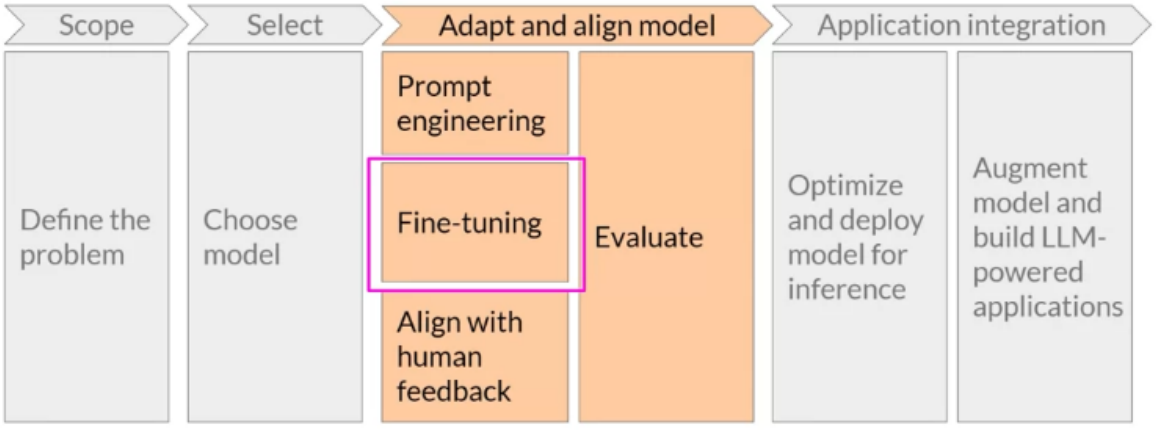
Fine-tuning LLMs with instruction

> Source: Coursera

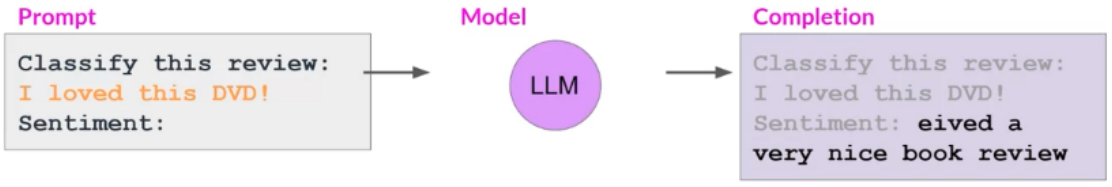
▼ Instruction fine-tuning



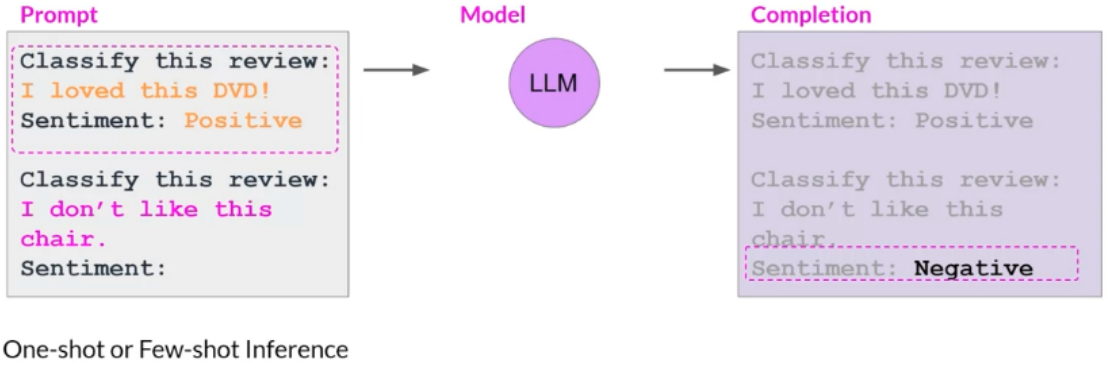
Methods exist for improving the performance of an existing model for a specific use case. Important metrics can also be used to evaluate the performance of the fine-tuned LLM and quantify its improvement over the starting base model.

▼ Fine-tuning an LLM with instruction prompts

Some models can identify instructions contained in a prompt and correctly carry out zero-shot inference. In contrast, others, such as smaller LLMs, may fail the task, as shown here.

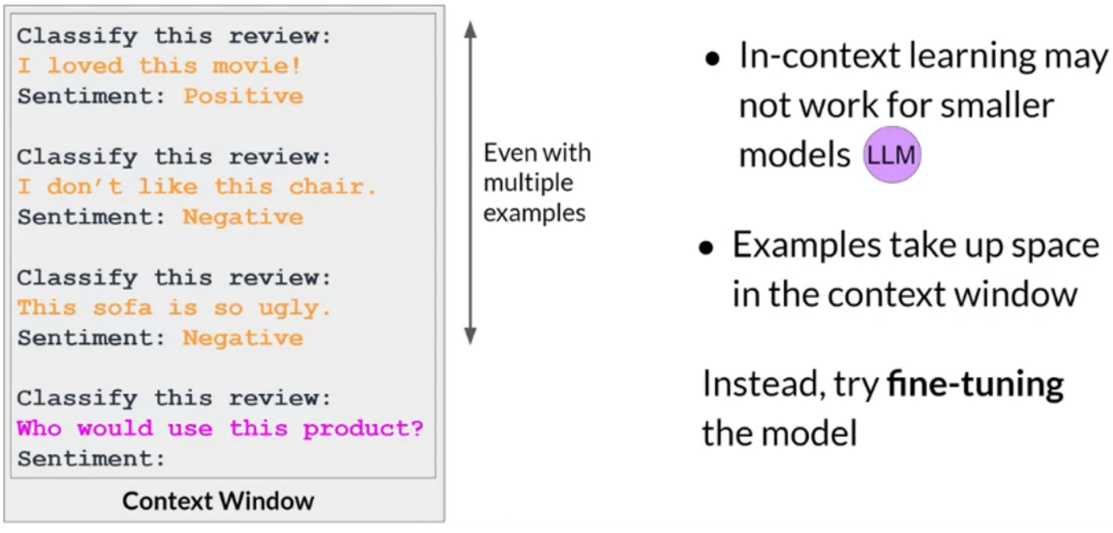


In-context learning (ICL) - zero shot inference



In-context learning (ICL) - one/few shot inference

Including one or more examples of what you want the model to do, known as one-shot or few-shot inference, can help the model identify the task and generate a good completion. However, this strategy has a couple of drawbacks. First, it doesn't always work for smaller models, even when five or six examples are included. Second, any examples included in the prompt take up valuable space in the context window, reducing the amount of room to include other useful information.

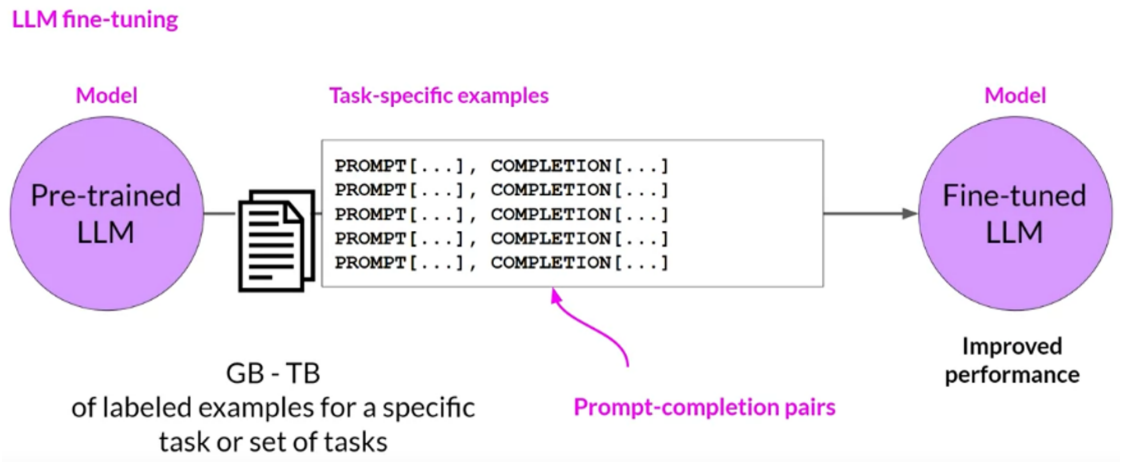


Limitations of in-context learning

Luckily, fine-tuning can train a base model further. In contrast to pre-training, where the LLM is trained using vast amounts of unstructured textual data via

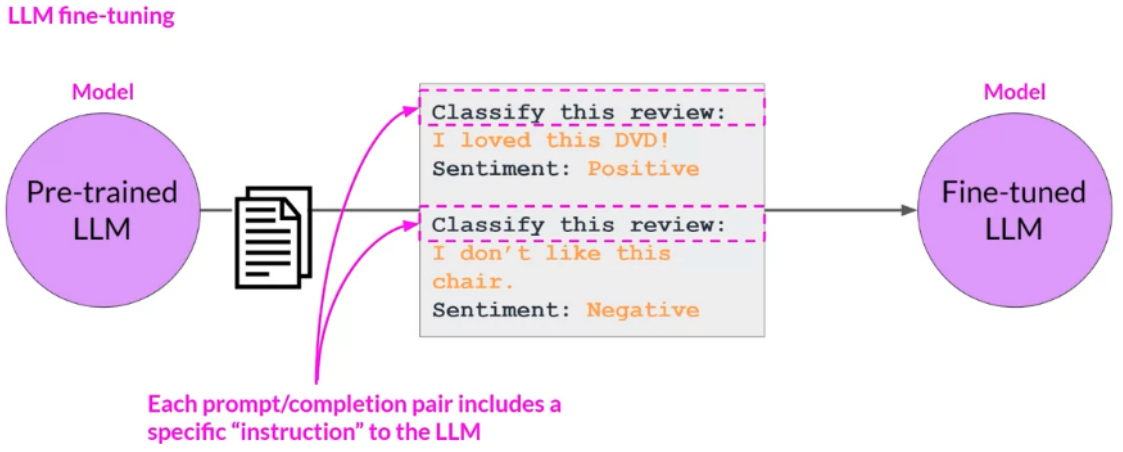
self-supervised learning, fine-tuning is a supervised learning process that uses a data set of labelled examples to update the LLM's weights.

