schema.json

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "company": {
      "type": "string",
      "description": "Exact employer name as it appears in the sentence"
    },
    "ai_causal": {
      "type": "string",
      "enum": ["yes", "no"],
      "description": "yes if AI/automation is the explicit reason for the staffing change"
    },
    "headcount": {
      "type": ["integer", "null"],
      "minimum": 0,
      "description": "Whole number of roles affected, or null when unspecified"
    },
    "job_titles": {
      "type": "array",
      "items": {
        "type": "string"
      },
      "description": "Zero-or-more occupational titles mentioned in the sentence"
    }
  },
  "required": ["company", "ai_causal", "headcount", "job_titles"],
  "additionalProperties": false
}
```

system_prompt.txt

text

```
You are an analyst who extracts structured facts from corporate layoff and hiring-freeze senter

Schema:
company        - string
ai_causal      - "yes" | "no"
headcount      - integer | null
job_titles     - array of strings (may be empty)

Version: 1.0
```

user_prompt_template.txt

text

```
Extract the required fields from the following sentence.

Sentence: "{{SENTENCE_HERE}}"

Extraction rules:
1. If the sentence does not specify a number of positions, set "headcount": null.
2. If AI, automation, robotics, or chatbots are described but **not blamed** for staffing chang
3. Company names should be written exactly as they appear.
4. If no job titles are stated, return an empty array [].

Return only the JSON object—do NOT wrap it in markdown, code fences, or add commentary.
```

few_shot_examples.jsonl

jsonl

```
{"sentence": "We will eliminate 700 customer-service roles because the new chatbot handles 80%
{"sentence": "Amazon laid off 150 warehouse workers due to automated sorting systems.", "compar
{"sentence": "Although we have invested in AI analytics, layoffs are driven by weak consumer de
{"sentence": "Tesla reduced headcount by terminating employees following budget constraints.",
{"sentence": "Microsoft is cutting 25% of its quality assurance team as automated testing tools
{"sentence": "Google froze hiring for software engineers while implementing machine learning re
{"sentence": "The bank eliminated 45 loan officers and 30 tellers after deploying AI-powered lo
{"sentence": "Meta announced workforce reductions affecting data entry clerks since automation
```

validate_return.py

python

```python
#!/usr/bin/env python3
import json
import sys

def validate_schema(data):
    """Validate that the JSON data conforms to the required schema."""

    # Check required fields
    required_fields = ["company", "ai_causal", "headcount", "job_titles"]
    for field in required_fields:
        if field not in data:
            return f"Missing required field: {field}"

    # Check for extra fields
    allowed_fields = set(required_fields)
    actual_fields = set(data.keys())
    extra_fields = actual_fields - allowed_fields
    if extra_fields:
        return f"Extra fields not allowed: {list(extra_fields)}"

    # Validate field types and values
    if not isinstance(data["company"], str):
        return "company must be string"

    if data["ai_causal"] not in ["yes", "no"]:
        return "ai_causal must be 'yes' or 'no'"

    if data["headcount"] is not None:
        if not isinstance(data["headcount"], int) or data["headcount"] < 0:
            return "headcount must be non-negative integer or null"

    if not isinstance(data["job_titles"], list):
        return "job_titles must be array"

    for title in data["job_titles"]:
        if not isinstance(title, str):
            return "job_titles must contain only strings"

    return None

def main():
    if len(sys.argv) != 2:
        print("Usage: python validate_return.py '<json_string>'")
        sys.exit(1)
```

```
    json_string = sys.argv[1]

    try:
        data = json.loads(json_string)
    except json.JSONDecodeError as e:
        print(f"Invalid JSON: {e}")
        sys.exit(1)

    error = validate_schema(data)
    if error:
        print(f"Schema validation error: {error}")
        sys.exit(1)

    print("Validation successful")
    sys.exit(0)

if __name__ == "__main__":
    main()
```

README.md

markdown

```
# AI-Causal Layoff Extraction Pipeline

This package provides assets for extracting structured facts from corporate layoff sentences, s

## Files Overview

- `schema.json`: JSON Schema Draft-07 specification for layoff data extraction
- `system_prompt.txt`: System message for the AI model (v1.0)
- `user_prompt_template.txt`: User message template with `{{SENTENCE_HERE}}` placeholder
- `few_shot_examples.jsonl`: 8 labeled examples covering various layoff scenarios
- `validate_return.py`: Python validation script for model responses

## Schema Fields

- `company`: Exact employer name as mentioned in sentence
- `ai_causal`: "yes" if AI/automation explicitly causes staffing changes, "no" otherwise
- `headcount`: Integer number of affected roles, or null if unspecified
- `job_titles`: Array of job titles mentioned (empty array if none)

## Usage

### OpenAI API Example

```python
import openai
import json

# Load prompts
with open('system_prompt.txt', 'r', encoding='utf-8') as f:
    system_prompt = f.read().strip()

with open('user_prompt_template.txt', 'r', encoding='utf-8') as f:
    user_template = f.read().strip()

# Process sentence
sentence = "Tesla laid off 200 assembly line workers due to new robotic manufacturing systems."
user_prompt = user_template.replace('{{SENTENCE_HERE}}', sentence)

response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt}
    ],
    temperature=0.0
```
```

```
    temperature=0.0
)

result = response.choices[0].message.content
```

Anthropic API Example

python

```python
import anthropic

client = anthropic.Anthropic(api_key="your-key")

# Load prompts (same as above)
message = client.messages.create(
    model="claude-3-sonnet-20240229",
    max_tokens=500,
    temperature=0.0,
    system=system_prompt,
    messages=[{"role": "user", "content": user_prompt}]
)

result = message.content[0].text
```

Validation

bash

```bash
python validate_return.py '{"company": "Tesla", "ai_causal": "yes", "headcount": 200, "job_titl
```

Token Usage Tips

- **System prompt**: ~60 tokens
- **User prompt**: ~80-100 tokens depending on sentence length
- **Expected response**: ~30-50 tokens
- **Total per call**: ~170-210 tokens

For high-volume processing:

1. Use temperature 0.0 for consistency
2. Batch sentences when possible
3. Cache company name variations
4. Log failed validations for model retraining

AI Causality Detection

The pipeline distinguishes between:

- **Explicit causality**: "laid off due to automation"
- **Correlation only**: "invested in AI, but layoffs due to market conditions"
- **Implicit causality**: "eliminated roles as robots handle the work"

Best Practices

1. **One sentence per call**: Maximizes accuracy and minimizes token cost
2. **Validate immediately**: Use `json.loads()` and schema validation on every response
3. **Audit trail**: Log prompt, model, and completion_id for each extraction
4. **Version control**: Update system prompt version when changing extraction rules
5. **Quality assurance**: Human review recommended for ambiguous cases

Common Edge Cases

- **Percentage-based layoffs**: "Cut 15% of workforce" → headcount: null
- **Multiple job titles**: Extract all mentioned roles
- **Company subsidiaries**: Use exact name as stated
- **Hiring freezes**: Treat as headcount: null with appropriate job_titles