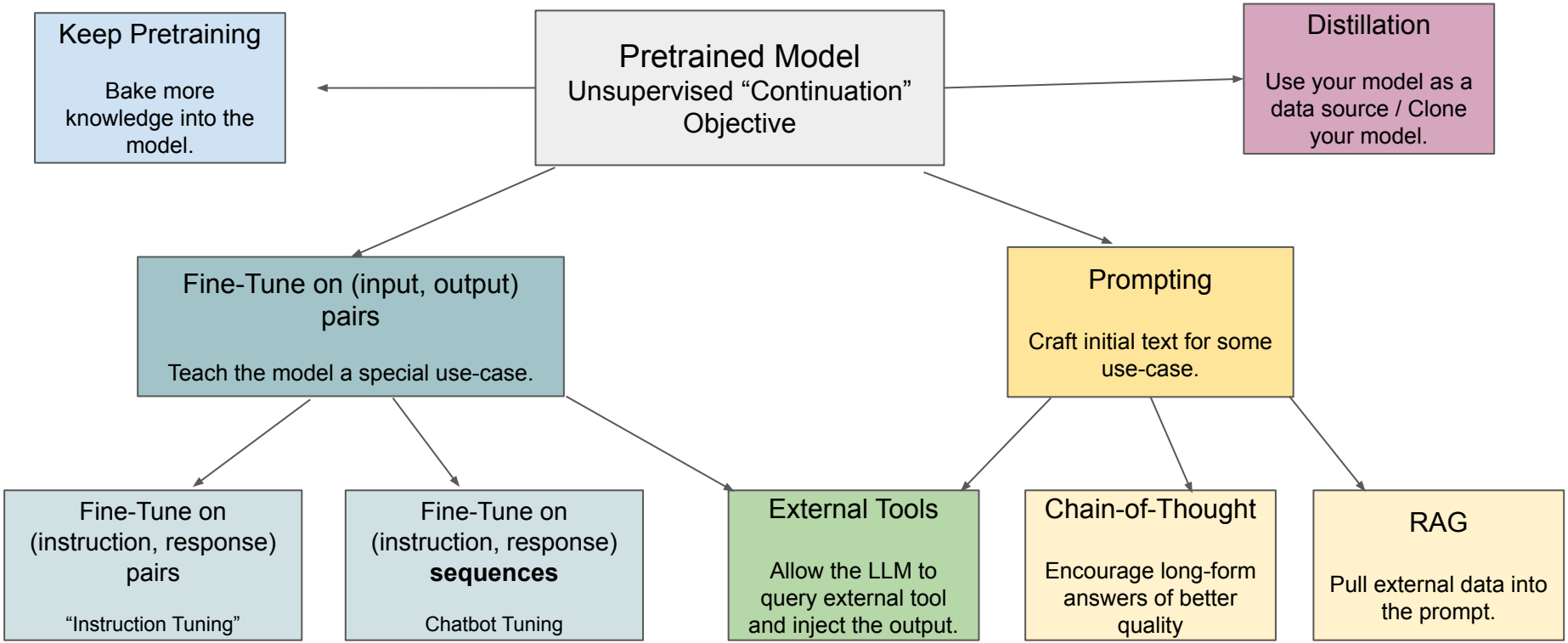
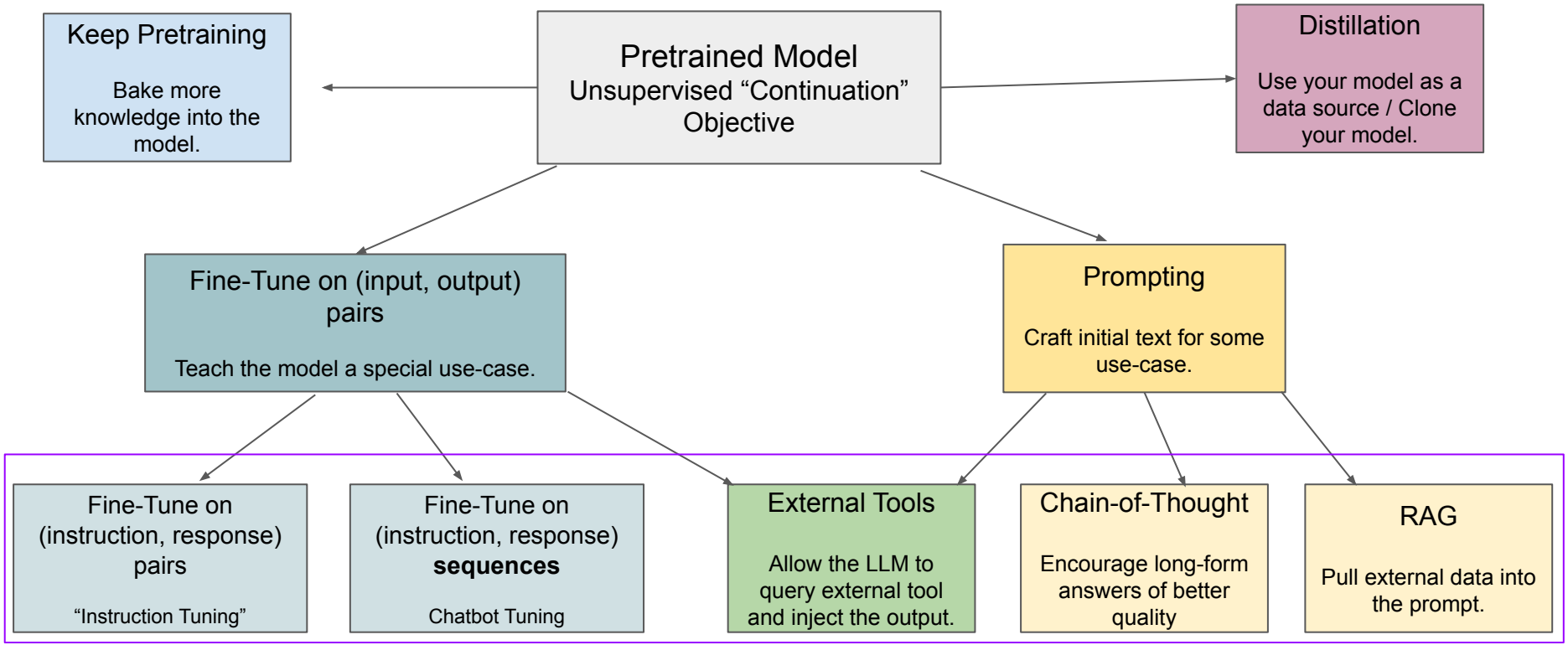
# Model Specialization

EN.705.743: ChatGPT from Scratch

# Lecture Outline: Specializing a Pretrained Model



# Lecture Outline: Specializing a Pretrained Model



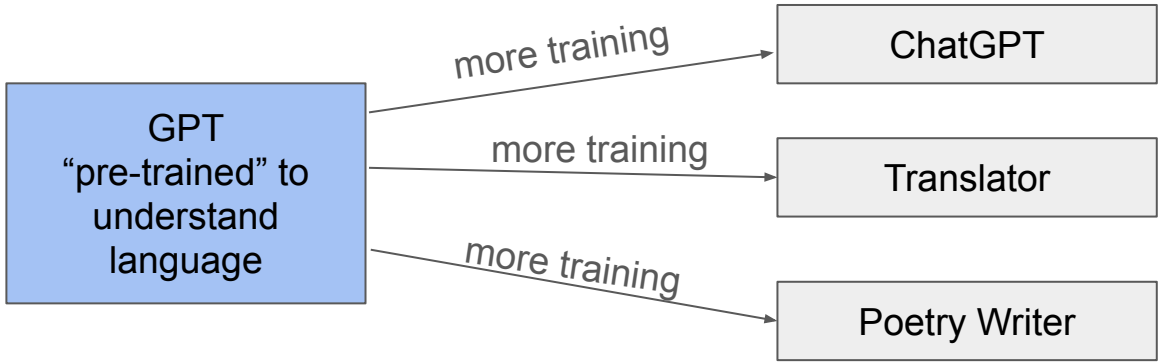

LLM Agent ~= Some combination of these

# Pros and Cons of Continuation Models

# Continuation

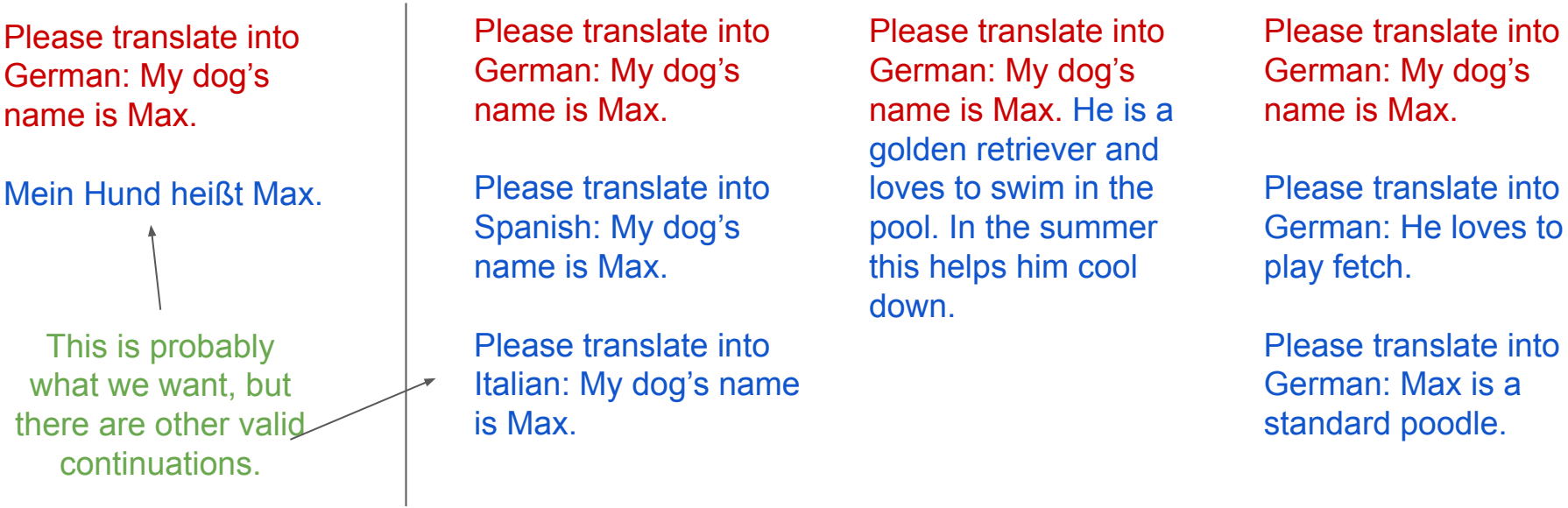
Recall that the idea behind pre-trained models is to create a common starting point for a variety of language-based tasks.

