

Using Large Language Models for Text Annotation in Social Science and Humanities: A Hands-On Python/R Tutorial

Qixiang Fang, Javier Garcia Bernardo, Erik-Jan van Kesteren

SoDa Science Team, Utrecht University, Netherlands

November 3, 2025

Abstract

Large language models (LLMs) have become an essential tool for social scientists and humanities (SSH) researchers who work with textual data. One particularly valuable use case is automating text annotation, traditionally a time-consuming step in preparing data for empirical analysis. Yet, many SSH researchers face two challenges: getting started with LLMs, and understanding how to evaluate and correct for their limitations. The rapid pace of model development can make LLMs appear inaccessible or intimidating, while even experienced users may overlook how annotation errors can bias results from downstream analyses (e.g., regression estimates, p -values), even when accuracy appears high. This tutorial provides a step-by-step, hands-on guide to using LLMs for text annotation in SSH research for both Python and R users. We cover (1) how to choose and access LLM APIs, (2) how to design and run annotation tasks programmatically, (3) how to evaluate annotation quality and iterate on prompts, (4) how to integrate annotations into statistical workflows while accounting for uncertainty, and (5) how to manage cost, efficiency, and reproducibility. Throughout, we provide concrete examples, code snippets, and best-practice checklists to help researchers confidently and transparently incorporate LLM-based annotation into their workflows.

Contents

1	Introduction	4
---	--------------	---

2 Understand LLMs for Text Annotation	6
2.1 From Logistic Regression to Language Modelling	7
2.2 A Segue to Prompting and Annotation	7
2.3 Summary	8
3 Research Project Set-up	8
3.1 Know Your Research Question and Data	9
3.2 Secure a Gold-Labelled Subset	9
3.3 Summary	12
4 Design Good Prompts	12
4.1 Taxonomy and Anatomy of Prompts	12
4.2 Prompt Engineering Techniques (Quick Wins).	13
5 Set Up the Computational Environment	17
5.1 A Quick Start with Cloud Notebooks	17
5.2 LLM APIs vs. Local LLMs	17
5.3 Obtain and Secure API Keys	18
5.4 Minimal Dependencies	18
5.5 Model Choice	19
5.6 Hyperparameters and Minimal Stability Settings	19
5.7 Summary	20
6 Run Your First Prompt Requests	20
6.1 Data Preparation	20
6.2 System Prompt and Model Initialisation	21
6.3 A Single Prompt Demo	22
6.4 Multiple Prompts Demo	22
7 Structuring LLM Annotations	23
7.1 Structured Output	23
7.2 A Single Prompt Demo	23

7.3	Multiple Prompts Demo	24
8	Evaluate Annotation Quality	25
8.1	Compute Agreement between LLM and Gold Labels	25
8.2	Identify Problematic Items and Examine Issues	27
8.3	Fix Data Issues, Improve Prompt, Iteratively	27
8.4	Summary	28
9	Consider Annotation Error in Downstream Analyses	29
9.1	Systematic Error vs. Random Error in Annotations	29
9.2	Effects on Downstream Analyses	29
9.3	Methods to Account for Annotation Error	30
9.4	Summary	31
10	Taking Your Annotations to the Next Level	31
10.1	Efficiency	31
10.2	Performance	32
10.3	Reproducibility	32
10.4	Common Pitfalls	33
10.5	Summary	33
11	Conclusion and Outlook	33
A	R Code	36
A.1	Obtain and Secure API Keys	36
A.2	Minimal Dependencies	36
A.3	Data Preparation	37
A.4	System Prompt and Model Initialisation	37
A.5	A Single Prompt Demo	38
A.6	Multiple Prompts Demo	38
A.7	Structuring LLM Annotations	39
A.7.1	Structured Output	39

A.7.2 A Single Prompt Demo	40
A.7.3 Multiple Prompts Demo	40
A.8 Evaluate Annotation Quality	40

1 Introduction

The recent proliferation of large language models (LLMs), such as GPT-4, Claude, and Gemini, has transformed the way researchers annotate and analyse text data (Brown et al., 2020; OpenAI, 2024a). Text annotation is a fundamental component of empirical research in the social sciences and humanities (SSH), serving as the basis for content analysis, sentiment measurement, stance detection, and narrative framing studies, among many others (e.g., Krippendorff, 2018a; Grimmer and Stewart, 2013; Mohammad, 2016; Conway et al., 2019). However, manual annotation is often costly, slow, and inconsistent across annotators (Artstein and Poesio, 2008; Schaefer et al., 2023; Poletto et al., 2020).

LLMs offer a promising way to automate or augment this process, allowing researchers to efficiently code large-scale textual data while maintaining flexibility and interpretability (Gilardi et al., 2023; Jurgens et al., 2023). The promise is that text annotation tasks (i.e., turning unstructured text into structured data on sentiment, topics, or other measurements) can be performed automatically, cheaply, and reliably with a few well-crafted prompts instead of extensive manual labelling.

Yet, for many SSH researchers, the barriers to adopting LLMs remain high. First, LLMs are technically challenging and difficult to integrate into existing research workflows. Setting up access rights for application programming interfaces (APIs), managing dependencies, or keeping up with the pace of model development can be overwhelming to newcomers. Second, even when researchers successfully deploy LLMs for annotation, they may underestimate the impact of annotation error on subsequent analyses. Imperfect labels—no matter how high the apparent accuracy—can propagate to downstream statistical models, biasing regression coefficients, inflating or deflating p -values, and ultimately affecting substantive conclusions (Egami et al., 2024).

This tutorial paper addresses both of these challenges. While several recent reviews have surveyed how LLMs and broader NLP methods are used in SSH research (e.g., Ziems et al., 2024; Abdurahman et al., 2025)), our goal is distinct: we provide a *hands-on, code- and project-focused guide* that enables researchers to implement, evaluate, and improve their own LLM-based annotation workflows. With an emphasis on transparent, reproducible, and research-centred practice, we hope to help both novice and experienced readers incorporate LLMs into their projects with ease and confidence.

Concretely, this paper **IS** about: (1) a concise overview of how LLMs work *for the purposes of text annotation* and what they can (and cannot) do; (2) a walk-through of API-based workflows for text annotation for both Python and R users; (3) methods to evaluate annotation quality and identify issues; (4) techniques to incorporate uncertainty in statistical models; and (5) cost-saving and reproducibility tips for scaling annotation projects. This paper **IS NOT** about: (1) the technical or architectural details of LLMs; (2) benchmarking or ranking LLMs for SSH applications; (3) fine-tuning or domain adaptation; or (4) replacing the need for gold-standard human annotations entirely. In summary, we situate LLMs as a new class of *measurement instruments*—powerful yet imperfect tools that can augment human expertise, rather than replace it entirely.

The remainder of this paper is organized as a practical guide on implementing an LLM text annotation workflow. Section 2 provides a brief, SSH-oriented background on LLMs for text annotation. Section 3 covers how to set up the research project in terms of research questions and data with ground-truth labels. Section 4 focuses on designing good prompts, often the most impactful decision when using LLMs to annotate text data. Section 5 describes how to use cloud notebooks to build a lightweight computational environment to get started quickly (for both Python and R). Section 6 demonstrates how to submit your first prompts and obtain LLM responses. Section 7 shows how to structure LLM outputs for improving both annotation performance and readability of LLM responses. Section 8 discusses how to evaluate annotation quality, fix potential issues and iterate the annotation pipeline to improve annotation performance. Section 9 shows the danger of ignoring annotation error and methods to account for it in downstream inferential analyses (e.g., linear regression). Section 10 offers additional tips on efficiency, performance, and reproducibility—especially when scaling up a project.

