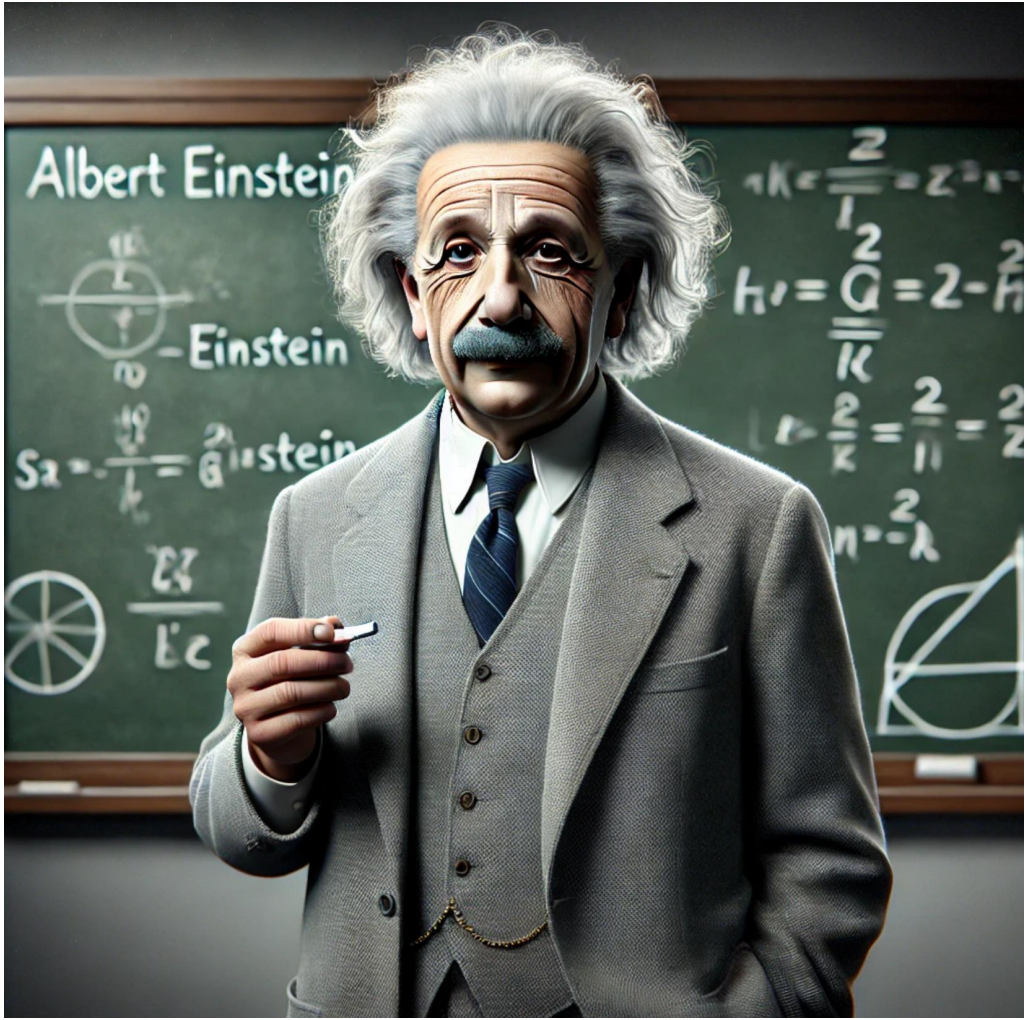
Prompting Large Language Models for Text Classification

Outline

1. 14:00 – 14:45 **Text annotation: key concepts, tasks, and best practices**
2. 14:45 – 16:00 **Prompting engineering**

   16:00 – 16:15 *break*
3. 16:15 – 17:00 **Automation with R**
4. 17:00 – 17:30 **Evaluating classification performance**


GPT-3
LLAMA
GPT-4
AYA
CLAUDE
LAMBDA


Albert Einstein
Einstein

EINSTEIN…

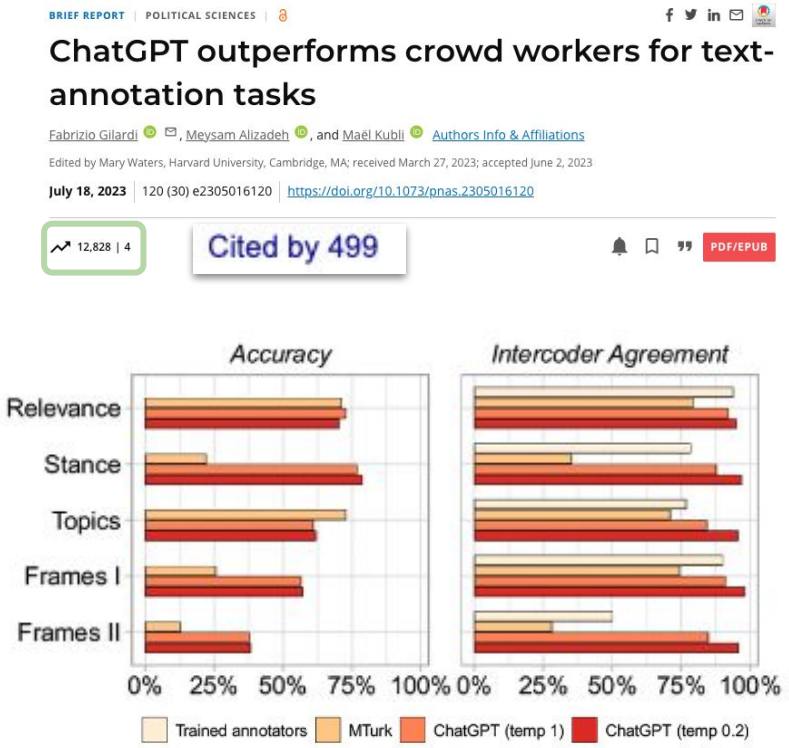
- can make mistakes
- can make wrong conclusions
- can confuse
- can misunderstand you

BUT THE BIGGEST LIMITATION IS YOUR IMAGINATION.

Motivation

LLMs as text annotators!?

