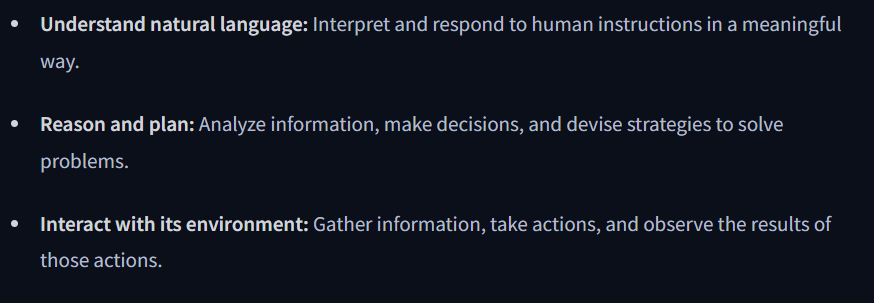
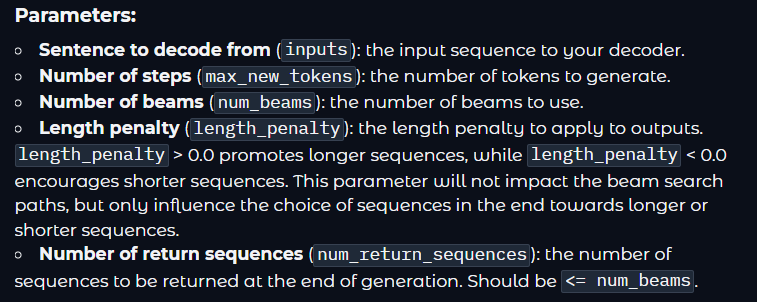
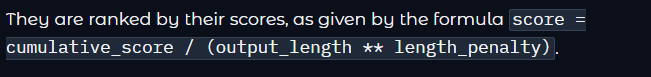
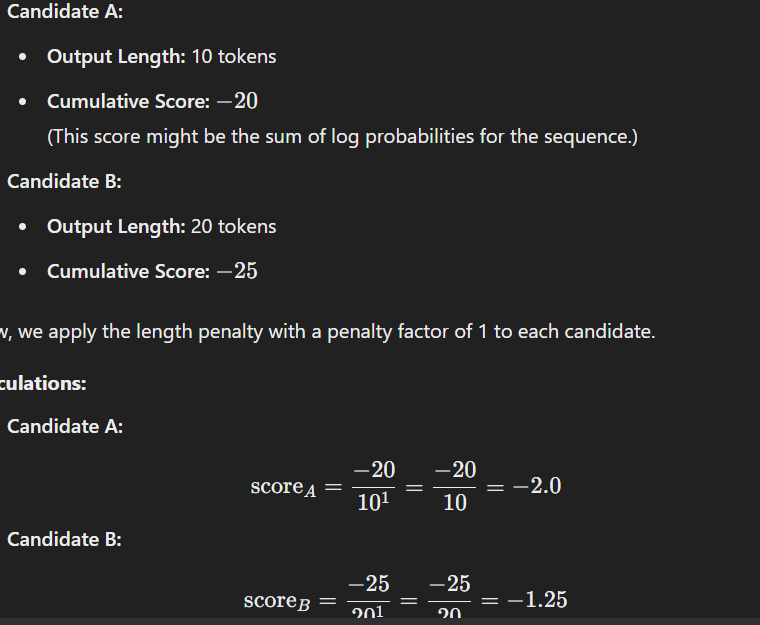
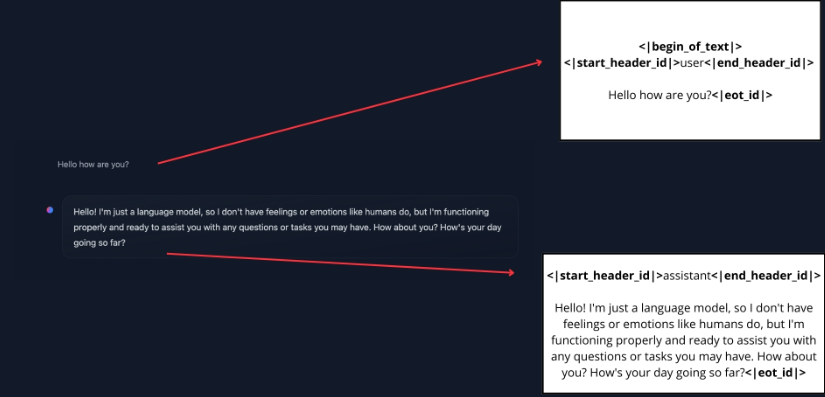
Agents – Week 1

