Prompt Patterns for Structured Data Extraction from Unstructured Text

Max Moundas
maximillian.r.moundas@vanderbilt.edu
Department of Computer Science
Vanderbilt University
Nashville, Tennessee, USA

Jules White
jules.white@vanderbilt.edu
Department of Computer Science
Vanderbilt University
Nashville, Tennessee, USA

Douglas C. Schmidt
dcschmidt@wm.edu
Department of Computer Science
William & Mary
Williamsburg, Virginia, USA

Abstract

Large language models (LLMs) show promise for extracting structured data from unstructured text by identifying patterns, keywords (including names, organizations, locations, and topics), and relations in text. However, the performance of LLMs on tasks like precision, clarity, replicability, and uniform interpretation of results depends heavily on how users express their prompts, which are instructions users give to LLMs. Understanding these performance measures is crucial to ensure consistent and reliable data extraction across various applications and users. To enhance these measures—and to make the extraction process more effective and repeatable for users—this paper introduces *structured data extraction prompt patterns*, which are reusable templates for prompting LLMs to extract desired data from unstructured text.

This paper provides three contributions to research on structured data extraction. First, we present a catalog of prompt patterns for common data extraction tasks, such as semantic data extraction. Second, we describe and evaluate methods for chaining prompt patterns into pattern compounds or pattern sequences to extract complex nested data, thereby enabling more effective use of LLMs for text mining and knowledge base construction from unstructured corpora. Finally, we present a structured approach to prompt engineering that supplies developers with cohesive and flexible templates, facilitating the creation of sophisticated data extraction workflows with more dependable results than *ad hoc* prompting.

CCS Concepts

• **Information systems** → **Search interfaces**; • **Computing methodologies** → **Information extraction**.

Keywords

LLMs, Prompt patterns, Prompt engineering, Data Extraction

ACM Reference Format:
Max Moundas, Jules White, and Douglas C. Schmidt. 2024. Prompt Patterns for Structured Data Extraction from Unstructured Text. In *Proceedings of 31st Conference on Pattern Languages of Programs, People, and Practices (PLoP 2024).* ACM, New York, NY, USA, 15 pages.

PLoP 2024, October 13–16, 2024, Skamania Lodge, Columbia River Gorge, Washington, USA.

Hillside ISBN 978-1-941652-20-6

Challenges of extracting structured data from unstructured text. A longstanding challenge in natural language processing is extracting structured data from unstructured text [13], including extracting product attributes (*e.g.*, price and brand) from product reviews or extracting patient data (*e.g.*, age and medical history) from clinical notes. Examples of this type of extraction include the following:

- *Pattern recognition*:
  - Text: "Car 1 is yellow. car 2 is blue, car 3 is red, car 4 is green."
  - Structured Output: "Car: 1, Color: Yellow; Car: 2, Color: Blue; Car: 3, Color: Red; Car: 4, Color: Green"
- *Keyword extraction*:
  - Text: "Black holes are regions in space with immense gravitational pull."
  - Structured Output: "Keywords: black holes, regions, space, gravitational pull"
- *Relation extraction*:
  - Text: "Paris is the capital of France."
  - Structured Output: "Relationship: Paris - is the capital of - France"

This extraction process is hard because unstructured text lacks explicit demarcations of fields, so the desired data must be identified and extracted using natural language understanding. Moreover, factors like ambiguity, complex linguistic arrangement, and diversity in textual representations make data extraction hard [11].

Traditional rule-based and statistical methods for data extraction rely on hand-engineered features and labeled training data. Recently, large language models (LLMs), including—but certainly not limited to—GPT-4, have shown promise for "few-shot" data extraction [14]. This few-shot approach enables LLMs to perform data extraction after seeing only a few demonstrations, unlike traditional supervised learning, which requires large labeled datasets.

For example, few-shot learning may involve just a few annotated sentences showing how to extract product details from text. LLMs can leverage their pre-trained knowledge and generalize based on these few examples, without needing extensive additional training on extraction tasks. Few-shot data extraction thus enables replicating extraction capabilities across topics and domains without costly data labeling efforts.