- November 2022: OpenAI releases ChatGPT, an LLM chat assistant fine-tuned on GPT 3.5
- quickly arouses interest of people in the text-as-data crowd:
  - "Can we stop collecting annotations from trained RAs or crowd coders?"
  - "Can we stop training/fine-tuning supervised text classifiers?"
- Gilardi et al. (2023) suggest "yes" 👉
- Others tend to agree (e.g. Ziems et al., 2023)
- While again others are more vary (e.g., Palmer et al., 2024; Reiss, 2023)



Today’s learning objectives

Main goal

learn how to use LLMs for text classification and annotation tasks (a.k.a. ***text coding***)

- convert coding instructions and schemes into prompts
- use python, the OpenAI API or ollama to automate the text coding process

On the fly …

- gain a basic understanding of how LLMs function and their characteristics
- learn about different approaches for LLM-based text coding and annotation
- know best practices for writing and optimizing LLM instructions ("prompts")
- know how to facilitate reproducibility when using LLMs

Text annotation

key concepts, tasks, and best practices

Quantitative content analysis

Definition
A methodology for systematically analyzing and quantifying communication phenomena in text (or other media)

Ingredients

1. a collection of to-be-analyzed **texts**
2. a **conceptualization** of the to-be-quantified phenomenon
3. a **measurement procedure**
    - ***manual* content analysis**: reading, interpreting, and coding by human coders
    - ***automated* content analysis**: automation with some instrument or model

Klaus Krippendorff's take on content analysis

> *a research technique for making replicable and valid inferences from texts ... relative to the contexts of their use* (Krippendorff, 2004, 19)

- "content" is an *emergent property* resulting from reading, interpreting, and analyzing a text in context
- text is always *qualitative*; categorizing textual units is the most elementary for form of measurement

History of content analysis in political science

overview of developments in *political (and communication) science* research

1. *1980*s and '*90*s: manual content analysis (e.g., *Comparative Manifesto Project*; cf. Budge & Laver, 1986; Klingelmann et al., 1994)
2. early *2002*s: first attempts at automation (e.g., party manifestos coding and scaling methods; Gabel & Huber, 2000; Laver & Garry, 2000; Laver, Benoit & Garry, 2003)
3. starting in the *2010*s: spill-over of ML techniques and bag-of-words NLP methods
    - supervised text classification (Hillard et al., 2008; D'Orazio et al., 2014)
    - topic modeling (Grimmer 2010; Quinn et al., 2010)
4. since ca. 2020: increasing adoption of deep learning-based NLP methods
    - word embeddings (Rodman, 2020; Rudkowsky et al., 2018; Rheault & Cochrane, 2020; cf. Rodriguez & Spirling, 2021)
    - neural network classifiers (e.g., van Atteveldt et al., 2021, Dai & Kustov, 2022)
5. since inception of Transformer models (BERT and GPT), increasing adoption of transfer learning approach

Producing quantitative data with CA

Krippendorff lists three crucial steps (2004, pp. 83-7)

1. unitizing
2. sampling
3. coding

Producing quantitative data with CA

Krippendorff lists three crucial steps (2004, pp. 83-7)

1. **unitizing**
2. sampling
3. coding

Unitizing

- communication phenomena can manifest at different textual levels
  - full documents (e.g., speeches)
  - sentences or paragraphs
  - multi-word expressions and phrases
- raw texts need to be processed and segmented into appropriate textual units (i.e., “unitized”) before they are distributed for manual coding

Producing quantitative data with CA

Krippendorff lists three crucial steps (2004, pp. 83-7)

1. unitizing
2. **sampling** (rather document/corpus selection)
3. coding

Sampling

texts selected for coding must enable researchers' to draw generalizable conclusions (classic case selection problem)

Important

If content-analyzed text units serve as inputs to **machine learning** procedures, there is an additional sampling step, where we (randomly) select texts that are coded by humans for training or validation

Producing quantitative data with CA

Krippendorff lists three crucial steps (2004, pp. 83-7)

1. unitizing
2. sampling
3. **coding**

Coding

- transfer analysts' *qualitative readings* of a text into *quantitative categories* based on observer-independent rules
- goal: generate numerical representations of text based on qualitative information

Coding instructions specify

1. the qualifications coders need to have
2. the training coders must undergo before coding
3. the **coding task** and **coding scheme**
4. the **output format** of human codings

Coding tasks

Document classification

assign documents to discrete set of pre-defined, mutually exclusive categories

- "document" can refer to a sentence, a paragraph, a headline, etc.
- *examples*
  - sentiment classification
  - policy topic classification
  - toxicity or incivility detection

Note: this is the default in applied political science research, thus often used simply referred to as "text classification" (although it is broader)

Variants

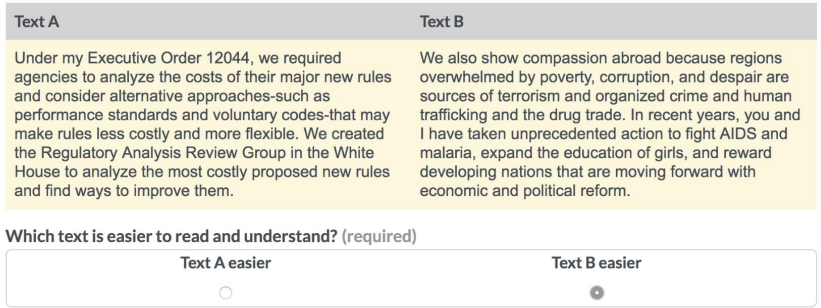
- *single*-label classification: each text can be assigned to one and only one category
  - binary classification: only two categories
  - multi-class classification: 3+ categories
- *multi*-label classification: each text can be assigned to none, one, or more categories (cf. Erlich et al., 2022)

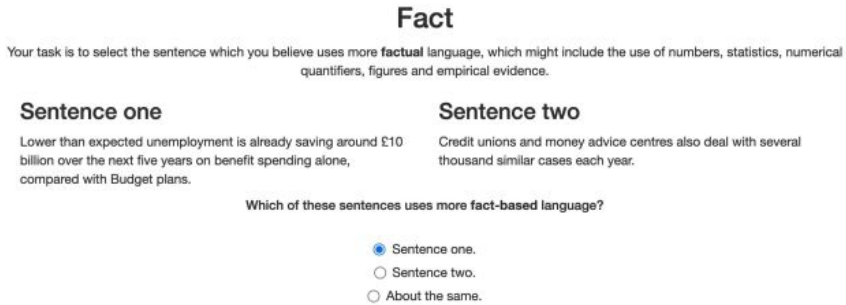
Coding tasks

Pairwise comparison

rank alternatives on a (latent) conceptual continuum in terms of their intensity

- *examples*
  - political sophistication (Benoit et al., 2019)
  - emotionality, factuality, human narrative, complexity, etc. (Hargrave & Blumenau, 2022)