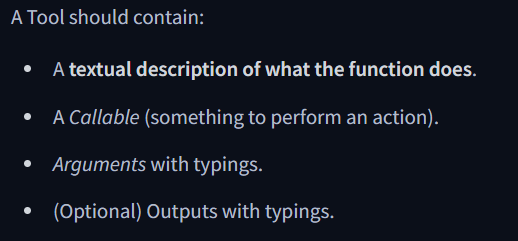
1. Ai model capable of reasoning , planning and interacting with its environment is called as agents.
2. *An Agent is a system that leverages an AI model to interact with its environment in order to achieve a user-defined objective. It combines reasoning, planning, and the execution of actions (often via external tools) to fulfill tasks.*
3. Parts of Agents:

* The Brain [AI Model ]
* Thinking happens .Handles reasoning and planning … to decide which actions to take based on situation
* Body [Capability and tools]
* What agent is equipped to do.. scope of possible actions

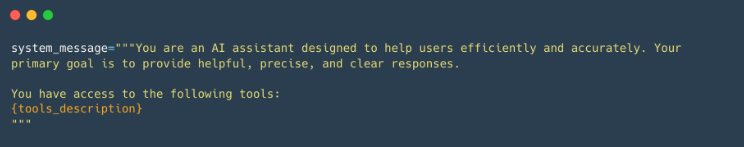
1. Input and output is text only
2. Agents can perform any tasks we implement via Tools to complete Actions.
3. Fix an agent to be PA.. if we want agent to send mail,, if we provide necessary code to send email … agents take this code as tool and send the email
4. *Note that****Actions are not the same as Tools****. An Action, for instance, can involve the use of multiple Tools to complete.*
5. Use case : personal virtual assistant, customer service chatbots, non playable character in video games
6. 
7. Encoder outputs dense representation of the texts
8. Decoder based transformer focus on generating new tokens to complete a sequence one token at a time
9. Beam search : 
10. 
11. Descending order is used
12. 
13. Llm is the brain of the agent

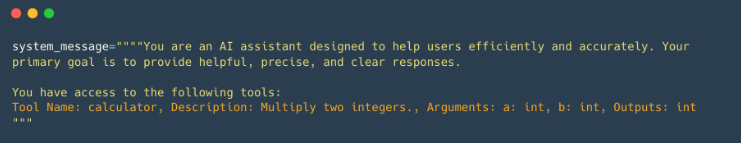
Message and special tokens

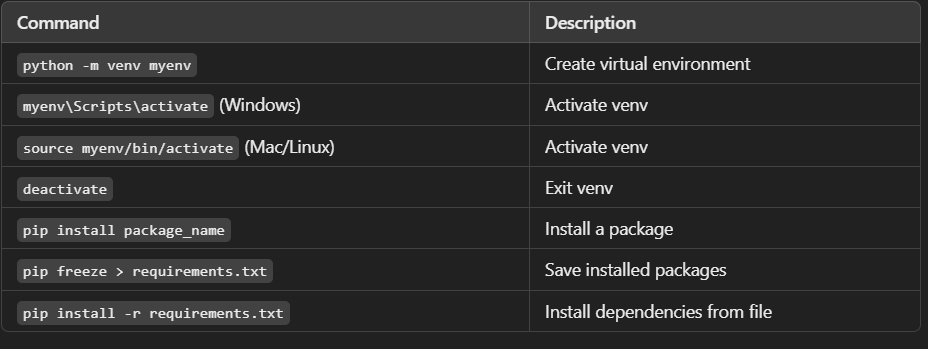
1. 
2. chat templates are essential for **structuring conversations between language models and users**. They guide how message exchanges are formatted into a single prompt.
3. To make a Base Model behave like an instruct model, we need to **format our prompts in a consistent way that the model can understand**. This is where chat templates come in.
4. *ChatML* is one such template format that structures conversations with clear role indicators (system, user, assistant).
5.  Jinja2 is a powerful templating engine in Python. It allows you to write templates that include placeholders, loops, conditionals, and other programming logic. In the context of chat templates:
6. **Placeholders:** Markers where content from the JSON messages (such as the system instruction, user message, or assistant reply) will be inserted.
7. **Logic:** Code that can loop over the list of messages, apply formatting, and arrange them in a clear order.
8. In transformers, chat templates include [Jinja2 code](https://jinja.palletsprojects.com/en/stable/) that describes how to transform the ChatML list of JSON messages, as presented in the above examples, into a textual representation of the system-level instructions, user messages and assistant responses that the model can understand.
9. This structure **helps maintain consistency across interactions and ensures the model responds appropriately to different types of inputs**.
10. from transformers import AutoTokenizer
11. tokenizer = AutoTokenizer.from\_pretrained("HuggingFaceTB/SmolLM2-1.7B-Instruct")
12. rendered\_prompt = tokenizer.apply\_chat\_template(messages, tokenize=False, add\_generation\_prompt=True)
13. Because each instruct model uses different conversation formats and special tokens, chat templates are implemented to ensure that we correctly format the prompt the way each model expects.
14. Tokenizer handles this … add\_generation\_prompt = true ..adds assistant at end to make it to generation stage
15. Tools:

* Actions happen through tools
* Tool is function given to llm with clear objective
* Good tool complements the power of LLM
* 
* What we mean when we talk about *providing tools to an Agent*, is that we **teach** the LLM about the existence of tools, and ask the model to generate text that will invoke tools when it needs to.
* For this to work, we have to be very precise and accurate about:
* **What the tool does**
* **What exact inputs it expects**

We tell these to LLM





* When this \_\_call\_\_ method is invoked on an instance, any positional arguments passed to it are captured in args (as a tuple) and any keyword arguments are captured in kwargs (as a dictionary).
* Using @tool decorator we can get those on top of function
* Tool Name: calculator, Description: Multiply two integers., Arguments: a: int, b: int, Outputs: int
* 
* In short, smolagents is a library that focuses on **codeAgent**, a kind of agent that performs **“Actions”** through code blocks, and then **“Observes”** results by executing the code.
* Function calling :
* LLM structure the conversation and letting them trigger Tools
* Way for llm to take action on its environment
* This capacity is learned by model using prompt only and not by other agents

Fine Tuning LLM

1. If two sentences are logically related or not [Natural Language Inference].
2. Glue 10 dataset benchmark for classification task … 8 out of 10 is pairs
3. Here, BERT is pretrained with token type IDs, and on top of the masked language modeling objective we talked about in [Chapter 1](https://huggingface.co/course/chapter1), it has an additional objective called next sentence prediction. The goal with this task is to model the relationship between pairs of sentences.
4. This works well, but it has the disadvantage of returning a dictionary (with our keys, input\_ids, attention\_mask, and token\_type\_ids, and values that are lists of lists). It will also only work if you have enough RAM to store your whole dataset during the tokenization (whereas the datasets from the 🤗 Datasets library are [Apache Arrow](https://arrow.apache.org/) files stored on the disk, so you only keep the samples you ask for loaded in memory).
5. To keep the data as a dataset, we will use the [Dataset.map()](https://huggingface.co/docs/datasets/package_reference/main_classes" \l "datasets.Dataset.map) method. This also allows us some extra flexibility, if we need more preprocessing done than just tokenization. The map() method works by applying a function on each element of the dataset.

SMOLAGENTS

1. Codeagents are primary type of agent in smolagents
2. Produce python code to perform actions
3. Toolcalling agets,json/text blob that system parse and interepret to execute the actions
4. You need a **lightweight and minimal solution.**
5. You want to **experiment quickly** without complex configurations.
6. Your **application logic is straightforward.**
7. Agent in smolagents operate as multi-step agents
8. Each multistep agent performs:

* One thought
* One tool call and execution

Steps in code agent smolagents

1. System prompt – sytemprompt step, user query – task step
2. While loop

* Agents.write\_memory\_to\_message() ->agents log into a list of llm chat messages
* Sent to model – to generate completion
* Parsed to extract action, code snippet
* Action is executed
* Results in memory at action step
* At end of each step,if agent includes any function call (agent.step\_callback), they are executed.

Tools:

1. Name, tool description, input types ad description, output type
2. Tool created using @tool decorator, creating a subclass of Tool
3. Tool with python class: name,description,inputs .. type and description, output type ,forward .. logic

Agentic RAG Systems