The labelled examples are prompt completion pairs. The fine-tuning process extends the model's training to improve its ability to generate good completions for a specific task.



LLM fine-tuning at a high level

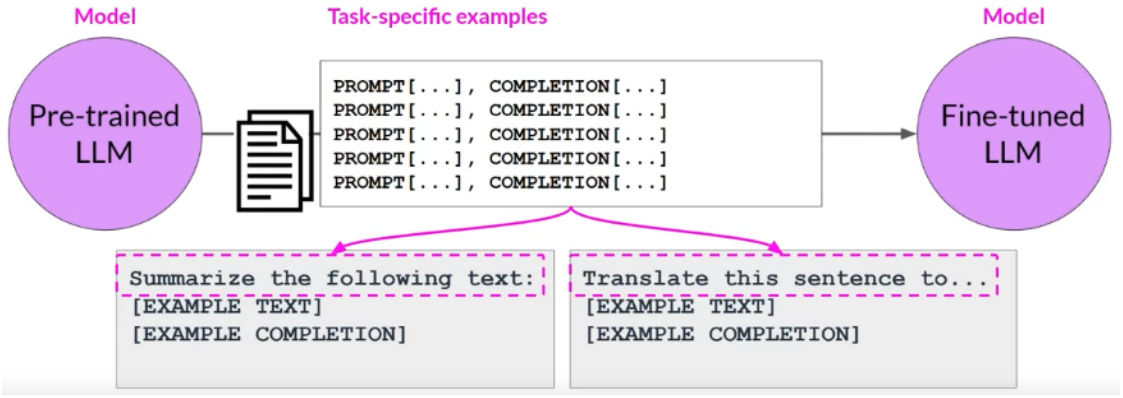
One strategy, instruction fine-tuning, is particularly effective at improving a model's performance on various tasks. Instruction fine-tuning trains the model using examples that demonstrate how it should respond to a specific instruction.



Using prompts to fine-tune LLMs with instruction

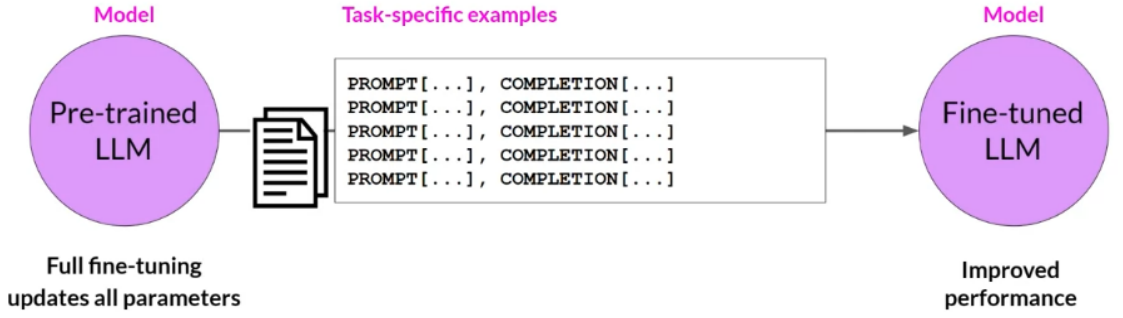
Here are a couple of example prompts to demonstrate this idea. In both examples, the instruction is to classify this review, and the desired completion is a text string that starts with sentiment followed by either positive or negative.

The data set for training includes many pairs of prompt completion examples, each of which includes an instruction.



For example, to fine-tune the model to improve its summarization ability, we'd build up a data set of examples that begin with the instruction summarize, the following text or a similar phrase. To improve the model's translation skills, the examples would include instructions like translating this sentence.

These prompt completion examples allow the model to learn to generate responses that follow the given instructions. Instruction fine-tuning, where all of the model's weights are updated, is known as full fine-tuning. The process results in a new version of the model with updated weights.



Like pre-training, full fine-tuning requires enough memory and a computing budget to store and process all the gradients, optimizers, and other updated components during training. This can benefit from memory optimization and parallel computing strategies.

The first step is to prepare the training data. Many publicly available datasets have been used to train earlier generations of language models, although most are not formatted as instructions. Luckily, developers have assembled prompt template libraries that can be used to take existing datasets, such as the large data set of Amazon product reviews, and turn them into instruction prompt datasets for fine-tuning. Prompt template libraries include many templates for different tasks and different data sets.

Classification / sentiment analysis

```
jinja: "Given the following review:\n{{review_body}}\npredict the associated rating\
  \ from the following choices (1 being lowest and 5 being highest)\n- {{ answer_choices\
  \ | join('\\n- ') }} \n|||\n{{answer_choices[star_rating-1]}}"
```

Text generation

```
jinja: Generate a {{star_rating}}-star review (1 being lowest and 5 being highest)
  about this product {{product_title}}.  |||  {{review_body}}
```

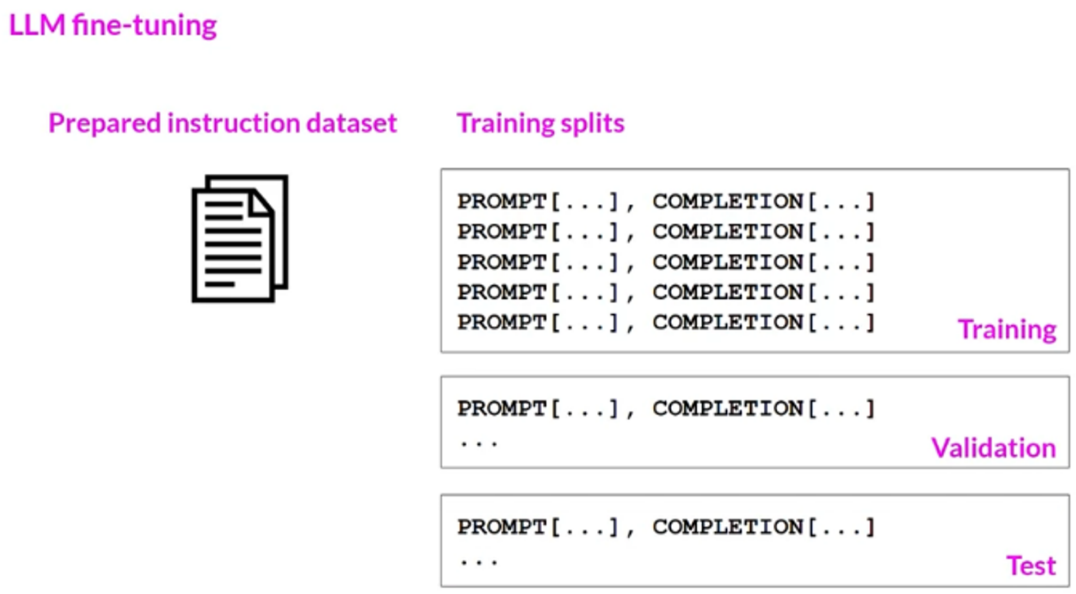
Text summarization

```
jinja: "Give a short sentence describing the following product review:\n{{review_body}}\
  \ \n|||\n{{review_headline}}"
```

Sample prompt instruction templates, Source: https://github.com/bigscience-workshop/promptsource/blob/main/promptsource/templates/amazon_polarity/templates.yaml

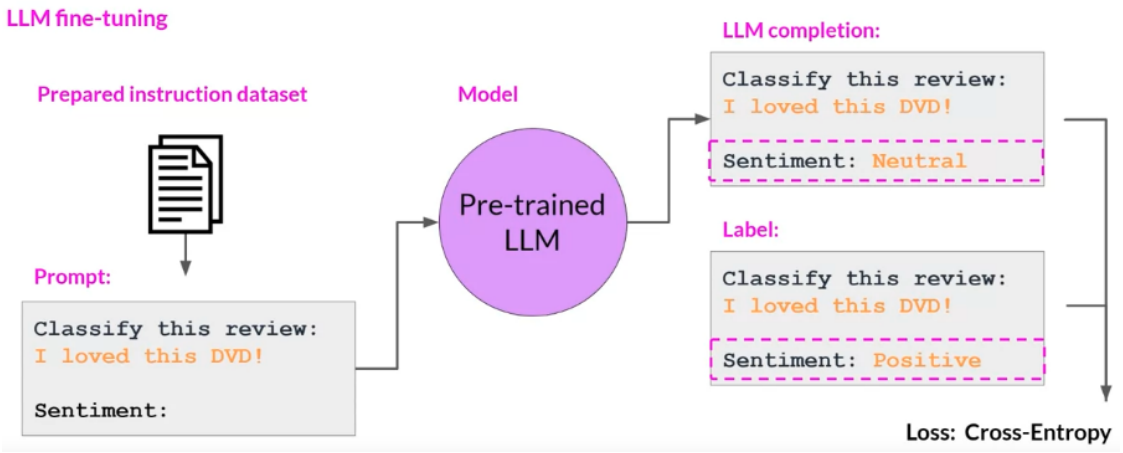
Here are three prompts designed to work with the Amazon reviews dataset that can be used to fine-tune models for classification, text generation and text summarization tasks. In each case, pass the original review, here called review_body, to the template, where it gets inserted into the text that starts with an instruction like predicting the associated rating, generating a star review, or giving a short sentence describing the following product review. The result is a prompt containing both an instruction and an example from the data set.