Coding tasks

Word-level classification

detect (and extract) phrases and multi-word expressions referring to mutually exclusive categories/types

- a.k.a. as token classification
- *examples*
  - *Named Entity Recognition*
  - social group mention detection (Licht & Sczepanski, 2024)

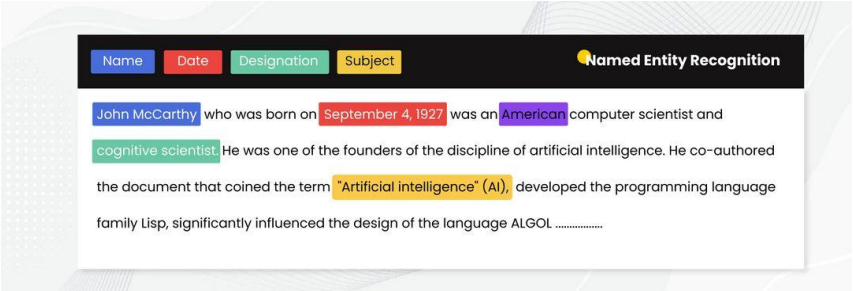


Table 1. Examples of group mentions in sentences drawn from British mainstream party manifestos. Highlighted text spans identify groups mentioned in each sentence.

We seek to bring about a fundamental change in the balance of power and wealth in favour of working people and their families.
Eight years of meanness towards the needy in our country and towards the wretched of the world.
The welfare of the old, the sick, the handicapped and the deprived has also suffered under Labour.
Labour recognises the special needs of people who live and work in rural areas.

Coding scheme and instructions

must provide

- a definition and description of the **concept** you want to measure
- the names of the **categories** into which you want to code text units
- definitions and descriptions of the categories
- instructions for how to record outputs (e.g., “select the label category and insert it in the column next to the text”)

Important

Careful ***conceptualization*** of the phenomenon you want to quantify is thus an integral part of the content-analytic approach

Consult the literature for guidance, e.g.,

- Collier (2008)
- Schedler (2010)
- Gortz (2020 [2006])

Exercise

Identify coding scheme ingredients

1. form a group with your table neighbor
2. consult the README.md file a paper in the labeled data folder in our Github repository to answer the following questions 👉

Examples

check out the original instructions used in Benoit et al. (2016)

Questions

1. What concept do they want to measure?
2. How do they define and describe the concept?
3. What coding categories do they provide?
4. How do they define and describe each of the coding categories (if at all)?
5. What further instructions and information do they provide (if any)?

Note: if you need the original paper and can't download it via your uni's VPN, ask us

PROMPT ENGINEERING

Medium Search Write

Prompt Engineering: The Key Skill for the 21st Century

Deepak Ravindran · Follow

2 min read · May 24, 2024

PROMPT ENGINEERING

Prompt Engineering: The New Literacy for an AI-Driven World

Goran Trajkovski
Senior Leader in Learning Science and Data Analytics | Expert in Educational Technology, Strategic Planning, and Program Development | Transforming Education Through Innovation in AI and Data Science

+ Folgen

April 17, 2024

In the rapidly evolving landscape of artificial intelligence, a new skill is emerging as a critical literacy for the 21st century: prompt engineering. Just as coding literacy has become essential in

Guest

Why prompt engineering is one of the most valuable skills today

VentureBeat/Midjourney

SHOT

SHOT = EXAMPLE

SHOT = EXAMPLE

ZERO SHOT PROMPTING

ONE SHOT PROMPTING

FEW SHOT PROMPTING

3 techniques for using LLMs for text coding and annotation

Zero-shot prompting

1. craft an instruction ("prompt") that describes the task at hand
2. feed the instruction and a to-be-classified text into the model
3. the model generates a response

the **response** is interpreted as the model's **classification**

the model's parameters are *not* updated; all "learning" occurs from instructions and examples *in the context*

Few-shot prompting

same as zero-shot prompting

but you include examples (inputs and desired responses)

- in the input *or*
- in the prompt itself

this demonstrates the desired response behavior to the model

Instruction tuning

1. combine your task-specific instruction with example inputs and desired responses
2. fine-tune the model using this text-to-text formatted training data ⇒ the model is optimized for your task
3. apply the fine-tuned model to unlabeled texts (+ the instructions)

Zero-shot in-context learning

Zero-shot Prompting

Prompting: Instruct an LLM to generate a response for some user input(s) *without* prior task-specific training.

Note: a more precise term is in-context learning (Brown *et al.*, 2020)

Requirements

1. translate coding instructions into a prompt
2. provide the LLM with texts that it should classify or annotate

Example

GPT-4o codes tweets' sentiment

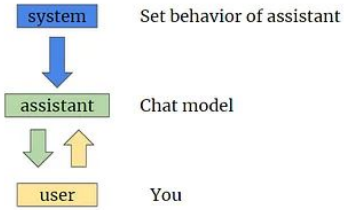
https://platform.openai.com/playground/p/GizUE1aEN2aIXkycL9SSZxZq?model=undefined&mode=chat

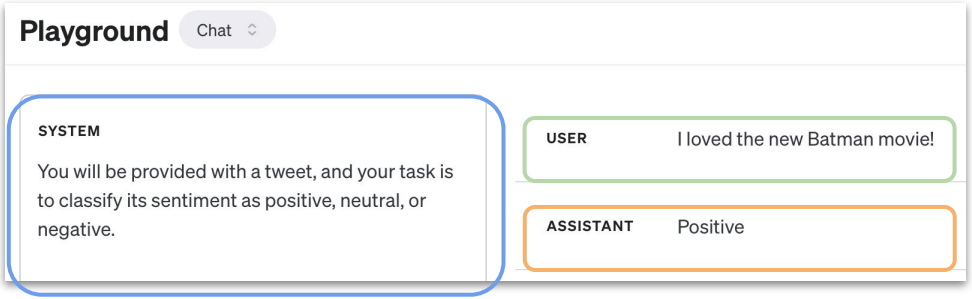
Zero-shot prompting with ChatGPT

- inputs and model responses are embedded in a ***conversation history***
- each *conversation history* is composed of three types of ***messages***
  - a system message
  - a user messages
  - an assistant messages

Role

```
messages =
[
      {“role”: “system”, “content”: “You are an assistant ..... “},
      {“role”: “user”, “content”: “tell me a joke“},
      {“role”: “assistant”, “content”: “Why did the chicken cross the road“},
      {“role”: “user”, “content”: “I don't know“},
      ....
]
```





Zero-shot prompting with ChatGPT

The system message

- provides the LLM assistant with behavioral instructions ⇒ conditions its behavior
- hence only defined once at the beginning of a conversation
- *after* the system message, conversations can be as short as one message or include many back and forth turns between the user and the assistant

Note: the system message is *optional* (without it, the model will behave as if it had received a generic system message like “You are a helpful assistant.”)