Even with few-shot learning, however, the performance of LLMs on data extraction tasks depends heavily on how users express prompts that provide LLMs with examples and instructions [10]. Until recently, this dependence required users to have a deep knowledge of how to interact with LLMs effectively. Unfortunately, this requirement poses a daunting impediment to broad user adoption and effective application of LLMs for data extraction.

Solution approach → Prompt patterns for structured data extraction. To simplify user learning curves, this paper presents a catalog of prompt patterns for structured data extraction. In general, *prompt patterns* [15] codify best practices for phrasing prompts to maximize extraction accuracy. They also provide knowledge transfer mechanisms that users can reference to problem-solve with LLMs more effectively via *prompt engineering* [12], which is the discipline of designing and optimizing user inputs (*i.e.*, prompts) to elicit desired outputs from LLMs.

The prompt patterns presented in this paper are reusable constructs that provide building blocks users can combine to prompt LLMs to extract desired data (including complex nested data) from unstructured text. In the context of data extraction, prompt engineering involves crafting prompts that guide LLMs to identify and extract specific data from unstructured text accurately. This approach leverages LLMs' natural language understanding capabilities to perform extraction tasks without extensive additional training.

This paper makes the following contributions to work on prompt engineering for structured data extraction from unstructured text:

(1) We present a catalog of prompt patterns that serve as reusable building blocks for data extraction pipelines, including patterns for extracting semantics, querying over input text, and managing context.
(2) We describe a systematic framework and method for constructing structured data extraction prompts by applying these reusable prompt patterns to provide the foundation for our principled prompt engineering approach.
(3) We demonstrate how these foundational patterns can be chained to form pattern compounds or pattern sequences [1] and then used to extract nested data structures from unstructured corpora, thereby enabling replicatable and transparent text extraction.
(4) We present many examples that showcase the flexibility of our prompt patterns when customizing extraction for diverse use cases while maintaining uniformity in execution.

Table 1 summarizes the five structured data extraction prompt patterns covered in this paper.

Table 1: Summary of Prompt Patterns for Structured Data Extraction

Pattern Category	Prompt Pattern
Data Extraction	*Semantic Extractor* (p. 3), *Adaptive Attribute Extractor* (p. 5), *Pattern Matcher* (p. 7)
Instances Query on Input	*Specify Constraints* (p. 8)
Input Specification	*Keyword Trigger Extractor* (p. 9)

Paper organization. The remainder of this paper is organized as follows: Section 1 gives an overview of structured data extraction using prompt engineering; Section 2 describes three prompt patterns for defining extraction formats and transforming input text into target structures; Section 3 discusses a prompt pattern for querying input text to filter results; Section 4 explores a prompt pattern for specifying input sources; Section 5 examines how chaining prompt patterns enhances the extraction process for complex scenarios; Section 6 reviews the evolution of structured data extraction methods, including traditional rule-based and statistical approaches, as well as emerging few-shot learning techniques and prompt engineering strategies; Section 7 presents concluding remarks that summarize key lessons learned from the prompt patterns and extraction pipelines we document in this paper; Appendix A discusses the applicability of our prompt patterns to multiple LLMs; and Appendix B provides an overview of our prompt pattern form, which should be familiar to readers acquainted with classic software pattern form [1, 5].

1 Overview of Structured Data Extraction Using Prompt Engineering

A systematic method for effective prompting is needed within the discipline of prompt engineering to extract structured data from unstructured text. To provide this capability, we devised a template tailored specifically for structured data extraction prompts. This template enhances the precision and clarity of extraction operations, while also ensuring replicability and uniform interpretation, regardless of the LLM or context of where prompts are applied. In addition, this template helps prompt engineers achieve consistency in executing and interpreting extraction tasks, thereby minimizing discrepancies.

