.

PAL LLM one-shot inference example

Input : Patrick already bowled 3 overs. He needs to bowl 2 more overs. In cricket each over consists of 6 balls to be bowled. How many balls has he bowled in total?

Generate

Output : Patrick has bowled 3 overs, and each over consists of 6 balls. So, he has bowled 3 overs * 6 balls per over = 18 balls already.

If he needs to bowl 2 more overs, that's an additional 2 overs * 6 balls per over = 12 balls.

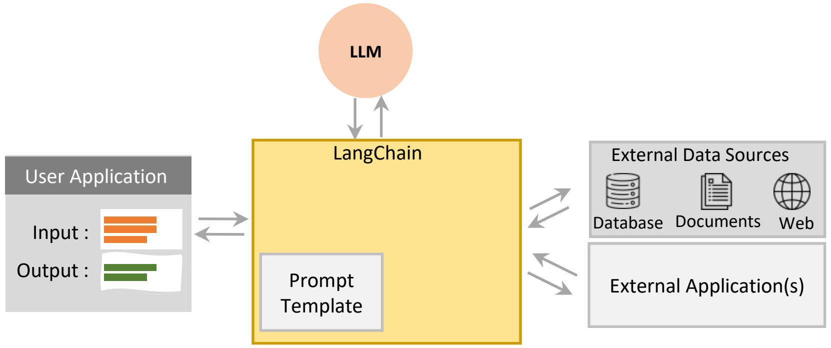
Therefore, in total, Patrick has bowled 18 + 12 = 30 balls.

Overview of LangChain

Depending on your LLM use case you may need to manage multiple decision points, validations actional and calls to external application.
One framework that can orchestrate these actions is LangChain.
LangChain framework provides you with modular pieces that contain the components necessary to work with LLMs.

Components of LangChain framework :

Prompt templates — Templates can used for many different use cases that you can use to format both input examples and model completions.



Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don't want LLM to predict most probable token instead we want accurate answer.

- One strategy that has demonstrated some success is prompting the model to think more like a human is by breaking the problem down into steps.
  - And allowing the LLM to interact with external application that are good at math for e.g. Python interpreter, this helps to overcome the LLM limitation.
  - One framework for augmenting LLMs in this way is called **P**rogram-**A**ided **L**anguage (**PAL**) models.
  - The method makes use of chain of thought prompting to generate executable Python scripts. The scripts that the model generates are passed to an interpreter to execute.

Overview of PAL

The strategy behind PAL is to have the LLM generate completions where reasoning steps are accompanied by computer code.
This code is then passed to an interpreter to carry out the calculations necessary to solve the problem.

PAL Example

- ❑ LLM has processed our request by creating a reasoning steps highlighted yellow and passing them as script to interpreter highlighted in gold.

- Variables are declared based on the text in each reasoning step
- The values of variables are assigned either directly, as #3 and #6 in the first line of test in this example, or as a calculations using numbers present in the reasoning text as you see in next sentence and in the second line.
- The model can also work with variables it creates in pervious steps, as you see in the third line.

PAL LLM one-shot inference example

Input : Patrick already bowled 3 overs. He needs to bowl 2 more overs. In cricket each over consists of 6 balls to be bowled. How many balls has he bowled in total?

Generate

Output : Patrick has bowled 3 overs, and each over consists of 6 balls. So, he has bowled 3 overs * 6 balls per over = 18 balls already.

If he needs to bowl 2 more overs, that's an additional 2 overs * 6 balls per over = 12 balls.

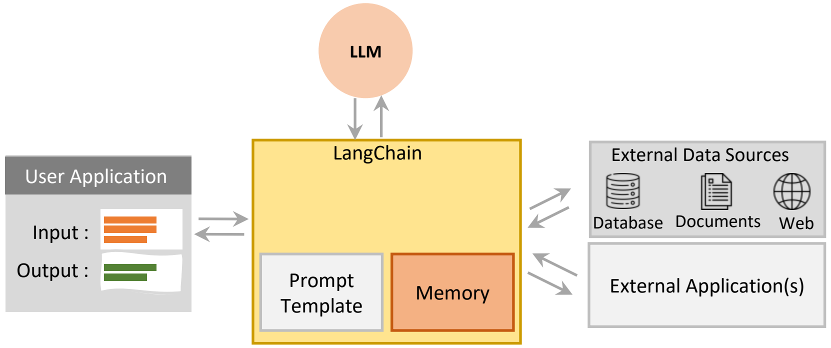
Therefore, in total, Patrick has bowled 18 + 12 = 30 balls.