The user message

- provides requests or comments for the assistant to respond to
- in our applications, the to-be-coded text

The assistant message

- the model's response given the instructions in the system message and the remaining conversation history)
- in our applications, the model’s classification or annotation(s) of the input text

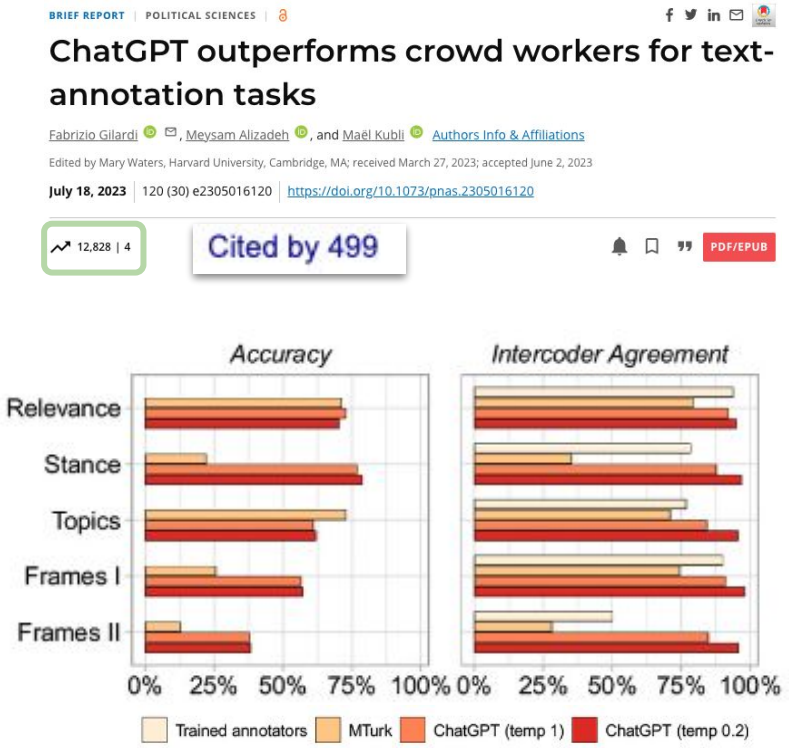
Let’s code: Using OpenAI’s chat models in python

- The instructions continue in the notebook llm_openai_chat_intro.ipynb

Motivation

LLMs as text annotators!?

- November 2022: OpenAI releases ChatGPT, an LLM chat assistant fine-tuned on GPT 3.5
- quickly arouses interest of people in the text-as-data crowd:
  - "Can we stop collecting annotations from trained RAs or crowd coders?"
  - "Can we stop training/fine-tuning supervised text classifiers?"
- Gilardi et al. (2023) suggest "yes" 👉
- Others tend to agree (e.g. Ziems et al., 2023)
- While again others are more vary (e.g., Palmer et al., 2024; Reiss, 2023)



Exercise

Identify coding scheme ingredients

1. form a group with your table neighbor(s)
2. go to the labeled data folder for the Benoit et al. (2016) paper
3. read the README.md and the files in the `instructions/` subfolder file to answer the following questions 👉

Take 20-30 minutes

Questions

1. What concepts do they want to measure?
2. How do they define and describe the concepts?
3. What coding categories do they provide?
4. How do they define and describe each of the coding categories (if at all)?
5. What further instructions and information do they provide (if any)?

Note: if you need the original paper and can't download it via your uni's VPN, ask us

Steps for automating zero-shot LLM classification

Step 1: Prepare the text data

1. Load the data into your python session, ideally as a `pandas` dataframe
2. clean the texts (e.g., remove line breaks, remove leading and trailing white spaces, remove fancy quote marks, etc.)
3. If you have labels for (some) of the texts: do not discard them, they'll be useful for evaluation

Step 2: Prepare the prompt

1. Describe what the input will look like
2. Describe the task
3. Name and define the coding categories
4. (optionally) give instructions for the desired output format (e.g., "only reply with the chosen category")

Steps for implementing few-shot LLM prompting

Step 3: Define a function for classifying a text

1. function should take (min.) three arguments:
    a. `text`
    b. `model`
    c. `system_message`
2. function then
    a. construct conversation history
    b. queries the model API for a response
    c. parses the response object for the generated text ⇒ the classification
    d. returns the classification

Step 4: Iterate over texts

1. convert the texts into a list of strings
2. iterate over individual texts
    a. call the classification function on each text on at a time
    b. save the classification in a list
3. add the LLM classifications to the data frame

Let’s code: Using LLMs for zero-shot text classification

The instructions continue in the notebook llm_zeroshot_classification_openai.ipynb

Writing prompts: best practices (source)

ask the model to adopt a “persona” to specify the system's desired response behavior

- a persona is description of a personality with certain characteristics (e.g., abilities or specializations)
- Example:

```
Act as a social science researcher versatile in analyzing political communication in parliamentary debates
```

specify the context and goal of your task in the instructions

- avoids that the model needs to guess the context of your request
- can results in more relevant answers
- Example:

```
You will be provided with a sentence taken from a speech delivered in the UK House of Commons, the lower house of the parliament of the United Kingdom. Your task is to …
```

Writing prompts: best practices

specify the steps required to complete a task

- whenever possible, break down your task into a sequence of steps (a.k.a a “chain of thought”)
- Example: In the system prompt write

```
Please follow the following steps to
generate your response:

1. Carefully read the text.
2. Assess whether it contains [CONCEPT]. If
   not, return “None” as your response.
3. Otherwise, classify the text into one of
   the following categories: …
```

specify the desired length and format of the output

- in the system message, state the desired length in terms of the count of words, sentences, paragraphs, bullet points, etc.
- Example:

```
Only respond with the name of the selected
category or “None”. Omit any further text or
explanations!
```

