Advice for using ChatGPT in Research

Neil He[1,3]

Special thanks to Kiran Dwivedi[2,3] ~~and GPT-4.o~~

Febuary 3, 2026

[1]Department of Economics, University of Southern California

[2]Department of Economics, Cornell University

[3]Microsoft Research

Intro: Why this talk

- LLMs are not “smart calculators”; they are **text-to-structure machines**.
- The win: convert messy language into variables you can regress on.
- Today: minimal API + prompts that actually work + how to trust the output.

Motivating Example: Performance Comparison

Metric	**GPT-4.0**	**Neil-1.0**
Reviews per minute (RPM)	33.3	3.8
Total Time (TT)	60 mins	DNF
Average Cost per Review (AC)	$\approx$ 10¢	$>$ 10¢

Note: No parallelization (feature unavailable for Neil-1.0).

Conclusion

This is basically data labeling. But for text. Patents, interviews, SEC filings, meeting minutes, policy documents, etc.

Motivating Example: Movie Reviews

Data

Captain Marvel movie reviews ($n = 2{,}000$) scraped from Rotten Tomatoes. Example:

> *"Already had low expectation [sic] but it still ended up disappointing."*

Task

Classify whether the sentiment is "politically motivated" (1) or not (0).

Before touching prompts: define the label (Rubric)

Operational definition

"Politically motivated" means the review explicitly frames the evaluation in terms of politics / ideology / culture-war language (e.g., woke/SJW/agenda, left/right, party politics, identity politics), rather than just movie quality.

Examples (from our dataset)

- **1 (political):** "This is pure woke messaging. I came for a superhero movie, got a lecture."
- **0 (not political):** "Boring pacing and flat dialogue. The villain was forgettable."
- **Borderline rule:** vague "agenda" with no explicit politics ⇒ label 0

Why this slide exists

If your definition is fuzzy, no prompt can save you.

Motivating Example: Intro to API (minimal)

Simple query function

```
def ask_GPT(prompt):
    response = client.chat.completions.create(
        model="gpt-4",
        messages=[
          {"role": "system", "content": "You are a helpful assistant."},
          {"role": "user", "content": prompt}
        ],
    )
    return response.choices[0].message.content
```

Example prompt

prompt = *"Below is a movie review. Return 1 if politically motivated, else 0: "* + `review_text`

I'll tell why you SHOULD NOT use this in a moment.

Step-by-step: what the notebook does

1. Load reviews (CSV)
2. Define rubric + JSON schema (Pydantic)
3. Call model and parse typed JSON (`parse()`)
4. Enforce word limit via second-pass rewrite
5. Write `labeled_reviews_output.csv`

Core design choice

Treat the LLM as a measurement instrument: text $\longrightarrow$ structured variables.

Step 1: Load / create the dataset

Goal

Get a simple table:

- `review_id`
- `review_text`
- optional human label for QC

```python
from pathlib import Path
import pandas as pd
import random

DATA_PATH = Path("fake_movie_reviews.csv")

def maybe_make_fake_dataset(path: Path, n: int = 30, seed: int = 7):
    random.seed(seed)
    political = [
        "This is woke propaganda dressed as a movie.",
        "Another SJW agenda push. Hard pass.",
        "Left/right culture-war nonsense ruined the plot.",
        "Pure identity politics. Not cinema.",
        "Party politics in superhero form. Obvious agenda.",
        "This felt like partisan messaging, not storytelling.",
    ]
    nonpolitical = [
        "Bad pacing and weak dialogue.",
        "The acting was fine but the plot was messy.",
        "Great visuals, mediocre script.",
        "Too long; the third act dragged.",
        "Sound mixing was awful in my theater.",
        "Fun movie, not perfect, but enjoyable.",
    ]
    rows = []
    for i in range(1, n+1):
        if random.random() < 0.35:
            txt = random.choice(political)
            y = 1
        else:
            txt = random.choice(nonpolitical)
            y = 0
        if random.random() < 0.25:
            txt += " " + random.choice(["Seriously.", "LOL.", "Just my opinion."])
        rows.append({"review_id": i, "review_text": txt.strip(), "true_is_political": y})
    pd.DataFrame(rows).to_csv(path, index=False)

if not DATA_PATH.exists():
    maybe_make_fake_dataset(DATA_PATH)

df = pd.read_csv(DATA_PATH)
df.head()
```

Step 2–4: Schema → parse() → enforce word limit

What matters

- Schema controls output shape
- `parse()` eliminates `json.loads` pain
- Second pass enforces hard constraints (words/style)

```python
def label_one(review_text: str) -> dict:
    prompt = build_prompt(review_text)
    completion = client.chat.completions.parse(
        model=MODEL,
        messages=[
            {"role": "system", "content": "Return JSON only."},
            {"role": "user", "content": prompt},
        ],
        response_format=PoliticalLabel,
    )
    msg = completion.choices[0].message
    if getattr(msg, "refusal", None):
        raise RuntimeError(msg.refusal)
    parsed = msg.parsed
    return parsed.model_dump() if hasattr(parsed, "model_dump") else parsed.dict()

def enforce_reasoning_limit(result: dict, max_words: int = 50) -> dict:
    prompt_2 = f"Rewrite reasoning to <= {max_words} words. Keep is_political unchanged. JSON only."
    completion = client.chat.completions.parse(
        model=MODEL,
        messages=[
            {"role": "system", "content": "Return JSON only."},
            {"role": "assistant", "content": str(result)},
            {"role": "user", "content": prompt_2},
        ],
        response_format=PoliticalLabel,
    )
    msg = completion.choices[0].message
    if getattr(msg, "refusal", None):
        raise RuntimeError(msg.refusal)
    parsed = msg.parsed
    out = parsed.model_dump() if hasattr(parsed, "model_dump") else parsed.dict()
    # hard guard
    if word_count(out["reasoning"]) > max_words:
        out["reasoning"] = " ".join(out["reasoning"].split()[:max_words])
    return out
```