Together, these sections aim to bridge the gap between conceptual understanding and technical implementation, empowering SSH researchers to use LLMs critically and effectively for text annotation. Lastly, the paper is accompanied by two Python and two R codebooks that mirror the steps in the text.¹

¹Access the codebooks here https://sodascience.github.io/workshop_llm_data_collection/

Running Example: Goal Setting and Planning in Learning

Throughout this paper, we use a running example drawn from educational research: annotating weekly dialogues between a student and a *rule-based chatbot*, with the goal of evaluating the *specificity of the student's goal setting and planning*. The chatbot asks students a *fixed* list of questions designed to help them set better learning goals. It functions as a structured questionnaire presented through a user-friendly conversational interface, rather than as an intelligent assistant like an LLM. The accompanying dataset is kindly provided by Gabrielle Martins van Jaarsveld based on her research on self-regulated learning ([Martins Van Jaarsveld et al., 2025](#)). A dialogue in the dataset looks like this:

Chatbot: Set an academic goal for the upcoming week.

Student: I would like to catch up on my geography reading

Chatbot: Add details to make your goal more specific.

Student: I need to either read the book from last week and this week, or read my friends notes on the reading to take notes of my own so I dont fall behind.

Chatbot: How will you measure progress on and achievement of your goal?

Student: by the number of pages I write per day

Chatbot: Why is this goal important to you in the context of your prior experiences and future goals?

Student: It is important to achieve because if I dont, I will fall behind and most likely wont be ready for the exam.

Chatbot: Create a step-by-step plan for achieving this goal in the coming week.

Student: 1. evaluate how much there is to do
2. get help from my friends
3. takes notes day by day

2 Understand LLMs for Text Annotation

LLMs such as GPT-4, Claude, and Gemini are built upon a family of deep learning architectures known as *transformers*([Vaswani et al., 2017](#)). At their core, they are predictive models: given a sequence of text, they estimate the probability of what comes next in the form of *tokens*. Tokens refer to the smallest recognisable units by the model, including words, subwords and punctuations. Although the underlying mathematics involves bil-

lions of parameters and high-dimensional vector spaces, the basic logic is not foreign to social scientists familiar with logistic regression.

2.1 From Logistic Regression to Language Modelling

In a logistic regression, we model the probability of an outcome $y \in \{0, 1\}$ (e.g., “positive” vs. “negative”) as a logistic transformation of a weighted combination of predictors:

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\beta}^\top \mathbf{x})}$$

where \mathbf{x} is a vector of predictor values (x_1, x_2, \dots, x_p) and $\boldsymbol{\beta}$ is a corresponding vector of coefficients $(\beta_1, \beta_2, \dots, \beta_p)$. Each coefficient β_j captures how strongly its predictor x_j influences the log-odds of the outcome.

An LLM performs a conceptually similar operation, but at a vastly larger scale. Instead of a handful of (typically manually specified) predictors, it internally computes thousands of *latent features* that represent meaningful patterns in the input text. These latent features are subsequently combined and weighted, much like predictors in a logistic regression, and then transformed into a probability distribution over all possible next tokens in the model’s pre-specified vocabulary.

Because the outcome space consists of many possible tokens rather than *two* classes, a *softmax* transformation is used instead of the logistic function:

$$P(\text{next token} = t_i \mid \text{input text}) = \frac{\exp(\mathbf{w}_i^\top \mathbf{h})}{\sum_j \exp(\mathbf{w}_j^\top \mathbf{h})},$$

where the latent feature vector \mathbf{h} represent the meaningful patterns of the input text (analogous to the predictor vector \mathbf{x}), and \mathbf{w}_i is a learned weight vector for each possible next token t_i (analogous to the coefficients $\boldsymbol{\beta}$ in regression). Both \mathbf{h} and \mathbf{w}_i share the same dimensionality, and the denominator sums over all tokens in the vocabulary. Conceptually, this makes an LLM a massively multivariate extension of logistic regression operating over sequences of text.

2.2 A Segue to Prompting and Annotation

When we provide a simple prompt like “*Classify the student’s goal as specific or vague:*”, the LLM conditions on this input and estimates the most likely continuation. Therefore, we can treat the model as a flexible text classifier (Gilardi et al., 2023; Jurgens et al., 2023). For instance, LLMs can help with SSH annotation tasks such as: (1) classifying text segments into theoretically meaningful categories; (2) extracting entities or themes

from open-ended narratives; (3) summarising or rephrasing qualitative data for coding; (4) turning unstructured text data into tabular data for integration with statistical software.

However, several caveats are worth-noting:

- **Probabilistic nature.** LLMs produce the most likely continuation, not necessarily the *true* or *correct* label. Small variations in model settings or prompt phrasing can yield different outputs.
- **Opacity of features.** Unlike logistic regression, where β coefficients have interpretable meanings, the internal weights of LLMs are distributed across billions of parameters and cannot be directly interpreted ([Bender et al., 2021](#)).
- **Training data and trainer dependence.** Model behaviour reflects the distributions of its training data and intentions of its developers, which may include biases or outdated information ([Mitchell et al., 2019](#)).
- **Reproducibility issues.** Models and their APIs may change silently over time; recording model version and prompt text can help with reproducibility, but even then recovering older models and their outputs can still be difficult.
- **Hidden costs.** Beyond API usage fees, employing LLMs for annotation requires time and resources for researcher training, prompt engineering, and iterative experimentation. On small research projects, manual annotations might still be cheaper and more efficient.

2.3 Summary

For SSH researchers, it is often useful to think of an LLM as an “expanded logistic regression” trained on the entire internet: it predicts repeatedly the next token (or label) using internally computed latent features that meaningfully represent some input text. When properly instructed via prompts, this predictive machinery can be repurposed for annotation tasks such as coding interview transcripts, classifying news articles, or analysing online discourse.

3 Research Project Set-up

In this section, we describe how to get started with a research project in which LLM-based text annotation will likely play a role.

3.1 Know Your Research Question and Data

Begin by clearly articulating your research questions and verifying whether your available data can meaningfully address them. In other words, ensure that the construct you aim to measure is actually represented in the textual material at hand. Some constructs—such as personality traits inferred from social media posts—are inherently difficult to capture because the texts often lack sufficient linguistic or contextual cues ([Stajner and Yenikent, 2021](#); [Fang et al., 2023](#)). Likewise, for highly subjective annotation tasks such as hate speech detection, even human annotators will naturally disagree. In both cases, large language models (LLMs) are likely to encounter similar challenges. These considerations highlight the importance of setting realistic expectations for LLM performance. Before beginning annotation, define an *a priori* target level of agreement that would render the LLM-generated labels useful for your downstream analyses. Establishing this target early not only clarifies what constitutes “good enough” performance but also helps you decide when further prompt tuning or model iteration is no longer necessary.

3.2 Secure a Gold-Labelled Subset

Establishing *gold standard labels* for a subset of your data is essential for verifying the performance of the LLM annotations. Aim for a balanced subset (e.g., across classes or score levels) so that evaluation is informative rather than dominated by the most frequent category. If you anticipate subgroup analyses (e.g., by time period or data source), stratify the gold standard labels accordingly to ensure that evaluation reflects those specific use cases. Keep this gold standard subset *strictly held out* from any prompt examples or model training data to avoid data leakage ([Kapoor and Narayanan, 2023](#)); this may artificially inflate the agreement between LLM and gold-standard ratings.

Obtaining such a subset usually requires human annotation. In typical SSH research, there are several possible scenarios:

- You already have a dataset with existing gold labels.
- You have a validated codebook or rubric from previous research and only need to recruit new annotators.
- You have neither and need to develop a new codebook and collect annotations from scratch.
- You plan to use a strong LLM to generate an initial gold-labelled subset, followed by selective manual verification to save time and resources.

Regardless of the scenario, it is crucial to ensure a strong conceptual link between your research question and the annotation instructions. The codebook serves as this bridge:

it operationalizes your theoretical constructs and later forms the backbone of prompt design—arguably the most important step in using LLMs for text annotation.