Writing prompts: best practices

use delimiters to clearly indicate distinct parts of the input

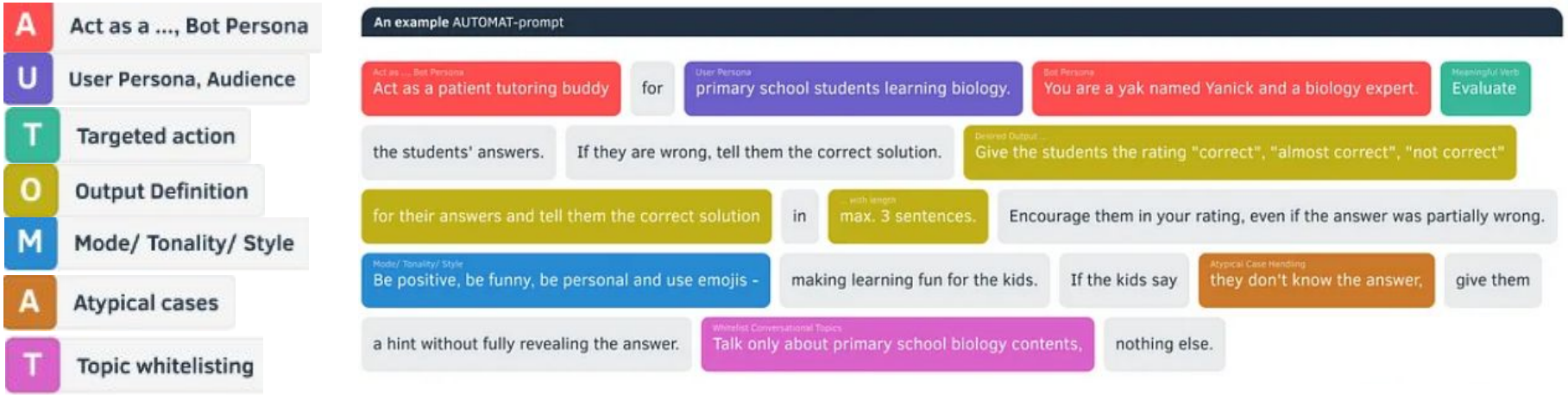
- Example: in the system message, add

  ```
  The text of the sentence will be enclosed
  in triple quotes.
  ```

  Then wrap the to-be-classified text in triple quotes like this:

  ```
  '''Bla bla bla'''
  ```

  Note: This focusses the model on the relevant text, even if it contains questions or quotes



https://big-picture.com/media/the_prompt_engineering_cheat_sheet.pdf

Exercise

Continue working with your group partners:

1. select one of the coding instructions files data/labaled/benoit_crowdsourced_2016/
2. translate the coding instruction into a prompt
3. load the corresponding data (loading_benoit_crowdsourced_2016_data.ipynb)
4. identify “gold standard” examples (text + desired classification) using the metadata__gold columns
5. in the OpenAI chat playground
    a. input your prompt as *system* message
    b. input a gold-standard example **text** as a *user* messages
    c. click “submit”
    d. get the models classification

Prompt template (for document classification)

```
You will be provided with a text as input. Your task is to <insert a one-sentence description of the task>.

<optional: insert a definition of the target concept>

Categorize the text into one of the following categories: <insert comma-separated list of categories>

<optional: insert definitions of the available categories>

Only reply with the selected category. Omit any other text or explanations.
```

Few-shot in-context learning

Techniques for using LLMs for text coding and annotation

Zero-shot prompting

1. craft an instruction (“prompt”) that describes the task at hand
2. feed the instruction and a to-be-classified text into the model
3. the model generates a response

the response is interpreted as the model's classification

Few-shot prompting

same as zero-shot prompting

but you include examples (inputs and desired responses)

- in the input *or*
- in the prompt itself

this demonstrates the desired response behavior to the model

but the model's parameters are not updated; all learning occurs *in context* ⇒ in-context learning

Prompt-based instruction tuning

1. combine your task-specific instruction with example inputs and desired responses
2. fine-tune the model using this text-to-text formatted training data ⇒ the model is optimized for your task
3. apply the fine-tuned model to unlabeled texts (+ the instructions)

Requires access to GPU computing resources

Using LLMs for few-shot text coding

LLMs can be provided with examples when tasked to classify/annotate texts

1. system message: provide coding instructions and specify the categories and output format
2. use one or more ***examples***, for each
   a. user message: example text
   b. assistant message: “true” label
3. user message: finally, the to-be-classified text

the response by the LLM containing its classification of the to-be-classified text

Few-shot in-context learning

> *the model is conditioned on a natural language instruction and/or a few demonstrations of the task and is then expected to complete further instances of the task simply by predicting what comes next. …*
>
> *We further specialize the description to “zero-shot”, “one-shot”, or “few-shot” depending on how many demonstrations are provided at inference time.*
>
> – Brown et al. (2020)

Example notebook

Let’s look at how to implement few-shot learning based on the notebook

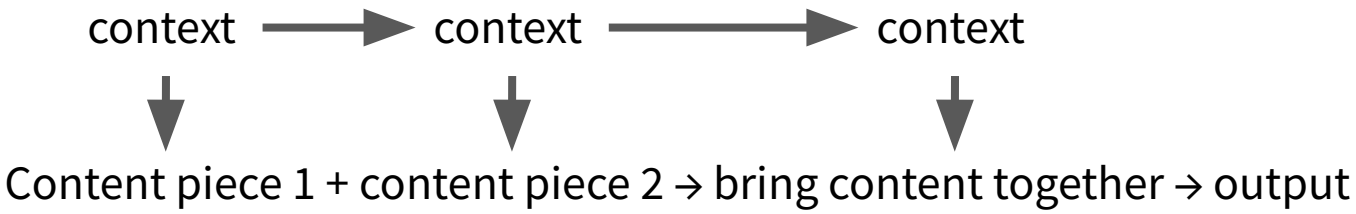
notebooks/llm_fewshot_classification_openai.ipynb

MULTIPROMPT APPROACH WORKS BEST if you have more than one moving part

SCENARIO 1

Context → Context → Context → Context → … → good output

SCENARIO 2



(Known) Pitfalls

LLMs tend to be ***biased*** towards certain answer categories during few-shot learning

1. **majority label bias**: predict the label class(es) that appear frequently in the few-shot prompt
2. **recency bias**: predict the label class(es) of examples that appear near the end of the few-shot prompt
3. **common token bias**: model prefer answer categories that are frequent in its pre-training data (e.g., the letters E and A compared to B, C, and D; source)

