UNIVERSITY OF MANNHEIM

Team Project HWS 2023

Benchmarking Function-Calling Enabled LLMs

Hallo

- **Prof. Dr. Christian Bizer**
- Professor for Information Systems V
- Research Interests:
  - Web-based Systems
  - Large-scale Data Integration
  - Data and Web Mining
- Room: B6, 26 - B1.15
- eMail: christian.bizer@uni-mannheim.de
- Consultation: Wednesday, 13:30-14:30

Hallo

- **M. Sc. Wi-Inf. Keti Korini**
- Graduate Research Associate
- Research Interests:
  - Table Annotation using Deep Learning
  - Schema Mapping
- Room: B6, 26, C 1.03
- eMail: kkorini@uni-mannheim.de

Agenda of Today‘s Kickoff Meeting

1. A round of introductions: You and Your Experience
2. Introduction to (Augmented) LLMs
3. Project Goals
4. Organization
5. Specific Subtasks
6. Schedule
7. Formal Requirements
8. Related Work

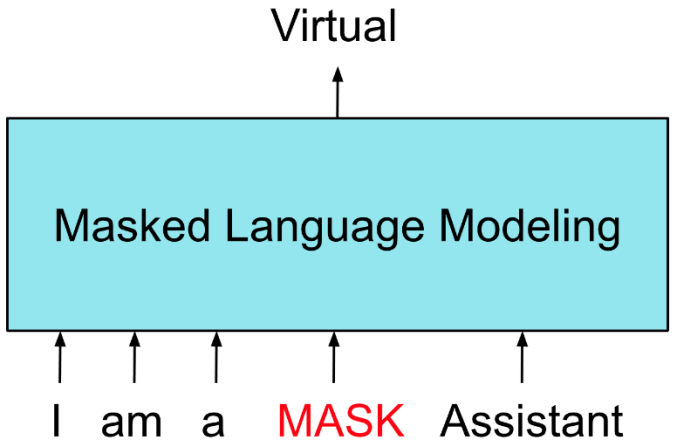
You and Your Experience

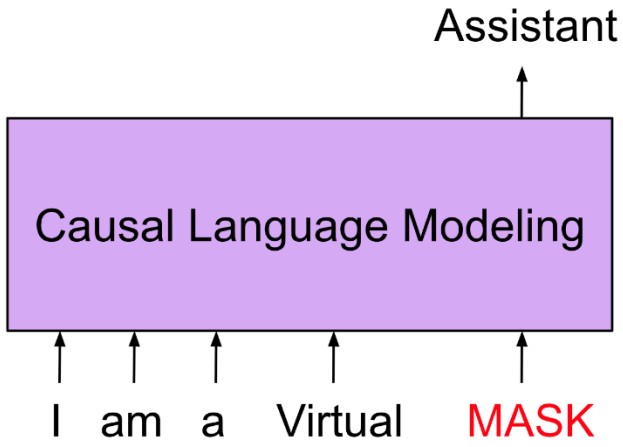
- A Short Round of Introductions
  - What are you studying? Which semester?
  - Which DWS courses did you already attend?
  - What are your programming and data wrangling skills?
  - What experience do you have with LLMs and prompt engineering?
- Participants

1. Eroglu, Zeynep
2. Jano, Stiliana
3. Bajri, Deidamea
4. Kutrolli, Serxhina
5. Koumi, Fatma
6. Heinz, Dennis
7. Khursheed, Saman

Introduction to Large Language Models (LLMs)

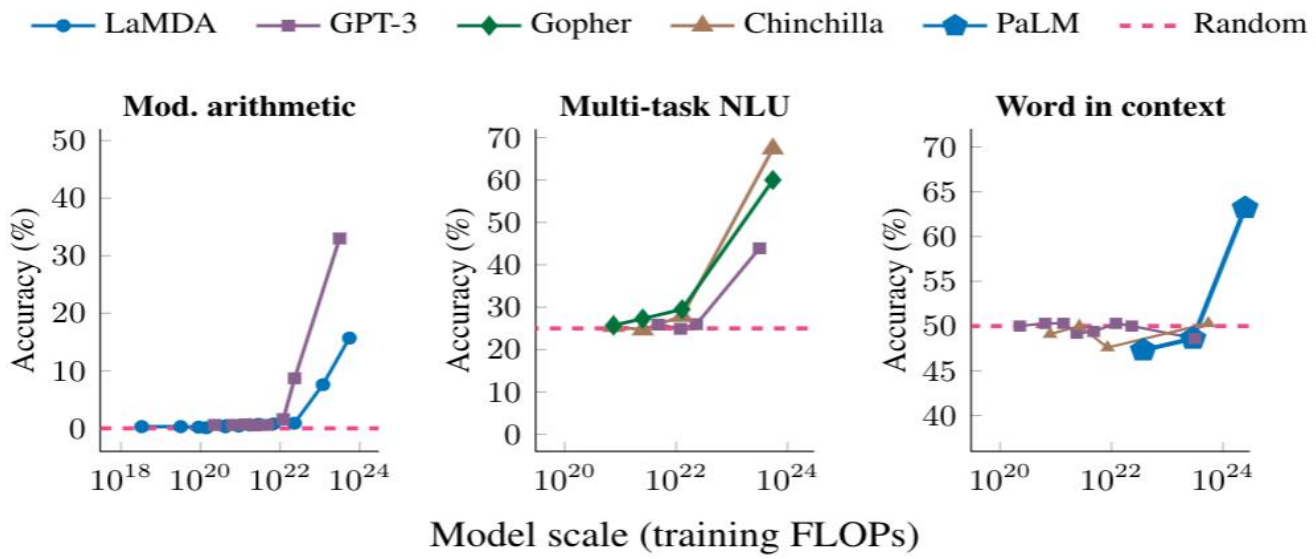
- Language Models model the joint probability for sequence of words
  - Example: If two words are given what is the most likely word to follow the other two?
  - They can be used to either generate new text or to asses whether a sequence is likely
- Some forms of Languge Models include:
  - Left-to-right auto-regressive **Causal Language Modeling** (predict next token in a sequence), like GPT models
  - **Masked Language Modeling** (captures bidirectional context: left and right tokens), like the BERT models





