Structured Data with LLMs Done Right

A Practical Guide to Text Classification, Information Retrieval and Generation with LLMs

Jesus Villota

CEMFI
jesus.villota@cemfi.edu.es

November 11, 2025

Outline

Three Core Tasks in Text Analysis

There are three fundamental ways of mapping text to structured data:

1. Text Classification

Assign predefined labels from a finite set of categories to text units.

$$f : \mathcal{D} \to \mathcal{C}$$

Example: Classify central bank statements as hawkish, dovish, or neutral

2. Information Retrieval

Extract specific data fields or entities from text.

$$g : \mathcal{D} \to \mathcal{V}_1 \times \mathcal{V}_2 \times \cdots \times \mathcal{V}_m$$

Example: Extract GDP forecasts, policy rates, and dates from Fed minutes

3. Structured Generation

Create new textual content adhering to a predefined schema.

$$h : \mathcal{D} \times \mathcal{S} \to \mathcal{T}$$

Example: Generate policy briefs with standardized sections

Motivation

LLMs are transforming empirical economics research:

- Central bank communication analysis (monetary policy stance)
- Financial sentiment from news and earnings calls
- ESG classification of corporate disclosures
- Contract information extraction (loan agreements, M&A)
- Policy text processing and analysis
- Literature review automation

The Problem

Current approaches relying on natural language prompting are fundamentally unreliable — threatening research validity and reproducibility

The Problem with Current Approaches

The Standard Approach: Prompting

Most researchers use natural language prompting:

```
prompt = f"""
Classify the following text as 'positive', 'negative', or 'neutral'.
Return the result in JSON format:
{{
  "sentiment": "positive",
  "confidence": 0.85
}}

Text: {text}
"""

response = llm.generate(prompt)
```

Why This Fails

The model is **instructed** but not **constrained** to follow the format.

Reliability Crisis

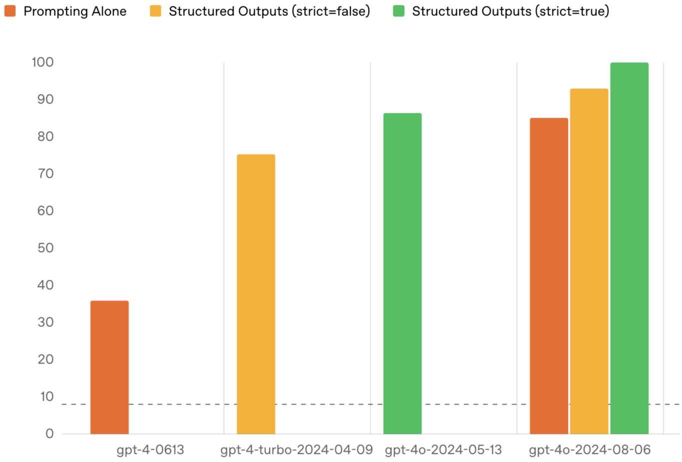
LLMs are stochastic processes $\Rightarrow$ they often fail to follow instructions:

- ✗ Return text that is not valid JSON
- ✗ Use different key names than specified
- ✗ Omit required fields or add unexpected fields
- ✗ Return values outside predefined categories
- ✗ Produce inconsistent formats across different inputs

Impact on Research

- ✗ **Broken** downstream processing **pipelines**
- ✗ Manual intervention introduces **bias**
- ✗ Poor **reproducibility** across model versions
- ✗ Impossible to **compare models** fairly

The Reliability Gap: Empirical Evidence



- **Prompting alone:** 85.1% adherence to schema instructions (GPT-4o)
- **Enriching prompt with structured schema:** 93% adherence to schema instructions (GPT-4o)
- **Structured schemas with strict mode:** **100% adherence** (GPT-4o)

The Solution: Function Calling Schemas

What are Function Calling Schemas?

Function calling (also: "tool use" or "tool choice") allows researchers to define precise schemas that **guarantee** structured, parseable results.

Brief History

- Pioneered by OpenAI in 2023 with Chat Completions API
- Quickly adopted by all major LLM providers:
  - OpenAI, Anthropic Claude, Google Gemini, xAI Grok
  - Mistral, Cohere, GroqCloud, Azure OpenAI
- **Strict mode** achieves 100% schema adherence
- Now an industry standard for structured output

Transform structured output from a **probabilistic formatting challenge** into a **deterministic structural contract**

Unified Framework Structure

Any structured output task can be formulated as a function calling (JSON) schema:

```
tools = [{
  "type": "function",
  "function": {
    "name": "function_name",           # What this function does
    "description": "Brief description",
    "strict": true,                    # Enforce 100% adherence
    "parameters": {
      "type": "object",
      "properties": {
        "property_1": {
          "type": "string",            # string, number, boolean, etc.
          "description": "Instructions for property_1"
        },
      },
      "required": ["property_1"],
      "additionalProperties": false
    }
  }
}]
```

