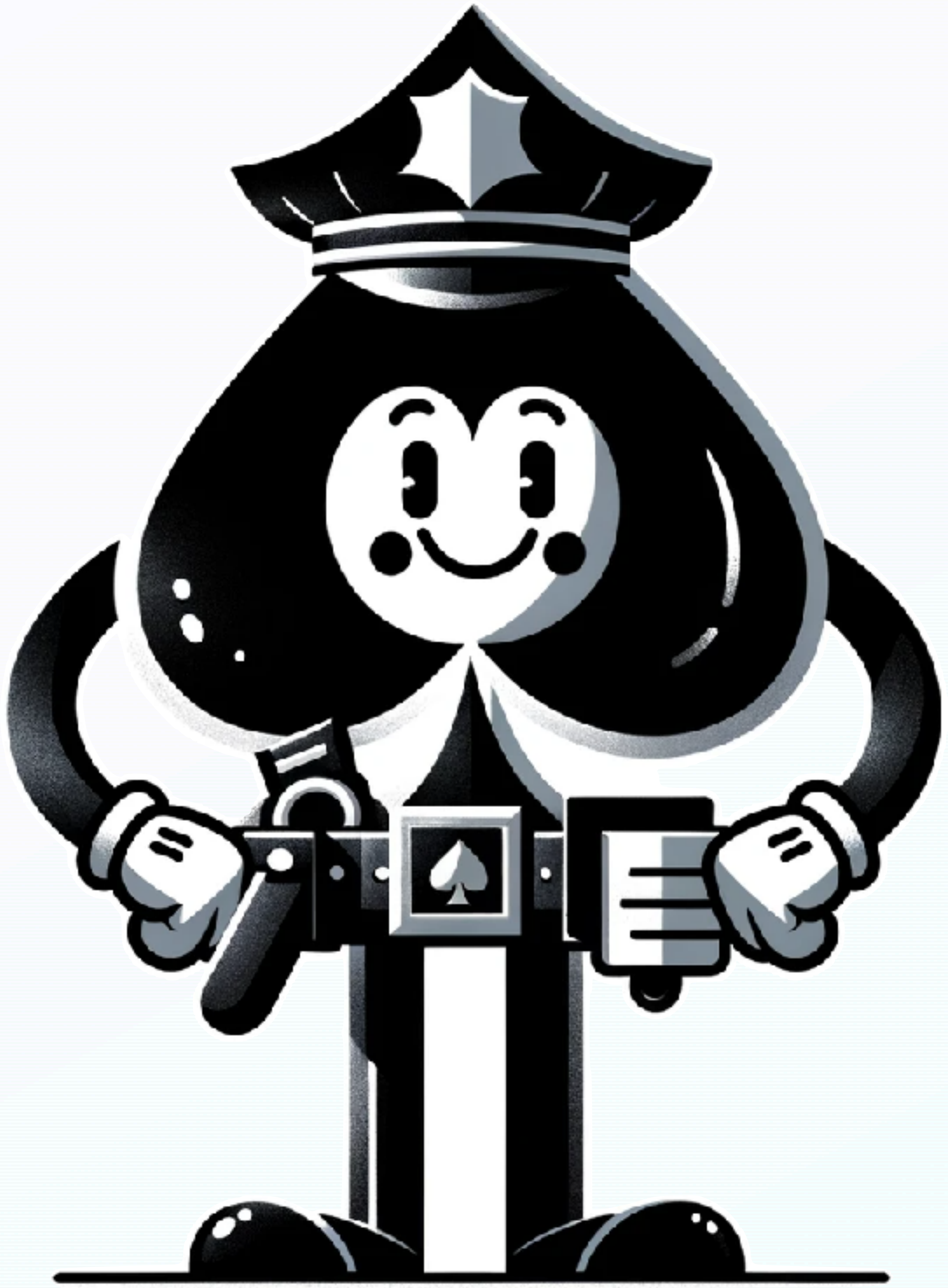
SPADE: A System for Prompt Analysis and Delta-Based Evaluation