This section describes how we leveraged our template to codify a catalog of patterns, derived from our rigorous research and practical applications conducted over the past two years. These patterns originated from our experience and serve as modular building blocks for assembling complex data extraction pipelines. Furnishing users with these customizable prompt patterns enables sophisticated extraction capabilities, thereby helping to ensure robust and accurate data retrieval across diverse scenarios, such as the following:

- Extracting contact data (*e.g.*, emails or phone numbers) from corporate websites,
- Gathering financial figures from annual reports,
- Parsing patient data from medical records,
- Retrieving citations from research papers,
- Compiling real estate listings from property websites, and
- Harvesting usage statistics from application logs.

Our prompt pattern template for data extraction leverages the unique properties and capabilities of LLMs, such as their ability to understand context, handle natural language instructions, and generate structured outputs. Our template provides a standardized framework that emphasizes flexibility so users can adapt it to specific use cases while maintaining its core elements. This template is expressed via the following structured elements:

```
Extract

GENERATION_CONSTRAINTS

in the format

EXTRACTION_PATTERN

(where INSTANCES_QUERY_ON_INPUT)
```

from

INPUT_SPECIFICATION

Where:

- GENERATION_CONSTRAINTS specifies constraints or limits on the data to extract. For example, setting a limit to the number of records retrieved could be represented as "UP TO Z instances of X."
- EXTRACTION_PATTERN defines the format or pattern in which data should be retrieved to ensure data elements are structured in a manner that aligns with the user objectives. Example objectives include normalizing raw text data into a consistent format for usage with predefined fields, converting records into structured entries loadable into databases, and parsing content into machine-readable formats to facilitate training machine learning models.
- INSTANCES_QUERY_ON_INPUT is akin to the 'WHERE' clause in the *Structured Query Language* (SQL), allowing the filtering of input based on certain conditions. This template element enables an LLM to focus on specific parts of the input, leveraging its contextual understanding capabilities without explicitly using the keyword "where."
- INPUT_SPECIFICATION specifies the source of the data or where the extraction should be performed. For example, it can instruct the LLM to only extract data directly following a particular keyword or to only extract data from the last ten lines of the full input text.

Our template provides a structured approach that aligns with LLM strengths in natural language understanding and generation. By decomposing the extraction task into these template elements, LLMs can more effectively parse and execute complex extraction tasks, while maintaining flexibility for various use cases. This decomposition enhances LLM performance in the following ways:

- *Improved parsing* – Each component targets a specific aspect of the extraction task, allowing an LLM to focus on one element at a time, thereby reducing cognitive load and potential confusion.
- *Enhanced execution* – The structured format guides an LLM through a step-by-step process, thereby ensuring all necessary data are considered and extracted in a logical order.
- *Increased accuracy* – By clearly defining the extraction parameters, the likelihood of irrelevant or incorrect data being included in an LLM's output is reduced.
- *Adaptability* – The modular nature of the template enables easy modification of individual components to suit different extraction scenarios without altering the overall structure.

For example, in the context of the healthcare domain, the same template can be adapted to extract patient diagnoses, medication dosages, and/or treatment outcomes by simply adjusting the EXTRACTION_PATTERN and INSTANCES_QUERY_ON_INPUT components. Similarly, in the financial domain, the template can be modified to extract quarterly earnings, stock prices, or market trends by fine-tuning these elements to specific data requirements. Likewise, in the environmental science domain, researchers can adapt this template to extract climate data from diverse sources by adjusting the EXTRACTION_PATTERN to focus on temperature readings, precipitation levels, or air quality indices, and modifying the INSTANCES_QUERY_ON_INPUT to filter for specific geographic regions or time periods. These adaptations demonstrate the template's versatility in facilitating data-driven research across various scientific disciplines and industry domains.

The remainder of this paper explores each of the four elements capitalized above. We explain the nuances of each element and present prompt patterns that can be employed effectively in practical scenarios. This detailed exploration provides a thorough understanding of structured data extraction, paving the way for more efficient and targeted data retrieval methodologies in future work, as discussed in Section 7.