A good codebook or annotation manual is *clear*, *complete*, and *easy to follow*, avoiding unnecessary jargon and ambiguity. Drawing on prior research, established rubrics, and domain experts can greatly improve its quality. While a detailed discussion of best practices for obtaining high-quality human annotations is beyond the scope of this paper (see, e.g., [Leech et al., 2003](#), for guidance), we emphasize that LLM-based annotation and human annotation are conceptually similar processes—both rely heavily on well-designed and unambiguous instructions.

Caveat Even carefully constructed gold-labelled subsets may contain errors or inconsistencies, which can lead to apparent disagreement between LLM and gold labels. In Section 9, we discuss methods for diagnosing and accounting for such discrepancies.

Running Example: Rubric

Here's the rubric that Gabrielle Martins van Jaarsveld developed after synthesizing prior research and expert opinions:

- **Characteristic Measured:** Goal specificity
- **Definition:** Goal must be specific rather than general. The context and details of the goal should be explicitly described, and all abstract terms are explained.
- **Scoring Rules:**
 - **Not Present (0):** Extremely broad, with no details about what this goal entails. States the goal using vague terms without providing any descriptions of what they mean. *Example student response: "Prepare for the tutorial."*
 - **Partially Present (1):** States the goal and offers some descriptions of the terms used, however there are still some vague terms which are not fully described. *Example student response: "Prepare for this week's Friday morning tutorial by reading all the materials."*
 - **Fully Present (2):** No vague terms which are not described. Clearly states the goal and uses clear descriptions to describe exactly what they want to achieve. OR Gives a boundary descriptor which offers context to the other 'unexplained' terms in the goal. *Example student response: "Prepare for this week's Friday morning tutorial by reading all 5 of the articles which are assigned on canvas."*

Running Example: Agreement and Expectation

In a pilot study involving four human annotators with PhD level expertise, an inter-annotator agreement above 0.80 was observed. Given this high agreement score, the clarity in the rubric and the presence of explicit textual cues, we can expect similar performance from an LLM on this task.

3.3 Summary

Before writing any prompt or code, confirm that (1) the data plausibly contain the signals needed to answer your research questions; (2) you have set a realistic target agreement; and (3) you have a good translation between your research questions and instructions (e.g., a codebook, rubric, or scoring manual); (4) have a balanced, held-out gold-standard subset for evaluation.

4 Design Good Prompts

4.1 Taxonomy and Anatomy of Prompts

A prompt is defined as natural language input or instructions (i.e., text) that describe the task that an LLM (and other AI systems) should perform. Prompt design is the most crucial part of obtaining consistent and interpretable annotations from LLMs.

Most LLM APIs separate a *system prompt* (which sets overall model behavior) and a *user prompt* (which contains the task input). In the context of LLM-based annotations, system prompts should describe the task *context*, the concrete *task* itself, and the expected *output*. In contrast, user prompts provide the *input* to the task (or say task content).

Running Example: Basic Prompt Design

System Prompt:

A university student was given a series of questions from a chatbot, guiding them through the process of setting and elaborating on an academic goal for the coming week. You will be provided with the entire conversation including the chatbot's questions and the student's answers. Your objective is to assess the specificity of the student's goal ^a on a scale of 0 to 2 based on the entire conversation.

User Prompt:

Chatbot: Set an academic goal for the upcoming week.

Student: I would like to catch up on my geography reading

Chatbot: Add details to make your goal more specific.

Student: [...] ^a

^aThe rest of the conversation is omitted.

Keeping these distinctions between system and user prompts makes the task persistent and context-independent, which can also improve LLM processing efficiency.

4.2 Prompt Engineering Techniques (Quick Wins).

Prompt engineering refers to the process of modifying an instruction in order to produce better outputs from LLMs. Several simple adjustments can markedly improve annotation stability ([OpenAI, 2023](#)):

1. **Clarity.** Be explicit. Avoid ambiguity and vague wording. For instance, verbs/phrases like “classify”, “assign one label” and “choose the best option” are preferred over “analyse” and “interpret” ([OpenAI, 2024b](#)).
2. **Role-based prompting.** Assign a concise role (e.g., “You are an educational expert”) to guide the LLM’s response style, tone and task interpretation ([OpenAI, 2024b](#)).
3. **Step-by-step reasoning.** When the task is complex, instruct the model to reason step by step before giving the final label (a light form of chain-of-thought prompting ([Wei et al., 2022](#))).
4. **Few-shot prompting.** Provide a few labelled examples within the system prompt to anchor decisions—especially helpful for nuanced categories ([Brown et al., 2020](#)).

We show below how to incorporate these four techniques into the system prompt of our running example.

Running Example: Improving Clarity in System Prompts

##TASK##

A university student was given a series of cues by a chatbot, guiding them through the process of setting and elaborating on an academic goal for the coming week. You will be provided with the entire conversation including the chatbot cues, and the student answers. Your objective is to assess all the students answers using a scoring rubric that evaluates the specificity of the goal on a scale of 0 to 2, representing the characteristic being not present, partially present, or fully present. Specificity is defined as: Goal must be specific rather than general. The context and details of the goal should be explicitly stated and described, and all terms are explained.

- Score of 0: Extremely broad, with no details about what this goal entails. States the goal using vague terms without providing any descriptions of what they mean. Or the goal is an abstract concept to improve or work towards, without any explanation of how this could be actionable or concrete.
- Score of 1: States an actionable or concrete goal and offers some descriptions of the terms used. However, there are still some vague terms which are not fully described.
- Score of 2: No vague terms which are not described. Clearly states the goal and uses clear descriptions to describe exactly what they want to achieve. OR gives a boundary descriptor which offers context to the other unexplained terms in the goal.

Running Example: Using Role-based Prompting in System Prompts

Insert the following at the beginning of the system prompt:

“You are an expert in educational assessment and goal evaluation, with specialized expertise in applying deductive coding schemes to score the quality and content of student goals. You have a deep understanding of scoring rubrics and are highly skilled at analysing goals for specific characteristics according to well-defined criteria.”

Running Example: Integrating Step-by-Step Reasoning in System Prompts

##INSTRUCTIONS##

1. Understand the scoring rubric: REVIEW the rubric provided for specificity to understand the criteria for scores of 0, 1, and 2. IDENTIFY the key elements that distinguish a low score (0) from a high score (2).
2. Analyse the conversation in relation to specificity: ASSESS the extent to which the goal is specific rather than general. Are context and details of the goal explicitly described, and all terms explained? Is the goal concrete and attainable and not something abstract?
3. Assign a score: Assign a score of 0, 1, or 2 based on the rubric. Use the provided scored examples as a reference to ensure consistency with previous assessments.
4. Provide a detailed rationale for the score: EXPLAIN why you assigned the score by directly referencing aspects of the goal that meet or fall short of the rubric criteria.
5. Check for consistency: Double-check that the score aligns with both the rubric criteria and the rationale provided. Maintain objectivity by strictly adhering to the rubric without introducing personal biases.

##EDGE CASE HANDLING##

If a goal is ambiguous or unclear, SCORE it on the lower end. If a goal appears to partially meet the criteria for two different scores, SELECT the score that best reflects the majority of the goals characteristics for that category.

##WHAT NOT TO DO##

Never apply personal opinion or assumptions outside the rubric criteria. never give a score without a detailed explanation, even if the scoring seems obvious. Never modify or assume student intent; score the goal exactly as written. Never ignore the rubric or provided examples when scoring.