Solutions

Calibration (Zhao et al. 2021)!? but maybe not (Zhang et al. 2023)

- an open question
- no clear best practices yet

The “class A”-prompt

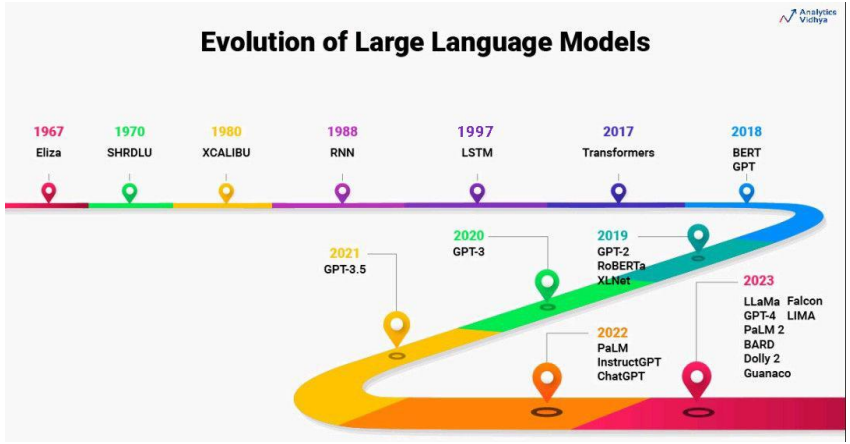
1. INSTRUCTIONS using the AUTOMATE framework
2. FEW SHOT LEARNING
3. DATA CONTEXT
4. OUTPUT FORMAT
5. WHAT HAPPENED BEFORE (context that happened immediately before)

LLM pre-training

Large Language Models

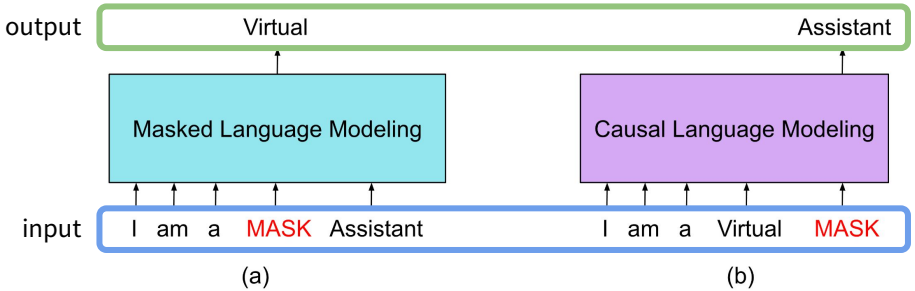
In simple terms …

- advanced *neural network models* designed for understanding, generating, and interacting with human language
- trained on large amounts of text data
- can recognize the nuances, contexts, and structures of language
- optimized to perform various tasks
  - answering questions
  - summarizing texts
  - translating languages
  - generating new text
  - **giving replies that follow instructions**



LLM pre-training

- **training** in machine learning: use labeled data (input–output pairs) to optimize a prediction model



Note: language modeling is "*self*-supervised"

- causal LM: the observed next word is the label
- masked LM : the masked-out word is the label

LLM pre-training

Motivation

- mimic humans' text generation behavior by training a model for causal language modeling task on large text corpus
- model learns
  - how words co-occur
  - how sentences, paragraphs, and documents are composed
  - what words, sentences, etc. mean in context
- can be used for *transfer learning*

Transfer learning

machine learning approach to reuse general-purpose model for a specific task

intuition knowledge gained while learning to perform a general task (e.g., language modeling) can be applied to quickly solve related task (e.g., sentiment classification)

premise general features learned during pre-training can be relevant for more specific tasks

From LLMs to chat assistants

- pre-trained LLMs like GPT 4 can generate text that resembles human-generated text
- *but* prompted with some text input, they just complete it with a probable sequence of words ⇒ often gives *undesirable* completions
- to **align LLMs' outputs with user's intent**, they need to be able to comprehend and **follow instructions**
- *conversational* LLMs like ChatGPT are so versatile and popular because of their instruction-following capabilities

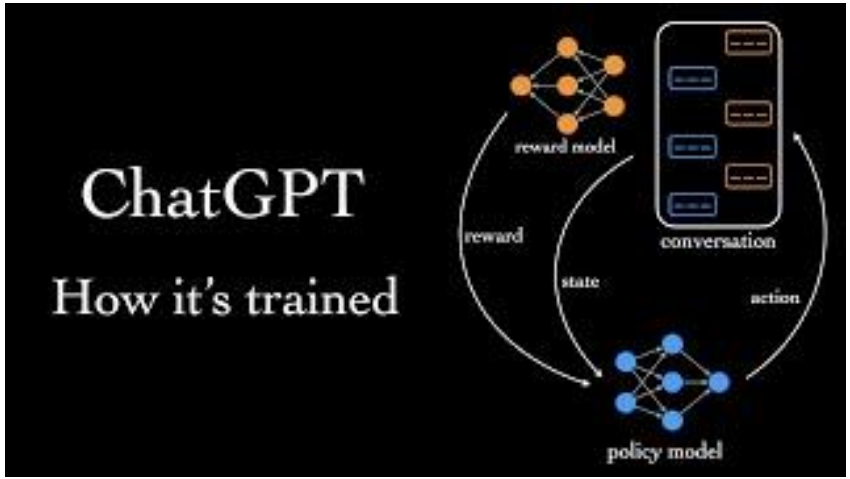
Methodological motivation for chat assistants

- there are variety of NLP tasks
  - sentiment analysis
  - question answering
  - (extractive) summarization
  - entity recognition
  - etc.
- have been studies separately with specialized methods
- these tasks can be converted into text-to-text problems ⇒ allows focusing on developing general-purpose instruction-following models
- makes NLP more accessible and flexible

ChatGPT

Building blocks

1. a GPT model: the underlying language model (generative pre-trained LLM)
2. instruction/conversational datasets for supervised fine-tuning
3. reinforcement learning with human feedback



https://www.youtube.com/watch?v=VPRSBzXzavo

ChatGPT **Building block 1: a GPT model**

Training data

- a diverse collection of internet text, including books, websites, blogs, online fora (e.g. stack overflow, reddit)
- covers a wide range of topics, languages, and writing styles

Tokenization

- text are broken down into smaller *tokens*
- tokens represent words, parts of words, or punctuation
- special tokens help the model understand text boundaries and structural elements