To train such a model, we give it the task of continuing any input text by one token. We can then “roll this out” to generate continuations of any starting text.



# Continuation

For better or worse, the task of “continuing text” is open-ended. There are often many ways that text could be plausibly continued:



# Prompting Examples

As we saw in Lecture 7, one way around this is to make the desired continuation more “obvious”. This is a bit cumbersome but can work well.

Please translate into German: Bananas are yellow.

Bananen sind gelb.

Please translate into German: Today it is sunny.

Heute ist es sonnig.

Please translate into German: My dog’s name is Max.

Extra input to encourage a specific behavior.

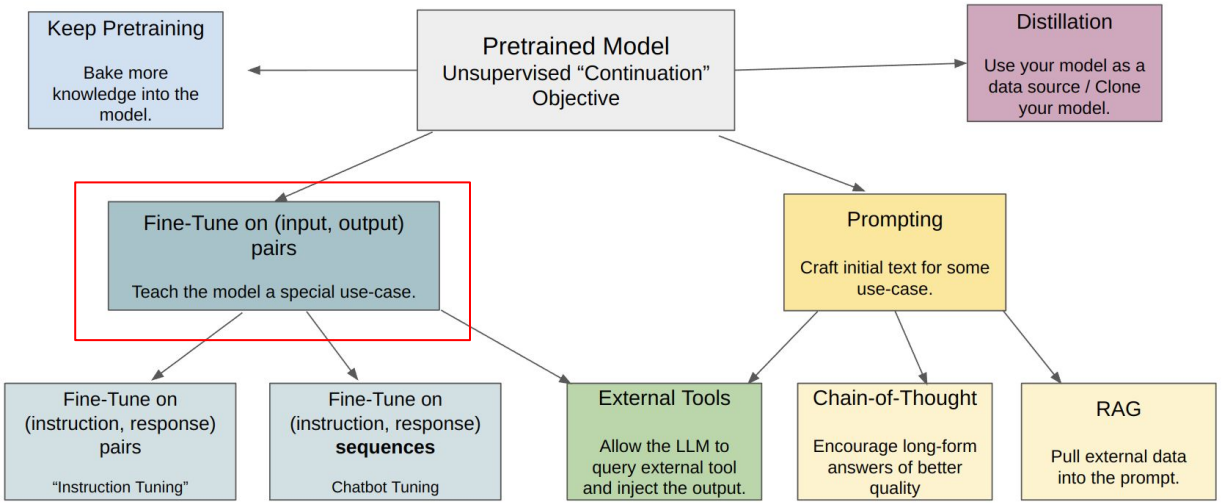
# Prompting Headaches

Although this can work extremely well for ad-hoc problems, there are a few downsides to this approach:

1) Expensive inference: We are adding tokens that do not directly relate to our answer.
2) Finite sequence length: What if each example is 500 tokens? How much room will remain our real query?
3) What if we have lots of examples, such as a small dataset that we want our model to understand?

Fundamentally these all come down to the same thing: Our model was trained to do continuation, but we are trying to force it into a “supervised” mold with the expectations of (input, output) behavior.

# Fine-Tuning

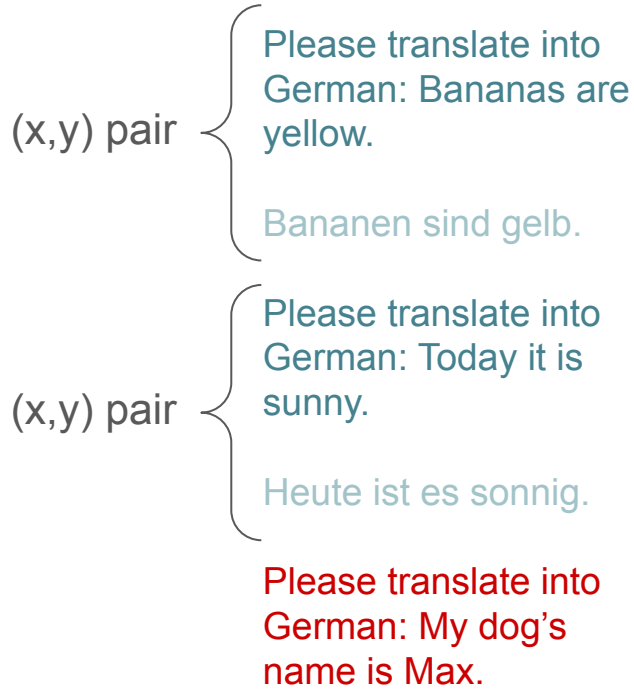

Keep Pretraining
Bake more knowledge into the model.
Pretrained Model
Unsupervised "Continuation" Objective
Distillation
Use your model as a data source / Clone your model.
Fine-Tune on (input, output) pairs
Teach the model a special use-case.
Prompting
Craft initial text for some use-case.
Fine-Tune on (instruction, response) pairs
"Instruction Tuning"
Fine-Tune on (instruction, response) sequences
Chatbot Tuning
External Tools
Allow the LLM to query external tool and inject the output.
Chain-of-Thought
Encourage long-form answers of better quality
RAG
Pull external data into the prompt.

# (Supervised) Fine-Tuning

Supervised Fine-Tuning (SFT) is an approach to convert our general, self-supervised base model into a narrow, supervised model that has a specific behavior.

Starting with our pre-trained model, we continue training on specific (x,y) pairs that demonstrate the desired use-case of the model.

These are just like the examples we might put into a prompt, except we are now training them into the model instead.




We have already seen what these (x,y) pairs look like- we use them in prompt engineering.

# SFT

Implementing SFT is similar to pre-training. There are two major differences:

(1) the format of the samples are usually standardized or at least more constrained than the pre-training data, and

(2) we do not need to learn how to “predict” the original input.

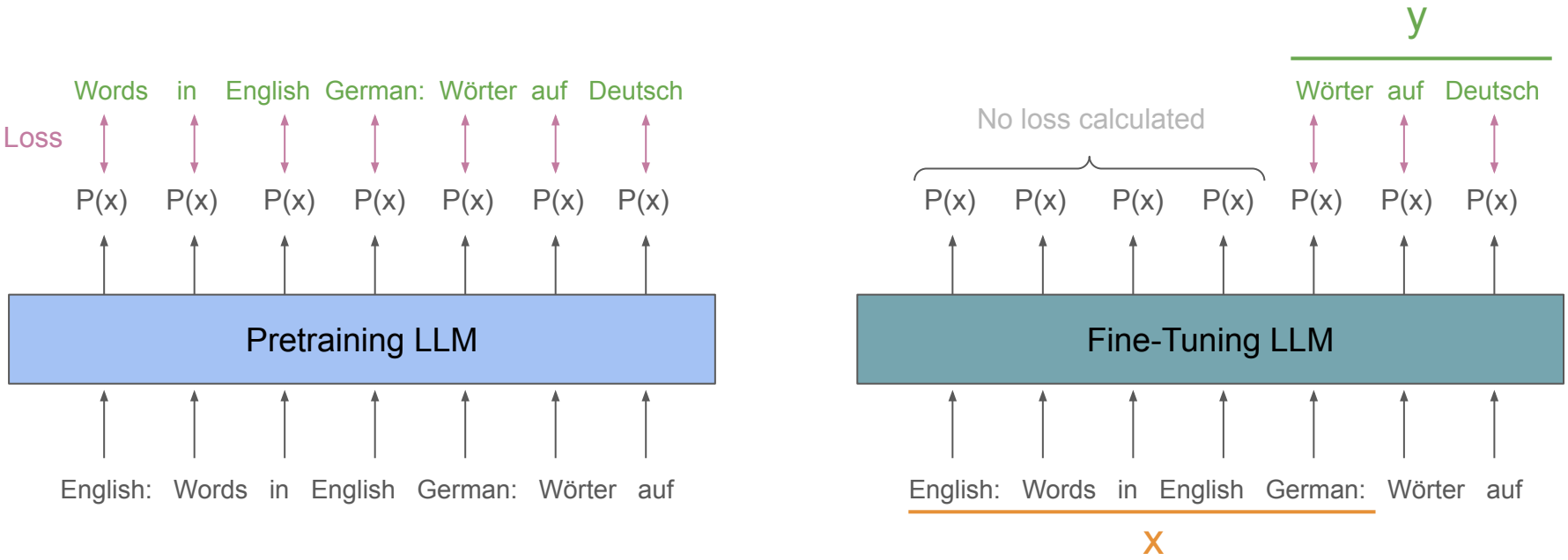
We might have an English-German translation dataset with samples like this:

English: <English sentence> German: <German sentence>

x y

# SFT Training

When we train our model, we do not compute loss for the input tokens:



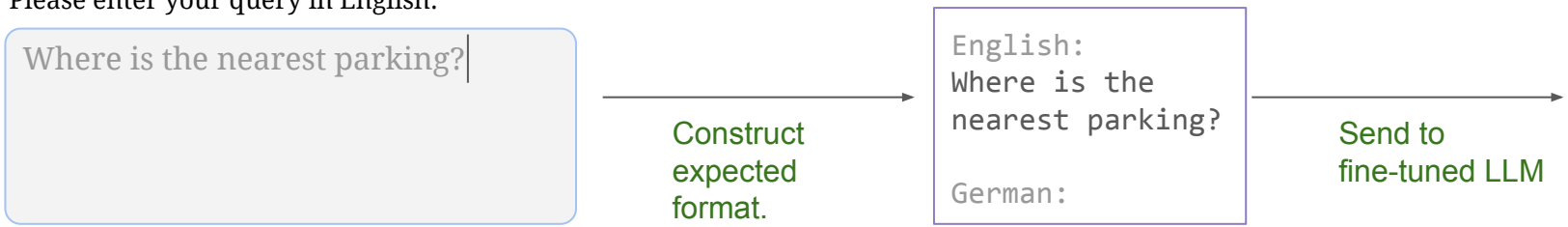
# SFT Requirements

To use SFT, you will need to:

- Have enough data to actually train the model (hundreds of examples or more)
- Have a similar amount of compute to pre-training, probably within an order of magnitude
  - We are still backpropagating through a huge LLM!
  - The only difference is training time- we have fewer samples so we do not need to parallelize purely for speed.
- Ensure that the use of the model will conform to the training format.
  - One we start SFT, this format is all that the model knows. Other formats are out-of-distribution.