Once the instruction data is ready, as with standard supervised learning, divide the data set into training validation and test splits. During fine-tuning, select prompts from the training data set and pass them to the LLM, generating completions.



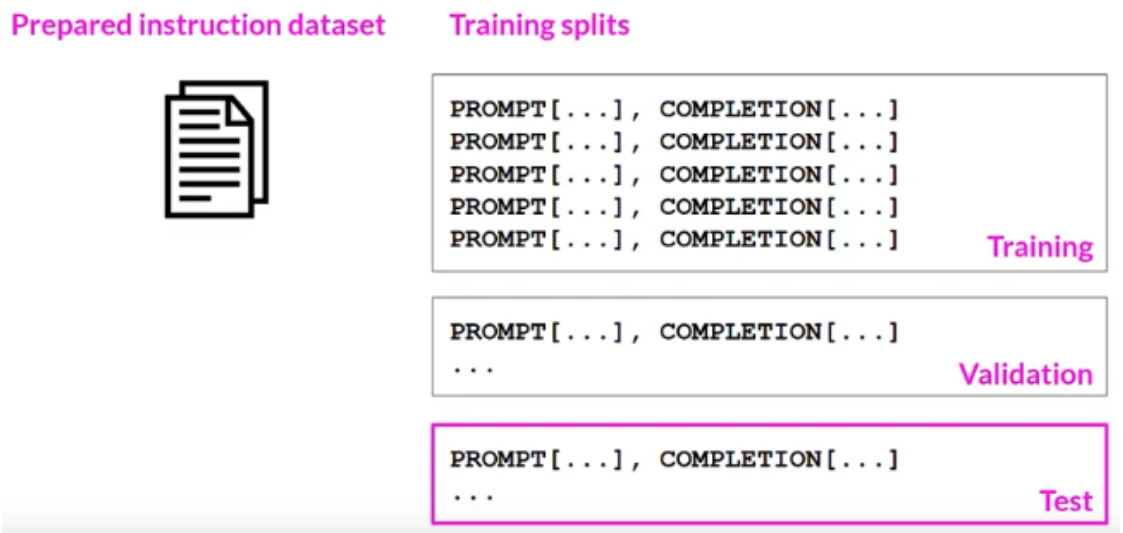
LLM fine-tuning process

Next, compare the LLM completion with the response specified in the training data. Here, the model didn't do a great job. It classified the review as neutral, which is a bit of an understatement. The review is very positive.



The output of an LLM is a probability distribution across tokens. So, we can compare the completion and training label distributions and use the standard cross entropy function to calculate the loss between them. Then, using the calculated loss, we can update the model weights in standard backpropagation. We'll do this for many batches of prompt completion pairs over several epochs and update the weights to improve the model's performance on the task.

As in standard supervised learning, we can define separate evaluation steps to measure the LLM performance using the holdout validation data set. This will give the validation accuracy. After we've completed the fine-tuning, we can perform a final performance evaluation using the holdout test data set. This will give the test accuracy.

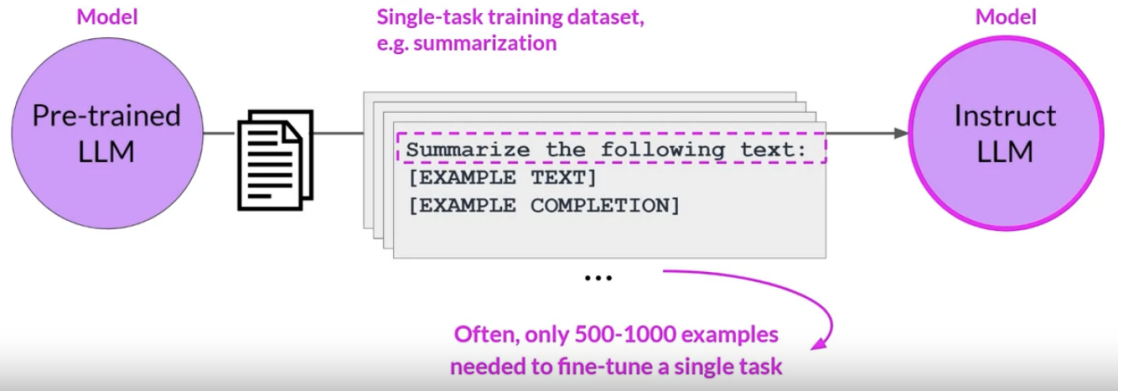


The fine-tuning process results in a new version of the base model, often called an instruct model, better at the tasks we are interested in. Fine-tuning with instruction prompts is the most common way to fine-tune LLMs. From this point on, when hearing or seeing the term fine-tuning, we can assume that it always means instruction fine-tuning.

Fine-tuning a single task

Single-task fine-tuning

While LLMs have become famous for their ability to perform many different language tasks within a single model, some applications may only need to perform a single task. In this case, fine-tune a pre-trained model to improve performance only on the task.



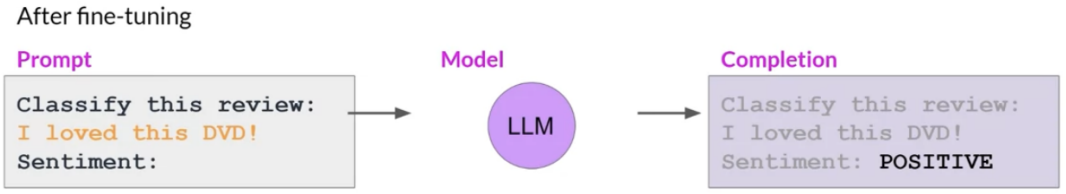
Fine-tuning a single task

For example, summarization can be done using a dataset of examples for that task. Interestingly, good results can be achieved with relatively few examples. Often, just 500-1,000 examples can result in good performance compared to the billions of pieces of text that the model saw during pre-training.

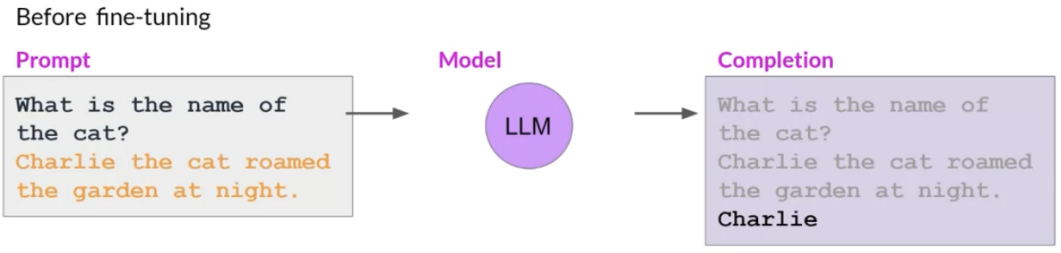
Catastrophic forgetting

However, there is a potential downside to fine-tuning on a single task. The process may lead to a phenomenon called catastrophic forgetting. Catastrophic forgetting happens because the full fine-tuning process modifies the weights of the original LLM. While this leads to great performance on a single fine-tuning task, it can degrade performance on other tasks. For example, while fine-tuning can improve the ability of a model to perform sentiment analysis on a review and result in a quality completion, the model may forget how to do other tasks.

- Fine-tuning can significantly increase the performance of a model on a specific task...

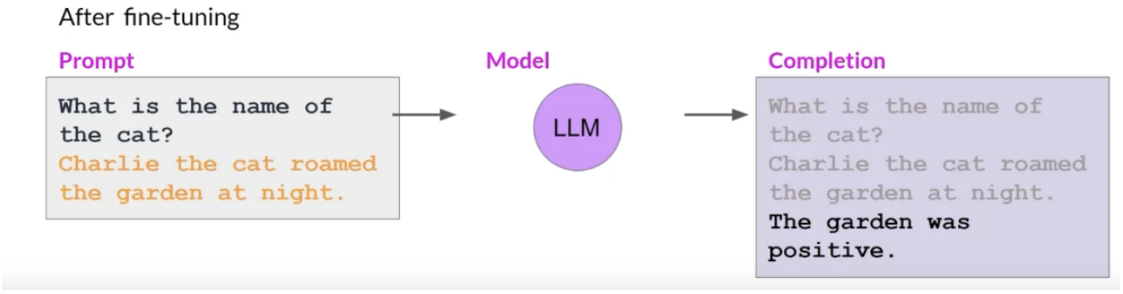


- ...but can lead to reduction in ability on other tasks



This model knew how to recognise a named entity before fine-tuning and correctly identifying Charlie as the cat's name in the sentence. However, after fine-tuning, the model can no longer carry out this task, confusing the entity it is supposed to identify and exhibiting behaviour related to the new task.

- ...but can lead to reduction in ability on other tasks



What options do we have to avoid catastrophic forgetting?