2 Data Extraction Patterns

This section presents the following three prompt patterns that transform input text into desired target data structures during extraction: *Semantic Extractor*, the *Adaptive Attribute Extractor*, and the *Pattern Matcher*. These patterns extract structured data from unstructured text through a variety of techniques—from flexible semantic parsing to strict pattern matching—providing users with multiple modular building blocks to meet their diverse extraction needs. Note that while the examples throughout this paper are simplified to omit JSON formatting for clarity, in practice these patterns typically generate properly formatted JSON output with appropriate brackets, quotation marks, and escape characters.

2.1 The Semantic Extractor Pattern

2.1.1 Summary. The *Semantic Extractor* pattern extracts specific structured data from unstructured text based on a semantic description rooted in the target data structure definition. Unlike traditional extraction methods [3, 4] that rely on explicit rules or patterns, this pattern uses a descriptive framework to identify and capture desired data implicit in the target structured specification. It thus enables better understanding and interpretation of the meaning or context of the desired data points, without the need for labor-intensive rule definition or manual intervention.

2.1.2 Problem. Traditional data extraction methods present several challenges that make efficient data extraction hard:

- Manual definition and maintenance of explicit rules and regular expressions can be tedious, error-prone, and time-consuming,
- Traditional pattern matching approaches are often brittle and fail to handle variations in input text,
- Existing approaches often miss contextual nuances in the data, and
- Complex data structures require increasingly complicated rule sets

2.1.3 Solution. The *Semantic Extractor* pattern provides an approach that leverages natural language descriptions within a structured format to guide the extraction process. This structured format is designed as follows:

- Use semantic descriptions embedded within the target data structure,

- Leverage LLM capabilities to understand and process natural language variations,
- Maintain a consistent structure while allowing flexible descriptions,
- Enable intuitive specification of extraction requirements without explicit rules, and
- Adapt to new extraction targets without significant structural changes

2.1.4 Structure. The *Semantic Extractor* pattern consists of the following three components working together:

- Extraction Command
  - Clear indicator of the task ("Extract:")
  - Sets up the extraction context
- Target Structure Definition
  - JSON-like structure defining the output format
  - Field names that identify the data points
  - Semantic descriptions for each field explaining what to extract
  - Optional nested structures for complex data
- Input Text Section
  - Clear separator ("from the following text:")
  - The unstructured text to process
  - Clear boundary between structure and input

Here's a general example of this type of structure:

```
Extract:

field1: semantic description of what to extract,
field2: semantic description of what to extract,

from the following text:

[Input text]
```

2.1.5 Implementation. More detailed examples of implementing the *Semantic Extractor* pattern are provided below.

Input:
Extract:

name: the name of the car
identifier: the VIN number

from the following text:

In today's review, we'll be diving deep into the latest sedan model, the 'EcoSprint'. This vehicle, with a VIN number of '12345ABCDE', has been making waves in the automotive industry with its innovative features.

Output:
name: EcoSprint
identifier: 12345ABCDE

Input:
Extract:

name: the name of the musician
instrument: the primary instrument they play

from the following text:

Ludwig van Beethoven, a maestro primarily known for his prowess with the piano, whose compositions have stood the test of time.

Output:
name: Ludwig van Beethoven
instrument: piano

Input:
Extract:

brand: brand of the phone
feature: its standout feature

from the following text:

The latest 'TechStar' phone boasts a revolutionary '3D touch' feature that has tech enthusiasts buzzing.

Output:
brand: TechStar
feature: 3D touch

2.1.6 Consequences. There are several benefits, limitations, and performance limitations of applying the *Semantic Extractor* pattern, as described below.

- Benefits:
  - Reduces need for explicit rule definitions
  - Handles input variations naturally via LLM capabilities
  - Easily adaptable to new extraction targets
  - More intuitive for users to define extraction requirements
  - Leverages natural language understanding for better accuracy
- Limitations:
  - Effectiveness depends on LLM quality and training
  - Requires clear and unambiguous semantic descriptors
  - May not be as precise as rule-based systems for highly structured data
  - Needs regular testing and refinement of prompts
  - Performance varies based on LLM's understanding of semantic descriptions
- Performance implications:
  - Can handle variations in input text without additional rules
  - Adapts to new fields without structural changes
  - May require balancing between description detail and constraints

  - Processing time depends on LLM capabilities

2.1.7 Related Patterns.

- *Adaptive Attribute Extractor* pattern, which is more suitable when precision and handling complex data structures are paramount.
- *Pattern Matcher* pattern, which is better for cases where strict templates and conditions are needed.