Running Example: Incorporating Few-shot Examples in System Prompts

```
##EXAMPLE SCORING##  
Example 1: [insert example conversation]  
Example 1 Scoring: [insert score] ([insert reasoning])  
  
Example 2: [insert example conversation]  
Example 2 Scoring: [insert score] ([insert reasoning])
```

Prompt Templates and Automatic Generators Several online tools can help researchers design effective prompts, such as OpenAI’s prompt examples (<https://platform.openai.com/examples>) and Anthropic’s prompt library (<https://docs.anthropic.com>). In addition, automatic prompt generators—such as Originality.ai Prompt Generator (<https://originality.ai/blog/ai-prompt-generator>, free) and Anthropic’s Prompt Generator (<https://console.anthropic.com/dashboard>, paid)—can serve as useful starting points. However, these resources should be treated primarily as inspiration: all prompts must ultimately be reviewed and adapted to the specific goals and context of your study.

Common Prompt Pitfalls Avoid typical sources of poor annotation performance:

- Lacking a clear structure or output indicator;
- Providing insufficient context or label definitions;
- Using inconsistent terminology.

A simple check is to give the same instruction to human coders—if they cannot agree on the intended meaning, the LLM likely will not either. Furthermore, when comparing LLM versus human annotations, keep the instructions as similar as possible to reduce interpretive discrepancies.

Summary Prompt engineering involves refining instructions to elicit higher-quality outputs from LLMs. A few straightforward techniques can substantially improve annotation quality: using clear and specific wording, assigning the model a role, prompting step-by-step reasoning, and providing labelled examples. Automatic prompt libraries (e.g., from OpenAI or Anthropic) can serve as useful inspiration, but their templates should always be adapted to the specific research context. Finally, avoid common pitfalls such as vague structure, missing contextual information, or inconsistent terminology. If a prompt appears ambiguous to human coders, it will almost certainly confuse the LLM.

5 Set Up the Computational Environment

5.1 A Quick Start with Cloud Notebooks

There are many valid setup options for working with LLMs, including local installations, cloud-based notebooks, virtual machines, and HPC clusters. For readers who are inexperienced with R, Python, or programming in general, we recommend starting with cloud-based notebooks such as Google Colab (for Python and R) and RStudio Cloud (by Posit, for R). These platforms remove the need to manage local installations or hardware and are free to use at the basic tier. Our accompanying Python and R notebooks are designed to run on these services, as they are also intended as reference materials for an interactive workshop. Throughout this paper, the step-by-step demonstrations assume a Google Colab session for Python and an RStudio Cloud session for R; however, the same commands can also be executed locally with only minor adjustments. Once you are familiar and comfortable with the cloud-based notebook approach as a quick starting point, we recommend moving away from this iterative workflow toward a local, project-based setup that is repeatable and reproducible—particularly for larger-scale text annotation projects (see Section 10.3).

5.2 LLM APIs vs. Local LLMs

LLMs can be accessed through:

1. **APIs**, which provide ready-to-use models such as OpenAI’s GPTs, Anthropic’s Claude, or Google’s Gemini. These require only an internet connection and an API key.
2. **Local or open-source models**, such as Llama 2 or Mistral, which can be deployed on institutional hardware or high-performance clusters.

For most SSH researchers, cloud-based APIs provide the most accessible entry point. They remove the need for dedicated GPU hardware while offering consistent performance across sessions. However, it is important to point out this approach involves sending research data to the API provider, which may raise privacy and data protection concerns under the GDPR. Unless explicitly permitted by the study participants or covered by an approved data management protocol, such data transfer is generally not allowed as part of the research process. In such cases, running local LLMs offers a feasible and GDPR-compliant, privacy-friendly alternative. Unfortunately, we do not cover the local deployment of LLMs in this paper.

In this tutorial, we use the `OpenAI` API as the default provider and rely on the `langchain` and `ellmer` libraries to facilitate interaction with LLM APIs in Python and R, respectively. These libraries ensure the annotation pipeline is independent of the provider,

so the same workflow is suitable for other API providers or local LLMs as well.

5.3 Obtain and Secure API Keys

All commercial LLM providers require an API key linked to a user account. After creating an account on the respective provider’s website, you can generate an API key in your user dashboard. The API key is a unique alphanumeric string that identifies you as the authorized user—for example, `c784df9e-bcf1-4145-83b6-a3c9ca281ee3`. This key grants access to the model endpoints and is used to monitor usage and manage billing.

A tip is to generate independent API keys for different projects such that you can have a better overview of the activities and costs associated with each project. To prevent accidental exposure of keys in shared repositories, store the key locally (and separately) in a hidden file or as an environment variable (never inside code).

In the accompanying codebooks, the following code prompts a user to paste their key interactively; the key is retained only for that session. Both the Python and the R code store the entered API key as an environment variable.

Python:

```
# Prompt user for API key
os.environ["OPENAI_API_KEY"] = getpass.getpass("Enter API key for OpenAI: ")
```

R: See Appendix [A.1](#).

5.4 Minimal Dependencies

Install only what the accompanying notebooks require. For Python, that means a lightweight data stack and an API client; for R, an API client and tidy data tooling.

Python:

```
# Install additional packages not included in Colab by default
%pip install krippendorff ppi-python
```

```
# Load required packages
import getpass # For secure password or API key input (hides typed text)
import os # For interacting with the operating system (e.g., paths, env vars)
import pandas as pd # Data manipulation and analysis (tables, CSVs, etc.)
import numpy as np # Numerical computing and array operations
from tqdm import tqdm # Progress bars for loops and iterations
```

```

from langchain.chat_models import init_chat_model # Initializes chat-based LLMs
from langchain_core.prompts import ChatPromptTemplate # Template for building
# structured LLM prompts
from pydantic import BaseModel, Field # Data validation and schema definition
import krippendorff # Computes Krippendorff's alpha
from ppi_py import ppi_ols_pointestimate, ppi_ols_ci # Implements correction of
# OLS regression estimates
import statsmodels.api as sm # Statistical modeling tools (e.g., regression)
import matplotlib.pyplot as plt # For plotting and visualizing data
import seaborn as sns # High-level visualization library

```

R: See Appendix [A.2](#).

5.5 Model Choice

Keep it simple at the start. Different models balance *accuracy*, *cost*, and *speed*, the various combinations of which may easily overwhelm users. To start with, we recommend picking any reasonably priced model recommended by its provider (on their website). In our tutorials, we use `gpt-4o-mini`, an OpenAI-recommended model for fast, affordable and focused task completions.

Later on, keep costs in mind. Annotation projects involving tens of thousands of texts can incur substantial costs due to the high number of tokens consumed. To address this issue, researchers should estimate total token usage by running a small pilot on a subset of their data. As a rough rule, one short paragraph (≈ 100 words) equals 120–150 tokens. Note that input tokens and output tokens are often priced differently.

5.6 Hyperparameters and Minimal Stability Settings

LLM outputs can vary slightly between runs due to their stochastic nature. You can control this variance through the following commonly available hyperparameters:

- **temperature**: sets output randomness (0 = deterministic, 1 = creative);
 - Low values (e.g., 0.3) make the model more deterministic and repetitive.
 - High values (e.g., 0.6 or higher) increase diversity and creativity but may reduce coherence.
- **top_k** and **top_p**: control sampling diversity.
 - `top_k`: Restricts sampling to the k most likely next tokens. A lower value (e.g., 10) makes output more deterministic. A higher value (e.g., 50 or 100) allows for more

diversity.

- `top_p`: Instead of picking from the k most probable tokens, it selects from the smallest set of tokens whose probabilities sum to p . Lower values (e.g., 0.3) make responses more deterministic. Higher values (e.g., 0.9) increase diversity.
- `max_tokens`: limits output length;
- `seed`: fixes random initialization for reproducibility.

For example, to ensure consistent outputs from LLMs, you can set `temperature = 0` and fix a `seed` if supported. Even so, a small degree of nondeterminism may remain due to factors such as server scheduling. Note also that not all of these hyperparameters are available for every LLM or API. Always consult the official documentation to verify which model hyperparameters are supported before use.