Overview of LangChain

Depending on your LLM use case you may need to manage multiple decision points, validations actional and calls to external application.
One framework that can orchestrate these actions is LangChain.
LangChain framework provides you with modular pieces that contain the components necessary to work with LLMs.

Components of LangChain framework :

Prompt templates	Templates can used for many different use cases that you can use to format both input examples and model completions.
Memory	To store interactions with an LLM.



Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don't want LLM to predict most probable token instead we want accurate answer.

- One strategy that has demonstrated some success is prompting the model to think more like a human is by breaking the problem down into steps.
  - And allowing the LLM to interact with external application that are good at math for e.g. Python interpreter, this helps to overcome the LLM limitation.
  - One framework for augmenting LLMs in this way is called **P**rogram-**A**ided **L**anguage (**PAL**) models.
  - The method makes use of chain of thought prompting to generate executable Python scripts. The scripts that the model generates are passed to an interpreter to execute.

Overview of PAL

The strategy behind PAL is to have the LLM generate completions where reasoning steps are accompanied by computer code.
This code is then passed to an interpreter to carry out the calculations necessary to solve the problem.

PAL Example

- ❑ LLM has processed our request by creating a reasoning steps highlighted yellow and passing them as script to interpreter highlighted in gold.

- Variables are declared based on the text in each reasoning step
- The values of variables are assigned either directly, as #3 and #6 in the first line of test in this example, or as a calculations using numbers present in the reasoning text as you see in next sentence and in the second line.
- The model can also work with variables it creates in pervious steps, as you see in the third line.

PAL LLM one-shot inference example

Input : Patrick already bowled 3 overs. He needs to bowl 2 more overs. In cricket each over consists of 6 balls to be bowled. How many balls has he bowled in total?

Generate

Output : Patrick has bowled 3 overs, and each over consists of 6 balls. So, he has bowled 3 overs * 6 balls per over = 18 balls already.

If he needs to bowl 2 more overs, that's an additional 2 overs * 6 balls per over = 12 balls.

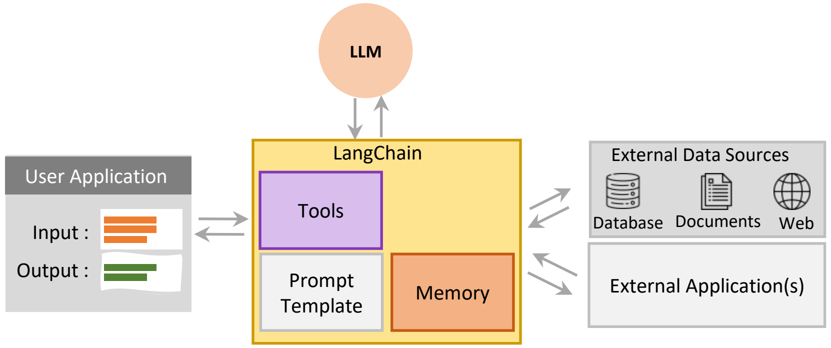
Therefore, in total, Patrick has bowled 18 + 12 = 30 balls.

Overview of LangChain

Depending on your LLM use case you may need to manage multiple decision points, validations actional and calls to external application.
One framework that can orchestrate these actions is LangChain.
LangChain framework provides you with modular pieces that contain the components necessary to work with LLMs.

Components of LangChain framework :

Prompt templates	Templates can used for many different use cases that you can use to format both input examples and model completions.
Memory	To store interactions with an LLM.
Pre-built tools	It has pre-built tools that enable to carry our wide variety of tasks, including calls to external data sources and APIs



Generative AI Project Lifecycle – LLMs Reason and Plan with chain-of-thoughts

Complex reasoning which involves advance maths can be challenging for LLMs. We don't want LLM to predict most probable token instead we want accurate answer.