Key Schema Components

Core Elements

- `name`: Function name
- `description`: What it does
- `properties`: Output fields
- `required`: Mandatory fields

Field Properties

- `type`: Data type (string, number, boolean, array, object)
- `description`: Instructions for this field
- `enum`: Restrict to specific values

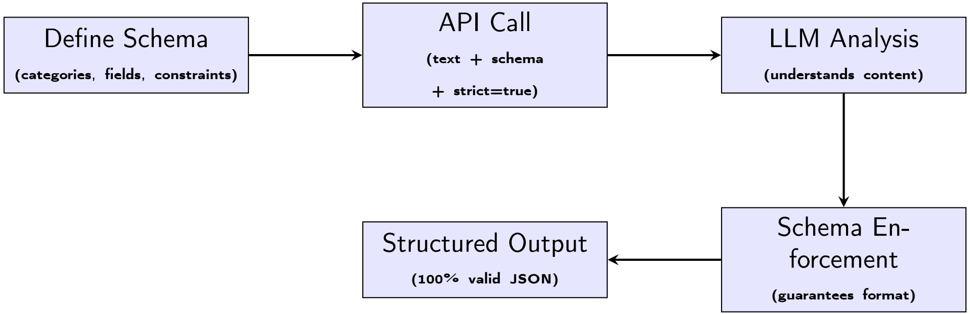
Available Types

- `string`: Text, dates, names
- `number`: Continuous values
- `integer`: Discrete counts
- `boolean`: Yes/no flags
- `array`: Lists of items
- `object`: Nested structures

Critical Setting

- `strict`: Enforce adherence
- `additionalProperties: false`
  Prevents unexpected fields

How It Works: API Flow



Separation of Concerns

- **Schema defines** what information to return and in what format
- **Model determines** the actual content based on text analysis
- **Strict mode guarantees** format compliance

Task 1: Text Classification

Classification Schema Template

Key element: `enum` constraint restricts model to predefined categories

```
tools = [{
  "type": "function",
  "function": {
    "name": "classify_<task>",
    "description": "Classify text according to <criterion>",
    "strict": true,
    "parameters": {
      "type": "object",
      "properties": {
        "classification": {
          "type": "string",
          "enum": ["c_1", "c_2", "c_3", ..., "c_k"],
          "description": "Detailed classification instructions"
        },
        "confidence": {"type": "number", "minimum": 0, "maximum": 1},
        "reasoning": {"type": "string"}
      },
      "required": ["classification", "confidence", "reasoning"],
      "additionalProperties": false
    }
  }
}]
```

Example: Classify FOMC statements by monetary policy stance

```
tools = [{
  "type": "function",
  "function": {
    "name": "classify_monetary_policy_stance",
    "description": "Classify central bank communication",
    "strict": true,
    "parameters": {
      "type": "object",
      "properties": {
        "policy_stance": {
          "type": "string",
          "enum": ["hawkish", "dovish", "neutral"],
          "description": """Classify as:
            - 'hawkish': preference for raising rates, inflation concerns
            - 'dovish': preference for lowering rates, growth concerns
            - 'neutral': balanced view or no clear direction"""
        },
        "confidence": {"type": "number", "minimum": 0, "maximum": 1},
        "reasoning": {"type": "string"}
      },
      "required": ["policy_stance", "confidence", "reasoning"],
      "additionalProperties": false
    }
  }
}]
```

Implementation

```
text = """Given the persistent upward pressure on prices
and the need to ensure inflation expectations remain well-anchored,
the Committee judges that a further tightening of monetary policy
is warranted to bring inflation back to our 2% target."""

response = client.chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "system", "content": "Expert in monetary policy."},
        {"role": "user", "content": f"Classify:  {text}"}
    ],
    tools=tools,
    tool_choice={"type": "function",
                 "function": {"name": "classify_monetary_policy_stance"}},
    temperature=0.0
)

result = json.loads(response.choices[0].message.tool_calls[0]
                   .function.arguments)
# Result: {"policy_stance": "hawkish", "confidence": 0.95, ...}
```

Guaranteed valid output every time!