5.7 Summary

Begin in a cloud notebook to avoid setup friction; use an API for a smooth first pass; secure keys via secrets or environment variables; and install only the minimal packages that our notebooks require. Set `temperature = 0` and fix a `seed` if reproducibility is desired; however, postpone more detailed hyperparameter decisions for later.

6 Run Your First Prompt Requests

6.1 Data Preparation

Start by deciding how you will store and access the text data to be annotated. For most researchers, a simple `.csv` file is sufficient: it can be inspected with Excel, Numbers, or any text editor, and can be easily read by both Python and R. More efficient formats such as Parquet or Arrow can reduce size and speed up I/O, but they are not necessary for small or medium-sized projects. Each row should correspond to one text unit (e.g., a sentence, paragraph, or post) with a unique identifier and any contextual metadata (e.g., participant ID, date, or source).

In our tutorials, we use a toy dataset from our running example of annotating goal specificity from student-chatbot conversations. The dataset contains the following columns:

Running Example: Dataset Description

- **id**: the identifier of the row (corresponding to a student's entire conversation);
- **conversation**: the text of the conversation to annotate;
- **score_specificity_llm**: the specificity score produced by a carefully prompted LLM, taking values 0, 1, or 2;
- **score_specificity_human**: the gold-standard specificity score assigned by human expert annotators (also 0–2);
- **performance**: the student's academic performance (1–10 scale).

To load the dataset:

Python:

```
# url where you can download our example data
data_url = "https://sodascience.github.io/workshop_llm_data_collection/data/" \
"srl_data_example.csv"

# Read CSV into dataframe
df = pd.read_csv(data_url)
```

R: See Appendix [A.3](#).

6.2 System Prompt and Model Initialisation

Python:

```
system_prompt = """
You are an expert in educational assessment and goal evaluation, with
specialized expertise in applying deductive coding schemes to score the quality
and content of student goals.

##TASK##
A university student was given a series of prompts, guiding them through the
process of setting and elaborating on an academic goal for the coming week. You
will be provided with the entire conversation including the prompts, and the
student answers. Your objective is to assess the specificity of of the student's
goal on a scale of 0 to 2 based on the entire conversation.

"""
```

```
model = init_chat_model("gpt-4o-mini",
                        model_provider="openai",
                        temperature=0,
                        max_tokens=1000)
```

R: See Appendix [A.4](#).

6.3 A Single Prompt Demo

Python:

```
prompt_template = ChatPromptTemplate([
    ("system", system_prompt),
    ("user", "{conversation}")
])

# Create a prompt request with the system prompt and the user prompt based on
# the first conversation from the dataset
prompt_request = prompt_template.invoke({"conversation": df.iloc[0,1]})

# Check the prompt
prompt_request.to_messages()

# Prompt the model and inspect the response!
response = model.invoke(prompt_request)
print(response.content)
```

R: See Appendix [A.5](#).

Voila! You have your first successful interaction with an LLM API!

Running Example: First Prompt Results

Results to be added.

6.4 Multiple Prompts Demo

Python:

```

ids = df.id[:10].tolist()
conversations = df.conversation[:10].tolist()

responses = {}
for id, conversation in tqdm(zip(ids, conversations),
                             total=len(ids),
                             desc="Processing Requests"):
    prompt_request = prompt_template.invoke({"conversation": conversation})
    response = model.invoke(prompt_request)
    responses[id] = response.content

print(responses['request_3'])

```

R: See Appendix [A.6](#).

7 Structuring LLM Annotations

7.1 Structured Output

Python:

To force the LLM to produce outputs in formats specified by you, you need to use the `BaseModel` and `Field` classes from the `pydantic` package. Below, we define our desired output format as:

- `specificity_score`: an integer (either 0, 1 or 2) reflecting the specificity of a conversation.
- `reasoning`: a string that provides the model's reasoning.

```

class SpecificityFormat(BaseModel):
    """Always use this tool to structure your response to the user."""
    specificity_score: int = Field(description="The specificity score of the" +
                                    "entire conversation on a scale of 0, 1 and 2.")
    reasoning: str = Field(description="Your reasoning process.")

```

R: See Appendix [A.7.1](#).

7.2 A Single Prompt Demo

Python:

```
# Bind responseformatter schema to the model
model_structured = model.with_structured_output(SpecificityFormat)

# Run a request through this new model
prompt_request = prompt_template.invoke({"conversation": df.iloc[0,1]})
structured_response = model_structured.invoke(prompt_request)

dict(structured_response)
```

R: See Appendix [A.7.2](#).

7.3 Multiple Prompts Demo

Being able to work with multiple prompts at the same time and obtain structured output will save you a substantial amount of time in research projects!

Python:

```
structured_responses = []
for id, conversation in tqdm(zip(ids, conversations),
                             total=len(ids),
                             desc="Processing Messages"):
    prompt_request = prompt_template.invoke({"conversation": conversation})
    structured_response = model_structured.invoke(prompt_request)

# Below we save only the specificity scores
structured_responses[id] = dict(structured_response)["specificity_score"]
structured_responses
```

R: See Appendix [A.7.3](#).

Running Example: Structured Output Results

Results to be added.

8 Evaluate Annotation Quality

8.1 Compute Agreement between LLM and Gold Labels

A crucial step in evaluating the quality of LLM-based annotations is to quantify their level of agreement with a reference or “gold-standard” dataset, typically obtained from trained human annotators. The choice of agreement metric depends on the nature of the annotation outcome—categorical, ordered, or continuous—and each metric carries specific assumptions, strengths and limitations.

Categorical outcomes. For nominal labels (e.g., news categories such as *politics*, *science*, *art*), the most commonly used metrics include simple accuracy, Cohen’s κ ([Cohen, 1960](#)), and Krippendorff’s α ([Krippendorff, 2018b](#)). Accuracy is intuitive and easy to interpret, but it ignores chance agreement and is sensitive to class imbalance (i.e., when there are rare categories). Therefore, accuracy is typically only used with balanced datasets. Cohen’s κ adjusts for chance agreement between two raters, providing a more conservative estimate, while Krippendorff’s α generalises this idea to multiple annotators and handles missing data. However, both κ and α can be unstable in cases of highly skewed class distributions or small sample sizes.

Ordered outcomes. When the labels are ordinal (e.g., Likert-type ratings), weighted versions of κ or α are preferred because they account for the magnitude of disagreement between categories. Linear or quadratic weights penalize larger discrepancies more heavily, thereby reflecting the underlying ordering of the labels ([Gwet, 2014](#)). Nevertheless, even weighted metrics may obscure systematic directional biases (e.g., when an LLM consistently rates higher or lower than human annotators), which should be examined separately.

Continuous outcomes. For continuous or quasi-continuous² variables (e.g., predicted probabilities, test scores, frequencies), agreement is typically quantified using correlation-based or error-based measures. The Pearson correlation coefficient (r) assesses the strength of linear association, while the intraclass correlation coefficient (ICC) ([Shrout and Fleiss, 1979](#)) measures both correlation and absolute agreement. Alternatively, mean absolute error (MAE) or root mean squared error (RMSE) can be used to express the average discrepancy in scale units. Correlation metrics are scale-invariant but do not capture systematic shifts (e.g., consistent over- or underestimation), whereas error-based metrics

²Quasi-continuous variables are discrete variables (often ordinal or interval-like) with many possible values that are treated as continuous for practical or statistical reasons.

are sensitive to such shifts but depend on the units of measurement.

Summary. In practice, no single metric fully captures annotation quality. A comprehensive evaluation often combines multiple indicators—for instance, reporting both κ and class-specific precision/recall for categorical data, or both ICC and MAE for continuous data—to balance interpretability and robustness. Additionally, these statistics should be complemented with qualitative inspection of disagreements, as LLM-human mismatches may stem from annotation ambiguity or under-defined label definitions rather than model error.