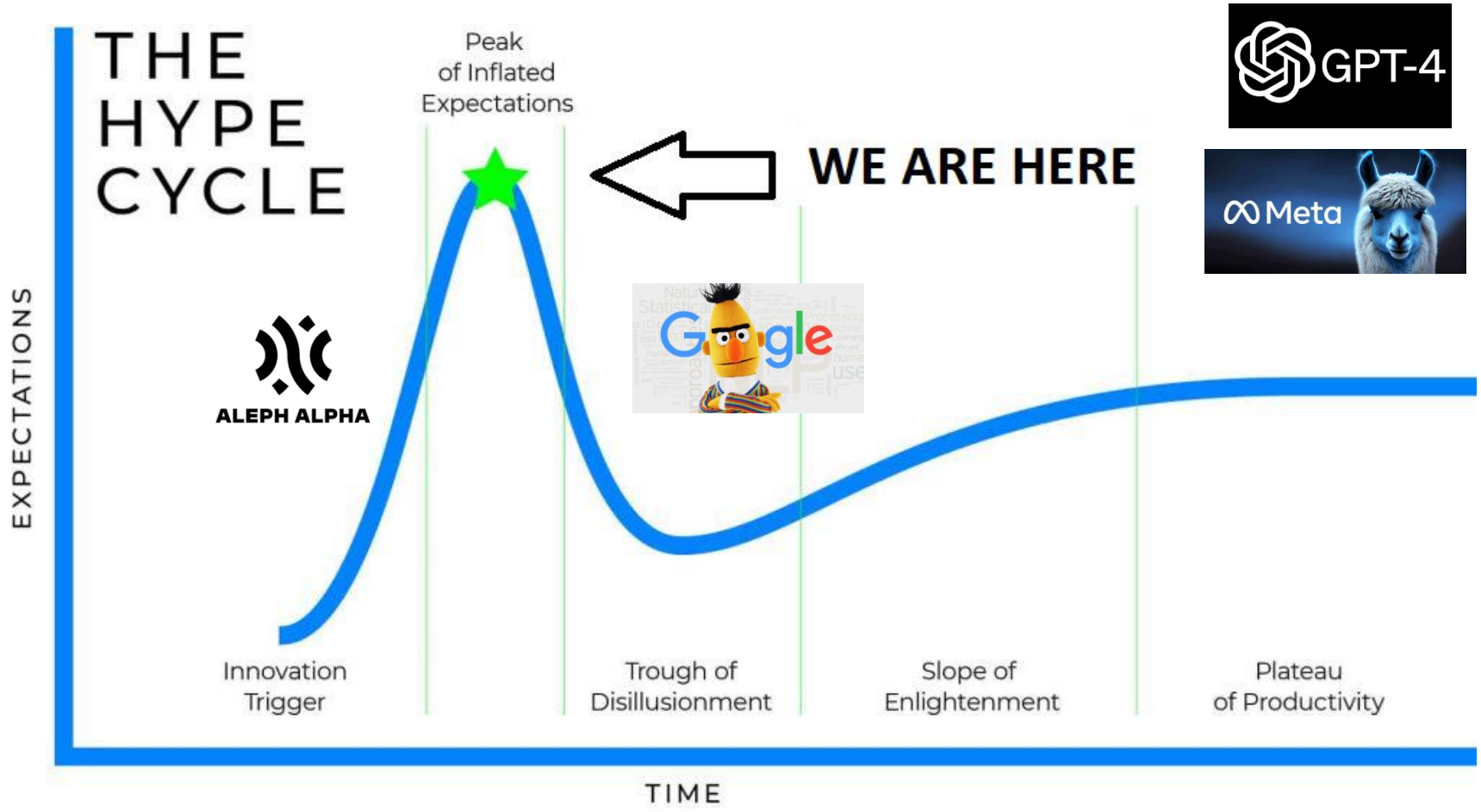
Large Language Models

- Large Language Model (LLMs) are LMs that have very large number of parameters
  - Example: GPT-3 has 175 billion parameters, while PaLM has 540 Million parameters
- LLMs display some **emergent abilities** not observed in PLMs
  - Some of these abilities are in-context learning, instruction following and step-by-step reasoning
  - No need to fine-tune them to reach good performance



Zhao, Wayne Xin, et al. "**A survey of large language models.**" arXiv preprint arXiv:2303.18223 (2023).

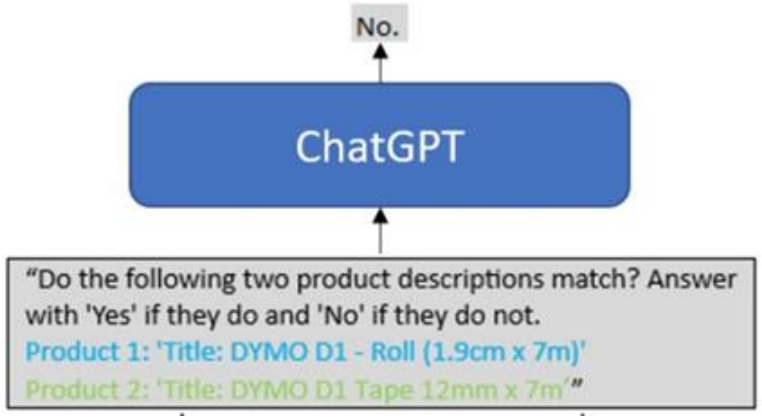
Large Language Models



Interacting with LLMs via Prompts

- Prompt

  A prompt is natural language text
  - describing the task that a model should perform.
  - posing a question that a model should answer.