More Classification Examples

Financial Sentiment

```
"sentiment": {
  "type": "string",
  "enum": ["positive",
           "negative",
           "neutral"]
},
"intensity": {
  "type": "integer",
  "enum": [1, 2, 3]
}
```

Applications: earnings calls, news articles, analyst reports

ESG Classification

```
"primary_dimension": {
  "type": "string",
  "enum": ["environmental",
           "social",
           "governance",
           "none"]
},
"commitment_level": {
  "enum": ["aspirational",
           "committed",
           "implemented",
           "reported"]
}
```

Applications: corporate disclosures, annual reports

Task 2: Information Retrieval

Information Retrieval Schema Template

```
tools = [{
  "type": "function",
  "function": {
    "name": "extract_data",
    "description": "Extract specific information from text",
    "strict": true,
    "parameters": {
      "type": "object",
      "properties": {
        "entity_name": {"type": "string"},
        "date": {"type": "string",
                 "description": "YYYY-MM-DD format.  Use null if not found."},
        "amount": {"type": "number"},
        "confidence": {"type": "number", "minimum": 0, "maximum": 1}
      },
      "required": ["entity_name", "date", "amount"]
      "additionalProperties": false
    }
  }
}]
```

Example: Extract forecasts and policy decisions from FOMC minutes

```
tools = [{
  "type": "function",
  "function": {
    "name": "extract_economic_indicators",
    "description": "Extract economic forecasts and policy decisions",
    "strict": true,
    "parameters": {
      "type": "object",
      "properties": {
        "gdp_forecast": {"type": "number"},
        "inflation_forecast": {"type": "number"},
        "unemployment_forecast": {"type": "number"},
        "policy_rate_change": {
          "type": "number",
          "description": "Rate change in basis points (e.g., 25 for 0.25%)"
        },
        "forecast_horizon": {"type": "string"},
        "confidence": {"type": "number", "minimum": 0, "maximum": 1}
      },
      "required": ["policy_rate_change", "forecast_horizon", "confidence"],
      "additionalProperties": false
    }
  }
}]
```

Implementation

```
text = """The Committee decided to raise the target range for
the federal funds rate by 25 basis points to 5.25-5.50 percent.
Participants project GDP growth of 2.1% in 2024 and inflation
to decline to 2.6% by year-end."""

response = client.chat.completions.create(
    model="gpt-4o",
    messages=[
        {"role": "system", "content": "Extract economic indicators."},
        {"role": "user", "content": text}
    ],
    tools=tools,
    tool_choice={"type": "function",
                 "function": {"name": "extract_economic_indicators"}},
    temperature=0.0
)

result = json.loads(response.choices[0].message.tool_calls[0]
                    .function.arguments)
# Result: {"gdp_forecast": 2.1, "inflation_forecast": 2.6,
#           "policy_rate_change": 25, "forecast_horizon": "2024", ...}
```

More Information Retrieval Examples

Loan Contract Terms

```
"loan_amount": {
  "type": "number"
},
"interest_rate": {
  "type": "number"
},
"maturity_date": {
  "type": "string"
},
"collateral_type": {
  "enum": ["real_estate",
           "equipment",
           "unsecured", ...]
},
"covenants_present": {
  "type": "boolean"
}
```

Corporate Events

```
"event_type": {
  "enum": ["merger",
           "acquisition",
           "earnings",
           "dividend", ...]
},
"event_date": {
  "type": "string"
},
"companies_involved": {
  "type": "array",
  "items": {"type": "string"}
},
"transaction_value": {
  "type": "number"
}
```

Applications: Credit markets, event studies, financial databases

Task 3: Structured Generation

Structured Generation Schema Template

```
tools = [{
  "type": "function",
  "function": {
    "name": "generate_output",
    "description": "Generate structured content",
    "strict": true,
    "parameters": {
      "type": "object",
      "properties": {
        "title": {"type": "string", "description": "Concise title (max 15 words)"},
        "summary": {"type": "string", "description": "Brief summary (max 200 words)"},
        "key_points": {
          "type": "array",
          "items": {"type": "string"},
          "description": "List of 3-5 key points",
          "minItems": 3,
          "maxItems": 5
        },
      },
      "required": ["title", "summary", "key_points"]
    }
  }
}]
```

Example: Generate standardized policy briefs from legislation

```
tools = [{
  "type": "function",
  "function": {
    "name": "generate_policy_brief",
    "description": "Generate structured policy brief",
    "strict": true,
    "parameters": {
      "type": "object",
      "properties": {
        "title": {"type": "string", "description": "max 15 words"},
        "executive_summary": {"type": "string", "description": "max 150 words"},
        "affected_sectors": {"type": "array", "items": {"type": "string"}},
        "fiscal_impact": {
          "type": "object",
          "properties": {
            "estimated_cost": {"type": "number"},
            "revenue_impact": {"type": "number"},
            "time_horizon": {"type": "string"}
          },
          "required": ["time_horizon"]
        },
        "key_provisions": {
          "type": "array",
          "items": {"type": "string"},
          "minItems": 3,
          "maxItems": 7
        }
      },
      "required": ["title", "executive_summary", "affected_sectors", "key_provisions"]
    }
  }
}]
```