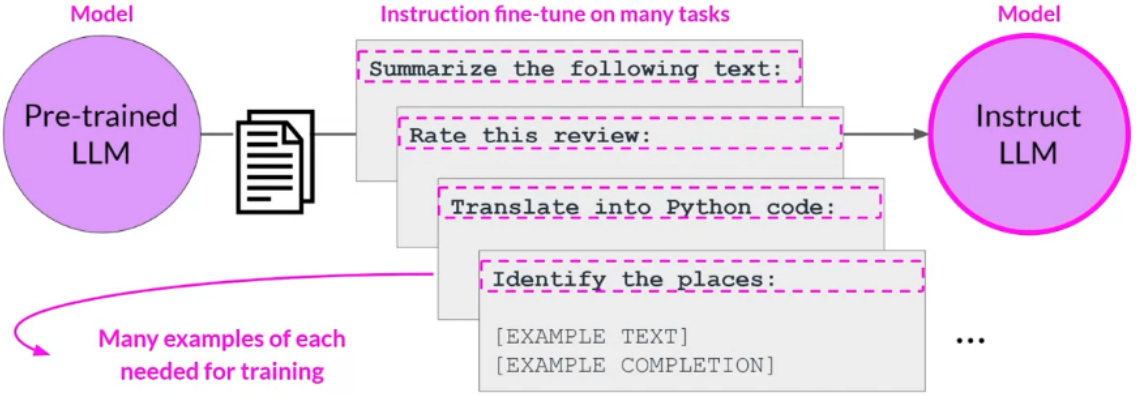
First, it is important to decide whether catastrophic forgetting impacts the use case. If reliable performance is the only need for a single fine-tuned task, it may not be an issue that the model can't generalize to other tasks. If the model needs to maintain its generalised multitasking capabilities, fine-tuning multiple tasks simultaneously can help.

Good multitasking fine-tuning may require 50-100,000 examples across many tasks, and so will require more data and computing to train.

The second option is to perform parameter-efficient fine-tuning, or PEFT for short, instead of full fine-tuning. 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. Since most pre-trained weights are left unchanged, PEFT is more robust to catastrophic forgetting.

▼ Multi-task instruction fine-tuning

Multitask fine-tuning is an extension of single-task fine-tuning, where the training dataset is comprised of example inputs and outputs for multiple tasks.



Multi-task, instruction fine-tuning

The dataset contains examples instructing the model to perform various tasks, including summarization, review rating, code translation, and entity recognition. Training the model on this mixed dataset so that it can improve the performance of the model on all the tasks simultaneously, thus avoiding the issue of catastrophic forgetting.

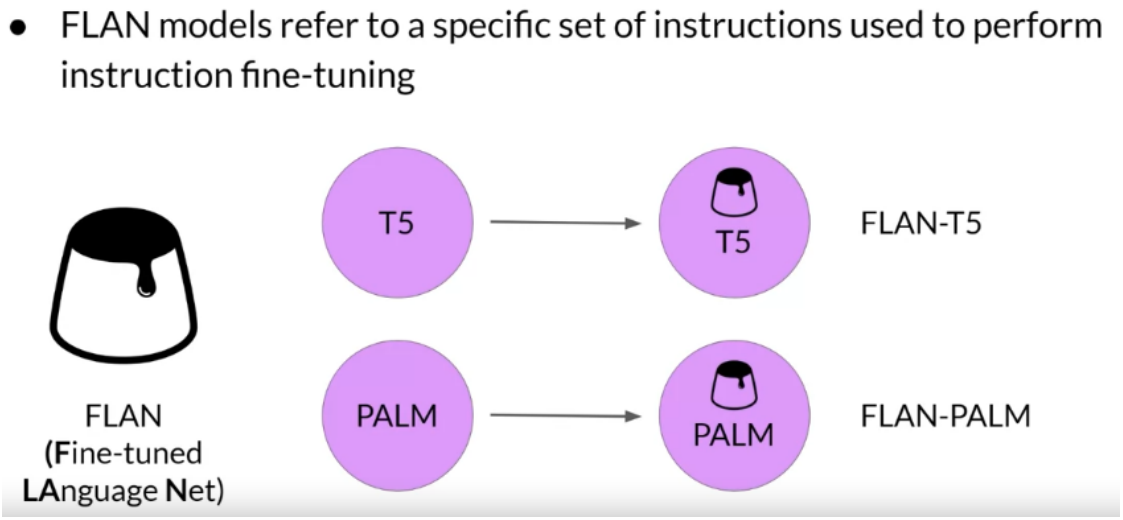
Over many epochs of training, the calculated losses across examples are used to update the model weights, resulting in an instruction-tuned model that is learned to be good at many different tasks simultaneously.

One drawback to multitask fine-tuning is that it requires a lot of data. The training set may need as many as 50-100,000 examples. However, assembling this data can be worthwhile and worth the effort. The resulting models are often very capable and suitable for use in situations where good performance at many tasks is desirable.

▼ Instruction fine-tuning with FLAN

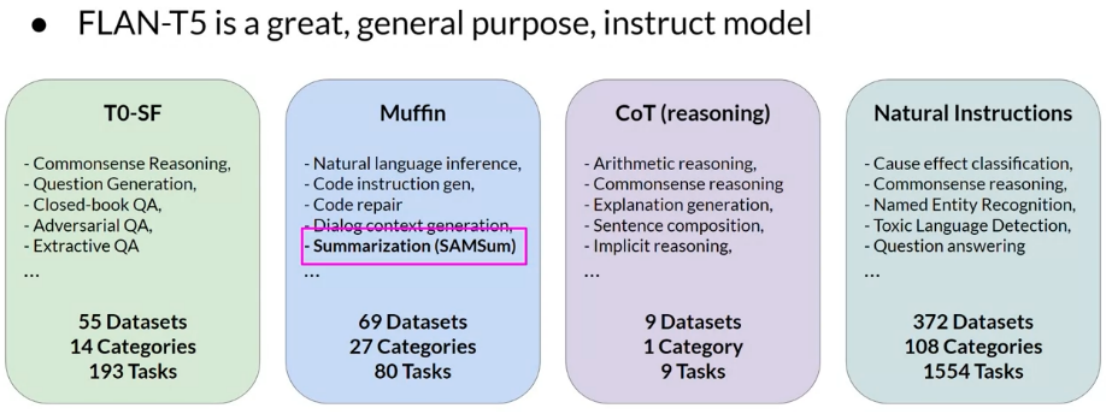
Let's take a look at one family of models that have been trained using multitask instruction fine-tuning. Instruct model variance differs based on the datasets

and tasks used during fine-tuning. One example is the FLAN family of models. FLAN, a fine-tuned language net, is a specific set of instructions to fine-tune different models.



Because their FLAN fine-tuning is the last step of the training process, the original paper's authors called it the metaphorical dessert to the main course of pre-training, which is quite a fitting name. FLAN-T5 is the FLAN instruct version of the T5 foundation model, while FLAN-PALM is the flattening struct version of the palm foundation model.

FLAN-T5 is a great general-purpose instruction model. It's been fine-tuned on 473 datasets across 146 task categories chosen from other models and papers, as shown here.



Source: Chung et al. 2022, "Scaling Instruction-Finetuned Language Models"

▼ SAMSum: A dialogue dataset

One example of a prompt dataset used for summarization tasks in FLAN-T5 is SAMSum. It's part of the muffin collection of tasks and datasets and is used to

train language models to summarize dialogue. SAMSum is a dataset with 16,000 messenger-like conversations with summaries.

Sample prompt training dataset (**samsum**) to fine-tune FLAN-T5 from pretrained T5

Datasets: **samsum** | Tasks: Summarization | Languages: English

dialogue (string)	summary (string)
"Amanda: I baked cookies. Do you want some? Jerry: Sure! Amanda: I'll bring you tomorrow :-)"	"Amanda baked cookies and will bring Jerry some tomorrow."
"Olivia: Who are you voting for in this election? Oliver: Liberals as always. Olivia: Me too!! Oliver: Great"	"Olivia and Olivier are voting for liberals in this election. "
"Tim: Hi, what's up? Kim: Bad mood tbh, I was going to do lots of stuff but ended up procrastinating Tim: What did…	"Kim may try the pomodoro technique recommended by Tim to get more stuff done."

SAMSum: A dialogue dataset. Source: https://huggingface.co/datasets/samsum, https://github.com/google-research/FLAN/blob/2c79a31/flan/v2/templates.py#L3285

Three examples are shown here: the dialogue on the left and the summaries on the right. Linguists crafted the dialogues and summaries to generate a high-quality training dataset for language models. The linguists were asked to create conversations similar to those they would write daily, reflecting the proportion of topics of their real-life messenger conversations. Language experts then created short summaries of those conversations that included important information and the names of the people in the dialogue.