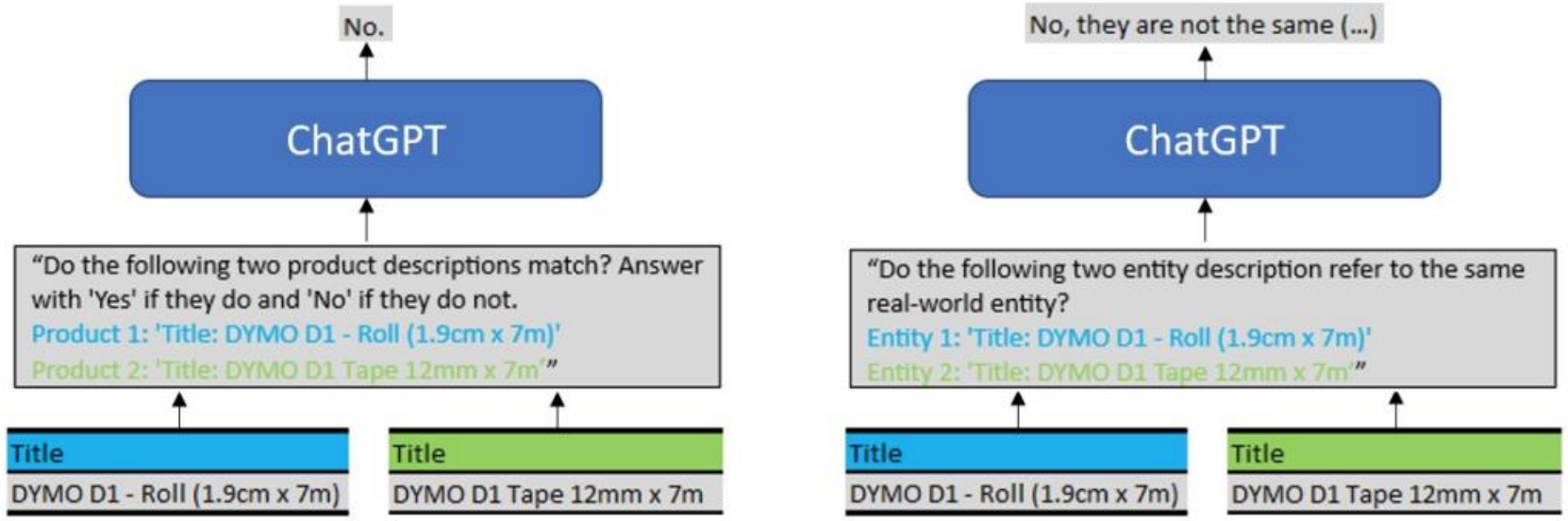
- Prompt Engineering

  Prompt engineering is the task of developing and optimizing prompts to efficiently use LLMs for a wide variety of applications.

Prompt Engineering Guides

https://www.promptingguide.ai/
https://learnprompting.org/docs/intro

Impact of Variations in the Prompt Formulation



Variation

- **general** vs. **domain-specific** wording
- **complex** vs. **simple** task description
- **free-form** vs. **forced** (restricted) answering

Impact of Variations in the Formulation of Prompts

Peeters, Bizer: Using ChatGPT for Entity Matching. https://arxiv.org/abs/2305.03423 (N=433 pairs)

Prompt	P	R	F1	Δ F1	cost (¢) per pair
general-complex-free-T	49.50	**100.00**	66.23	-	0.11
general-simple-free-T	70.00	98.00	81.67	15.44	0.10
general-complex-forced-T	63.29	**100.00**	77.52	11.29	0.14
general-simple-forced-T	75.38	98.00	85.22	18.99	0.13
general-simple-forced-BT	79.66	94.00	**86.24**	20.01	0.13
general-simple-forced-BTP	71.43	70.00	70.70	4.47	0.13
domain-complex-free-T	71.01	98.00	82.35	16.12	0.11
domain-simple-free-T	61.25	98.00	75.38	9.15	0.10
domain-complex-forced-T	71.01	98.00	82.35	16.12	0.14
domain-simple-forced-T	74.24	98.00	84.48	18.25	0.13
domain-simple-forced-BT	76.19	96.00	84.96	18.73	0.13
domain-simple-forced-BTP	54.54	84.00	66.14	-0.09	0.13
Narayan-complex-T	85.42	82.00	83.67	17.44	0.10
Narayan-simple-T	**92.86**	78.00	84.78	18.55	0.10

- Precision and recall strongly vary depending on the prompt formulation.

- Three patterns emerge:
  1. domain-specific wording leads to more stable results
  2. describing the task in simpler language works better
  3. forcing the model to answer with simple “Yes” or “No” is helpful

In-Context Learning

Provide demonstrations in a prompt on how to perform the task.

Task Description	Given the following information about matching product descriptions:
In-context Examples	**Matching:** **Product 1: 'Title: DYMO D1 Labelling Tape 45803 Black on White 19 mm x 7 m'** **Product 2: 'Title: Dymo Label Casette D1 (19mm x 7m - Black On White)'** **Non-matching:** **Product 1: 'Title: DYMO D1 Tape 24mm Black on Yellow'** **Product 2: 'Title: Dymo 45803 D1 19mm x 7m Black on White Tape'**
Task Description	Do the following two product descriptions refer to the same product? Answer with 'Yes' if they do and 'No' if they do not.
Task Input	Product 1: 'Title: DYMO D1 - Glossy tape - black on white - Roll (1.9cm x 7m) - 1 roll(s)' Product 2: 'Title: DYMO 45017 D1 Tape 12mm x 7m sort p rd, S0720570'

- How to select in-context demonstrations
  - **Related**: Use similarity metric to find most similar demonstrations in a training set
  - **Random**: Randomly choose pairs from training set
  - **Handpicked**: Domain expert chooses a small set of demonstrations