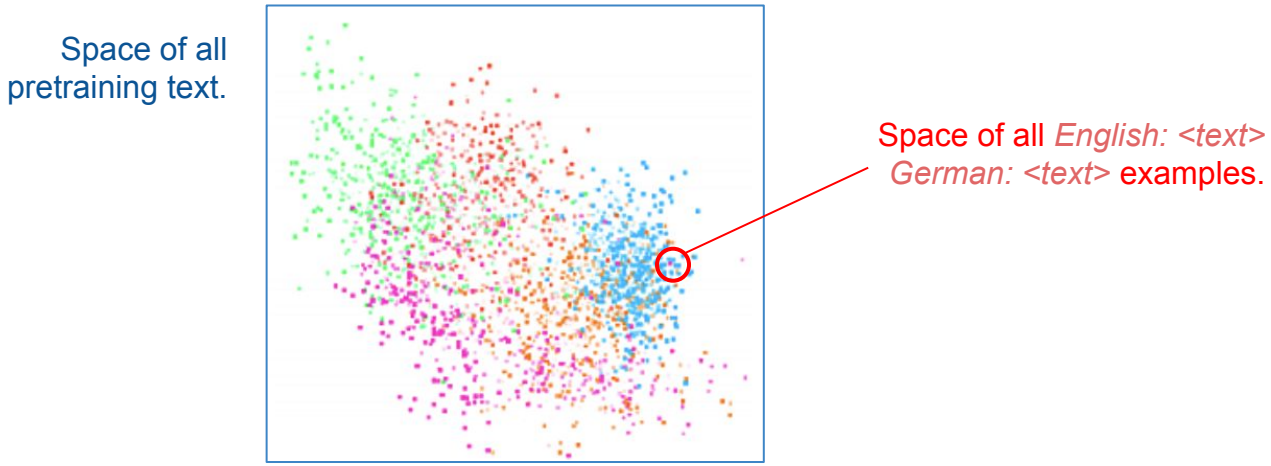
# Conforming to the Format

Many applications will adhere to the SFT data format by simply hiding it from the user. If we made a translation app powered by our LLM, we would have wrappers that inject the user's query into our format:

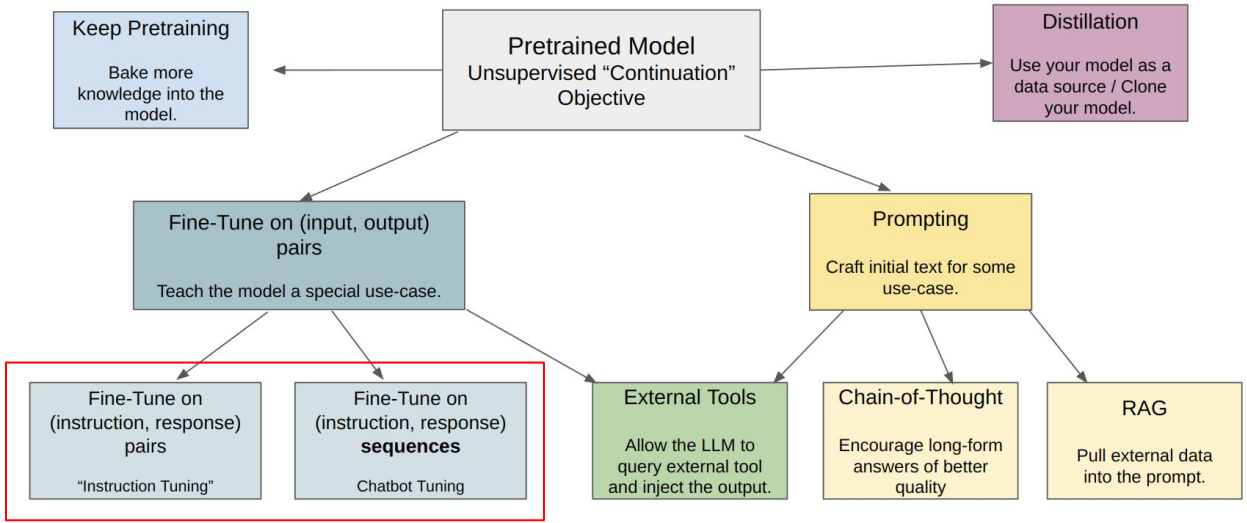


# SFT

One perspective: this is more constrained than the pre-training text, which can generally take any form. Since this is a much more specific use-case, we hope we can specialize our model to this task (and we are okay with forgetting other tasks).



# Instruction-Tuning



# Fine-tuning for General Use

Let's look at the motivating example again- it's pretty clear which one is desired. Even if all of these are valid continuations, only one is a "response" to the user's query.

Please translate into German: My dog's name is Max.

Mein Hund heißt Max.

This one actually responds to the user.

Please translate into German: My dog's name is Max.

Please translate into Spanish: My dog's name is Max.

Please translate into Italian: My dog's name is Max.

Please translate into German: My dog's name is Max. He is a golden retriever and loves to swim in the pool. In the summer this helps him cool down.

Please translate into German: My dog's name is Max.

Please translate into German: He loves to play fetch.

Please translate into German: Max is a standard poodle.

# Breaking out of the continuation mold

For a generally useful model, what we really want is to avoid outputs that merely “continue” and more towards the outputs that are more a “response” or “answer”.

We can craft a dataset that follows the (instruction, response) format, and use the same recipe as fine-tuning. If we keep our topics very wide, then we can ideally show the model how to behave without sacrificing its general abilities. This is called **instruction tuning**, sometimes abbreviated IFT or IT. Typically these datasets have order 100k examples.

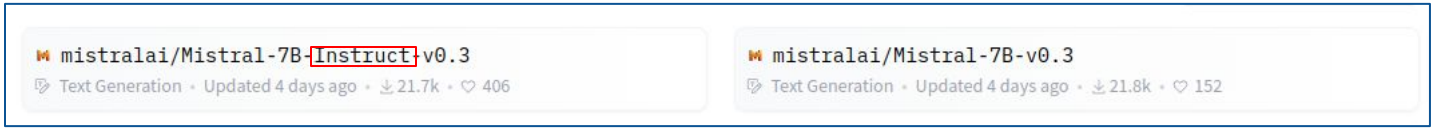
| | |
|---|---|
| Write a short story about a day in the life of a software engineer. | John was a software engineer and had been coding for the past 5 years. Every day he… |
| What's the difference between a virus and a worm? | The main difference between a virus and a worm is that a virus requires user… |
| Explain the concept of the blockchain. | The blockchain is a distributed ledger technology that is used to store and record… |
| What is the contraction of "they are"? | The contraction of "they are" is "they're". |
| Create a list of items for a picnic. | A picnic list should include items such as: sandwiches, chips, fruit, vegetables,… |

Example instruction dataset

# Instruction Models

Models trained with instruction tuning are often more useful to a user than a base model. Many LLMs are now distributed in multiple sizes and also in base/instruct forms.



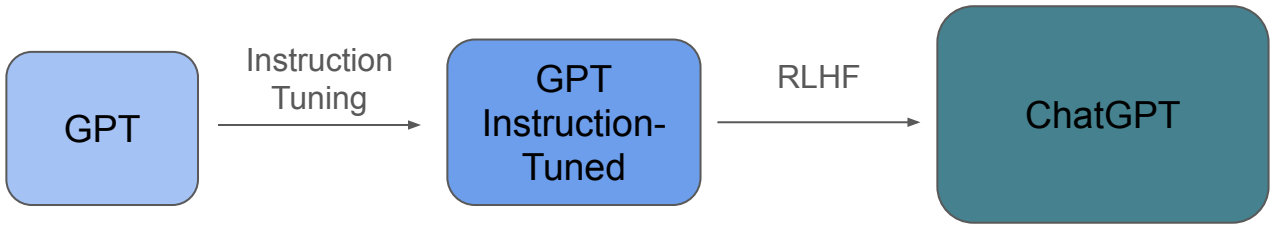



Three open-source models from the last year, distributed as both a base model and an instruction-tuned model.