Sample FLAN-T5 prompt templates

```
"samsum": [
       ("{dialogue}\n\Briefly summarize that dialogue.", "{summary}"),
       ("Here is a dialogue:\n{dialogue}\n\nWrite a short summary!",
        "{summary}"),
       ("Dialogue:\n{dialogue}\n\nWhat is a summary of this dialogue?",
        "{summary}"),
       ("{dialogue}\n\nWhat was that dialogue about, in two sentences or less?",
        "{summary}"),
       ("Here is a dialogue:\n{dialogue}\n\nWhat were they talking about?",
        "{summary}"),
       ("Dialogue:\n{dialogue}\nWhat were the main points in that "
        "conversation?", "{summary}"),
       ("Dialogue:\n{dialogue}\nWhat was going on in that conversation?",
        "{summary}"),
]
```

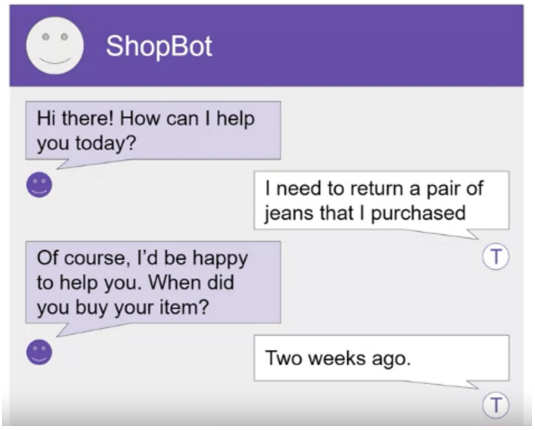
Sample FLAN-T5 prompt templates

This prompt template is designed to work with this SAMSum dialogue summary dataset. The template comprises several instructions that ask the model to do the same thing: summarize a dialogue. For example, briefly summarize that dialogue. What is a summary of this dialogue? What was going on in that conversation? Including different ways of saying the same instruction helps the model generalize and perform better.

In each case, the dialogue from the SAMSum dataset is inserted into the template wherever the dialogue field appears. The summary is used as the label. After applying this template to each row in the SAMSum dataset, it can be used to fine-tune a dialogue summarization task.

▼ Improving FLAN-T5's summarization capabilities

Meanwhile, FLAN-T5 is a great general-use model capable of many tasks. However, tasks for the specific use case still need improvement.

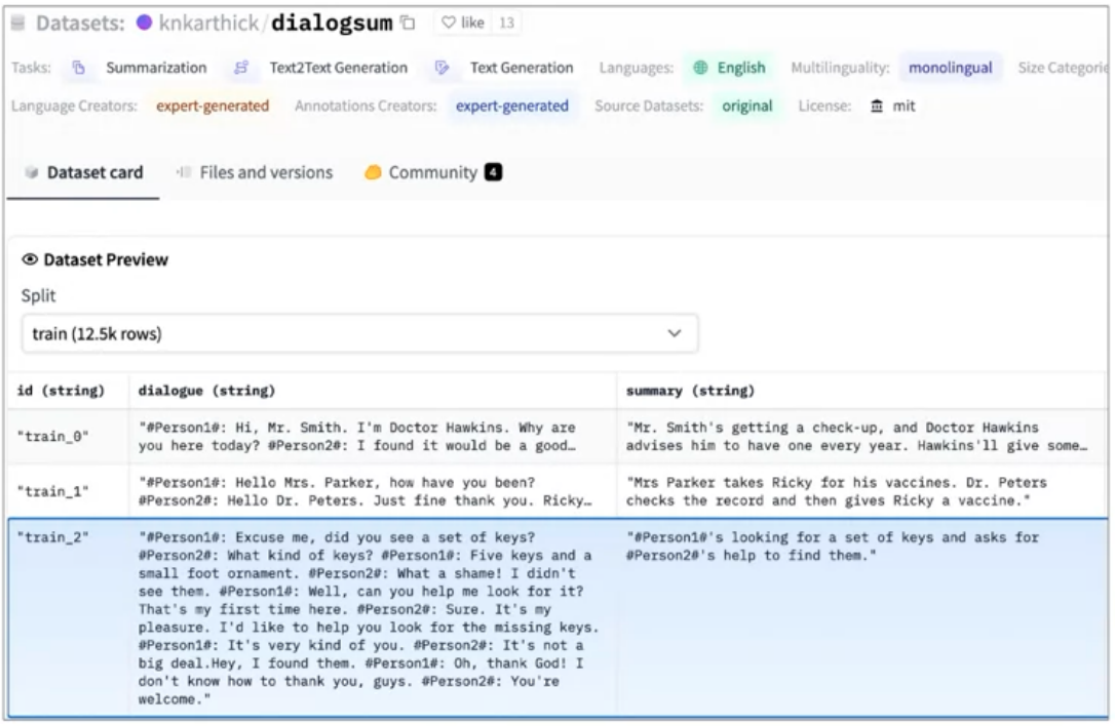


For example, there is an app to support the customer service team and process requests received through a chatbot, like the one shown here. The customer service team needs a summary of every dialogue to identify the key actions the customer requests and determine what actions should be taken in response. The SAMSum dataset gives FLAN-T5 some ability to summarize conversations.

However, the examples in the dataset are mostly conversations between friends about day-to-day activities and don't overlap much with the language structure observed in customer service chats. Using a dialogue dataset that is much closer to the conversations with the bot can perform additional fine-tuning of the FLAN-T5 model.

▼ Improving FLAN-T5's summarization capabilities

There is an additional domain-specific summarization dataset called dialogsum to improve FLAN-T5's ability to summarize support chat conversations. This dataset consists of over 13,000 support chat dialogues and summaries. The dialogue in some datasets is not part of the FLAN-T5 training data, so the model has not seen these conversations before.



Further fine-tune FLAN-T5 with a domain-specific instruction dataset (dialogsum)

Let's look at the example from the dialogue sum and discuss how a further round of fine-tuning can improve the model. This support chat is typical of the examples in the dialogue sum dataset. The conversation is between a customer and a staff member at a hotel check-in desk. A template was applied to the chat so that the instructions for summarizing the conversation were included at the start of the text.

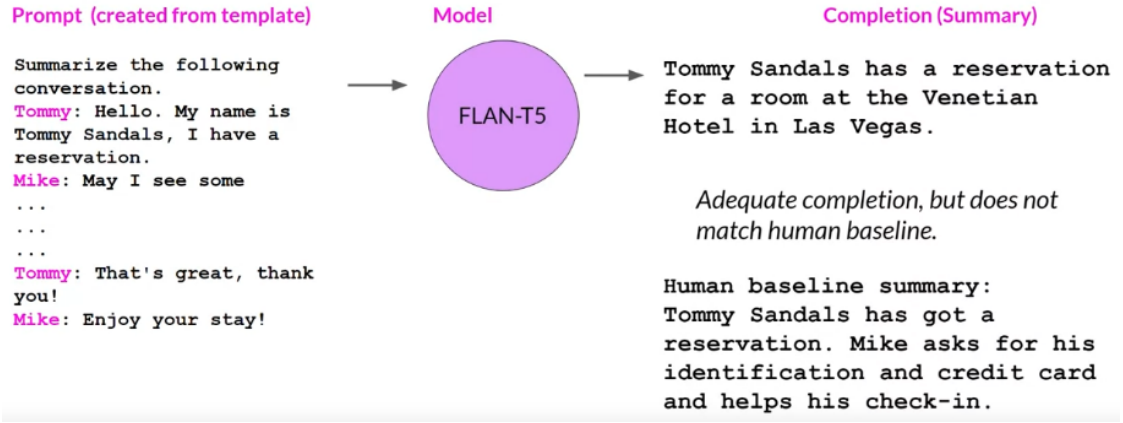
Prompt (created from template)

```
Summarize the following conversation.
Tommy: Hello. My name is Tommy Sandals, I have a reservation.
Mike: May I see some identification, sir, please?
Tommy: Sure. Here you go.
Mike: Thank you so much. Have you got a credit card, Mr.
Sandals?
Tommy: I sure do.
Mike: Thank you, sir. You'll be in room 507, nonsmoking,
queen bed.
Tommy: That's great, thank you!
Mike: Enjoy your stay!
```

Example support-dialog summarization, Source: https://huggingface.co/datasets/knkarthick/dialogsum/viewer/knkarthick--dialogsum/

Now, let's look at how FLAN-T5 responds to this prompt before doing additional fine-tuning. Note that the prompt is now condensed on the left to give more room to examine the model's completion. Here is the model's

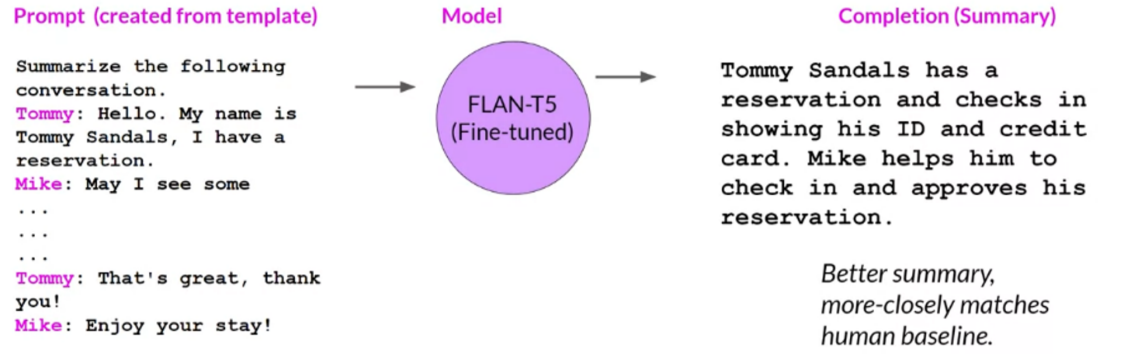
response to the instruction. The model does as it's able to identify that the conversation was about a reservation for Tommy.



Summary before fine-tuning FLAN-T5 with the dataset

However, it does not do as well as the human-generated baseline summary, which includes important information, such as Mike asking for information to facilitate check-in and the model's completion. It also invented information not included in the original conversation—specifically, the name of the hotel and the city in which it was located.

After fine-tuning the dialogue with some datasets, let's look at how the model works. This is closer to the human-produced summary. There is no fabricated information, and the summary includes all the important details, including the names of both people participating in the conversation.