Key Features for Generation

Length Constraints

Specify in descriptions: "max 200 words", "3-5 bullet points"

Array Constraints

Use `minItems` and `maxItems` to enforce list lengths

- Ensures consistent structure across generated outputs
- Prevents overly brief or verbose lists

Nested Objects

Use `type: "object"` for complex hierarchical structures

- Example: `fiscal_impact` contains multiple related fields
- Maintains logical grouping of related information

Style Guidance

More Generation Examples

Company Profiles

```
"financial_highlights": {
  "type": "object",
  "properties": {
    "revenue": {"type": "number"},
    "net_income": {"type": "number"},
    "year": {"type": "integer"}
  }
},
"key_risks": {
  "type": "array",
  "items": {"type": "string"},
  "minItems": 3,
  "maxItems": 5
}
```

Application: Cross-firm comparison

Literature Summaries

```
"main_research_question": {
  "type": "string"
},
"key_findings": {
  "type": "array",
  "items": {"type": "string"},
  "minItems": 3,
  "maxItems": 5
},
"limitations": {
  "type": "array",
  "items": {"type": "string"},
  "minItems": 2,
  "maxItems": 4
}
```

Application: Meta-analysis prep

Applications: Research automation, systematic reviews, database creation

Implementation & Best Practices

Development Workflow

1. **Pilot Testing**
   - Test schema on 30-50 labeled examples
   - Validate accuracy and refine descriptions
   - Use `reasoning` field to understand decisions
2. **Iterative Refinement**
   - Adjust category descriptions based on errors
   - Clarify ambiguous cases in field descriptions
   - Add examples in descriptions when helpful
3. **Human Validation**
   - Validate random sample (10-20%) against human coding
   - Assess reliability and identify systematic errors
   - Document agreement metrics in paper
4. **Full Deployment**
   - Process complete dataset
   - Monitor confidence scores
   - Flag low-confidence outputs for review

Model Selection and Configuration

Model Comparison

- Test multiple models on validation set
- Consider accuracy-cost tradeoffs
- Options: GPT-4o, Claude Sonnet, Gemini, Llama, etc.
- Use smaller models (GPT-4o-mini) for testing

Temperature Settings

- `temperature=0.0` for classification & extraction (max consistency)
- `temperature=0.3-0.5` for generation (some variation OK)

Operational Considerations

- **Batch Processing:** Use batch APIs for cost savings
- **Rate Limiting:** Implement proper throttling
- **Error Handling:** Retry logic for API failures
- **Caching:** Cache common prompts when supported

Cost Management

Estimate total costs before full implementation using token counts

Reproducibility Checklist

Document in Your Paper

- ✓ Exact model version (e.g., `gpt-4o-2024-08-06`)
- ✓ Complete schema with all properties and descriptions
- ✓ System prompts and user message templates
- ✓ All API parameters (`temperature`, `tool_choice`, etc.)
- ✓ Fixed `seed` parameter when available

Replication Materials

- ✓ Store complete schemas in supplementary materials
- ✓ Include example API calls
- ✓ Document any schema modifications during research
- ✓ Version output data if schema changes
- ✓ Report validation metrics (human agreement, confidence distributions)

Cross-Provider Compatibility

Function calling is supported by all major LLM providers:

- OpenAI (GPT-4, GPT-4o)
- Anthropic Claude (Sonnet, Opus)
- Google Gemini
- xAI Grok
- Mistral AI
- Cohere
- GroqCloud (fast inference)
- Azure OpenAI
- Local models (via APIs)

Portability

The JSON schemas in this paper work across providers with minimal syntax adjustments

- Core schema structure is universal
- API call syntax varies slightly
- Consult provider documentation for specifics

Conclusion

Key Takeaways

1. **The Problem is Fundamental**
   - Prompting alone: 35-85% format adherence
   - This is unacceptable for scientific research
2. **The Solution: Function Calling Schemas**
   - Strict mode: 100% format adherence
   - Industry standard across all major providers
   - Deterministic formatting + analytical flexibility
3. **Universal Framework**
   - Same structure for classification, retrieval, generation
   - Easily combined for multi-task applications
   - Ready-to-use templates for common economic research tasks
4. **Reproducible Research**
   - Document model version, complete schema, parameters
   - Output format is deterministic
   - Fair model comparisons possible

Call to Action

Stop relying on prompting alone

Adopt function calling schemas

for reliable, reproducible, rigorous research

Thank You!

Questions?

Jesus Villota-Miranda
CEMFI
jesus.villota@cemfi.edu.es

Structured Data with LLMs Done Right
A Practical Guide to Text Classification, Information Retrieval and Generation