Table 1: Common Agreement Metrics by Data Types.

Metric	Data Type	Strengths	Limitations
Accuracy	Categorical	Simple and interpretable	Ignores chance agreement; affected by class imbalance
Cohen’s κ	Categorical	Adjusts for chance agreement	Sensitive to skewed distributions and class imbalance
Krippendorff’s α	Categorical OR ordered	Handles missing data; generalizes to many raters	Computationally intensive; unstable in small samples
Weighted κ	Ordered	Accounts for distance between labels	May hide systematic directional bias
Pearson r	Continuous	Simple linear association	Ignores scale or bias differences
ICC	Continuous	Captures correlation and absolute agreement	Requires model choice; assumes residual normality
MAE / RMSE	Continuous	Expressed in scale units; intuitive	Sensitive to outliers and units of measurement

We showcase below code implementations of Krippendorff’s α for its wide application in SSH research.

Python:

Implement a handy function to calculate Krippendorff’s Alpha (i.e., agreement) between two lists of specificity scores.

```
def compute_krippendorff_alpha(x: list[int], y: list[int]):
    # Format data into a reliability matrix (rows=raters, cols=items)
    data_krippendorff = np.array([x, y])
    # Compute Krippendorff's Alpha (interval metric)
    kripp_alpha = krippendorff.alpha(reliability_data=data_krippendorff,
                                    level_of_measurement='interval')
    return kripp_alpha
```

Let's check the agreement between the specificity scores we got from the LLM above and the human expert-coded specificity scores!

```
score_specificity_human = df.score_specificity_human[:10].tolist()
structured_response_values = list(structured_responses.values())
print("Krippendorff's Alpha:",
compute_krippendorff_alpha(structured_response_values, score_specificity_human))
```

R:

Running Example: Annotation Quality

Results to be added.

8.2 Identify Problematic Items and Examine Issues

Regardless of the overall agreement score, it is essential to examine and, where possible, correct potentially problematic items. Quantitative agreement metrics can mask specific sources of error, so visual inspection of cases with disagreements between the LLM and the gold labels is highly recommended. Such inspection may reveal that (a) the gold annotation itself is incorrect due to human oversight; (b) the LLM output violates task specifications, for example by producing a value outside the permitted range; or (c) the LLM fails to account for certain nuances, exceptions, or contextual cues described in the prompt. Identifying these patterns not only improves label quality but also clarifies whether disagreement stems from annotation ambiguity, model behavior, or flaws in the instruction design.

Running Example: Identified Issues

Results to be added.

8.3 Fix Data Issues, Improve Prompt, Iteratively

Once potential issues have been identified, the next step is to iteratively refine both the data and the prompting procedure. This process typically involves three complementary actions: (1) fixing the data itself, (2) improving the prompt or task specification, and (3) re-running the annotation workflow to verify improvements.

- 1. Correct or flag problematic items.** If disagreements stem from clear human or LLM errors (e.g., invalid label values, missing entries, or mismatched examples), these cases should be corrected or flagged for exclusion. When corrections are uncertain, it is good practice to keep both the original and revised versions in a transparent audit trail, noting the rationale for the change. In projects with multiple annotators, such cases can also be reviewed by an adjudicator or majority vote to ensure consistency.
- 2. Refine prompt instructions.** When disagreements reflect systematic LLM misinterpretations, the prompt itself likely needs revision. Common improvements include: clarifying category definitions, adding boundary examples, explicitly stating how to handle ambiguous or multi-faceted cases, and enforcing stricter output formatting. For instance, if the LLM tends to overestimate specificity, the prompt can be refined with counterexamples (e.g., “Responses that only restate the goal without a concrete plan should receive a 0”). Iterative prompt adjustments are particularly effective when combined with structured outputs (e.g., JSON schema) that enforce the required response format.
- 3. Re-run and re-evaluate.** After each refinement round, re-run the annotation and re-calculate agreement metrics to assess improvement. A well-documented iteration log including date, prompt version, and agreement statistics allows researchers to track progress and determine whether changes in agreement reflect genuine prompt improvement or random variation. If multiple prompt versions are tested, a simple comparison table or plot can visually summarise which modifications led to more reliable results.

Running Example: Iterative Refinement

Results to be added.

8.4 Summary

Evaluating LLM-based annotations is not a one-off step but a cyclic process of measurement, diagnosis, and improvement. Quantitative agreement metrics such as κ , α , and ICC provide essential summaries of reliability, yet they must be interpreted alongside qualitative inspection of disagreements and iterative prompt adjustments. Poor agreement does not necessarily imply model failure but often signals ambiguous instructions, unclear category boundaries, or inconsistencies in the gold data itself. An effective evaluation workflow therefore combines agreement assessment, error diagnosis and iterative refinement. Through this systematic and transparent approach, researchers can ensure that LLM annotations not only replicate human judgments but also reveal where human

definitions and instructions themselves may need clarification, turning evaluation into a process of both quality control and conceptual refinement.

9 Consider Annotation Error in Downstream Analyses

While inter-rater agreement provides an indication of annotation quality, researchers should also consider how annotation error—both random and systematic—can influence downstream analyses. Ignoring such error may lead to biased parameter estimates, attenuated relationships, or incorrect inferences.

9.1 Systematic Error vs. Random Error in Annotations

Annotation errors can broadly be classified as either *random* or *systematic*. Random error reflects unsystematic noise in the annotation process, such as inconsistent judgments or stochastic variation in LLM outputs across repeated runs. These errors increase measurement variance but do not bias estimates on average. In contrast, systematic error occurs when annotations consistently deviate from the true construct in a particular direction—for instance, when an LLM systematically overestimates sentiment intensity or misclassifies a specific subgroup due to training data biases. Systematic error distorts the meaning of measured variables, leading to biased estimates even in large samples.

Distinguishing between these two sources of error is critical: while random error primarily reduces reliability and statistical power, systematic error threatens validity by introducing predictable distortions in downstream findings.

9.2 Effects on Downstream Analyses

Annotation error affects both descriptive and inferential analyses. In simple descriptive statistics, random error in an annotated variable Y^* inflates its variance and attenuates estimated means toward the center of the scale. When Y^* is used as a dependent or independent variable in a regression model, such measurement error leads to biased or inconsistent coefficient estimates.

For instance, if the observed variable Y^* represents an error-prone version of the true latent construct Y , we can express the measurement model as

$$Y^* = Y + \varepsilon,$$

where ε denotes the annotation error term. In regression settings of the form

$$Y^* = \beta_0 + \beta_1 X + \eta,$$

random error in Y^* (classical measurement error) inflates residual variance but leaves $\hat{\beta}_1$ unbiased, whereas random error in a predictor variable $X^* = X + \varepsilon$ typically attenuates $\hat{\beta}_1$ toward zero (therefore also called attenuation error) (Fuller, 1987). The magnitude of distortions due to random error depends on the reliability of the annotated variable, often expressed as

$$\rho = \frac{\text{Var}(Y)}{\text{Var}(Y^*)} = 1 - \frac{\text{Var}(\varepsilon)}{\text{Var}(Y^*)}.$$

Low reliability (ρ close to 0) implies severe attenuation of correlations and regression coefficients, whereas high reliability (ρ close to 1) indicates negligible distortion.

Systematic error, by contrast, arises when the measurement error has a nonzero mean or correlates with predictors. A constant additive bias (e.g., when all annotations are consistently rated 0.5 points higher) shifts the intercept of the regression model but does not affect the estimated slope. However, when the bias varies with the true variable or with other predictors—such that $E[\varepsilon|X] \neq 0$ or $\text{Cov}(X, \varepsilon) \neq 0$ —it introduces directional bias in the slope as well. In such cases, the estimated effect sizes may be systematically over- or underestimated, leading to distorted substantive conclusions even when the observed variable appears reliable.