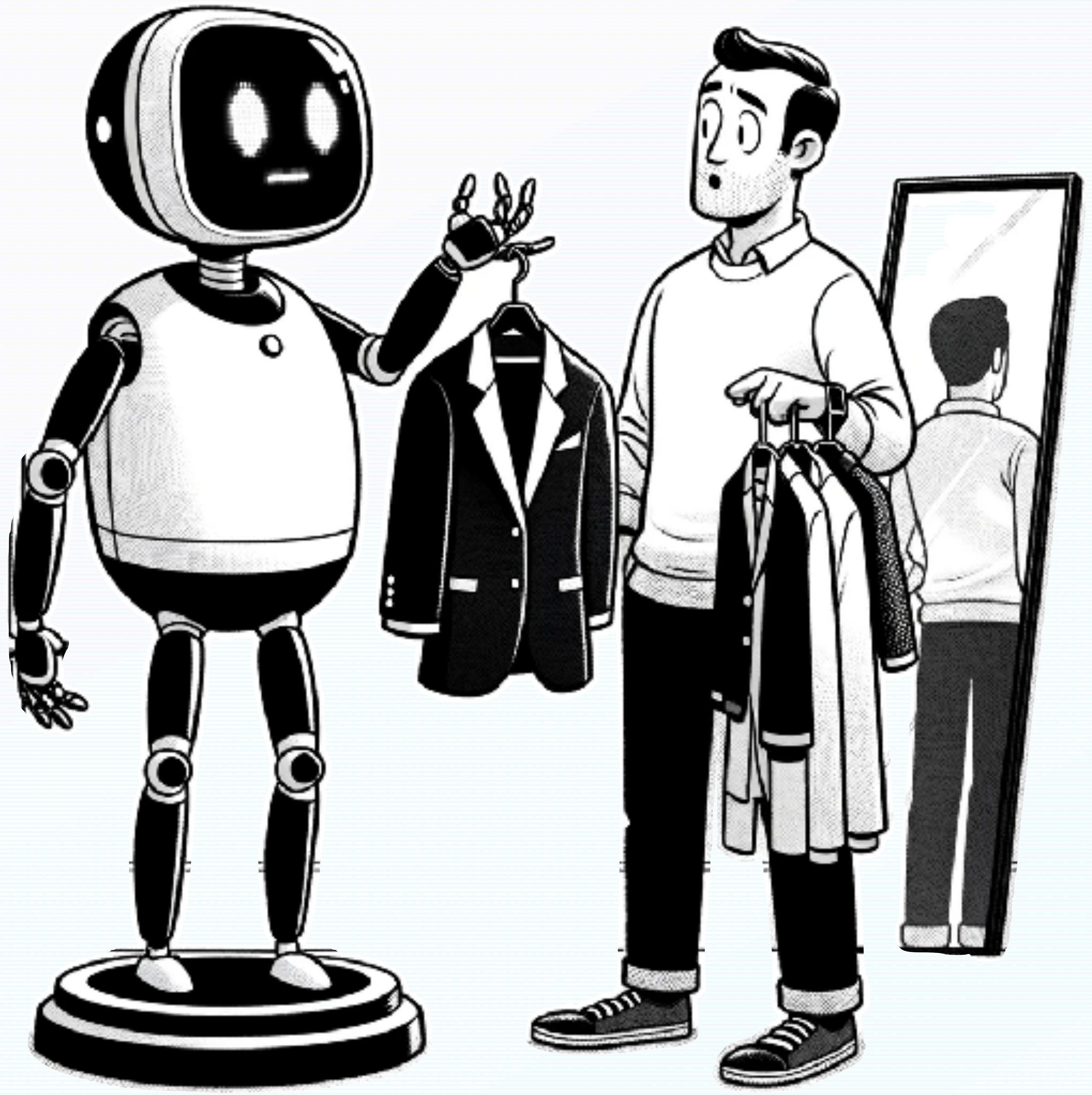
Shreya Shankar
November 2023

EPIC DATA lab
UC Berkeley



How do people write custom LLM pipelines?