# Towards ChatGPT

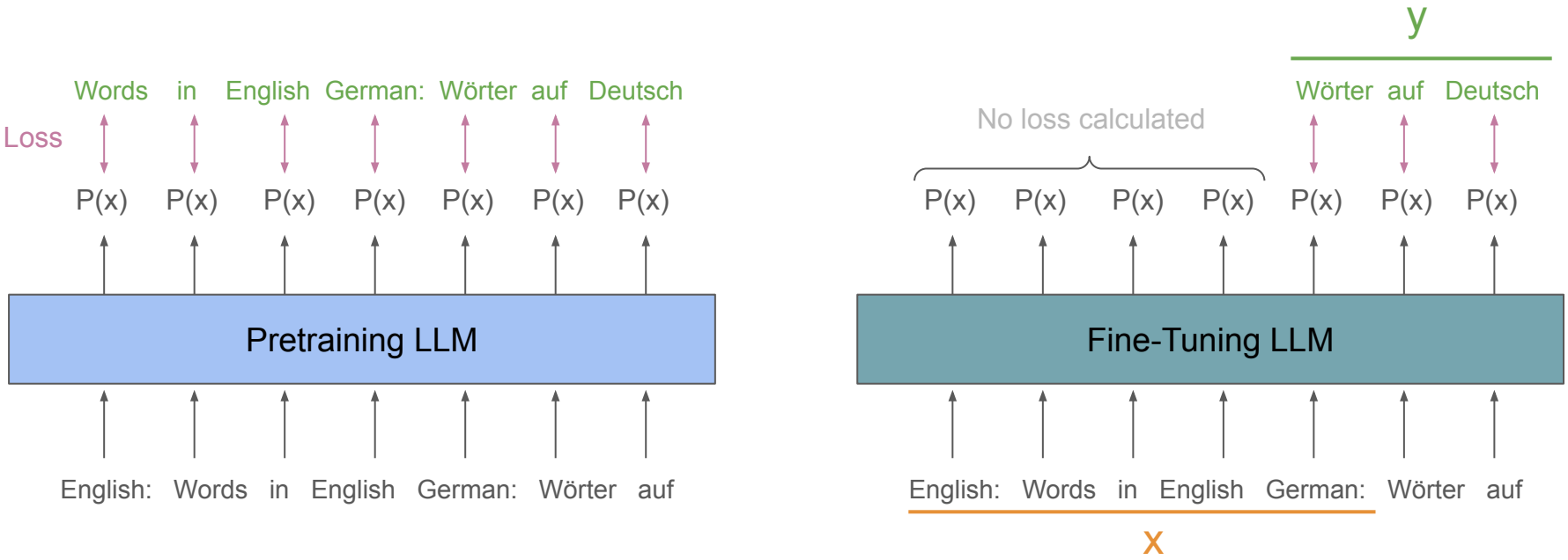
In the beginning of the course we discussed that ChatGPT is really a base model (GPT) with lots of additions. Instruction tuning is one of the key steps towards making a conversational and helpful AI like ChatGPT.

The other major step is RLHF (Lecture 13).



# SFT Training

When we train our model, we do not compute loss for the input tokens:

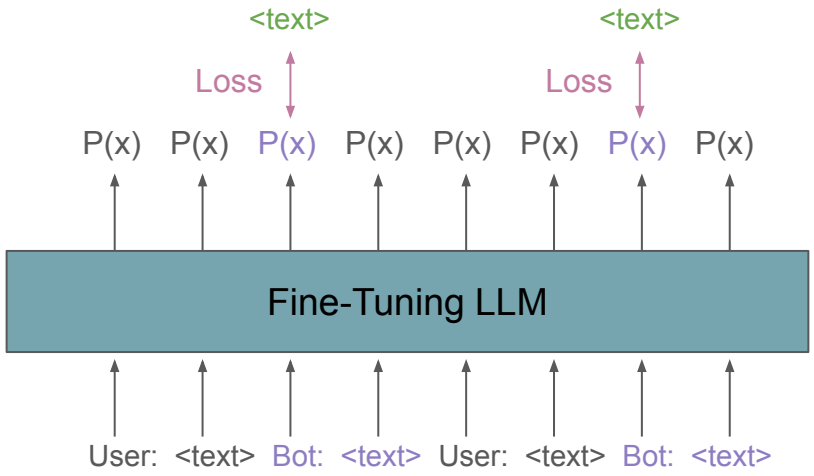


# Instructions vs Chat vs Assistant

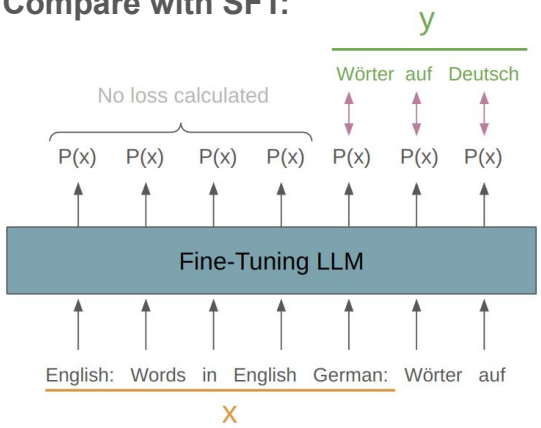
Sometimes instruction models will be called something else, like “chat” or “assistant”. This is blurry and only sometimes means anything.

My definition would be that an “instruction-tuned” model handles a single request, whereas a “chat” model is trained on several back-and-forths.

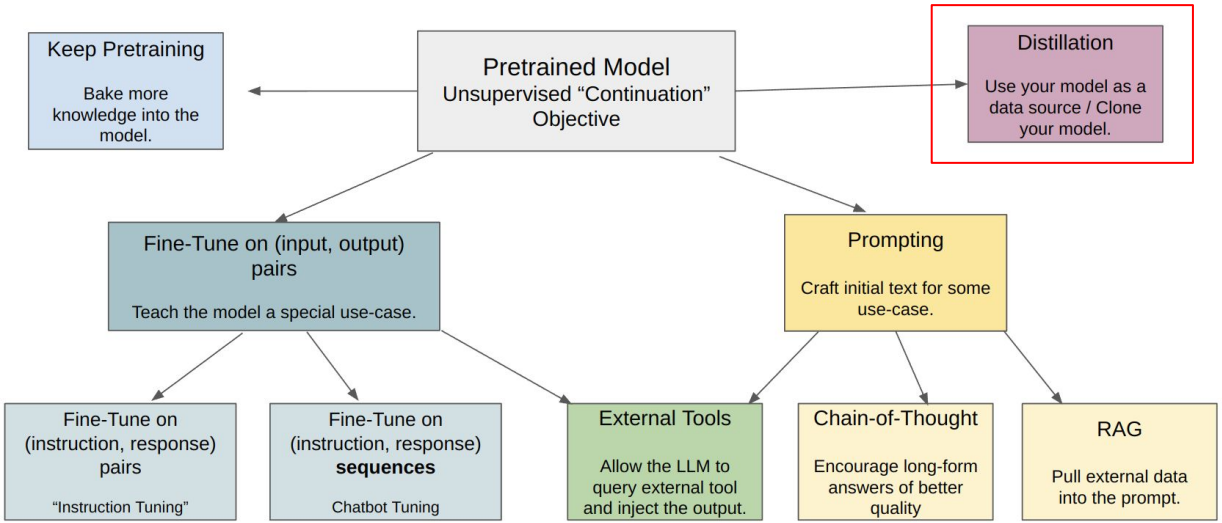
You can specifically train for chat by extending the fine-tuning setup to only calculate a loss for model responses:




**Compare with SFT:**

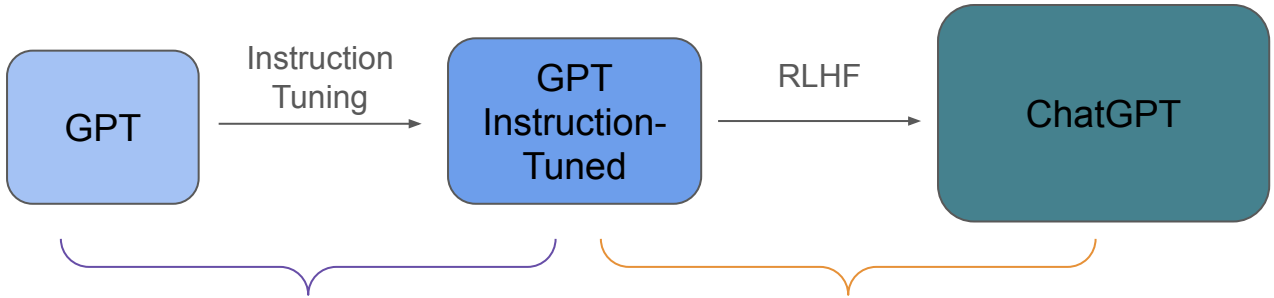


# Synthetic Data / Distillation



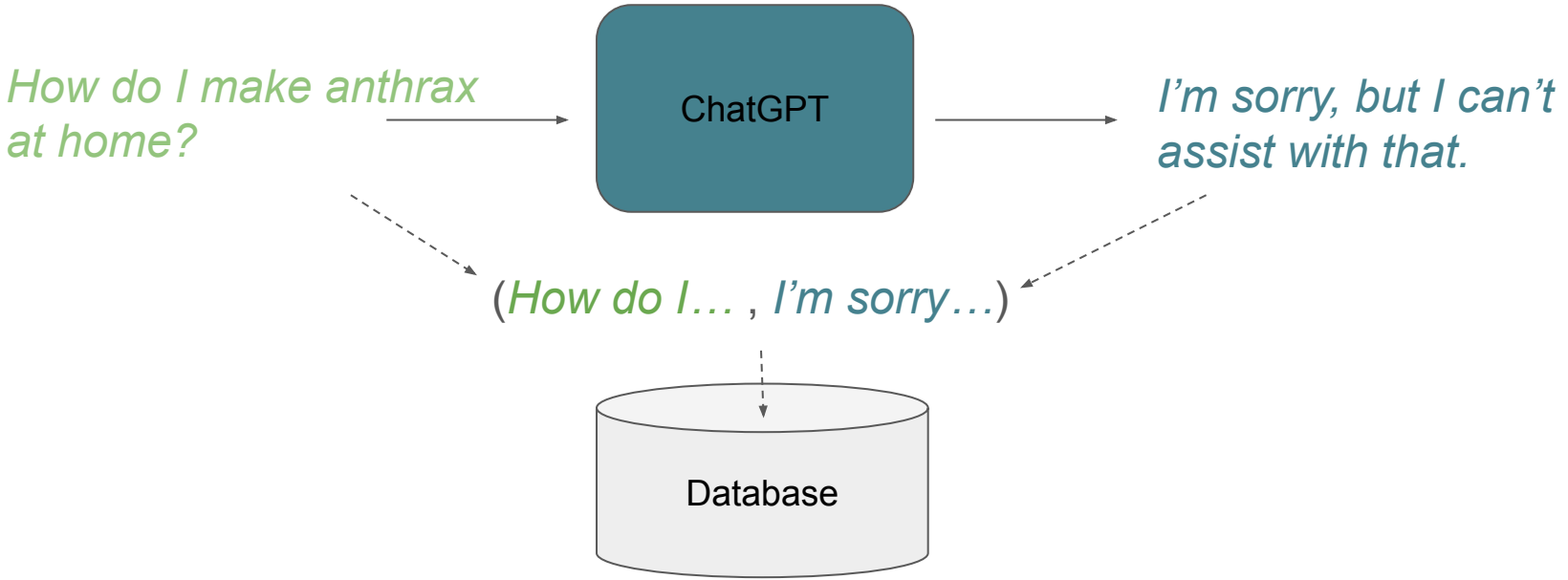
# ChatGPT as a Data Source

Instruction tuning may be insufficient for creating a model that has additional desirable qualities (polite, helpful, etc). We will talk more about adding these in Module 11.