- One strategy that has demonstrated some success is prompting the model to think more like a human is by breaking the problem down into steps.
  - And allowing the LLM to interact with external application that are good at math for e.g. Python interpreter, this helps to overcome the LLM limitation.
  - One framework for augmenting LLMs in this way is called **P**rogram-**A**ided **L**anguage (**PAL**) models.
  - The method makes use of chain of thought prompting to generate executable Python scripts. The scripts that the model generates are passed to an interpreter to execute.

Overview of PAL

The strategy behind PAL is to have the LLM generate completions where reasoning steps are accompanied by computer code.
This code is then passed to an interpreter to carry out the calculations necessary to solve the problem.

PAL Example

- ❑ LLM has processed our request by creating a reasoning steps highlighted yellow and passing them as script to interpreter highlighted in gold.
  - Variables are declared based on the text in each reasoning step
  - The values of variables are assigned either directly, as #3 and #6 in the first line of test in this example, or as a calculations using numbers present in the reasoning text as you see in next sentence and in the second line.
  - The model can also work with variables it creates in pervious steps, as you see in the third line.

PAL LLM one-shot inference example

Input : Patrick already bowled 3 overs. He needs to bowl 2 more overs. In cricket each over consists of 6 balls to be bowled. How many balls has he bowled in total?

Generate

Output : Patrick has bowled 3 overs, and each over consists of 6 balls. So, he has bowled 3 overs * 6 balls per over = 18 balls already.

If he needs to bowl 2 more overs, that's an additional 2 overs * 6 balls per over = 12 balls.

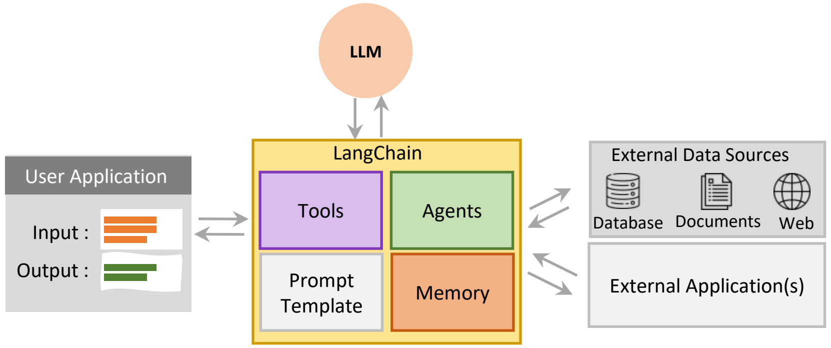
Therefore, in total, Patrick has bowled 18 + 12 = 30 balls.

Overview of LangChain

Depending on your LLM use case you may need to manage multiple decision points, validations actional and calls to external application.
One framework that can orchestrate these actions is LangChain.
LangChain framework provides you with modular pieces that contain the components necessary to work with LLMs.

Components of LangChain framework :

Prompt templates	Templates can used for many different use cases that you can use to format both input examples and model completions.
Memory	To store interactions with an LLM.
Pre-built tools	It has pre-built tools that enable to carry our wide variety of tasks, including calls to external data sources and APIs
Agents	Agents can be used to interpret the input from user and determine which tool or tools to use to complete the task



Kumar

LLM Application Architecture

Several key components are required to create end-to-end solutions for LLM applications.

LLM Application Architecture

Several key components are required to create end-to-end solutions for LLM applications.

infrastructure layer	This layer provides the compute, storage, and network to serve the LLMs, as well as to host the application components. This layer can be provisioned via on-demand and pay-as-you-go cloud services.