Step 5: Batch run + write labeled CSV

Output

A flat file you can merge into your analysis pipeline:

- `pred_is_political`
- `reasoning` (debug only)
- `reasoning_words`

```python
rows = []
for _, r in df.iterrows():
    out = label_one(r["review_text"])
    out = enforce_reasoning_limit(out, max_words=50)
    rows.append({
        "review_id": int(r["review_id"]),
        "review_text": r["review_text"],
        "true_is_political": int(r["true_is_political"]),
        "pred_is_political": int(out["is_political"]),
        "reasoning": out["reasoning"],
        "reasoning_words": word_count(out["reasoning"]),
    })

out_df = pd.DataFrame(rows)
out_df.head()
```

Advice 1: Don't let the output freestyle

Don't do

```
def ask_GPT(content):
    response = client.chat.completions.create(
        model="gpt-4",
        messages=[
          {"role":"system","content":"You are helpful"},
          {"role":"user","content":content}
        ],
    )
    return response.choices[0].message.content
```

Reason

Inconsistent formats $\Rightarrow$ painful post-processing: "Yes", "1", "Political", "Not political but ..."

Advice 2 (biggest takeaway): Force JSON

Do: specify a schema (keep it small)

```
JSONschema = {
  "is_political": "int (0/1)",
  "reasoning": "string (<=50 words)"
}
```

Why JSON is worth a whole slide

- **is_political** goes straight into your dataset
- **reasoning** is your debugging signal (not for the paper)

Pro tip

Put the schema in the system/user prompt and demand: **JSON only**.

Advice 3: Debug prompts using reasoning (not vibes)

Do

Ask for reasoning when you are iterating on the prompt. Then read the failures and fix the prompt systematically.

What a typical response looks like

```
{
  "is_political": 0,
  "reasoning": "The review criticizes movie quality and does not reference politics or culture-war
        framing."
}
```

How you actually use this

If the model is wrong, the reasoning tells you **which part of your rubric/prompt is unclear**.

Advice 4: Word limits are *soft* — enforce constraints explicitly

Problem

"¡=50 words" in the prompt is a **soft constraint**. It helps, but it is not guaranteed (even with GPT-5).

Don't do

Re-run and pray it becomes shorter / cleaner.

Do: two practical ways

1. **Token cap (API-level, hard-ish):** set `max_output_tokens` to prevent runaway verbosity
2. **Sequential rewrite (hard + best quality):** second pass rewrites to meet word/format constraints

Advice 4 (cont.): Token cap vs sequential rewrite

Option 1: Token cap (cheap, but tokens $\neq$ words)

```
response = client.chat.completions.create(
  model="gpt-5-mini",
  max_output_tokens=160,
  response_format={"type":"json_object"},
  messages=[
    {"role":"system","content":"Return a JSON object only."},
    {"role":"user","content": prompt_1}
  ]
)
```

Option 2: Sequential rewrite (best for word limits + formatting)

```
prompt_2 = "Rewrite reasoning to <=50 words. JSON only."

messages = [
  {"role":"system","content":"Return a JSON object only."},
  {"role":"user","content": prompt_1},
  {"role":"assistant","content": answer_1},
  {"role":"user","content": prompt_2}
]
```

Token cap controls cost/latency. Sequential rewrite controls **style/wording/length**.

Advice 5: Evaluate like it's measurement (because it is)

Minimal Quality Control loop

1. Build a small gold set (e.g., 50 items) with human labels
2. Run GPT on the same set
3. Read disagreements + reasoning; bucket failures (sarcasm/ambiguity/etc.)
4. Update rubric/prompt; re-run the same gold set

Key point

Prompt = hyperparameter. Treat it like model selection.

Advice 6: Keep runs reproducible (lowest effort version control)

Write down 4 things every time you run

- **model** (gpt-4 / gpt-5-mini / ...)
- **prompt_version** (prompt_v1, v2, v3...)
- **schema_version** (schema_v1, v2...)
- **run_id/date**

Common failure modes (for this task)

What goes wrong in “politically motivated” labeling

- **Over-expanding the concept:** strong emotion $\neq$ political
- **Ambiguity:** “agenda” with no explicit political content
- **Sarcasm:** literal text says A, intent is not-A
- **Long reviews:** model focuses on the first sentence and misses the key one

Fixes you can do with just prompting

Tighten rubric + add counterexamples + require reasoning during iteration + gold-set feedback.

Wrap-up: The whole workflow in one slide

From text to variables

Rubric → Prompt(v1) → JSON labels → Gold set QC → Prompt(v2) → Full run → Analysis

Thank you!

Questions / feedback welcome.

`neil.he@usc.edu`