- Consider an example use of LLMs: an AI-powered personal stylist
- Prompt: “what should I wear to a conference?”
- LLM response: “Certainly, I can help you decide what to wear to your conference!....”
- A developer needs to turn this into a template that they can run for *different queries* and *reliably* extract output from without human supervision



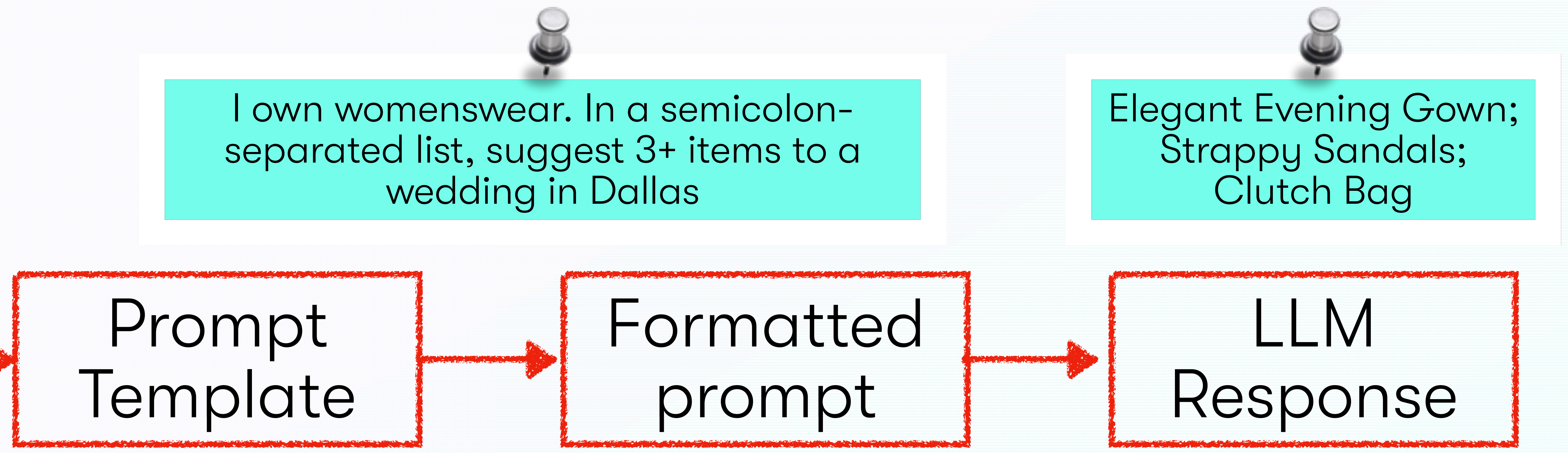
Custom LLM pipelines

- Chat GPT prompt: “what should I wear to a conference?”

Placeholder in a prompt template

- A prompt template for an automated pipeline: “I own {gender}. Suggest 3+ items to wear to {event} in a semicolon-separated list”

Inputs
{“gender”: “womenswear”, “event”: “a wedding in Dallas”}
{“gender”: “menswear”, “event”: “my company holiday party”}
{“gender”: “womenswear”, “event”: “hiking Mt. Diablo”}
{“gender”: “menswear”, “event”: “a friend’s 30th bday party”}



Monitoring LLM Response Quality is Hard

- Prompt templates get deployed to production with no clear sense of how well the pipeline will perform
- Accuracy and “good” are poorly defined for free-form responses. No clear way to evaluate custom tasks.
- Holistic evaluation may be subjective, but every task has some objective indicators of correctness
- Can we automatically recommend assertions for LLM pipelines?