Results: In-Context Learning

Selection heuristic	Shots	P	R	F1	Δ F1	Cost (¢) per pair	Cost increase	Cost increase per Δ F1
ChatGPT-zeroshot	0	71.01	**98.00**	82.35	-	0.14	-	-
ChatGPT-random	6	78.33	94.00	85.45	3.10	0.77	450%	145%
	10	79.66	94.00	86.24	3.89	1.13	707%	182%
	20	78.95	90.00	84.11	1.76	2.07	1379%	783%
ChatGPT-handpicked	6	76.19	96.00	84.86	2.51	0.72	414%	165%
	10	80.00	96.00	87.27	4.92	1.00	614%	125%
	20	79.66	94.00	86.24	3.89	2.03	1350%	347%
ChatGPT-related	6	80.36	90.00	84.91	2.56	0.68	386%	151%
	10	**89.58**	86.00	87.76	5.41	1.05	650%	120%
	20	88.46	92.00	**90.20**	7.85	1.97	1307%	167%
GPT3.5-handpicked	10	61.97	88.00	72.72	-9.63	10.54	7429%	771%
	20	61.43	86.00	71.67	-10.68	19.71	13979%	1309%
GPT3.5-related	10	67.69	88.00	76.52	-5.83	10.04	7071%	1213%
	20	61.43	86.00	71.67	-10.68	20.34	14429%	1351%

- Performance increase of ~**3%** F1 with just small number of examples
- Best performance: **20 related** examples lead to ~**8%** F1 increase
- Increased performance comes with a **cost increase** of > 100% per gained percentage point of F1

Provide Domain Knowledge in a Prompt

Task Description	Your task is to decide if two product descriptions match. The following rules need to be observed:
Rules	1. **The brand of matching products must be the same if available** 2. **Model names of matching products must be the same if available** 3. **Model numbers of matching products must be the same if available** 4. **Additional features of matching products must be the same if available**
Task Description	Do the following two product descriptions refer to the same product? Answer with 'Yes' if they do and 'No' if they do not.
Task Input	Product 1: 'Title: DYMO D1 - Glossy tape - black on white - Roll (1.9cm x 7m) - 1 roll(s)' Product 2: 'Title: DYMO 45017 D1 Tape 12mm x 7m sort p rd, S0720570'

- Approach: Provide simple human created matching rules
- Try to guide the reasoning capability of the LLM
- Intrinsic understanding of product features necessary

Results – Domain Knowledge

Table 5: Matching Knowledge results

Prompt	Shots	P	R	F1	Δ F1	Cost (¢) per pair	Cost increase	Cost increase per Δ F1
ChatGPT-zeroshot	0	71.01	**98.00**	82.35	-	0.14	-	-
ChatGPT-zeroshot with rules	0	80.33	**98.00**	88.29	5.94	0.28	100%	17%
ChatGPT-related	6	80.36	90.00	84.91	2.56	0.68	386%	151%
	10	89.58	86.00	87.76	5.41	1.05	650%	120%
	20	88.46	92.00	**90.20**	7.85	1.97	1307%	167%
ChatGPT-related with rules	6	90.70	78.00	83.87	1.52	0.79	464%	305%
	10	90.91	80.00	85.11	2.76	1.17	736%	267%
	20	**91.11**	82.00	86.32	3.97	2.09	1393%	351%

- Matching rules lead to increase in ~9% Precision and ~6% F1
- Similar but not as strong effect as providing related in-context examples
- Rules are cheaper to derive, cost of a query is lower

Disadvantages of LLMs

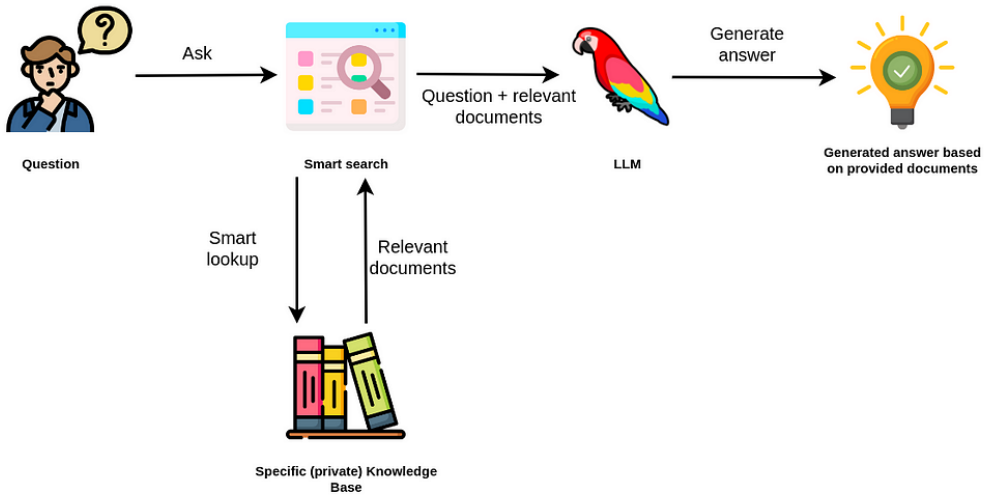
As at its core they are statistical models and can generate text based on the pre-training patterns they have learned, LLMs display certain disadvantages:

- They **have problems with advanced reasoning**, e.g. mathematical reasoning
- They display **factual errors**, this problem is also referred as *hallucinations*
- LLMs **do not contain detailed information about many long-tail entities**, such as products, events, local businesses, or music recordings
- Knowledge stored in LLMs may be **outdated or incorrect,** as it depends on the training corpus

Borji, Ali. **"A categorical archive of chatgpt failures."** arXiv:2302.03494 (2023).
Bang, Yejin, et al. **"A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity."** arXiv:2302.04023 (2023).

Augmented LLMs

- To overcome disadvantages, LLMs can be *augmented* with *information* and *tools*
  - Pairing with an LLM a **python interpreter** to perform mathematical reasoning
  - LLMs can be combined with **visual language models** in order to controll physical robots
  - augment with **retrieved documents, external APIs** to overcome non-factual and outdated information
- **Retrieval Augmented Question Answering**



Mialon, et al.: **Augmented Language Models: a Survey**. arXiv:2302.07842 [cs.CL]
He, Hangfeng, Hongming Zhang, and Dan Roth. "**Rethinking with retrieval: Faithful large language model inference.**" arXiv:2301.00303 (2022).