Summary: After fine-tuning FLAN-T with our dataset

This example uses the public dialogue and some datasets to demonstrate fine-tuning on custom data. The most important part of fine-tuning is using the company's internal data. For example, there are support chat conversations from the customer support application. This will help the model learn how the company likes summarising conversations and what is most useful to the customer service colleagues.

▼ Reading: Scaling instruct models

This paper introduces FLAN (Fine-tuned LAnguage Net), an instruction fine-tuning method, and presents the results of its application. The study demonstrates that by fine-tuning the 540B PaLM model on 1836 tasks while incorporating Chain-of-Thought Reasoning data, FLAN improves generalization, human usability, and zero-shot reasoning over the base model. The paper also provides detailed information on how each aspect was evaluated.

The image from the lecture slides illustrates the fine-tuning tasks and datasets employed in training FLAN. The task selection expands on previous works by incorporating dialogue and program synthesis tasks from Muffin and integrating them with new Chain of Thought Reasoning tasks. It also includes subsets of other task collections, such as T0 and Natural Instructions v2. Some tasks were held out during training and later used to evaluate the model's performance on unseen tasks.

▼ Model evaluation

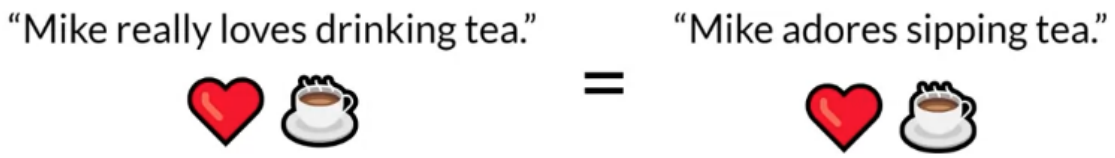
Some statements, like the model demonstrated good performance on this task or this fine-tuned model, showed a large improvement in performance over the base model. What do statements like this mean? How do we formalize the improvement in the performance of the fine-tuned model over the pre-trained model? Let's explore several metrics used by developers of large language models that can be used to assess the performance of the models and compare them to other models out in the world.

▼ LLM Evaluation - Challenges

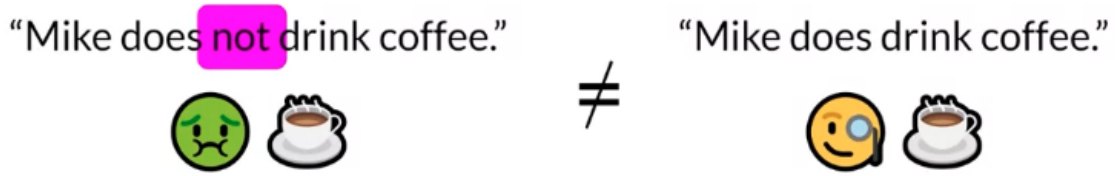
In traditional machine learning, we can assess a model's performance on training and validation data sets where the output is already known. We can calculate simple metrics such as accuracy, which states the fraction of all correct predictions because the models are deterministic; however, with large language models where the output is non-deterministic, the language-based evaluation is much more challenging.

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

Take, for example, the sentence, Mike really loves drinking tea. This is quite similar to Mike adores sipping tea. But how do you measure the similarity?



Let's look at these other two sentences. Mike does not drink coffee, and Mike does drink coffee.



There is only one-word difference between these two sentences. However, the meaning is completely different. As humans with squishy organic brains, we can easily see the similarities and differences between them. training a model on millions of sentences needs an automated, structured way to make measurements.

▼ LLM Evaluation - Metrics

ROUGE and BLEU are two widely used evaluation metrics for different tasks. ROUGE, or recall-oriented understudy for jesting evaluation, is primarily employed to assess the quality of automatically generated summaries by comparing them to human-generated reference summaries. On the other hand, BLEU, or bilingual evaluation understudy, is an algorithm designed to evaluate the quality of machine-translated text by comparing it to human-generated translations.

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

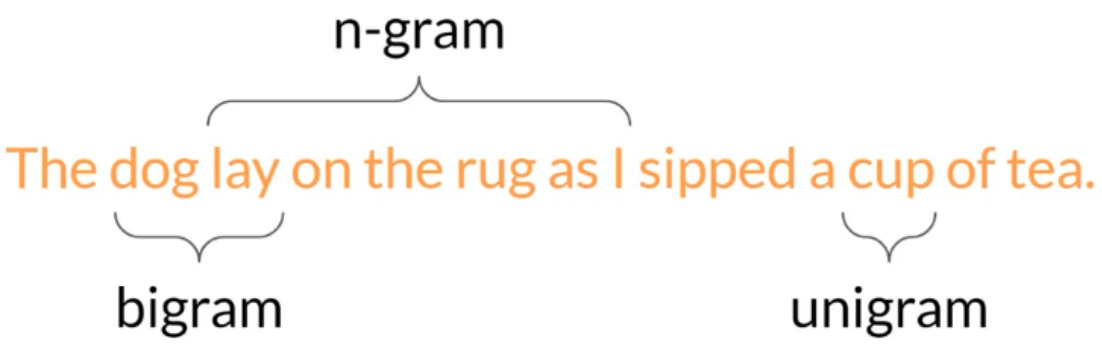
- Used for text translation
- Compares to human-generated translations

The word BLEU is French for blue. People might call this blue.

▼ LLM Evaluation - Metrics - Terminology

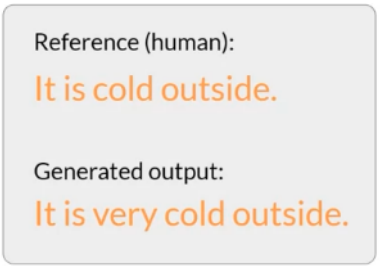
In the anatomy of language:

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



It's pretty straightforward stuff.

▼ LLM Evaluation - Metrics - ROUGE-1



First, let's look at the ROUGE-1 metric. To do so, let's look at a human-generated reference sentence. It is cold outside, and the generated output is very cold outside. You can perform simple metric calculations similar to other machine-learning tasks using recall, precision, and F1.

The recall metric measures the number of words or unigrams matched between the reference and the generated output divided by the number of words or unigrams in the reference. In this case, it gets a perfect score of one, as all the generated words match the reference words. Precision measures the unigram matches divided by the output size.

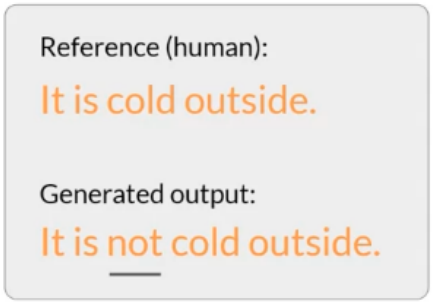
$$\text{ROUGE-1 Recall} = \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$$

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

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

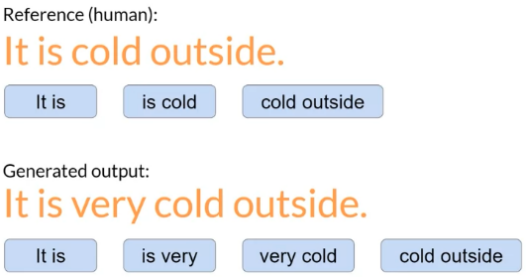
The F1 score is the harmonic mean of both of these values.

These are very basic metrics that only focus on individual words, hence the one in the name, and don't consider the ordering of the words. It can be deceptive. Generating sentences that score well but would be subjectively poor is easily possible.

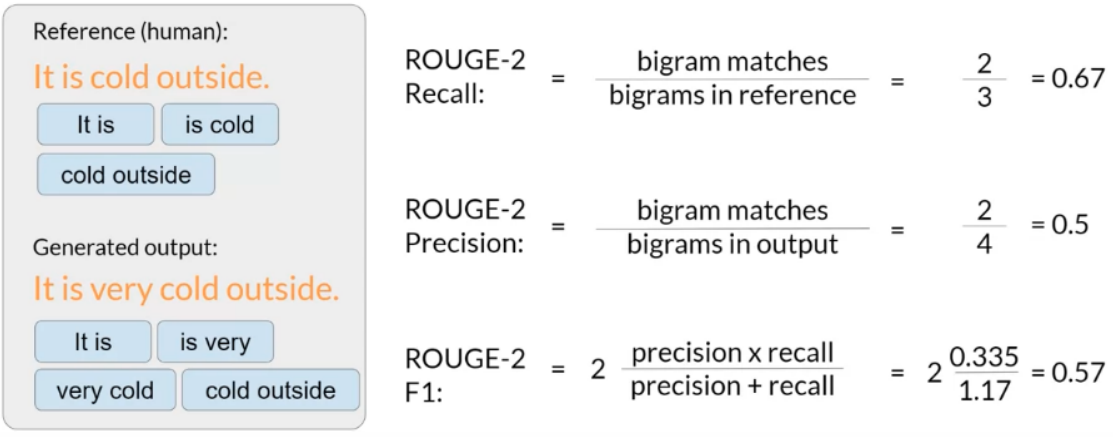


Now, the sentence generated by the model differed by just one word “Not”. so it is not cold outside. The scores would be the same.

▼ LLM Evaluation - Metrics - ROUGE-2

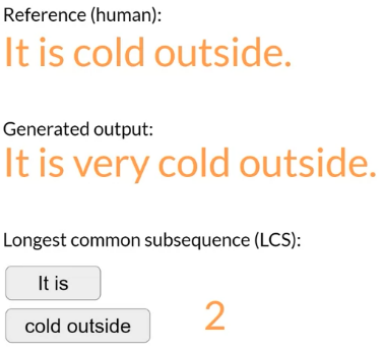


Taking into account bigrams or collections of two words at a time from the reference and generated sentence can give a slightly better score. By working with pairs of words, we're acknowledging the ordering of the words in the sentence.