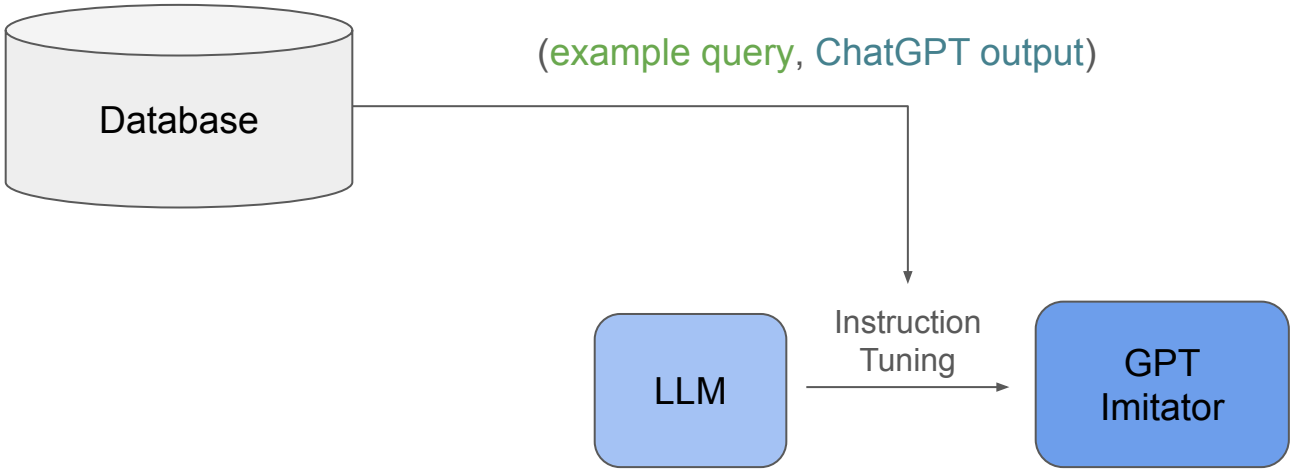
# ChatGPT as a Data Source

If you already have a model with the right qualities, you can collect (input, output) pairs:



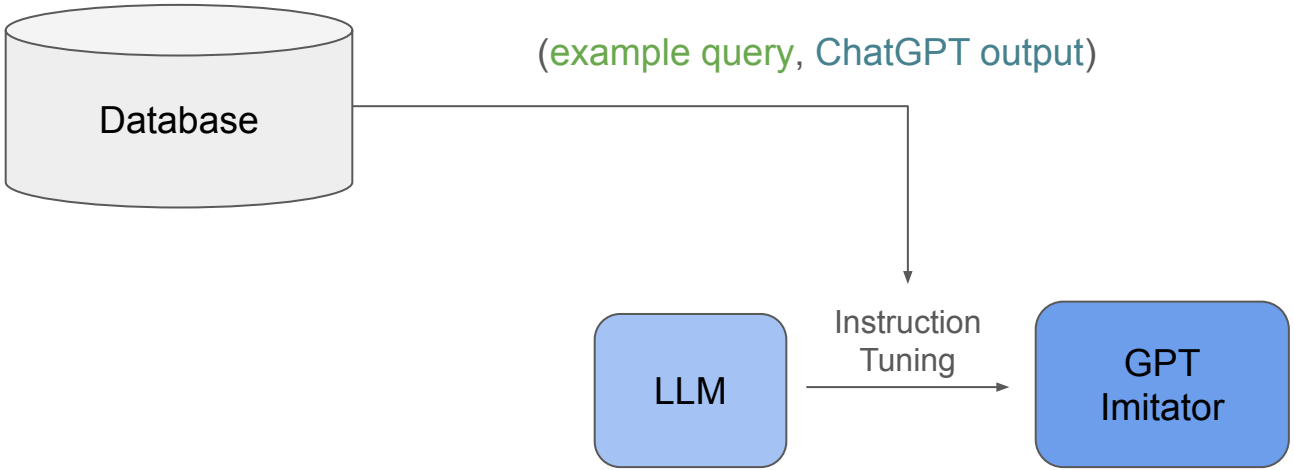
# ChatGPT as a Data Source

You can then fine-tune on this dataset, and it will hopefully demonstrate the desired behavior.



# ChatGPT as a Data Source

You can then fine-tune on this dataset, and it will hopefully demonstrate the desired behavior.

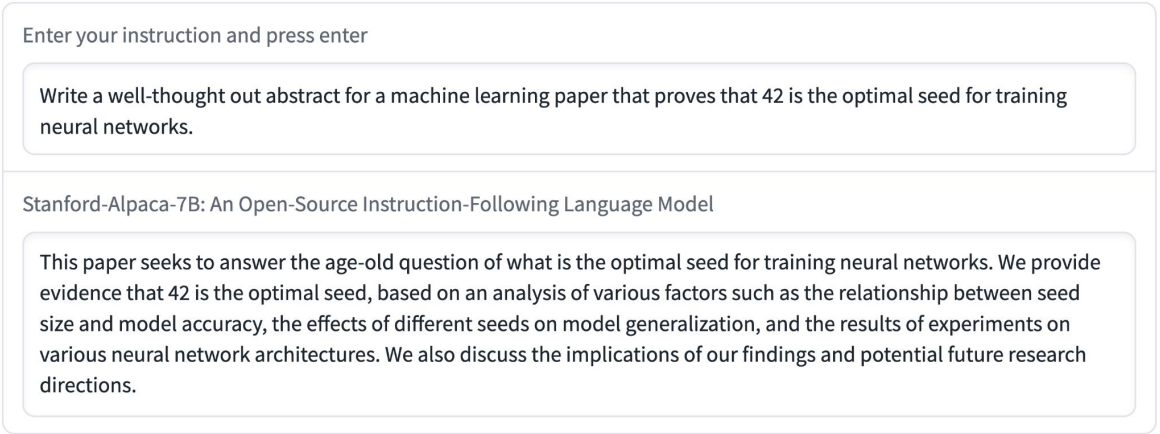



This is also called “distillation”: we are *distilling* ChatGPT’s behavior into a new model.

# Alpaca

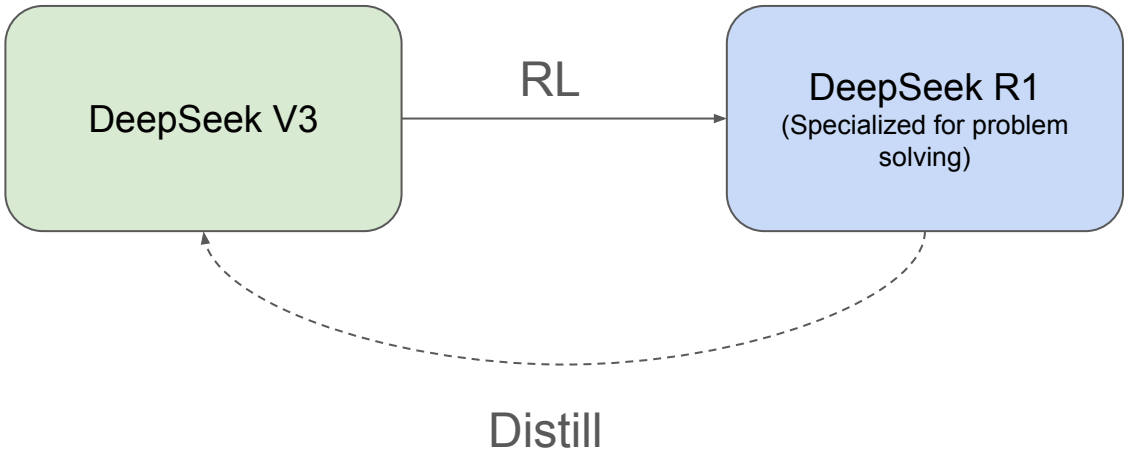
Following the open-source release of LLaMA, this technique was used to make low-cost model with similar behavior to GPT3 Instruct.