The *Semantic Extractor* pattern can also be combined with other patterns to form pattern compounds or pattern sequences that balance flexibility and precision, as demonstrated in the Technical Specification Extraction Chain pattern sequence (Section 5.1).

2.1.8 Example Applications. Example applications of the *Semantic Extractor* pattern include the following:

- Document Processing:
  - Extracting contact information from business documents
  - Processing customer feedback forms
  - Analyzing product specifications
- Content Analysis:
  - Extracting key details from news articles
  - Processing social media posts for sentiment and key information
  - Analyzing product reviews for specific attributes
- Data Migration:
  - Converting unstructured legacy data to structured formats
  - Extracting specific fields from varied document types
  - Standardizing information across different sources

2.2 The Adaptive Attribute Extractor Pattern

2.2.1 Summary. The *Adaptive Attribute Extractor* pattern extracts and refines structured data attributes dynamically from unstructured text, focusing on flexibility and constraint specification. This pattern adapts the data extraction process to the specific attributes present within the data instances, thereby ensuring accurate and comprehensive data capture while allowing for customizable constraints.

2.2.2 Problem. When extracting data from unstructured text, entities are often described with varying attributes, leading to several challenges:

- Rigid, predefined data extraction methods can miss unique features not captured by fixed templates
- Alternative methods might achieve uniformity but at the expense of valuable data
- Natural variation and richness of attributes in actual instances can be lost
- Need for balance between flexibility and consistency in extraction
- Risk of incomplete data representation when using fixed extraction approaches

2.2.3 Solution. The *Adaptive Attribute Extractor* pattern provides a flexible extraction framework that:

- Begins with a broad extraction structure that adjusts based on specific attributes found in the text
- Allows specification of constraints to guide the extraction process
- Facilitates nuanced data extraction that can handle diversity while ensuring complete attribute representation
- Combines dynamic discovery with structured output requirements

2.2.4 Structure. The *Adaptive Attribute Extractor* pattern consists of the following four components working together:

- Target Format Definition
  - Defines basic structure with required fields
  - Includes semantic descriptions for clarity
- Adaptive Attribute Section
  - Uses ellipses (...) or alternative syntax to denote open-ended attributes
  - Can employ various syntactic approaches:
    - Specific delimiters (###, $$$)
    - Structured prompt language
    - Natural language instructions
- Constraint Specification
  - Defines rules for attribute consistency
  - Sets requirements for attribute selection
  - Controls variation between instances
- Discovery Mechanism
  - Optional initial phase for attribute identification
  - Guides subsequent targeted extraction

Here's a generalized overview of this type of structure:

```
Extract:

name: semantic description of the primary identifier,
...attributes related to [entity type] [constraint specification]...

from the following text:

[Input text containing entity descriptions with
various attributes]
```

2.2.5 Implementation. More detailed examples of implementing the *Adaptive Attribute Extractor* pattern are provided below.

Input:
Extract:

name: the name of the gadget,
...attributes related to the gadget [each instance can have different attributes]...

from the following text:

The latest gadgets introduced this year include the SmartHome 3000, a revolutionary device that enables full control of your home environment through a single app. This gadget features voice recognition, energy monitoring, and compatibility with various smart devices. Another exciting release is the EcoCharge Pro, a portable charger made from sustainable materials, boasting a capacity of 20,000 mAh

and fast-charging capabilities. Additionally, tech enthusiasts are raving about the PixelVision Camera, which offers 50 MP resolution, advanced night mode, and built-in Wi-Fi for seamless photo sharing. Each of these gadgets brings something unique to the table, catering to different needs and preferences of users. Lastly, the Wearable Health Tracker has gained popularity for its heart rate monitoring, sleep analysis, and integration with fitness apps, making it a must-have for health-conscious individuals.