9.3 Methods to Account for Annotation Error

Several statistical approaches can adjust for the presence of annotation error in downstream analyses. When independent reliability estimates are available (e.g., from repeated human or LLM annotations), one can apply *reliability correction* or *attenuation adjustment*, scaling observed coefficients by the estimated reliability ratio (Fuller, 1987). Alternatively, *error-in-variables (EIV)* or *latent-variable* models explicitly incorporate measurement error into the estimation process. These methods treat the true construct as a latent variable and model the noisy observed annotation as an imperfect indicator. Another option is to conduct a *sensitivity analysis*, assessing how conclusions vary under plausible levels of measurement error variance.

Our accompanying notebooks demonstrate two practical implementations:

- **Python:** The `ppi_py` package provides functions such as `ppi_ols_pointestimate()` and `ppi_ols_ci()` for correcting regression coefficients under classical measurement error assumptions. Example code is provided in the accompanying Python notebook.
- **R:** The `dsl` package implements a declarative syntax for specifying measurement mod-

els and performing bias correction in regression and structural equation settings. Example R code can be found in the accompanying R notebook.

Together, these approaches enable researchers to quantify and, where possible, mitigate the influence of annotation error on downstream estimates, thereby improving the validity and interpretability of LLM-assisted analyses.

Running Example: Regression Correction

Results to be added.

9.4 Summary

When annotations are used as variables in downstream analyses, annotation error can substantially influence both descriptive statistics and inferential models. Random error primarily reduces reliability, inflating variance and attenuating observed associations, while systematic error introduces directional bias that can distort estimated means, slopes, and substantive interpretations. To ensure valid conclusions, researchers should assess the reliability of annotated variables, examine potential sources of systematic bias, and apply statistical corrections such as reliability adjustment, error-in-variables modelling, or sensitivity analysis. By explicitly accounting for annotation error, downstream analyses can more accurately reflect the underlying constructs that LLMs are intended to measure.

10 Taking Your Annotations to the Next Level

Once the basic annotation workflow has been established, researchers can further enhance the *efficiency*, *performance*, and *reproducibility* of their LLM-assisted annotation pipelines. This section outlines several best practices and advanced strategies, as well as common pitfalls to avoid.

10.1 Efficiency

Caching. When running iterative or large-scale annotation tasks, repeated prompts can lead to unnecessary API calls and cost inefficiencies. To minimize redundancy, ensure that the starting tokens of your system prompt (i.e., the instruction and context that precede the variable content) remain identical across calls. This allows effective caching and reuse of precomputed token embeddings, reducing both latency and expenses.

Batch processing. For large datasets, asynchronous batch submission is considerably more efficient than sending prompts sequentially. The OpenAI Batch API allows researchers to upload a JSONL file of requests and retrieve results within roughly 24 hours at approximately 50% of the regular price. A detailed step-by-step tutorial is available at https://odissei-soda.nl/tutorials/llm_batch_structured_output/.

10.2 Performance

Model hyperparameter tuning. The most influential hyperparameter for annotation stability is the *temperature*. Lower temperatures (e.g., 0 to 0.3) produce more deterministic outputs and are recommended for classification or coding tasks. Higher values increase variability and may encourage creativity, but at the cost of consistency. Where possible, evaluate multiple temperature settings using a small validation subset to identify the best balance between accuracy and diversity.

Consistency prompting. LLMs can exhibit stochastic variation even under identical prompts. A simple yet powerful method to increase reliability is to generate multiple responses for each item and aggregate the results—for example, by taking the majority vote for categorical outputs or the mean for continuous scores. This ensemble-like strategy smooths out random fluctuations and provides a more stable annotation signal.

10.3 Reproducibility

Metadata documentation. Reproducibility requires precise documentation of all components involved in the annotation process. This includes the full prompt text, model identifiers and version numbers (with links to model cards), data descriptions (via data cards), hyperparameter settings, and timestamps of API calls. Maintaining such metadata enables both replication and transparent reporting.

Version control and archiving. Store annotation scripts, prompts, and metadata in a version-controlled repository (e.g., GitHub or GitLab). For long-term preservation and citation, deposit tagged releases on platforms such as Zenodo or OSF. This practice ensures that future researchers can reproduce your annotation pipeline exactly, even if model versions or API defaults evolve over time.

10.4 Common Pitfalls

Data contamination. Avoid publishing or sharing your data and labels before completing the annotation experiment. Public exposure of the dataset may lead to unintended inclusion of your materials in the model’s training corpus, thereby inflating performance.

Data leakage in few-shot examples. When providing few-shot examples within the prompt, ensure that these examples are drawn exclusively from the training portion of your data. Leakage of test instances or related materials into few-shot demonstrations can yield overly optimistic performance estimates and compromise validity.

10.5 Summary

This section highlighted practical strategies to optimize annotation efficiency, performance, and reproducibility. Caching and batch processing can substantially reduce computational cost, while careful hyperparameter tuning and consistency prompting improve reliability. Transparent documentation and version control ensure that annotations remain verifiable and reusable. Finally, avoiding contamination and leakage safeguards the integrity of both model evaluation and scientific inference.

11 Conclusion and Outlook

LLM-assisted annotation presents a promising new paradigm for large-scale text analysis in the social sciences and humanities. Yet, as with any measurement instrument, its utility depends on careful validation and transparent methodological design. By systematically evaluating annotation quality, modeling annotation error, and adopting reproducible practices, researchers can leverage LLMs to generate data that are both efficient and scientifically credible.

Looking ahead, improvements in open-weight LLMs, alignment techniques, and interpretability tools will further expand the methodological toolkit for automated annotation. Integrating these advances with principles from measurement theory and psychometrics will be key to ensuring that LLM-based annotations serve as valid and reliable instruments for empirical research.

References

- Abdurahman, S., Ziabari, A. S., Moore, A. K., Bartels, D. M., and Dehghani, M. (2025). A primer for evaluating large language models in social-science research. *Advances in Methods and Practices in Psychological Science*, 8(2):25152459251325174.
- Artstein, R. and Poesio, M. (2008). Inter-coder agreement for computational linguistics. *Computational Linguistics*, 34(4):555–596.
- Bender, E. M., Gebru, T., McMillan-Major, A., and Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of FAccT*.
- Brown, T. B., Mann, B., Ryder, N., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*.
- Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.
- Conway, M., Kenski, K., and Wang, D. (2019). Political behavior in 140 characters or less: The promise and perils of using twitter data for social science research. *Political Science & Politics*, 52(4):755–760.
- Egami, N., Hinck, M., Stewart, B. M., and Wei, H. (2024). Using large language model annotations for the social sciences: A general framework of using predicted variables in downstream analyses. Working paper; this version November 17, 2024.
- Fang, Q., Giachanou, A., Bagheri, A., Boeschoten, L., van Kesteren, E.-J., Kamalabad, M. S., and Oberski, D. (2023). On text-based personality computing: Challenges and future directions. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10861–10879.
- Fuller, W. A. (1987). *Measurement Error Models*. Wiley, New York.
- Gilardi, F., Alizadeh, M., and Kubli, M. (2023). Chatgpt outperforms crowd workers for text annotation tasks. *Proceedings of the National Academy of Sciences*.
- Grimmer, J. and Stewart, B. M. (2013). Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political Analysis*, 21(3):267–297.
- Gwet, K. L. (2014). *Handbook of Inter-Rater Reliability: The Definitive Guide to Measuring the Extent of Agreement Among Raters*. Advanced Analytics, LLC, Gaithersburg, MD, 4th edition.