**Stanford Alpaca 7b** - fine-tuned LLaMA on 52K examples generated from OpenAI's text-davinci-003.

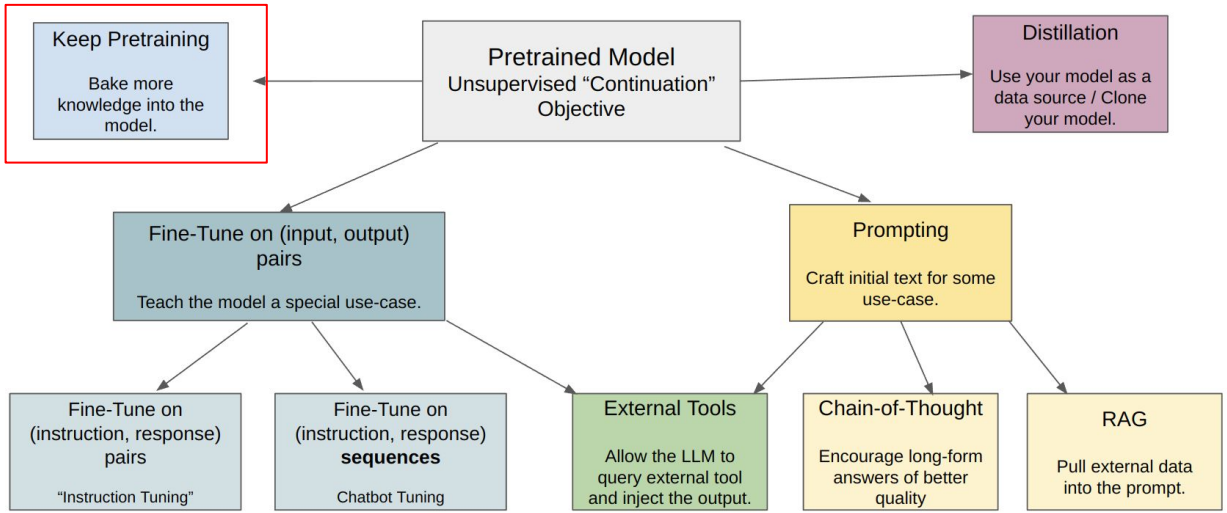


# DeepSeek V3

Recently, DeepSeek had success doing a sort-of self distillation:



# Knowledge vs. Behavior



# Recommendations

If you have a few examples: Construct a prompt

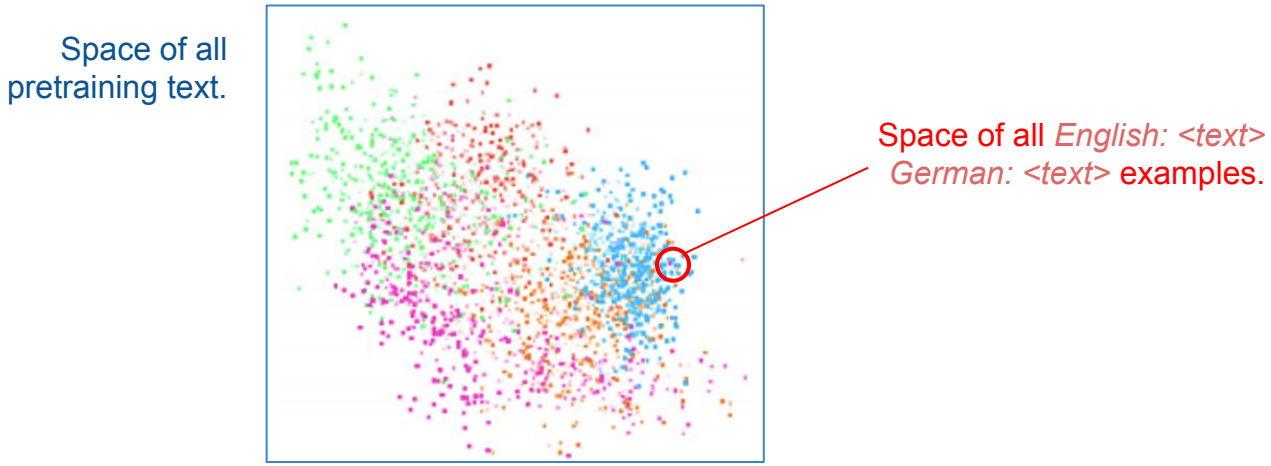
If you have many examples of desired format: fine-tune

If you want to make a base model that can follow instructions, try instruction tuning.

**What if you want to learn something new from your dataset?** For example, take a base LLM and teach it about a specific scientific discipline that was not in the pre-training set.

# Recall SFT

SFT can be viewed as specialization to existing knowledge. Note that we are not introducing something new per se (besides maybe a specific text format).



# One Critical Caveat

In previous slides we mainly discuss the model learning a “behavior” or learning to operate according to some “format”. The underlying assumption is that the model already “knows” (from pretraining) all the correct responses, and we are just showing it how to properly construct an output as a response.

The consensus among most research is that a model learns “knowledge” during pre-training, and “behaviors” can be added by fine-tuning. **However, adding new knowledge during fine-tuning is really hard, if not impossible.**

This is an open research area.

# Repercussions

The consensus among most research is that a model learns “knowledge” during pre-training, and “behaviors” can be added by fine-tuning.

**New Knowledge**

To really add new information to a model (in a deep sense, not just a few examples that could go in a prompt), you need to keep pretraining.

A really interesting case study is ChipNeMo, which continues to train a model on ~25B tokens about computer chips.

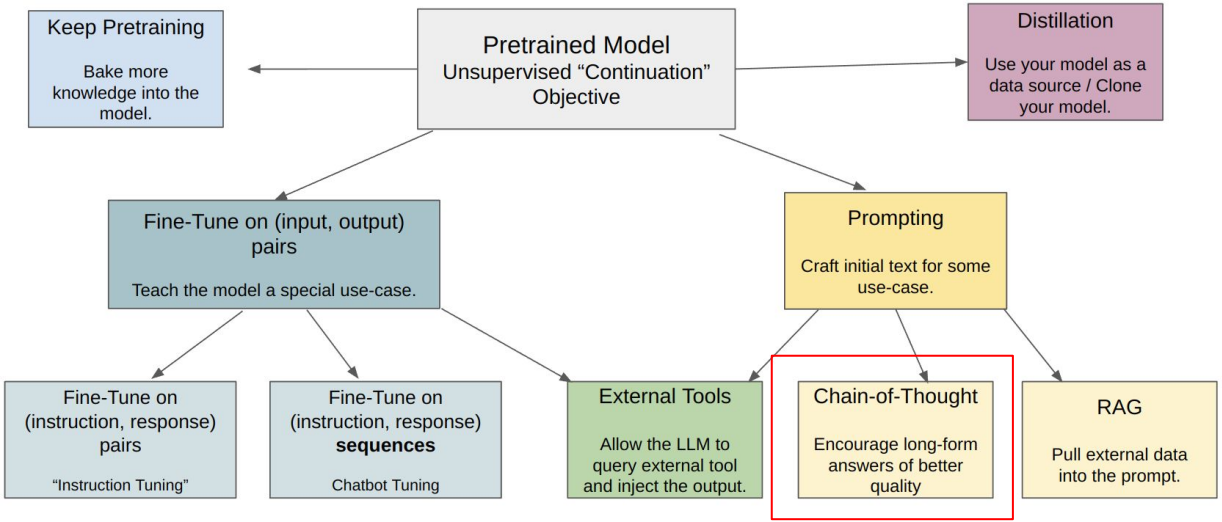
See appendix of this slidedeck.

**Bring Knowledge to Surface**

Other side of the coin- the model actually knows a ton of stuff, more than is apparent from simple interaction.

We can quickly bring some of this knowledge to the surface by only tuning a few parameters. This is a good idea if we think that what we care about was in the pretraining dataset. This will be a main topic for next week.

# Advanced Prompting



# Prompting Recap

We saw last week that we can use a prompt to provide the model with information that may be helpful in generation. A common example is to provide examples of the behavior we are looking for.

More generally, we can include all sorts of things that would be useful for the model when continuing our inputs (or replying to our instruction).

Please translate into German: Bananas are yellow.

Bananen sind gelb.

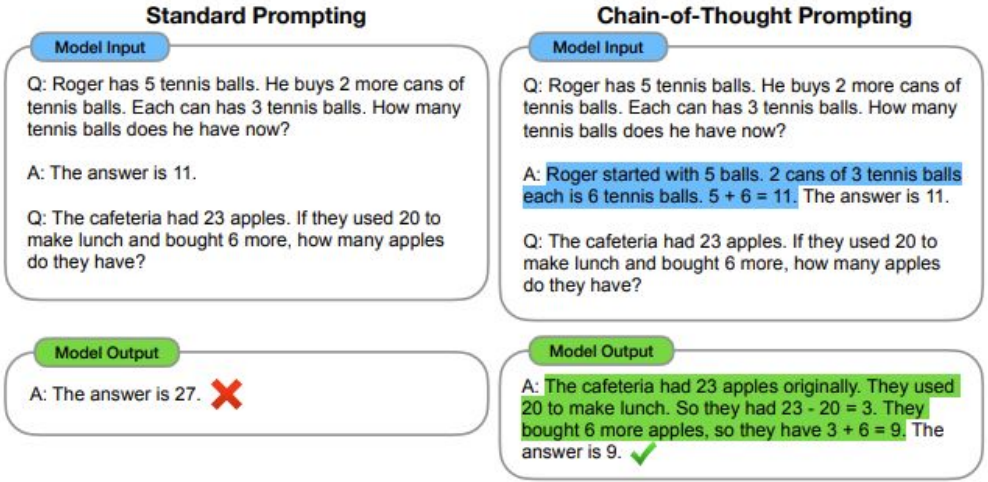
Please translate into German: Today it is sunny.

Heute ist es sonnig.

Please translate into German: My dog's name is Max.

# Chain-of-Thought (CoT)

Another thing we can do with our prompts is encourage the model to “think out loud” via examples or instructions. This is a special prompting case called “chain of thought”, which has been found to lead to higher accuracy on reasoning problems.




Above: example of CoT prompting. On the left, a simple (input, output) example is shown. On right, the same example is shown but with a long-form answer, leading to better model behavior.

CoT paper: https://arxiv.org/pdf/2201.11903

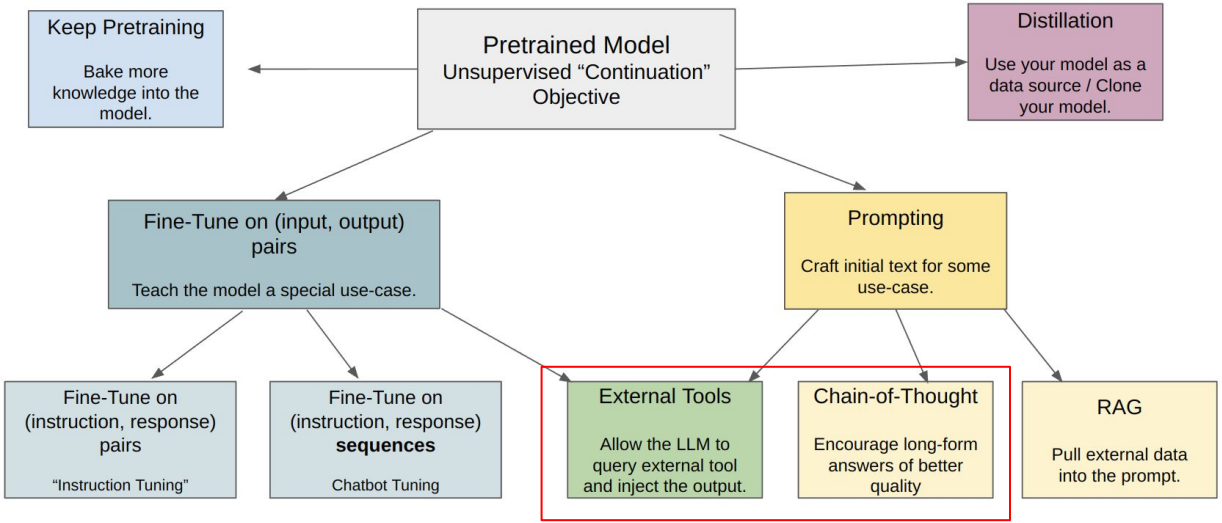
# Chain-of-Thought (CoT)

This can be taken to an extreme by asking a model to repeatedly reflect or revise its own outputs, or executing the generation many times over and taking a majority vote.