SPADE ♠: System for Prompt Analysis and Delta-Based Evaluation

1. Identify phrases in the prompt that indicate potential assertions
2. Write assertions as Python functions that operate on formatted prompt & response pairs
3. Reduce redundancies & inaccuracies in the assertions

...For wedding-related events, don't suggest any white items unless the client explicitly states that they want to be styled for their wedding...

```
def check_excludes_white_wedding(prompt: str, response: str) -> bool:
    """
    This function checks if the response does not include white items
for wedding-related events,
    unless explicitly stated by the client.
    """
    # Check if event is wedding-related
    if "wedding" in prompt.lower() and "my wedding" not in
prompt.lower():
        # Check if the response includes the word "white"
        return "white" not in response.lower()
    else:
        return True
```

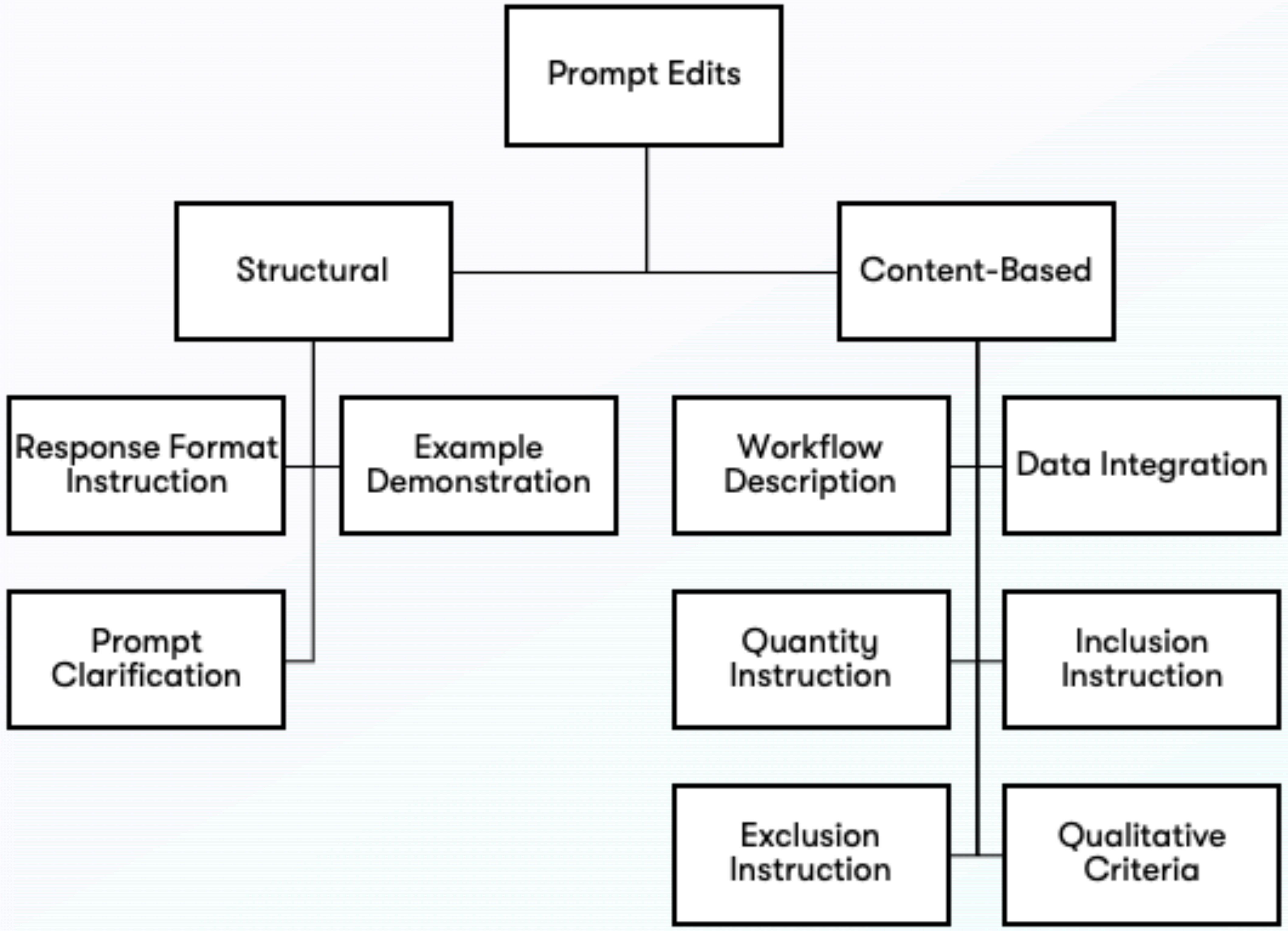
Identifying Assertion Concepts

- Without explicitly asking the developer to write assertion criteria…
  - What's important to the engineer?
  - What LLMs are uniquely bad at?
- Prompt template provenance