Function Calling

- ChatGPT and GPT-4 models were **fine-tuned to decide when functions should be called** to improve results. The models reply with the parameters to call the function.
- **Function calling** can be used to augment LLMs:

Retrieval Augmented Question Answering

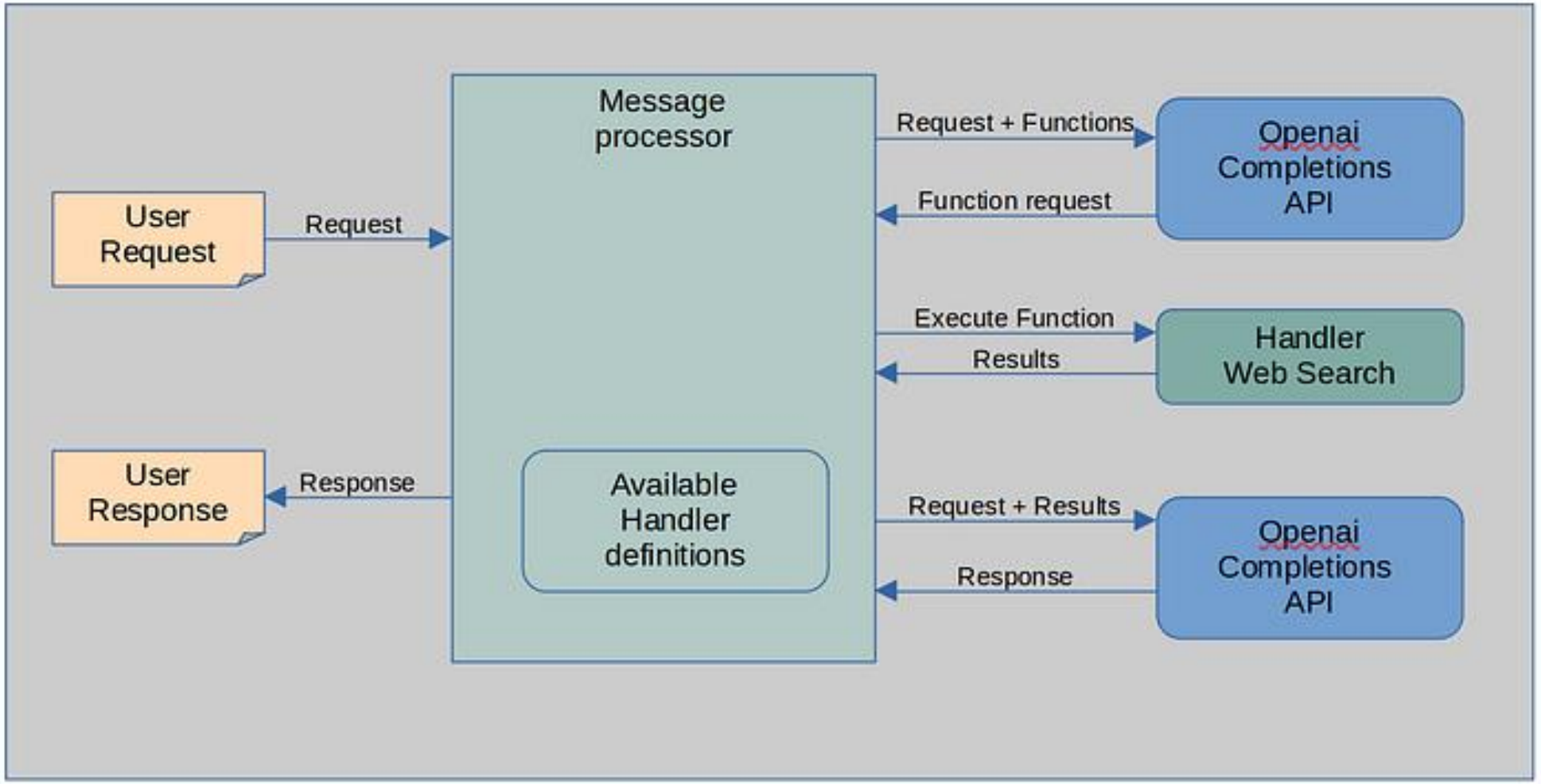
- **Example:** Ask for the current weather in a city.
- **Question:** What is the current weather in Mannheim?
- **Function:** get_weather(location: string, unit: “Celsius”|”Fahrenheit”)

Extract Structured Data from Text

- **Question:** Extract the brand and model attributes of a product from the following text:
  SAMSUNG GU55CU7179 LED TV
- **Function:** extract_attributes(product:[{brand: string, model: string}])