You can even use the output of CoT as a training target for more finetuning:

- Input query
- Deep and repeated CoT to determine a good answer
- Isolate the final answer
- Store (input query, good answer) as a training example

# Tool Use


Keep Pretraining
Bake more knowledge into the model.
Pretrained Model
Unsupervised “Continuation” Objective
Distillation
Use your model as a data source / Clone your model.
Fine-Tune on (input, output) pairs
Teach the model a special use-case.
Prompting
Craft initial text for some use-case.
Fine-Tune on (instruction, response) pairs
“Instruction Tuning”
Fine-Tune on (instruction, response) sequences
Chatbot Tuning
External Tools
Allow the LLM to query external tool and inject the output.
Chain-of-Thought
Encourage long-form answers of better quality
RAG
Pull external data into the prompt.

# Toolformer

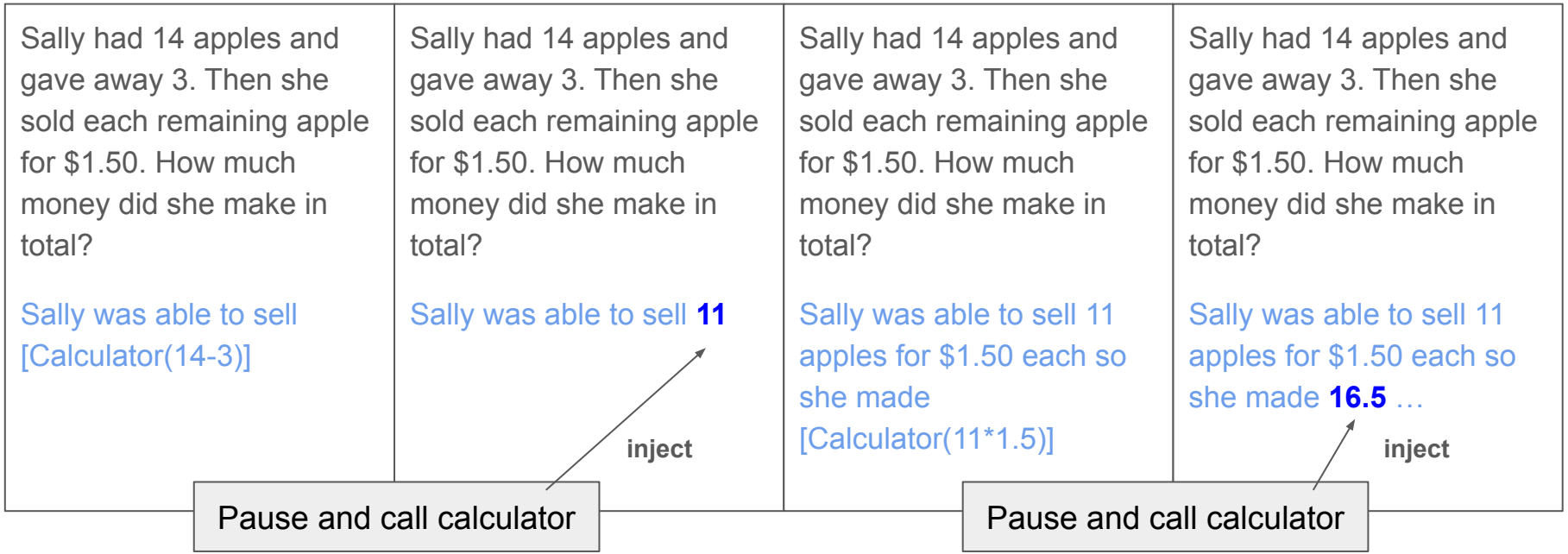
Just as RAG looks up external knowledge and inserts it into the prompt, we can also query external data and insert it *during generation*. Both of these fall under “tool usage”, which is an LLM querying an external tool to assist in generation.

Toolformer is a 2023 paper that introduces the following mechanism:

- Generate a special syntax that means “call external tool”
- If this is generated, pause generation and query the tool.
- Replace the special syntax with the result
- Continue generating

Toolformer: Timo Schick Feb 2023, Meta AI, https://arxiv.org/pdf/2302.04761

# Toolformer Example



# Toolformer: Prompting and Tuning

Toolformer has to first encourage the LLM to synthesize the correct tool syntax in places where appropriate.

To do this, they first construct prompts for each tool that show examples of intended tool use.

These are used to find successful examples of using tools, which are then used as fine-tuning examples.

# What tools are possible?

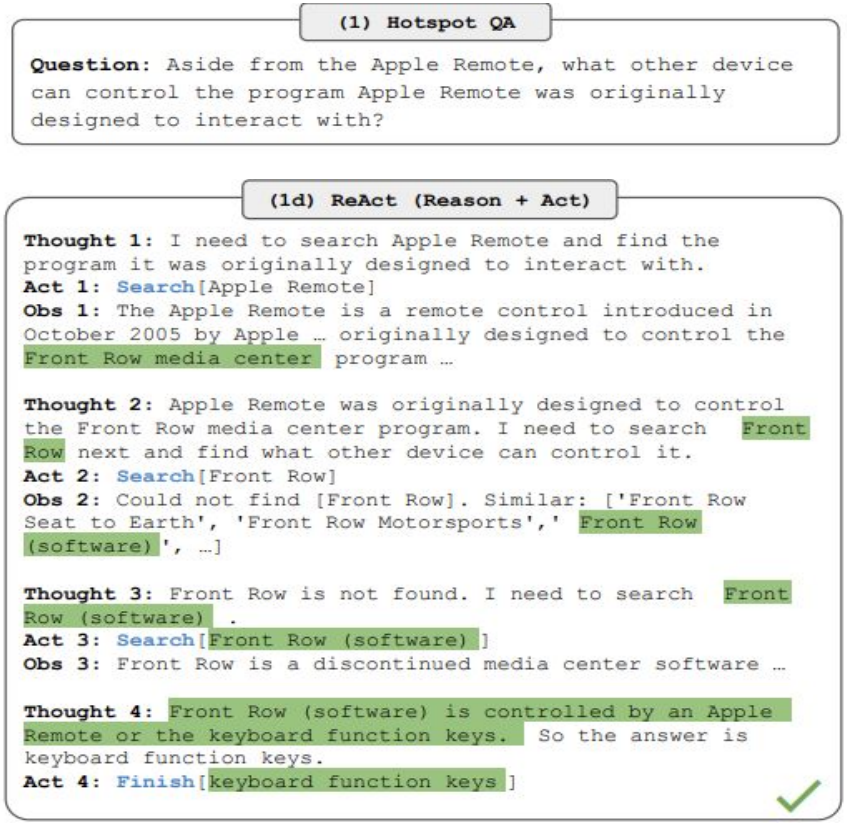
In general, all sorts of tools are possible as long as the accept text as an input (the LLM has to generate the call to the tool). These can include things like:

- Calculators
- Web queries
- Database lookups
- Executing code (maybe that the LLM authored!)
- Another LLM!

# ReAct

There are lots of variations of CoT and tool calling. One popular framework is called “ReAct” (unrelated to the javascript library).

ReAct combines CoT and tool calling together by explicitly asking the LLM to generate a thought, an action, and reflect on the results (an “observation”). This can continue until a desired result is reached.

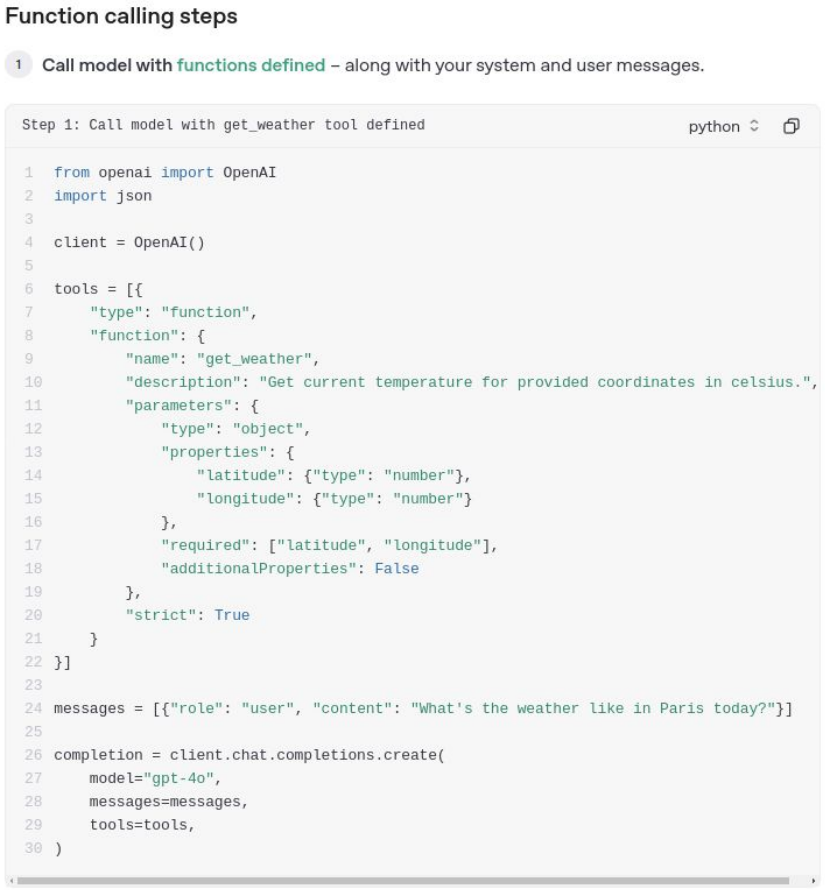



ReAct: Shunyu Yao Oct 2022, Princeton and Google, https://arxiv.org/pdf/2210.03629