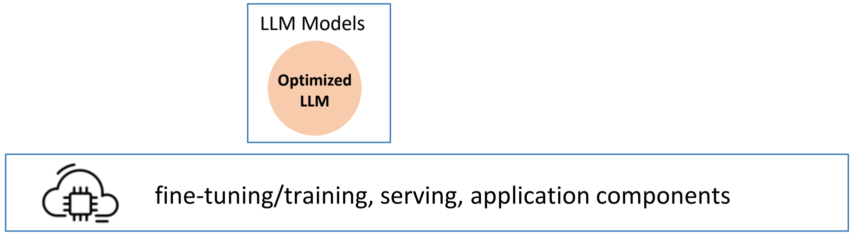
fine-tuning/training, serving, application components

Kumar

LLM Application Architecture

Several key components are required to create end-to-end solutions for LLM applications.

infrastructure layer	This layer provides the compute, storage, and network to serve the LLMs, as well as to host the application components. This layer can be provisioned via on-demand and pay-as-you-go cloud services.
LLM	Large Language Models you want to use in your application. • These could include foundation models, as well as the models you have adapted to your specific task. • The models are deployed on the appropriate infrastructure for your inference needs. Taking into account whether you need real-time or near-real-time interaction with the model.

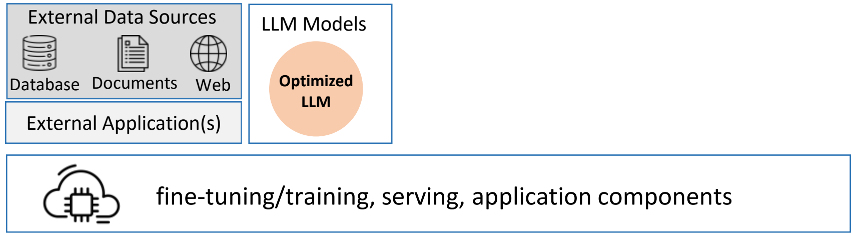


Kumar

LLM Application Architecture

Several key components are required to create end-to-end solutions for LLM applications.

infrastructure layer	This layer provides the compute, storage, and network to serve the LLMs, as well as to host the application components. This layer can be provisioned via on-demand and pay-as-you-go cloud services.
LLM	Large Language Models you want to use in your application. • These could include foundation models, as well as the models you have adapted to your specific task. • The models are deployed on the appropriate infrastructure for your inference needs. Taking into account whether you need real-time or near-real-time interaction with the model.
External Data Sources	You may also have the need to retrieve information from external sources or applications for use the cases where you need to implement RAG.

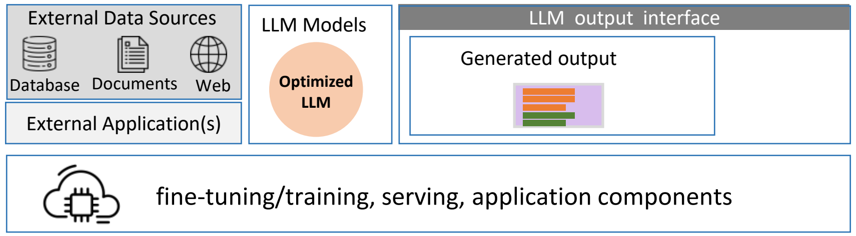


Kumar

LLM Application Architecture

Several key components are required to create end-to-end solutions for LLM applications.

infrastructure layer	This layer provides the compute, storage, and network to serve the LLMs, as well as to host the application components. This layer can be provisioned via on-demand and pay-as-you-go cloud services.
LLM	Large Language Models you want to use in your application. • These could include foundation models, as well as the models you have adapted to your specific task. • The models are deployed on the appropriate infrastructure for your inference needs. Taking into account whether you need real-time or near-real-time interaction with the model.
External Data Sources	You may also have the need to retrieve information from external sources or applications for use the cases where you need to implement RAG.
LLM output interface	The LLM application will return the completions from selected LLM to the user or consuming application. Depending on the use case, you may need to implement a mechanism to capture and store the outputs. For example, you could build the capacity to store user completions during a session to augment the fixed contexts window size of your LLM.