Output:
SmartHome 3000
- Voice recognition
- Energy monitoring
- Compatibility with various smart devices

EcoCharge Pro
- Portable charger
- Made from sustainable materials
- Capacity of 20,000 mAh
- Fast-charging capabilities

PixelVision Camera
- 50 MP resolution
- Advanced night mode
- Built-in Wi-Fi for seamless photo sharing

Wearable Health Tracker
- Heart rate monitoring
- Sleep analysis
- Integration with fitness apps

Input:
Extract:

brand: brand of the beverage,
...attributes related to the beverage [all instances should have at least one common attribute]...

from the following text:

In the bustling heart of the city, Joe's Café has become famous for its exceptional beverages. Among their offerings is the renowned Cool Breeze, a refreshing sparkling water from Breeze Brands, available in classic 500ml and family-friendly 1.5-liter sizes. This beverage is infused with natural mint and lemon flavors, making it a perfect thirst-quencher. Another highlight is the Rich Roast, a bold coffee crafted by Dark Brew Co. It comes in sizes of 250ml for a quick pick-me-up and a larger 750ml option for those who like to savor their coffee throughout the day. Additionally, they serve the Sweet Harmony tea by Tranquil Teas, which is offered in delightful 300ml cups and generous 1-liter pots, featuring a blend of chamomile and honey for a soothing experience. Each beverage at Joe's Café is carefully selected to ensure that customers can enjoy their drink in the size that suits them best.

Output:
- Brand: Breeze Brands
- Beverage Name: Cool Breeze
- Attributes:
- Type: Sparkling Water
- Sizes: 500ml, 1.5-liter
- Flavors: Natural Mint, Lemon

- Brand: Dark Brew Co.
- Beverage Name: Rich Roast
- Attributes:
- Type: Coffee
- Sizes: 250ml, 750ml

- Brand: Tranquil Teas
- Attributes:
- Type: Tea
- Sizes: 300ml, 1-liter
- Flavors: Chamomile, Honey

2.2.6 Consequences. There are several benefits and limitations of applying the *Adaptive Attribute Extractor* pattern, as described below.

- Benefits:
  - Adaptability to handle unexpected variations in text inputs
  - Ability to capture rich, diverse datasets
  - Flexibility in attribute extraction while maintaining constraints
  - No need for constant prompt adjustments
  - Reduced reliance on predefined rules
- Limitations:
  - Challenges in maintaining consistency across extracted data
  - Risk of overlooking important attributes
  - Requires well-crafted constraints
  - Balance required between flexibility and guidance

For optimal implementation of this pattern, we recommend a two-phase extraction process: first, conduct an initial discovery phase to analyze text and identify relevant attributes, then perform a specific extraction phase using these discovered attributes for targeted extraction. This approach helps ensure comprehensive and accurate attribute extraction while maintaining the pattern's adaptability.

2.2.7 Related Patterns.

- *Semantic Extractor* pattern is similar in providing structure for target name but with less flexibility.

2.2.8 Example Applications. Example applications of the *Adaptive Attribute Extractor* pattern include the following:

- Document Analysis

  - Processing technical documents with varying levels of detail
  - Extracting relevant attributes while maintaining document-specific constraints
- Content Aggregation
  - Combining information from multiple sources
  - Maintaining consistency while accommodating source-specific attributes

2.3 The Pattern Matcher Pattern

2.3.1 Summary. The *Pattern Matcher* pattern precisely extracts structured data from unstructured text by searching for instances that match a predefined template exactly. It operates similar to a regular expression within an LLM context, prioritizing precision over flexibility in data extraction tasks.