Text preprocessing

- raw content cleaned to remove any non-textual information
- cleaned texts converted into standardize input format

Model Architecture

an autoregressive decoder-only Transformer

- an handle long-range dependencies in text
- optimized for generating text/text completion

see Brown et al. (2020)

ChatGPT Building block 2: supervised fine-tuning

Supervised training for instruction following

- take the pre-trained GPT model (building block 1)
- take datasets recording prompts and correct completions
- input prompt, predict response
- compare to “correct” completions to optimize the model’s responses (fine-tuning)

Intuition the model learns to imitate an “ideal” chat bot’s response behavior (conditional on prompts and conversation histories)

introduced for *InstructGPT* (Quyang et al., 2022)

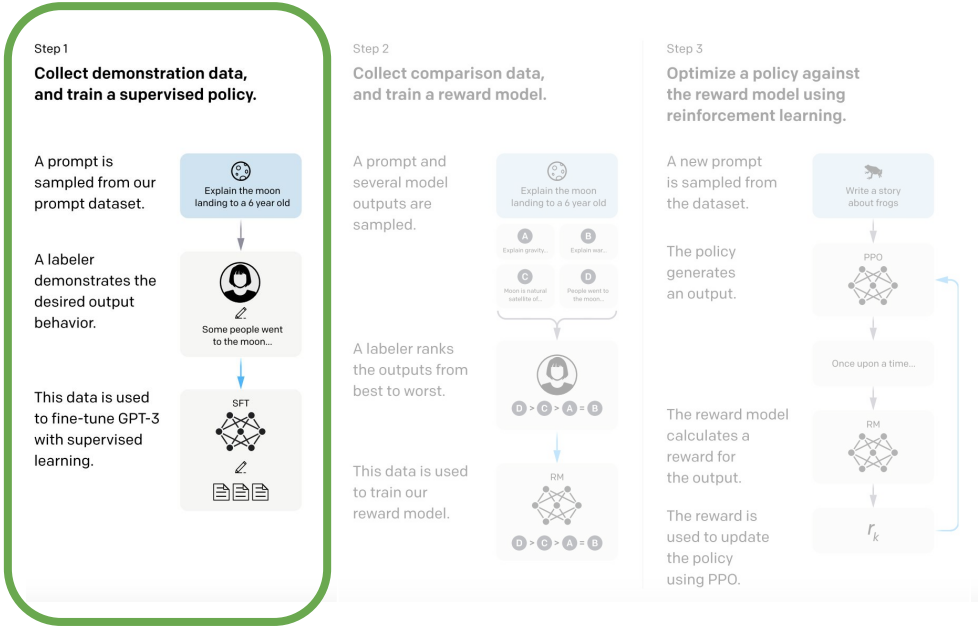


Figure 2 in Quyang et al. (2022)

ChatGPT **Building block 3: reinforcement learning with human feedback**

Intuition

- treat the model as an agent (not as a passive observer)
- fine-tune the model to generate human-preferred responses

Advantages

- this shifts the focus from mimicking the pre-training data to producing desirable/favorable completions
- consequently, the model adapts its understanding of users' desired outputs

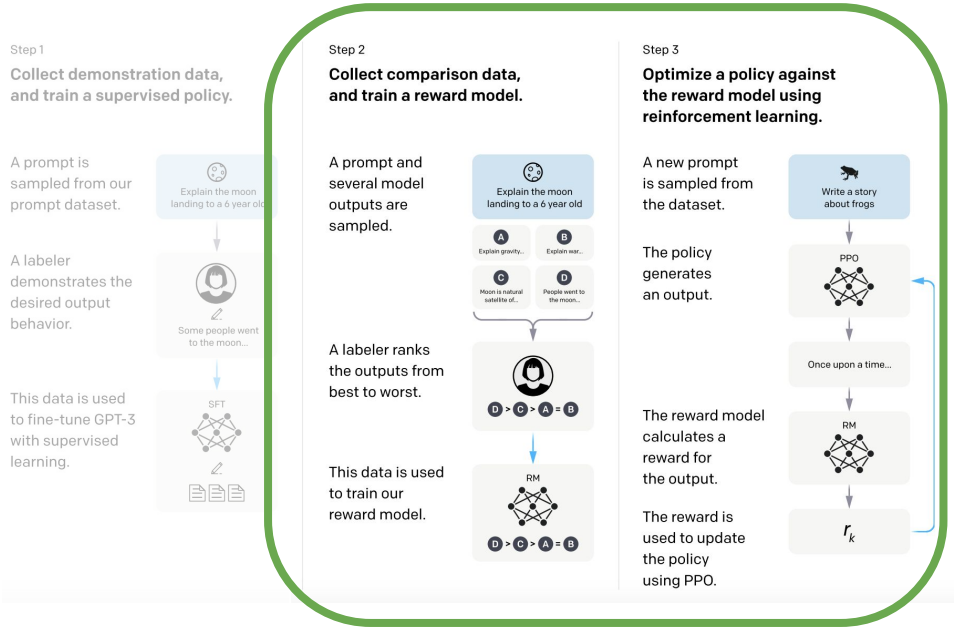


Figure 2 in Quyang et al. (2022)

ChatGPT **Building block 3: reinforcement learning with human feedback**

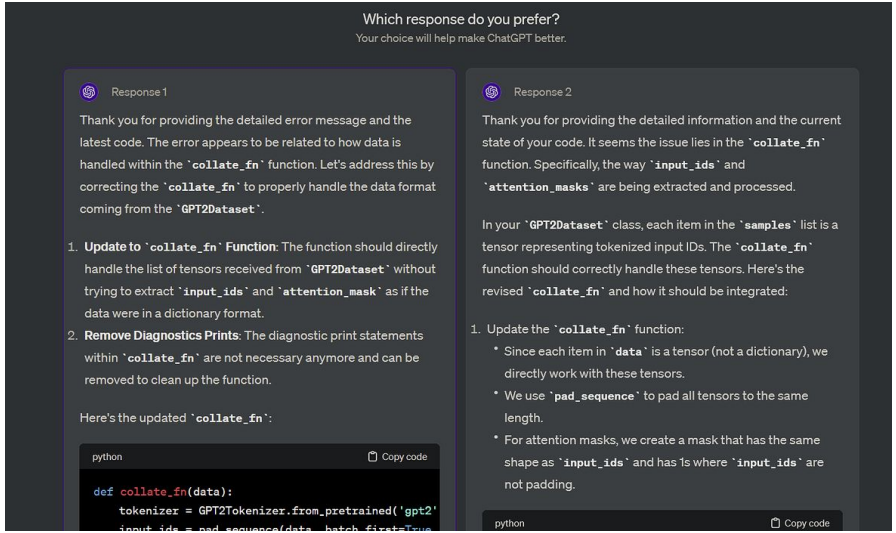
Intuition

the instruction-finetuned model is used as

1. *reward model*: learns to predicts reward associated with a generated completion (given prior conversation history) based on human feedback
2. *policy model*: selects “optimal” completions given prior conversation history and predicted reward