https://openai.com/blog/function-calling-and-other-api-updates

Function-Calling Augmented Question Answering



Function Calling Steps

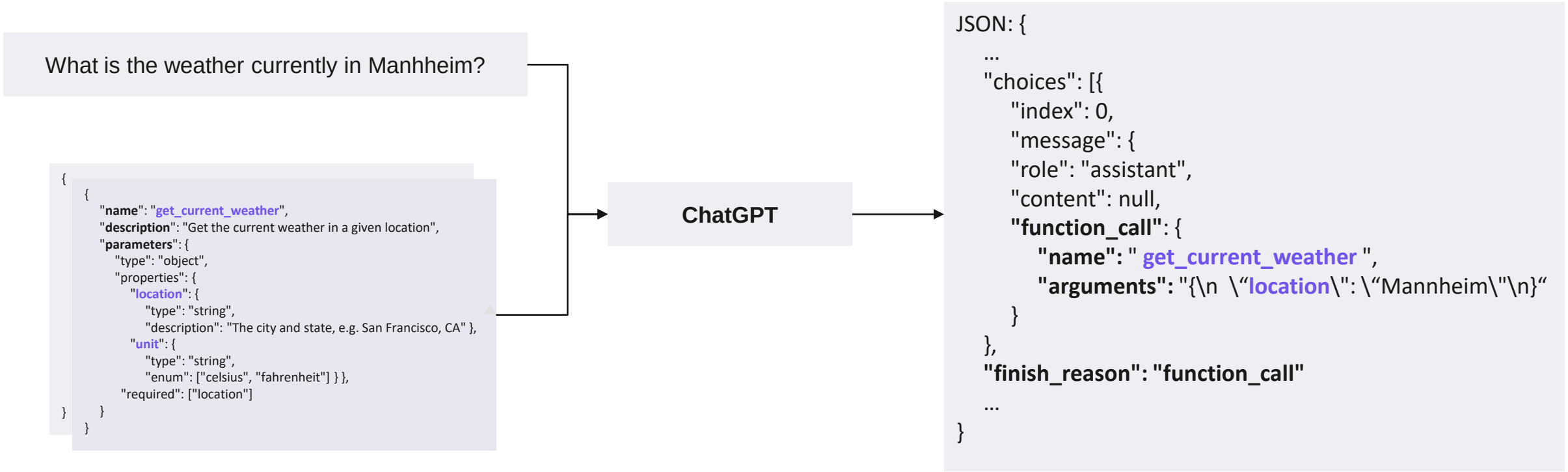
1. Model (ex. ChatGPT) is called using a query and a **set of functions**.
   - Functions are formulated as dictionaries that have the **key parameters**: name, description and its parameters

```
{
    "name": "get_current_weather",
    "description": "Get the current weather in a given location",
    "parameters": {
        "type": "object",
        "properties": {
            "location": {
                "type": "string",
                "description": "The city and state, e.g. San Francisco, CA" },
            "unit": {
                "type": "string",
                "enum": ["celsius", "fahrenheit"] } },
         "required": ["location"]
    }
}
```

https://openai.com/blog/function-calling-and-other-api-updates

Function Calling Steps

1. Model (ex. ChatGPT) is called using a **query** and a **set of functions**
2. The **Model decides** if it will call one of the functions passed to it.
   - If it decides to call one, it will return as **finish reason** a function call and will give the **parameters** with which to call the function.



https://openai.com/blog/function-calling-and-other-api-updates

Function Calling Steps

1. Model (ex. ChatGPT) is called using a query and a set of functions
2. Model decides if it will call one of the functions passed to it.
3. The function can be called **by the client** with the parameters returned from the model

```
JSON: {
    …
    "choices": [{
        "index": 0,
        "message": {
        "role": "assistant",
        "content": null,
        "function_call": {
            "name": " get_current_weather ",
            "arguments": "{\n  \“location\": \“Mannheim\"\n}“
        }
    },
    "finish_reason": "function_call"
    …
}
```

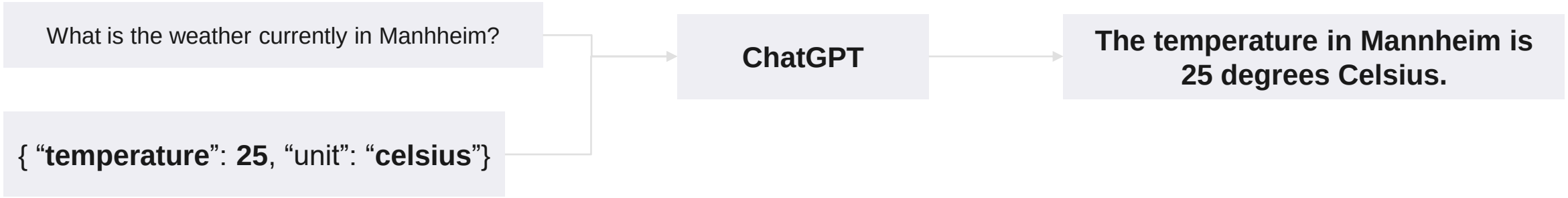
get_current_weather(location:string, unit:“Celsius”|”Fahrenheit”)

Function that queries an external weather API

https://openai.com/blog/function-calling-and-other-api-updates

Function Calling Steps

1. Model (ex. ChatGPT) is called using a query and a set of functions
2. Model decides if it will call one of the functions passed to it.
3. The function can be called with the parameters returned from the model
4. The function result together with the question are passed to the model again to summarize the result



https://openai.com/blog/function-calling-and-other-api-updates

Project Goals

1. Experiment and benchmark the function calling aspect of LLMs
   - Does the LLM decide to invoke the right functions?
   - How good is the LLM at translating textual questions into function calls?
   - Does it properly combine multiple external answers into an overall result?
2. Create a benchmark by designing test cases that require invoking functions
3. Evaluate the answers of ChatGPT and GPT-4 and conduct an error analysis to better understand the results
4. Publish the benchmark and your analysis on the Web

Learning Targets

Improve your technical skills

- Improve your technical expertise concerning **Large Language Models**
- Learn how to design and set up **Web APIs**
- Learn how to design **benchmarks** for testing AI models
- Improve your programming skills
- Work on the leading-edge AI topics

Improve your soft skills

- Work as part of a bigger team on a more complex project
- Organize yourself and assign tasks based on your skills
- Communicate and coordinate your work

Team Project Organization

Duration: 6 months (2 October 2023 – 22 March 2024)