Version	Prompt Template
1	Suggest 5 apparel items to wear to {event}. Return your answer as a Python list of strings.
2	A client ({client_genders}) wants to be styled for {event}. Suggest 5 apparel items for {client_pronoun} to wear. Return your answer as a Python list of strings.
3	A client ({client_genders}) wants to be styled for {event}. Suggest 5 apparel items for {client_pronoun} to wear. For wedding-related events, don't suggest any white items unless the client explicitly states that they want to be styled for their wedding. Return your answer as a python list of strings.

Identifying Assertion Concepts

- An edit to a prompt can indicate potential assertion concepts that developers care about!
  - E.g., If someone adds “Don’t include X” then LLM responses should never include X
  - Challenging to consider conditionals, e.g., “For wedding-related events, don’t suggest white”
- Looking at the deltas (i.e., diffs) between prompt template versions for 19 pipelines from LangChain, we identified 9 types of deltas



Identifying Assertion Concepts

- We use an LLM to dissect a prompt into the relevant *categories* and *concepts*
- Can be done with open-source models like Mistral or Llama
- Runs for every prompt template version
- A concept is associated with a category, prompt template, and source (phrase in the prompt template)

Category	Example Addition or Edit to a Prompt	Concept
Response Format Instruction	*"Return your answer as a Python dictionary"*	Can be parsed as dictionary
Example Demonstration	*"Here is an example question and response: Question: What should I wear to a workout class? Answer: {"tops": "black moisture-wicking tank top",*	Dictionary includes keys "tops," "bottoms," etc.
Prompt Clarification	*"~~Return~~ Give me a descriptive list"*	N/A (as long as the meaning of the prompt is unchanged)
Workflow Description	*"First, identify the dress code of the event. Then..."*	Dress code is correct for the event
Data Integration	*"The user does not like {dislikes_placeholder}"*	N/A
Quantity Instruction	*"The outfit must have at least 3 items"*	>= 3 items
Inclusion Instruction	*"Make sure your outfit is complete, i.e., it includes a top, shoe, and lower-body garment"*	Dictionary includes keys "tops," "shoes," "bottoms"
Exclusion Instruction	*"Do not suggest sneakers for wedding-related events"*	"Sneakers" not in response if "wedding" in example
Qualitative Criteria	*"Include a statement piece in your suggestion"*	Check if there is a statement piece (need LLM or human)

SPADE ♠: System for Prompt Analysis and Delta-Based Evaluation

1. Identify phrases in the prompt that indicate potential assertions

2. **Write assertions as Python functions that operate on formatted prompt & response pairs**

3. Reduce redundancies & inaccuracies in the assertions

Write Assertions as Python Functions

- Given the concepts found in a prompt template, e.g., “quantity instruction” and “>= 3 items,” GPT-4 generates Python functions
  - Functions can call *ask_llm*
  - Functions can batch concepts
- Depending on how many prompt versions exist, 10s or even 100s of assertions get generated! 🤯
- We could only generate assertions for the last prompt version, but we may lose important criteria, and/or the LLM may not recall all criteria if the prompt is very long