The policy model is optimized on batches of human-rated prompt–response pairs

Human feedback through pairwise comparison



https://community.openai.com/t/incorrect-use-of-ui-ux-when-choosing-which-response-do-you-prefer/596139

ChatGPT **Building block 3: reinforcement learning with human feedback**

Intuition

the instruction-finetuned model is used as

1. *reward model*: learns to predicts reward associated with a generated completion (given prior conversation history) based on human feedback
2. *policy model*: selects “optimal” completions given prior conversation history and predicted reward

The policy model is optimized on batches of human-rated prompt–response pairs

For more details, I recommend watching this lecture:

https://www.youtube.com/live/2MBJOuVq380?si=gxqYl1EOqFIrRAUg&t=493

Using open-weight LLMs

Closed- vs. open-weights LLMs

Closed-weights LLMs

- developed, trained, and owned by companies (e.g, OpenAi's GPTs, Anthropics *Claude*)
- often impressively performant

but

- no free use of model's weights/parameters
- no access to code and training data
  ⇒ limits transparency and replicability
- questionable data usage ethics and regulations ⇒ problematic when your data is sensitive (e.g., survey or interview data)

Open-weights LLMs

- developed by companies (e.g. META's Llama models) or the community (e.g., BLOOM)
- model weights released for public use
- can perform (almost) as well as closed-weights models (dep. on the task)
- great for transparency, reproducibility, and data privacy
- can be run locally on some consumer-grade hardware (especially when "quantized")

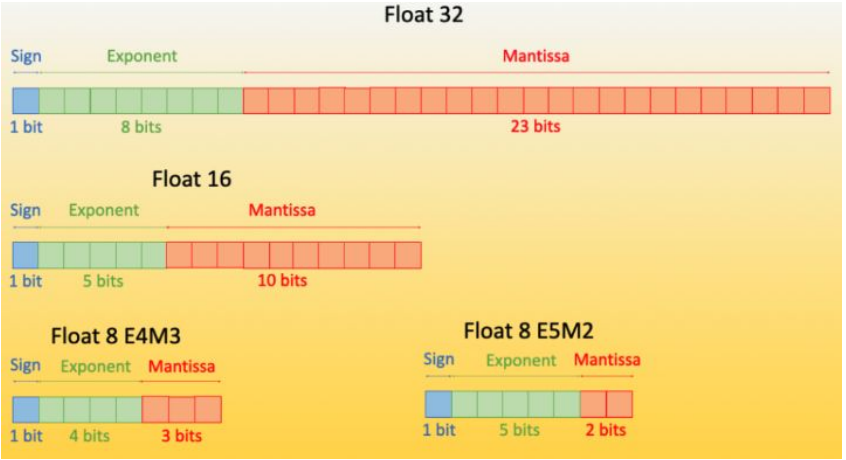
Examples

Llama 3.1, Phi 3, Gemma Mixtral, etc. (see here)

Using open-source LLMs through quantization

Quantization

- pre-trained LLMs have several billion parameters, resulting in very large model file sizes ⇒ prohibitive RAM requirements
- *quantization* is a method for reducing model file size ⇒ allows loading and computing und basic consumer hardware (e.g. MacBook Pro M1)
- works by reducing the *precision* of model parameters (see figure to the left) and uncompressing them on demand during computation



Example

`Llama 3.1 8B` has 8 billion parameters but its 4-bit quantized version is only 4.7 GB large

Using quantized LLMs with `ollama`

`ollama`

`ollama` is an open-source software package for running open-source quantized LLMs on consumer grade hardware (e.g., your MacBook or Laptop)

- install: https://ollama.com/
- download models: see here https://ollama.com/library

Using `ollama` in python

the `ollama` library allows calling LLMs run with ollama in your python session

- documentation: https://github.com/ollama/ollama-python
- example code:

```
from ollama import Client
client = Client()
MODEL = "llama3.1:8b"
messages = [
    {
        "role": "user",
        "content:" "What is the GESIS Fall Seminar?"
    }
]
client.chat(
     model= MODEL,
     messages=messages,
)
```

notebooks/llm_zeroshot_classification_ollama.ipynb

Reproducibility (and its limits)

Replicability and Reproducibility

Reproducibility

- ability to generate the reported results of an existing study again using the same data and methods
- this verifies the operationality and accurateness of the reported data analysis and computational procedures

Note: Reproducibility is also a requirement for successful acceptance at an increasing number of journals

Replicability

- replication means to conduct a new study with the same conceptual and methodological framework of an existing study but using *new* data
- tests the generalizability and robustness of findings across different conditions by ensuring they are not specific to particular datasets or analytical choices
- should build confidence in study

Reproducibility with LLMs

LLMs are non-deterministic! (see here or here)

- floating point operations in modern GPUs are inherently non-deterministic (see here and here)
- also: modern LLM architectures, maybe incl. GPT-4, use mixtures of experts ⇒ which component of the model contributes (how much) to the text generation is stochastic
- this leads to non-deterministic behavior ⇒ ***limits reproducibility***

Best practices

- set the *temperature* parameter to 0 (see here)
- set a *seed* (if possible, see here)
- fix all other request parameters

Caveat: all this makes model responses more "consistent" but *not* 100% reproducible.

Reproducibility with LLMs

Best practices

- set the *temperature* parameter to 0 (see here)
- set a *seed* (if possible, see here)
  - caveat: missing for openai R package
- fix all other request parameters
- save the system fingerprint (only with the OpenAI API)

Caveat: all this makes model responses more "consistent" but *not* 100% reproducible.

Implementation

openai

```
client.chat.completions.create(
     model = MODEL,
     messages=messages,
     temperature=0.0,
     seed=42
)
```

ollama

```
client.chat(
     model=MODEL,
     messages=messages,
     options = dict(
         temperature = 0.0,
         seed = 42
     )
)
```