Participants: 8 people

Type of work: Team and subgroup based

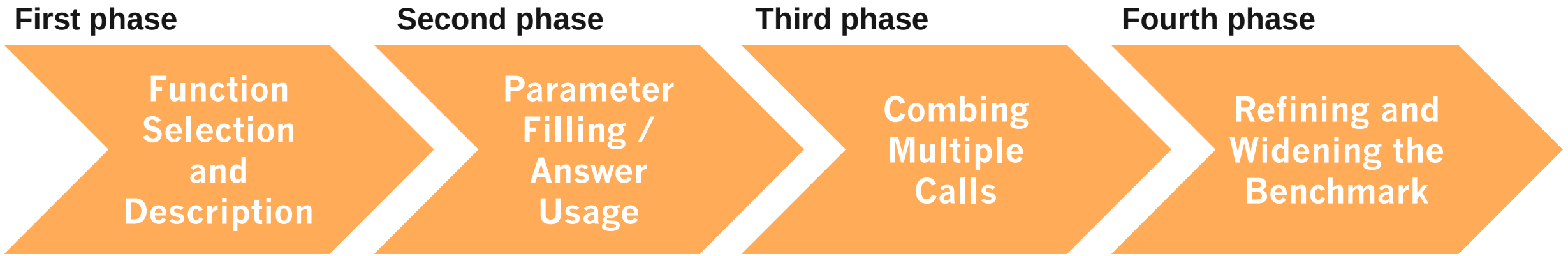
Milestones: 4 project phases

ECTS Points: 12

Evaluation

- Intermediate presentations
- Benchmark including documentation and code
- Individual contribution to the deliverables is graded

Team Project Organization: Project Phases



Phase 1: Function Selection and Description

Participants: all team members

Duration: 5 weeks

Sub-phases:

Phase 1.1: Topic Understanding (1 week)

- Read papers about the topic of LLMs, get familiar with OpenAI models, look at further examples regarding the LangChain library and function calling
- Sub-phase duration: 2 October 2023 – 9 October 2023

Phase 1.2: Use Case Selection and Initial Setup (3 weeks)

Phase 1.3: Analysis of Results (1 week)

Intermediate Presentation (Monday, 6 November 2023)

Getting familiar with the OpenAI API and OpenAI Models

- **First step:** Create OpenAI account https://openai.com/
  - To be able to use the models and also get a starting credit, you need to setup a paid account: https://platform.openai.com/account/billing/overview (later, we will cover some of your API costs)
  - You can also check how much you have spent in a day: https://platform.openai.com/account/usage
- **Second step:** Get familiar how the API works
  - A quick start tutorial by OpenAI: https://platform.openai.com/docs/quickstart
  - How are words tokenized in GPT models: https://platform.openai.com/tokenizer
  - Learn how to calculate your prompt token length so that you know beforehand how much a prompt will cost you: https://platform.openai.com/docs/guides/gpt/managing-tokens (Go to the Deep Dive: Counting tokens for chat API calls section)
  - To try some first prompts, you can use the Playground: https://platform.openai.com/playground (These prompts are also calculated in your costs!)
- Check **OpenAIs models**
  - https://platform.openai.com/docs/models/overview
  - For each model, its name, description and maximum token length is given.

LangChain Library

- Framework built for interacting with LLMs
- Offers **prompt templates** and **agent modules** which can be used for function calling
- Tutorials for getting started with LangChain:
  - **Youtube playlist**: https://www.youtube.com/watch?v=nE2skSRWTTs&list=PLIUOU7oqGTLieV9uTIFMm6_4PXg-hlN6F&ab_channel=JamesBriggs
  - **Handbook with code**: https://www.pinecone.io/learn/series/langchain/langchain-intro/
  - Code from our **repository**: https://github.com/wbsg-uni-mannheim/TabAnnGPT/tree/main
  - **OpenAI code examples** for function calling: https://github.com/openai/openai-cookbook/blob/main/examples/How_to_call_functions_with_chat_models.ipynb

Phase 1b: Use Case Selection and Initial Setup

- Use Case Selection, API Setup and Question Formulations
  1. select two use cases and related open license data sets
     - see Web Data Integration course for data sources
  2. form a sub-team to work on each use case
  3. select subset of the data and define a simple API over the data:
     - how to sample data e.g. 1000 entities, functions should involve single parameter
  4. formulate simple questions and different descriptions of API functions for your prompts
- Sub-phase duration: 9 October 2023 – 30 October 2023 (3 weeks)
- Deliverables: Use cases, function definitions, your own API endpoints descriptions (plus some examples for these endpoints) and draft of first questions

API selection: Use case ideas where to start

- Start by thinking about **use cases**:
  - **Music**: Formulate questions regarding music recordings, albums, genres
  - **Hotels and restaurants**: Combine questions about both domains, example you would want to find a hotel that is close to some restaurant that you like etc.
  - **Sports**: Some sport championships questions that use information about players, stadiums etc.
- Select some data sources/APIs and for the two chosen use case and create subgroups for each of them so that you work in parallel
- Example: Music Brainz offers information about artists, albums, recordings etc. https://musicbrainz.org/doc/MusicBrainz_API
- Set up your own APIs using Python and Flask
  - Some simple tutorials:
  - https://www.youtube.com/watch?v=MF75aNH3Gjs&ab_channel=JamesBriggs
  - https://www.youtube.com/watch?v=zsYIw6RXjfM&ab_channel=TechWithTim

Question Formulation: Ideas where to start

- In this phase **function selection** and **function descriptions** need to be tested. Questions related to these two points should be formulated for each use case that is chosen
- Start with at least **5 questions** per category and per use case and test different variations of them, here you can test for only **one parameter functions**
- Example questions that can test **functions selection**:
  - Retrieve the music recordings that were released in the spring of year 2000. → model needs to select the function that retrieves music recordings not music artists
  - Multiple versions of one function can exist: one that retrieves music recording based on the date and one that retrieves them based on the artist, therefore for the question above the model needs to chose the first function
- Example questions that can test **function description**:
  - Use words that are not included in the function descriptions
  - Function description: Retrieves music recording from external source.
  - Question: Retrieve the songs that were released in the spring of year 2000.