# OpenAI Tools

OpenAI (and others, i.e. Anthropic) provide models that are tuned to call tools. Given a variety of function signatures and their descriptions, the model will send queries as appropriate (same type of setup as Toolformer).

https://platform.openai.com/docs/guides/function-calling?api-mode=chat#overview



Function calling steps

1 Call model with functions defined – along with your system and user messages.

Step 1: Call model with get_weather tool defined

python

```python
from openai import OpenAI
import json

client = OpenAI()

tools = [{
    "type": "function",
    "function": {
        "name": "get_weather",
        "description": "Get current temperature for provided coordinates in celsius.",
        "parameters": {
            "type": "object",
            "properties": {
                "latitude": {"type": "number"},
                "longitude": {"type": "number"}
            },
            "required": ["latitude", "longitude"],
            "additionalProperties": False
        },
        "strict": True
    }
}]

messages = [{"role": "user", "content": "What's the weather like in Paris today?"}]

completion = client.chat.completions.create(
    model="gpt-4o",
    messages=messages,
    tools=tools,
)
```

# All Together

# Combining Approaches

A fully-fledged LLM system like ChatGPT will use several of these at once:

- Instruction or chatbot-style tuning to have proper conversation behavior
- Specialized prompts to guide behavior via examples
- RAG to include information from previous conversations
- Search tools to fold in external data or current event data
- Chain-of-thought to reason through a complex query

# Extra (Not in Live Lecture)

# Case Study

# Case Study: ChipNeMo

All of these techniques can be used together to create a highly specialized model.

We will quickly go over an excellent example: a model called ChipNeMo from NVIDIA.

https://research.nvidia.com/publication/2023-10_chipnemo-domain-adapted-llms-chip-design

# ChipNeMo

NVIDIA makes its money selling computer chips, so they decided to build an LLM that knew everything there was to know about chips.

They have other LLM's called "NeMo", so this specialized model is called "ChipNeMo".

However, they started with LLaMA 2, not NeMo.

# Overview

They use several techniques to turn LLaMA 2 into an expert on computer chip design:

1) Start with LLaMA 2 70B
2) Add domain-specific language to the vocabulary of the model
3) Continue pretraining on 24 B tokens just about chips.
4) Instruction-tune the model
5) Setup a RAG system with a custom encoder

# Dataset

The ChipNeMo dataset includes about 22B tokens of internal NVIDIA documents about chip design and verification, including code. They augment this with about 2B tokens of general knowledge (wikipedia) and general programming knowledge (github).

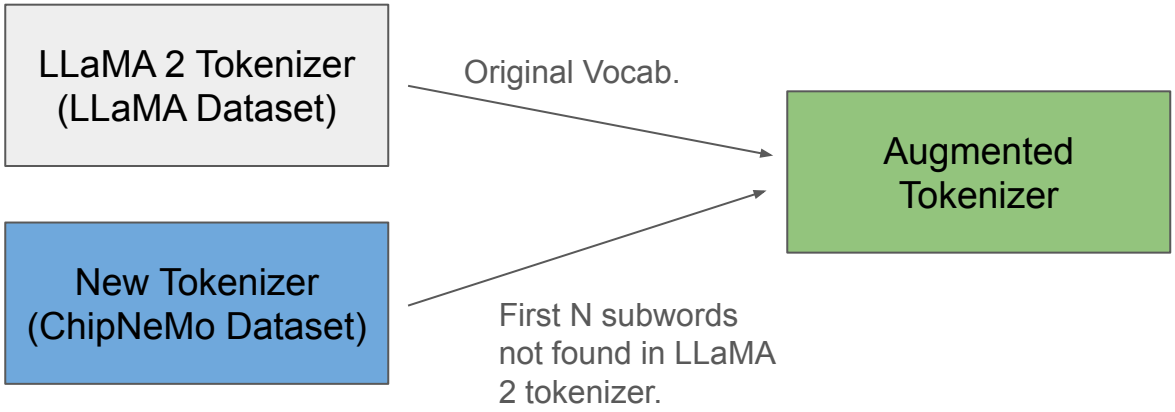
| Data Source Type | Data Percentage (%) | Data Tokens (B) | Training Percentage (%) | Training Tokens (B) |
|---|---|---|---|---|
| Bug Summary | 9.5% | 2.4 | 10.0% | 2.4 |
| Design Source | 47.0% | 11.9 | 24.5% | 5.9 |
| Documentation | 17.8% | 4.5 | 34.0% | 8.2 |
| Verification | 9.1% | 2.3 | 10.4% | 2.5 |
| Other | 7.9% | 2.0 | 12.0% | 2.9 |
| Wikipedia | 5.9% | 1.5 | 6.2% | 1.5 |
| Github | 2.8% | 0.7 | 3.0% | 0.7 |
| Total | 100.0% | 25.3 | 100.0% | 24.1 |

TABLE I: Breakdown of Data by Source. Token count measured with original LLaMA2 tokenizer.

# Adding Vocabulary

Since we are moving to a specific domain, there may be frequent words (or subwords) that are not in the vocabulary of the base model.

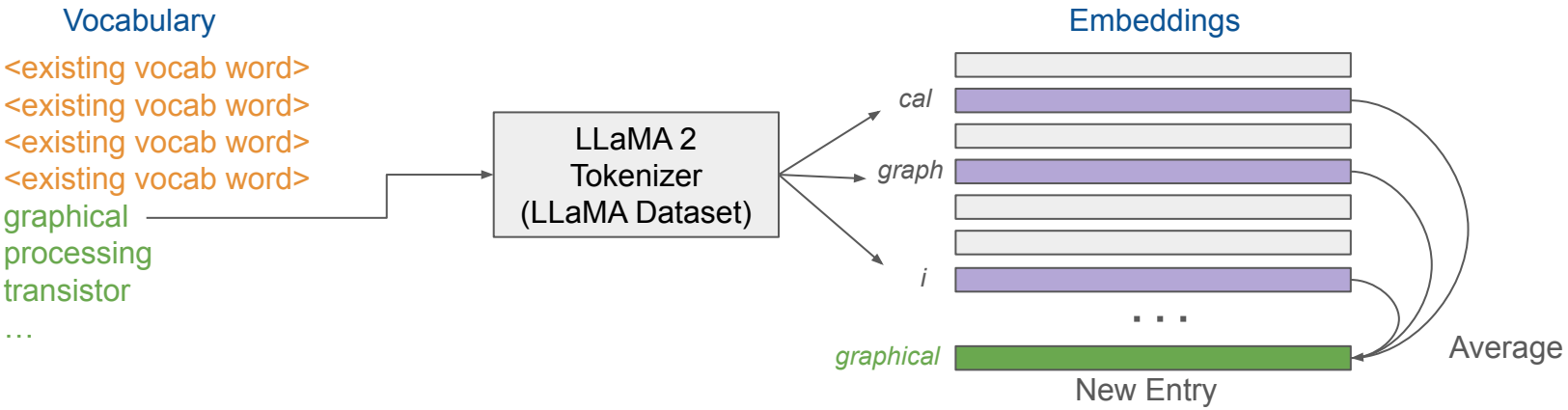
To find these we can train a new tokenizer on the new dataset, and find the first N entries that are not in the original tokenizer.



# Adding Vocabulary

Adding to the vocabulary also means that the model embeddings need to be expanded.

To initialize the embeddings, the new words are tokenized with the LLaMA 2 tokenizer and the average of the embeddings of those tokens is used.



# Continued Pretraining

Since fine-tuning does not generally add knowledge, ChipNeMo continues to pretrain on domain-relevant data.

While a fine-tuning dataset may be 10s or 100s of millions of tokens, this dataset is 24B tokens (more like a small pretraining dataset, which is what it is used for.)

The authors simply train for one epoch over this data at a small fixed learning rate.

# Instruction Tuning

Next, ChipNeMo is instruction tuned, using a dataset of 128k samples of (instruction, response).

Since this data is about behavior, not knowledge, most of it is actually irrelevant to computer chips. About 1% of the data contains examples that are relevant to computer chip design.

# Custom RAG

Finally, a RAG system is created so that ChipNeMo can reference the training corpus during inference to look up specific bits of knowledge.

They also build a custom text encoder for their data lookup, which is specifically trained to retrieve text about computer chips.

# Results

When humans rate model responses out of 10, ChatNeMo is preferred.

When RAG is added, humans slightly prefer LLaMA2-70B (about 5 times the size of ChipNeMo at 13B).

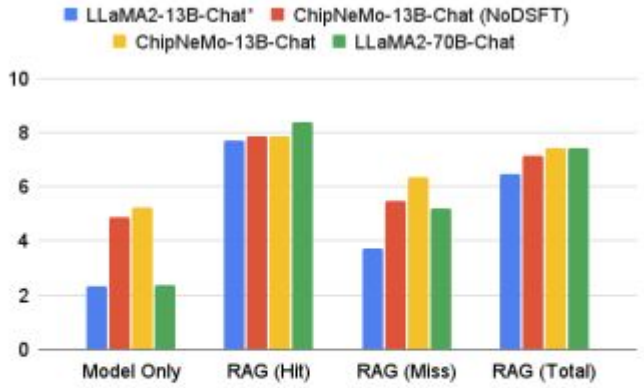



Fig. 8: Human Evaluation of Different Models. Model Only represents results without RAG. RAG (Hit)/(Miss) only include questions whose retrieved passages hit/miss their ideal context, RAG (Total) includes all questions.

NoDSFT = “No Domain SFT”, only doing instruction tuning on non-domain instructions.