In this case, bigrams are used to calculate a ROUGE-2. Using bigram matches instead of individual words can calculate the recall, precision, and F1 score. Notice that the scores are lower than the ROUGE-1 scores. With longer sentences, there's a greater chance that bigrams don't match, and the scores may be even lower.

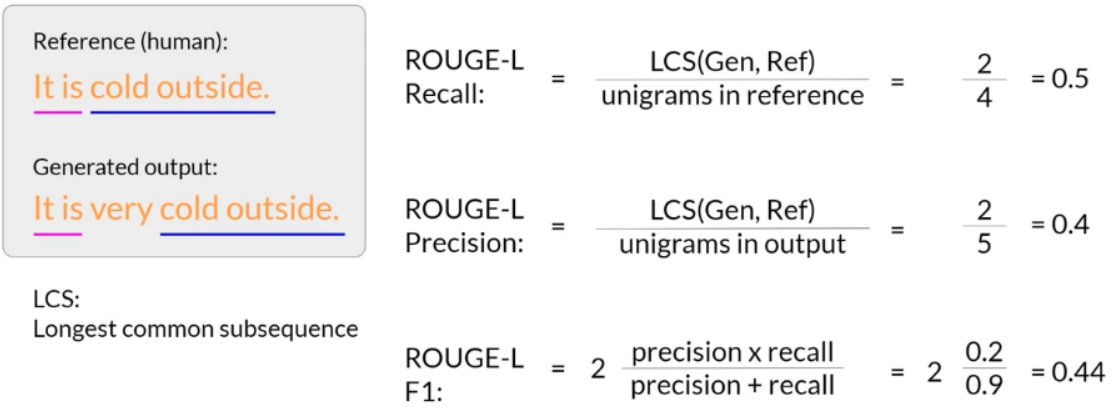
▼ LLM Evaluation - Metrics - ROUGE-L



Rather than continue with ROUGE numbers growing bigger to n-grams of three or fours, let's take a different approach. Instead, look for the longest common subsequence in the generated and reference outputs.

In this case, the longest matching sub-sequences are, it is and cold outside, each with a length of two.

Now, we can use the LCS value to calculate the recall precision and F1 score, where the numerator in both the recall and precision calculations is the length of the longest common subsequence, in this case, two.

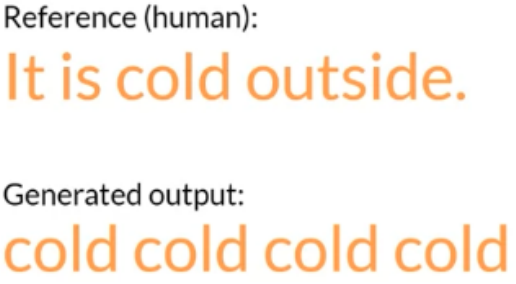


Collectively, these three quantities are known as the Rouge-L score. As with all Rouge scores, we need to contextualise the values. We can only use the scores to compare the capabilities of models if they were determined for the same task, such as summarization.

▼ LLM Evaluation - Metrics - ROUGE hacking

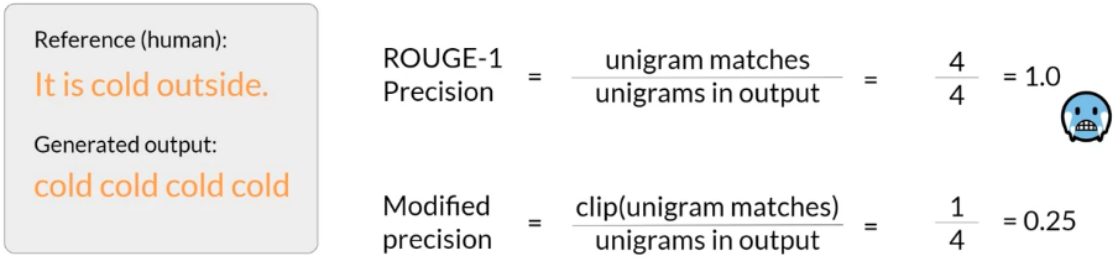
Rouge scores for different tasks are not comparable. A problem with simple rouge scores is that a bad completion can result in a good score.

For example, this generated output: cold, cold, cold, cold. This generated output contains one of the words from the reference sentence, so it will score quite highly, even though the same word is repeated multiple times. The Rouge-1 precision score will be perfect.

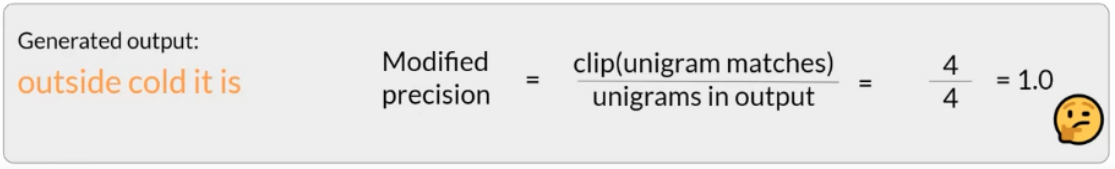


▼ LLM Evaluation - Metrics - ROUGE clipping

One way to counter this issue is to use a clipping function to limit the number of unigram matches to the maximum count for that unigram within the reference. In this case, there is one appearance of cold and the reference, and so modifying the precision with a clip on the unigram matches results in a dramatically reduced score.



However, it will still be challenged if the words they generate are all present but in a different order.



For example, with this generated sentence, outside cold it is. This sentence was called perfectly even with the modified precision with the clipping function, as all the words and the generated output are present in the reference.

Using a different rouge score can help when experimenting with an n-gram size, but the score that will calculate the most usefully will depend on the sentence, the sentence size, and the use case.

Many language model libraries, such as Hugging Face in the first week's lab, include implementations of rouge scores that can easily evaluate the model's output. The rouge score compares the model's performance before and after fine-tuning in this week's lab.

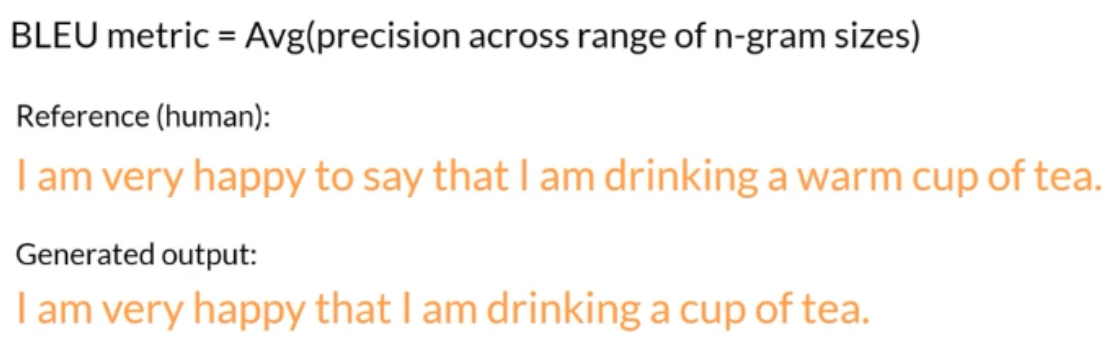
▼ LLM Evaluation - Metrics - BLEU

The other score that can be useful in evaluating the model's performance is the BLEU score, which stands for bilingual evaluation under study.

The BLEU score is useful for evaluating the quality of machine-translated text. The score itself is calculated using the average precision over multiple n-gram

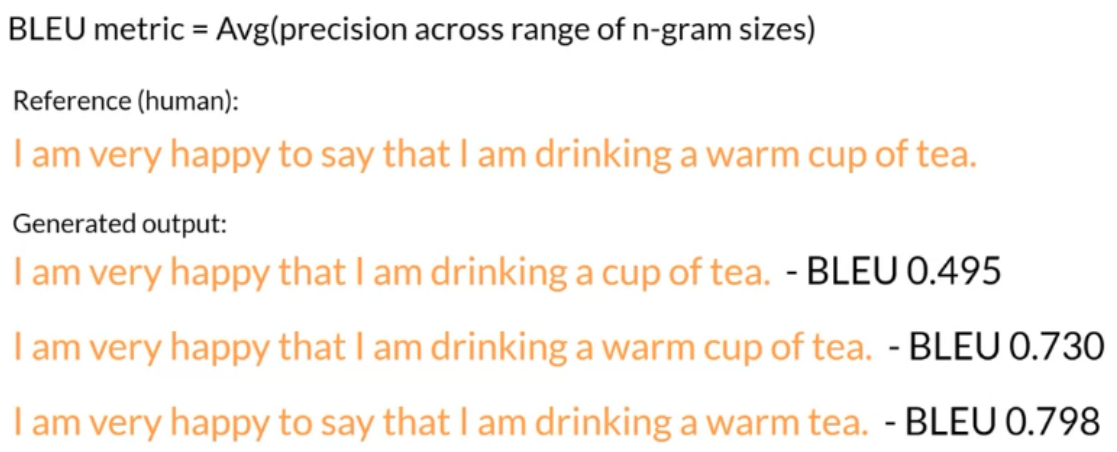
sizes. Like the Rouge-1 score we looked at before, we calculated for a range of n-gram sizes and then averaged.

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, average precision across a range of n-gram sizes.