Phase 1c: Analysis of Results

Analyze the answers from questions from **both categories** of function selection and description and compare to non-augmented answers

- Error analysis: Categorize into correctly/incorrectly called functions, correctly/incorrectly sent parameters and correctly/incorrectly answer given.
- **Sub-phase duration:** 30 October 2023 – 6 November 2023
- **Deliverables:**
  1. Error analysis report and comparison results
  2. API python script
  3. test code which includes the functions implemented

Phase 2: Parameter Detection and Answer Usage

Participants: one subgroup per use case

Duration: 4 weeks

Sub-phases:

1. Questions formulations and functions implementation for testing parameter detection
   - In this part functions that have multiple parameters can be tested (example: from 2 to 4 parameters).
   - Build straightforward and questions that do not directly fit the wordings of the description/naming of the parameters of the function (e.g. if one parameter of the function is called place, write in the question „location“)
   - **Sub-phase duration:** 6 November 2023 – 27 November 2023
   - **Deliverables:** List of questions and test code that includes the functions for each of the use cases
2. Analysis of Results
   - Analyze the answers: Are the right parameter values returned in all cases? When are they not?
   - **Sub-phase duration:** 27 November 2023 – 4 December 2023
   - **Deliverables:** Evaluation and error analysis
3. **Intermediate Presentation** (4 December 2023)

Phase 3: Combining Multiple Calls

Participants: all team members or two subgroups per use case

Duration: 6 weeks

Sub-phases:

1. Questions formulations and functions implementation for testing multi-call workflows
   - Questions that require combining the results of calling several API functions
   - Experiment with Plan-and-Solve Prompting
     - Wang, Lei, et al. “Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models.” arXiv preprint arXiv:2305.04091 (2023).
     - python.langchain.com/docs/modules/agents/agent_types/plan_and_execute
   - **Sub-phase duration:** 4 December – 15 January
   - **Deliverables:** List of questions and functions (code) for each of the use cases
2. Analysis of Results
   - Analyze the answers: Are the right parameter values returned in all cases? When are they not?
   - **Sub-phase duration:** 15 January – 29 January
   - **Deliverables:** List of questions and functions (code) for each of the use cases

Phase 4: Refinement and Widening of Benchmark

Participants: all team members or two subgroups per use case

Duration: 8 weeks

Sub-phases:

1. Refine and Widen the Benchmark
   - Go back to the questions that you have formulated in all previous project phases and refine the based on your analysis and add further questions to widen the benchmark
   - **Sub-phase duration:** 29 January - 26 February
2. Final Error Analysis
   - **Sub-phase duration:** 26 February - 8 March 2024
3. Finalize Documentation for Publication
   - HTML page describing the benchmark and your experiments
   - Well documented code for API installer and test driver
   - **Sub-phase duration:** 9 March 2024 – 22 March 2024

Schedule

Date	Session
Tuesday, 26.09.2023	Kickoff meeting (today)
	Phase 1.1: Setup and Topic Understanding
	Phase 1.2: API Setup and Question Formulations
	Phase 1.3: Analysis of Results
Monday, 06.11.2023, 10:00	1st Intermediate Presentation: Use Cases, API, Questions, and Error Analysis
	Phase 2.1: Parameter detection and answer usage
	Phase 2.2: Error Analysis and evaluation
Monday, 04.12.2023	2nd Intermediate Presentation: Use Cases, API, Questions, and Error Analysis
	Phase 3.1: Combing Multiple Calls: Questions and Implementation
	Phase 3.2: Analysis of Results
Monday, 29.01.2024	3nd Intermediate Presentation: API, Questions Requiring Multiple Calls, and Error Analysis
	Phase 4.1: Combining Multiple Calls: Questions and Implementation
	Phase 4.2: Analysis of Results
	Phase 4.3: Finalizing the artefacts: Benchmark Documentation, API installer, test driver
Friday, 22.03.2024	Final Deliverable Deadline: Benchmark Documentation, API Installer, Test Driver
	Easter Break

Formal Requirements & Consultation

Deliverables

1. On the presentation dates provide us via e-mail with:
   - **Presentation slides**
   - **Task to member report**: excel sheet stating which team member conducted which subtask
   - **Code/Data**: link or zipped folder with your code and data

2. Final Deliverables
   - **Webpage** describing the benchmark and your experiments
   - **API Installer** for setting up the APIs locally in order to run benchmark
   - **Test Driver** to run benchmark and calculate results

All deliverables should be sent to Keti & Chris!

Formal Requirements & Consultation

Final grade

- 20% for each phase
- 20% for final deliverables
- Late submission: -0.3 per day

Consultation

- Send one e-mail per team or subgroup stating your questions to Keti
- Regular meetings at least every two weeks

Useful Software

- LangChain documentation
  - https://python.langchain.com/docs/get_started/introduction
  - https://www.pinecone.io/learn/series/langchain/langchain-intro/
- Team Cooperation
  - GitHub/Lab for the code base
  - Project Management Tool of your choice
- Processing and GPUs
  - Teaching GPU-Server
  - Google Colab: https://colab.research.google.com/
  - BwUniCluster2.0: https://wiki.bwhpc.de/e/Category:BwUniCluster_2.0

Related Work: Large Language Models

- Zhao, Wayne Xin, et al. "**A survey of large language models.**" arXiv:2303.18223 [cs.CL] (2023).
- Mialon, et al.: **Augmented Language Models: a Survey**. arXiv:2302.07842 [cs.CL]
- Ouyang, Long, et al. "**Training language models to follow instructions with human feedback.**" Advances in Neural Information Processing Systems 35 (2022): 27730-27744.
- Borji, Ali. "**A categorical archive of chatgpt failures.**" arXiv:2302.03494 (2023).
- Bang, Yejin, et al. "**A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity.**" arXiv:2302.04023 (2023).
- Wei, Jason, et al. "**Emergent abilities of large language models.**" arXiv:2206.07682 (2022).
- Wei, Jason, et al. "**Chain-of-thought prompting elicits reasoning in large language models.**" Advances in Neural Information Processing Systems 35 (2022): 24824-24837.
- Wang, Lei, et al. "**Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models.**" arXiv preprint arXiv:2305.04091 (2023)
- Schick, Timo, et al. "**Toolformer: Language models can teach themselves to use tools.**" arXiv:2302.04761 (2023).
- He, Hangfeng, Hongming Zhang, and Dan Roth. "**Rethinking with retrieval: Faithful large language model inference."** arXiv preprint arXiv:2301.00303 (2022).

Questions?