fashion_experiment > dataset_creation.py > ...

```python
async def collect_all_evals(prompt_templates: List[str]):
    evals = []
    for template in prompt_templates:
        reply_json, eval_functions, messages = await suggest_evals("", template)
        print(reply_json)
        evals += eval_functions

    # Get all eval functions
    eval_code = "\n\n".join([e["code"] for e in evals])
    return eval_code


async def check_JSON_format_1(prompt: str, response: str) -> bool:
    """
        Check if the response is in valid JSON format by trying to load it with
        the json
    library.
    """
    try:
        json.loads(response)
        return True
    except json.JSONDecodeError:
        return False


async def check_replaced_none(prompt: str, response: str) -> bool:
    """
    Check if 'None' values have been replaced by some item descriptions.
    """
    loaded_response = json.loads(response)
    # Iterate through all the values to check if any is 'None'
    return all(i != "None" for i in loaded_response.values())


async def check_description_inclusion(prompt: str, response: str) -> bool:
    """
    Check if the response includes a short, descriptive phrase for every item.
    """
    loaded_response = json.loads(response)
```

SPADE ♠: System for Prompt Analysis and Delta-Based Evaluation

1. Identify phrases in the prompt that indicate potential assertions

2. Write assertions as Python functions that operate on formatted prompt & response pairs

3. **Reduce redundancies & inaccuracies in the assertions**

Assertion Examples

- ask_llm(“Is this outfit appropriate for the weather?”)
- ask_llm(“Is this outfit appropriate for the temperature and season?”)
- ask_llm(“Does this outfit make sense given the possible weather?”)

```
def check conciseness(prompt, response):
  return len(response.split(“ “)) < 5
```

Eliminate Redundant Assertions

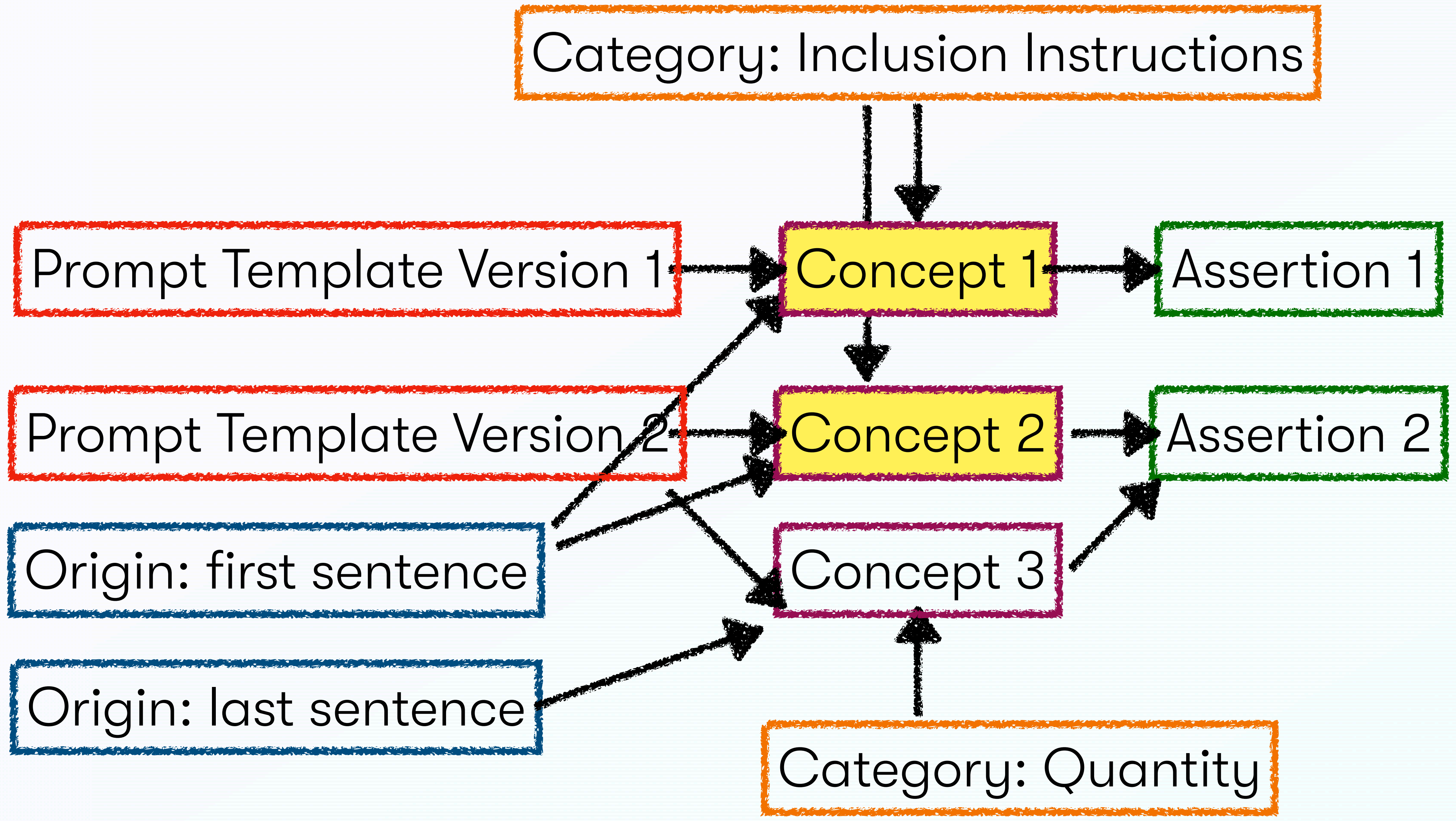
- We want the fewest # of assertions where our chosen assertions:
  - *(Recall constraint)* Cover all the actual bad examples
  - *(Accuracy constraint)* Don't fail too many examples that the developer believes to be good, i.e., "false failure rate"
- Since this is pre-deployment, we can't assume access to a representative set of examples
- Recall constraint -> Cover all the concepts found by SPADE. Need to do concept deduplication/ entity resolution to get the set of all concepts.

```
def check_professional(prompt: str,
response: str) -> bool:
    """
    This function checks if the response
includes professional attire for work-
related formal events.
    """
    return ask_llm(f"Is the outfit
suggestion {response} professional enough
for the event described in {prompt}?")
```