If we were to calculate this by hand, we would carry out multiple calculations and then average all the results to find the BLEU score. For this example, look at a longer sentence to better understand the score's value. The reference human-provided sentence is, “I am very happy to say that I am drinking a warm cup of tea”.

This is the result of BLEU using a standard library. Calculating the BLEU score is easy with pre-written libraries from providers like Hugging Face, it needs to be done for each candidate sentence.



The first candidate is “I am very happy that I am drinking a cup of tea”. The BLEU score is 0.495. As we get closer and closer to the original sentence, we get a score that is closer and closer to one.

Both rouge and BLEU are quite simple metrics and relatively inexpensive to calculate. They can be used for simple reference as we iterate over the models,

but we shouldn't use them alone to report the final evaluation of a large language model.

Use rouge for diagnostic evaluation of summarization tasks and BLEU for translation tasks. However, for the overall evaluation of the model's performance, we will need to look at one of the evaluation benchmarks developed by researchers.

▼ Benchmarks

LLMs are complex, and simple evaluation metrics like the rouge and blur scores can only tell about the model's capabilities. To measure and compare LLMs more holistically, we can use preexisting datasets and associated benchmarks established by LLM researchers specifically for this purpose.

Selecting the right evaluation dataset is vital to accurately assessing an LLM's performance and understanding its true capabilities. It is useful to select datasets that isolate specific model skills, like reasoning or common sense knowledge, and those that focus on potential risks, such as disinformation or copyright infringement.

An important issue is whether the model has seen the evaluation data during training. Evaluating its performance on data it hasn't seen before will give us a more accurate and useful sense of the model's capabilities.

Benchmarks, such as GLUE, SuperGLUE, or Helm, cover various tasks and scenarios. They do this by designing or collecting datasets that test specific aspects of an LLM.

▼ Evaluation benchmarks - GLUE

Corpus	|Train|	|Test|	Task	Metrics	Domain
			Single-Sentence Tasks		
CoLA	8.5k	**1k**	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
			Similarity and Paraphrase Tasks		
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	**391k**	paraphrase	acc./F1	social QA questions
			Inference Tasks		
MNLI	393k	**20k**	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	**146**	coreference/NLI	acc.	fiction books

The tasks included in the SuperGLUE benchmark. Source: Wang et al. 2018, "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding"

GLUE, or General Language Understanding Evaluation, was introduced in 2018. GLUE is a collection of natural language tasks, such as sentiment analysis and question-answering. GLUE was created to encourage the development of models that can generalize across multiple tasks and be used to measure and compare the model's performance.

▼ Evaluation benchmarks - SuperGLUE

As a successor to GLUE, SuperGLUE was introduced in 2019 to address the limitations of its predecessor. It consists of a series of tasks, some of which are not included in GLUE, and some are more challenging versions of the same tasks. SuperGLUE includes tasks such as multi-sentence reasoning and reading comprehension.

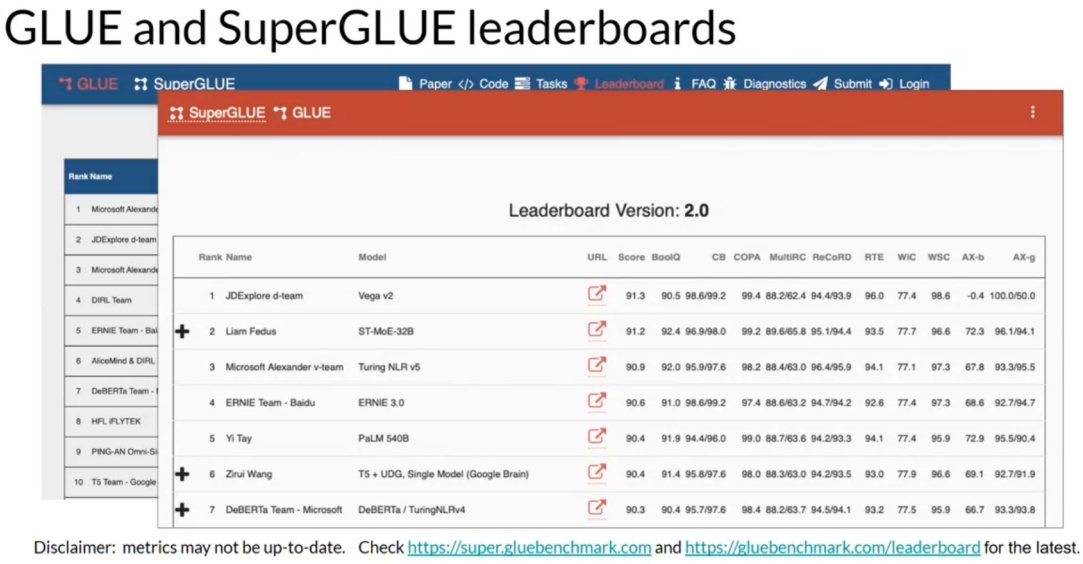
The tasks included in SuperGLUE benchmark:

Corpus	|Train|	|Dev|	|Test|	Task	Metrics	Text Sources
BoolQ	9427	3270	3245	QA	acc.	Google queries, Wikipedia
CB	250	57	250	NLI	acc./F1	various
COPA	400	100	500	QA	acc.	blogs, photography encyclopedia
MultiRC	5100	953	1800	QA	$F1_a$/EM	various
ReCoRD	101k	10k	10k	QA	F1/EM	news (CNN, Daily Mail)
RTE	2500	278	300	NLI	acc.	news, Wikipedia
WiC	6000	638	1400	WSD	acc.	WordNet, VerbNet, Wiktionary
WSC	554	104	146	coref.	acc.	fiction books

Source: Wang et al. 2019, "SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems"

The tasks included in the SuperGLUE benchmark. Source: Wang et al. 2019, "SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems"

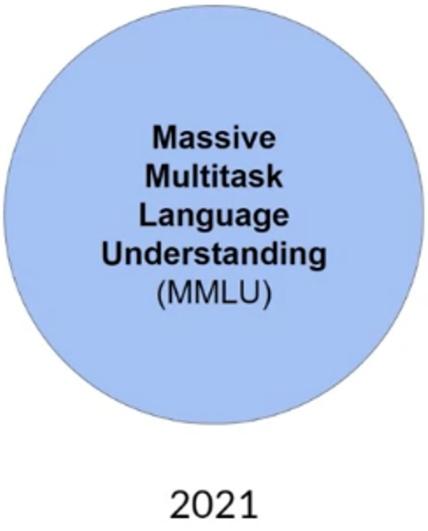
The GLUE and SuperGLUE benchmarks have leaderboards that can compare and contrast evaluated models.



GLUE and SuperGLUE leaderboards. Disclaimer: metrics may not be up-to-date. Check https://super.gluebenchmark.com and https://gluebenchmark.com/leaderboard for the latest.

The results page is another great resource for tracking the progress of LLMs. As models get larger, their performance against benchmarks such as SuperGLUE starts to match human ability on specific tasks. That's to say that models are able to perform as well as humans on the benchmark tests, but subjectively, we can see that they're not performing at a human level at tasks in general.

▼ Evaluation benchmarks - MMLU

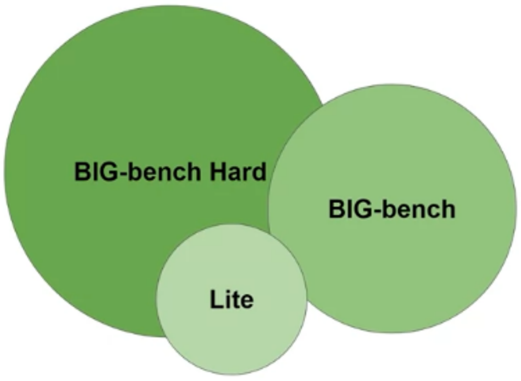


Benchmarks for massive models. Source: Hendricks, 2021. "Measuring Massive Multitask Language Understanding"

There is essentially an arms race between LLMs' emergent properties and the benchmarks that aim to measure them. Here are a couple of recent benchmarks that are pushing LLMs further.

Massive Multitask Language Understanding, or MMLU, is designed for modern LLMs. To perform well, models must possess extensive world knowledge and problem-solving ability. Models are tested on elementary mathematics, US history, computer science, and law.

▼ Evaluation benchmarks - BIG-bench



2022

Source: Suzgun et al. 2022. "Challenging BIG-Bench tasks and whether chain-of-thought can solve them."

In other words, tasks that extend far beyond basic language understanding. BIG-bench currently consists of 204 tasks, covering linguistics, childhood development, math, common sense reasoning, biology, physics, social bias, software development, and more.

BIG-bench comes in three different sizes, and part of the reason for this is to keep costs achievable, as running these large benchmarks can incur high inference costs.

▼ Evaluation benchmarks - HELM

A final benchmark is the Holistic Evaluation of Language Models, or HELM. The HELM framework aims to improve model transparency and offer guidance on which models perform well for specific tasks.

HELM takes a multi-metric approach, measuring seven metrics across 16 core scenarios, exposing trade-offs between models and metrics. One important feature of HELM is that it assesses metrics beyond basic accuracy measures, like the precision of the F1 score.

The benchmark also includes metrics for fairness, bias, and toxicity, which are becoming increasingly important to assess as LLMs become more capable of human-like language generation and exhibiting potentially harmful behaviour.

HELM is a living benchmark that aims to continuously evolve by adding new scenarios, metrics, and models. On the results page, we can browse the LLMs that have been evaluated and review scores pertinent to the project's needs.