- Jurgens, D., Hovy, D., et al. (2023). Large language models for computational social science. *Computational Communication Research*.
- Kapoor, S. and Narayanan, A. (2023). Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*, 4(9).
- Krippendorff, K. (2018a). *Content Analysis: An Introduction to Its Methodology*. Sage Publications.
- Krippendorff, K. (2018b). *Content Analysis: An Introduction to Its Methodology*. SAGE Publications, Thousand Oaks, CA, 4th edition.
- Leech, G., McEnery, T., Wynne, M., and Wilson, A., editors (2003). *Developing Linguistic Corpora: A Guide to Good Practice*. Oxbow Books, Oxford, UK. Arts and Humanities Data Service.
- Martins Van Jaarsveld, G., Wong, J., Baars, M., Specht, M., and Paas, F. (2025). Scaling goal-setting interventions in higher education using a conversational agent: Examining the effectiveness of guidance and adaptive feedback. In *Proceedings of the 15th International Learning Analytics and Knowledge Conference*, LAK '25, page 328–338, New York, NY, USA. Association for Computing Machinery.
- Mitchell, M. et al. (2019). Model cards for model reporting. *Proceedings of the Conference on Fairness, Accountability, and Transparency*.
- Mohammad, S. M. (2016). Sentiment analysis: Detecting valence, emotions, and other affectual states from text. In *Emotion Measurement*, pages 201–237. Elsevier.
- OpenAI (2023). Best practices for prompt engineering with the openai api. Accessed: 2025-10-31.
- OpenAI (2024a). Gpt-4 technical report. <https://cdn.openai.com/papers/gpt-4.pdf>.
- OpenAI (2024b). Prompt engineering best practices for chatgpt. Accessed: 2025-10-31.
- Poletto, F., Basile, V., Sanguinetti, M., Bosco, C., and Patti, V. (2020). Resources and benchmark corpora for hate speech detection: A survey. In *Proceedings of the Twelfth Language Resources and Evaluation Conference (LREC)*.
- Schaefer, R., Hovy, D., and Plank, B. (2023). Annotation uncertainty in natural language processing: A survey. *Computational Linguistics*, 49(3):687–720.
- Shrout, P. E. and Fleiss, J. L. (1979). Intraclass correlations: Uses in assessing rater reliability. *Psychological Bulletin*, 86(2):420–428.

Stajner, S. and Yenikent, S. (2021). Why is MBTI personality detection from texts a difficult task? In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3580–3589. Association for Computational Linguistics.

Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q., and Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS) 35*, pages 24824–24837. Curran Associates, Inc. Pre-print / arXiv version: arXiv:2201.11903.

Ziems, C., Held, W., Shaikh, O., Chen, J., Zhang, Z., and Yang, D. (2024). Can large language models transform computational social science? *Computational Linguistics*, 50(1):237–291.

A R Code

A.1 Obtain and Secure API Keys

```
# Prompt user for API key
openai_api_key <- readline(prompt = "Enter API key for OpenAI: ")
Sys.setenv(OPENAI_API_KEY = openai_api_key)
```

A.2 Minimal Dependencies

```
# Install additional packages not included in Colab by default
install.packages(c("ellmer", "patchwork", "irr", "gridExtra"))
library(devtools)
install_github("naoki-egami/dsl", dependencies = FALSE)
```

```
# Load required packages
library(tidyverse)    # Core collection of R packages for data manipulation,
# visualization, and analysis (e.g., dplyr, ggplot2)
library(patchwork)    # Combines multiple plots into a single frame
library(gridExtra)    # Arranges multiple plots or tables on one page
```

```

library(ellmer)      # R interface for working with LLM APIs
library(irr)         # Computes inter-rater reliability metrics
library(dsl)         # Implements correction of OLS regression estimates

```

A.3 Data Preparation

```

# url where you can download our example data
data_url <- paste0(
  "https://sodascience.github.io/workshop_llm_data_collection/data/",
  "srl_data_example.csv"
)

# Read CSV into dataframe
df <- read_csv(data_url)

```

A.4 System Prompt and Model Initialisation

```

system_prompt <- "
You are an expert in educational assessment and goal evaluation, with
specialized expertise in applying deductive coding schemes to score the quality
and content of student goals.

##TASK##
A university student was given a series of prompts, guiding them through the
process of setting and elaborating on an academic goal for the coming week. You
will be provided with the entire conversation including the prompts, and the
student answers. Your objective is to assess the specificity of of the student's
goal on a scale of 0 to 2 based on the entire conversation.

"

```

```

# Define a function that makes a call to the OpenAI API for gpt-4o-mini.
call_openai_api <- function(system_prompt, user_prompt) {
  chat <- chat_openai(
    model = "gpt-4o-mini",
    system_prompt = system_prompt,
    seed = 42,
    api_args = list(temperature = 0, max_tokens = 1000),

```

```

    echo = "none" #suppress the output from being printed
)
return(chat$chat(user_prompt))
}

```

A.5 A Single Prompt Demo

```

# Create a prompt request with the system prompt and the user prompt based on
# the first conversation from the dataset.

response <- call_openai_api(
  system_prompt = system_prompt,
  user_prompt   = df[["conversation"]][1]
)

# Check the first response
cat(response)

```

A.6 Multiple Prompts Demo

```

# initialize output list
responses <- list()

for (i in 1:10) {
  # extract current info
  cur_convo <- df |> slice(i) |> pull("conversation")
  cur_id     <- df |> slice(i) |> pull("id")

  # get rating from llm
  response <- call_openai_api(
    system_prompt = system_prompt,
    user_prompt   = cur_convo
  )

  # assign to output list
  responses[[cur_id]] <- response
}

```

```

# report progress (does not work well in colab)
cat(i, "/ 10 completed.\n")
}

# Let's inspect the responses!
cat(responses[[2]])

```

A.7 Structuring LLM Annotations

A.7.1 Structured Output

To force the LLM to produce outputs in formats specified by you, you need to use the ‘\$extract_data()‘ method instead of the ‘\$chat‘ method.

Below, we define our desired output format as:

- "specificity_score": an integer (either 0, 1 or 2) reflecting the specificity of a conversation.
- "reasoning": a string that provides the model’s reasoning.

```

output_structure <- type_object(
  specificity_score = type_integer("The specificity score of the entire
  conversation on a scale of 0, 1 and 2."),
  reasoning = type_string("Your reasoning process.")
)

call_openai_api_structured <- function(system_prompt, user_prompt) {
  chat <- chat_openai(
    model = "gpt-4o-mini",
    system_prompt = system_prompt,
    seed = 42,
    api_args = list(temperature = 0, max_tokens = 1000),
    echo = "none" #suppress the output from being printed
  )
  response <- chat$extract_data(user_prompt, type = output_structure)
  return(response)
}

```

A.7.2 A Single Prompt Demo

```
structured_response <- call_openai_api_structured(system_prompt, df[["conversation"]][1])
print(structured_response)
```

A.7.3 Multiple Prompts Demo

```
structured_responses <- list()
for (i in 1:10) {
  # extract current info
  cur_convo <- df |> slice(i) |> pull("conversation")
  cur_id     <- df |> slice(i) |> pull("id")

  # get rating from llm
  response <- call_openai_api_structured(
    system_prompt = system_prompt,
    user_prompt   = cur_convo
  )

  # assign to output list
  structured_responses[[cur_id]] <- response

  # report progress (does not work well in colab)
  cat(i, "/ 10 completed.\n")
}
```

A.8 Evaluate Annotation Quality

The `kripp.alpha()` function from the `irr` package can be used to calculate agreement (i.e., Krippendorff's Alpha) of specificity scores between two raters (e.g., LLMs and human experts). Let's check the agreement between the specificity scores we got from the LLM above and the human expert-coded specificity scores!

```
# create rating matrix (rows = raters, cols = items)
rating_matrix <- rbind(
  df |> slice(1:10) |> pull(score_specificity_human),
  response_df |> pull(specificity_score)
)
```

```
# compute agreement (0 - 1)
kripp.alpha(rating_matrix, method = "interval")
```