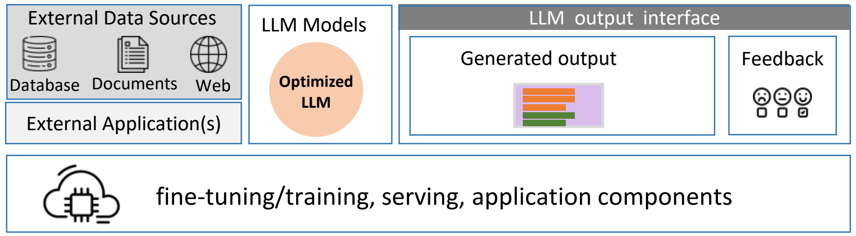


Kumar

LLM Application Architecture

Several key components are required to create end-to-end solutions for LLM applications.

infrastructure layer	This layer provides the compute, storage, and network to serve the LLMs, as well as to host the application components. This layer can be provisioned via on-demand and pay-as-you-go cloud services.
LLM	Large Language Models you want to use in your application. • These could include foundation models, as well as the models you have adapted to your specific task. • The models are deployed on the appropriate infrastructure for your inference needs. Taking into account whether you need real-time or near-real-time interaction with the model.
External Data Sources	You may also have the need to retrieve information from external sources or applications for use the cases where you need to implement RAG.
LLM output interface	The LLM application will return the completions from selected LLM to the user or consuming application. Depending on the use case, you may need to implement a mechanism to capture and store the outputs. For example, you could build the capacity to store user completions during a session to augment the fixed contexts window size of your LLM.
LLM Feedback Interface	You can also gather feedback from users that may be useful for additional fine-tuning, alignment, or evaluation as your application matures.

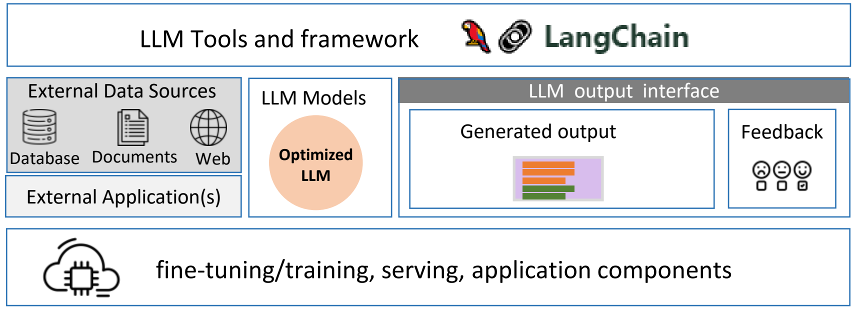


Kumar

LLM Application Architecture

Several key components are required to create end-to-end solutions for LLM applications.

infrastructure layer	This layer provides the compute, storage, and network to serve the LLMs, as well as to host the application components. This layer can be provisioned via on-demand and pay-as-you-go cloud services.
LLM	Large Language Models you want to use in your application. • These could include foundation models, as well as the models you have adapted to your specific task. • The models are deployed on the appropriate infrastructure for your inference needs. Taking into account whether you need real-time or near-real-time interaction with the model.
External Data Sources	You may also have the need to retrieve information from external sources or applications for use the cases where you need to implement RAG.
LLM output interface	The LLM application will return the completions from selected LLM to the user or consuming application. Depending on the use case, you may need to implement a mechanism to capture and store the outputs. For example, you could build the capacity to store user completions during a session to augment the fixed contexts window size of your LLM.
LLM Feedback Interface	You can also gather feedback from users that may be useful for additional fine-tuning, alignment, or evaluation as your application matures.
LLM Tools	You may need to use additional tools and frameworks for large language models that help you easily implement. As an example, you can use LangChain built-in libraries to implement techniques like PAL, ReAct or Chain of Thought prompting.

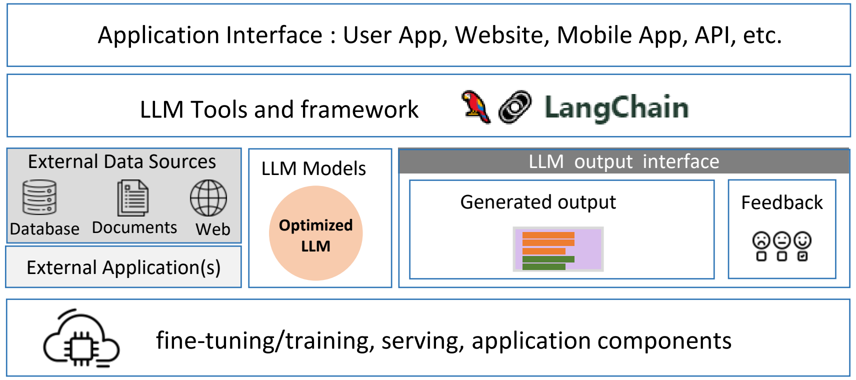


Kumar

LLM Application Architecture

Several key components are required to create end-to-end solutions for LLM applications.