Might not be 100% accurate!

Eliminate Redundant Assertions

- Concept deduplication — entity resolution problem
- Each assertion is associated with 1+ concepts (e.g., check_num_items checks ability to parse into a list & number of items)
- Each concept is associated with a category, prompt template version, & origin (phrase in prompt)
- Blocking rules:
  - Concepts with same category from our taxonomy
  - Concepts with same origin
  - Concepts from different template versions



Eliminate Redundant Assertions

- Each assertion has 1+ concepts, 1+ categories, and a version #
- Estimate accuracies with a sample of synthetically-generated and human-labeled ~50 examples
- Set cover problem:
  - Let f_i = assertion i
  - Let ffr_i = % examples that are good and f_i fails, i.e., false failure rate
  - Minimize sum $ffr_i * z_i$, where z_i indicates whether f_i is chosen, subject to coverage constraint?

$\boxed{ffr_{i \text{ and } j} \leq ffr_i + ffr_j}$

Eliminate Redundant Assertions

- Set cover problem:
  - Let f_i = assertion i
  - Let ffr_i = % examples that are good and f_i fails, i.e., false failure rate
  - Minimize sum ffr_i * z_i, where z_i indicates whether f_i is chosen, subject to coverage constraint?
- Not fully optimal, because ffr of the conjunction of chosen assertions F' can be less than the sum of individual ffr's for f_i in F'
  - max (ffr_i, ffr_j) <= ffr_{i and j} <= sum (ffr_i, ffr_j)
- Can implement a branch & bound solution. Branch step = including or excluding a function f_i, bound when ffr > threshold, # functions > size of best set so far, or feasibility

SPADE ♠: System for Prompt Analysis and Delta-Based Evaluation

1. Identify phrases in the prompt that indicate potential assertions
2. Write assertions as Python functions that operate on formatted prompt & response pairs
3. Reduce redundancies & inaccuracies in the assertions

Summary and Looking Ahead

- Deploying custom LLM pipelines requires some assertions
- It's hard to come up with assertions automatically, without much data or insight into what developers care about
- To align with what developers care about, we analyze edits to prompts
- To create assertions, we use LLMs + implement a pipeline to remove redundant assertions
- Need to run experiments