infrastructure layer	This layer provides the compute, storage, and network to serve the LLMs, as well as to host the application components. This layer can be provisioned via on-demand and pay-as-you-go cloud services.
LLM	Large Language Models you want to use in your application. • These could include foundation models, as well as the models you have adapted to your specific task. • The models are deployed on the appropriate infrastructure for your inference needs. Taking into account whether you need real-time or near-real-time interaction with the model.
External Data Sources	You may also have the need to retrieve information from external sources or applications for use the cases where you need to implement RAG.
LLM output interface	The LLM application will return the completions from selected LLM to the user or consuming application. Depending on the use case, you may need to implement a mechanism to capture and store the outputs. For example, you could build the capacity to store user completions during a session to augment the fixed contexts window size of your LLM.
LLM Feedback Interface	You can also gather feedback from users that may be useful for additional fine-tuning, alignment, or evaluation as your application matures.
LLM Tools	You may need to use additional tools and frameworks for large language models that help you easily implement. As an example, you can use LangChain built-in libraries to implement techniques like PAL, ReAct or Chain of Thought prompting.
Application/ User Interface	The final layer is user interface that the application will be consumed through, such as a website or a rest API. This layer is where you'll also include the security components required for interacting with your application.

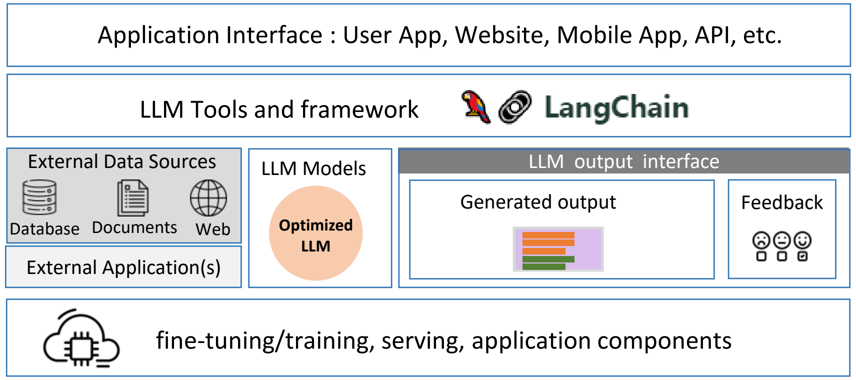


Kumar

LLM Application Architecture

Several key components are required to create end-to-end solutions for LLM applications.

infrastructure layer	This layer provides the compute, storage, and network to serve the LLMs, as well as to host the application components. This layer can be provisioned via on-demand and pay-as-you-go cloud services.
LLM	Large Language Models you want to use in your application. • These could include foundation models, as well as the models you have adapted to your specific task. • The models are deployed on the appropriate infrastructure for your inference needs. Taking into account whether you need real-time or near-real-time interaction with the model.
External Data Sources	You may also have the need to retrieve information from external sources or applications for use the cases where you need to implement RAG.
LLM output interface	The LLM application will return the completions from selected LLM to the user or consuming application. Depending on the use case, you may need to implement a mechanism to capture and store the outputs. For example, you could build the capacity to store user completions during a session to augment the fixed contexts window size of your LLM.
LLM Feedback Interface	You can also gather feedback from users that may be useful for additional fine-tuning, alignment, or evaluation as your application matures.
LLM Tools	You may need to use additional tools and frameworks for large language models that help you easily implement. As an example, you can use LangChain built-in libraries to implement techniques like PAL, ReAct or Chain of Thought prompting.
Application/ User Interface	The final layer is user interface that the application will be consumed through, such as a website or a rest API. This layer is where you'll also include the security components required for interacting with your application.



At a high level, this architecture stack represents the various components to consider as part of generative AI applications.

Kumar

The End