2.3.2 Problem. Many production datasets contain structured data that follows fixed formats (phone numbers, postal codes, emails, etc.). While traditional tools like regex can handle these tasks, integrating pattern matching within LLM workflows presents unique challenges:

- Need for precise extraction without interpretation or variation
- Maintaining consistency within larger LLM-based systems
- Balancing pattern rigidity with practical data variations
- Determining when LLM-based pattern matching is preferable to traditional tools
- Managing extraction across multiple pattern instances

2.3.3 Solution. The *Pattern Matcher* pattern requires:

- Definition of an exact search pattern matching the target data structure
- Clear specification of delimiters, boundaries, and formatting requirements
- Configuration for handling multiple pattern instances
- Strict adherence to pattern definition without contextual interpretation
- Implementation within appropriate LLM workflows where traditional tools aren't optimal

2.3.4 Structure. The *Pattern Matcher* pattern consists of the following two components working together:

- Pattern Definition
  - Fixed text elements
  - Variable placeholders with format specifications
  - Delimiter and boundary markers
- Search Instructions
  - Exact matching requirements
  - Multiple instance handling
  - Error handling for non-matching cases

Here's a general example of this type of structure:

> Extract all instances of the pattern:
>
> `[pattern definition]`
>
> from the following text:
>
> `[Input text potentially containing instances of [pattern definition] in the specified format]`

2.3.5 Implementation. More detailed examples of implementing the *Pattern Matcher* pattern are provided below.

> **Input:**
> Extract all instances of the pattern:
>
> Order ID: [8 digit number]
>
> from the following text:
>
> Your order from Elite Hardware was successfully placed. Make note of the Order ID: 12345678 for future reference.
>
> **Output:**
> Order ID: 12345678

> **Input:**
> Extract all instances of the phone number pattern:
>
> `[3 digits]-[3 digits]-[4 digits]`
>
> from the following text:
>
> Both John's and Jane's phone numbers, 123-456-7890 and 987-654-3210, were written down.
>
> **Output:**
> 123-456-7890
> 987-654-3210

> **Input:**
> Extract all instances of the email address pattern:
>
> `[a-z0-9._%-+]@[a-z0-9].[a-z]`
>
> from the following text:
>
> Email me your questions at 'yourprofessor@myuniversity.edu'
>
> **Output:**
> yourprofessor@myuniversity.edu

2.3.6 Consequences. There are several benefits and limitations of applying the *Pattern Matcher* pattern, as described below.

- Benefits:
  - High precision in data extraction
  - Reliable results when patterns are well-defined

  - Seamless integration with LLM workflows
  - Clear validation criteria
- Limitations:
  - Brittle against pattern variations
  - May miss valid data in unexpected formats
  - Requires thorough understanding of data structure
  - May be less efficient than traditional tools
  - Limited flexibility compared to context-based extraction

2.3.7 Related Patterns.

- *Adaptive Attribute Extraction*, which provides a more flexible alternative for variable formats.
- *Context Conveyor*, which manages broader extraction context.
- *Specify Constraints*, which defines extraction boundaries and rules.

2.3.8 Example Applications. Example applications of the *Pattern Matcher* pattern include the following:

- Financial Document Processing
  - Extract standardized account numbers
  - Identify transaction IDs
  - Process routing numbers
- Healthcare Records
  - Extract patient IDs
  - Process insurance claim numbers
  - Identify medical record numbers
- Legal Document Analysis
  - Extract case numbers
  - Process document identifiers
  - Identify standardized legal citations

3 Instances Query on Input Pattern

This section presents the *Specify Constraints* prompt pattern that was first introduced in previous research [15] under the name *Context Conveyor*.[1] While its broader usage includes managing conversational flow and topic relevance, this paper emphasizes applying *Specify Constraints* specifically to structured data extraction from unstructured sources.

3.1 The Specify Constraints Pattern

3.1.1 Summary. The *Specify Constraints* pattern enables precise control over LLM data extraction by allowing users to explicitly define constraints that focus the model's attention on specific aspects of the input text. This pattern helps achieve more accurate and relevant data extraction by clearly delimiting the scope of consideration.

3.1.2 Problem. When extracting structured data from unstructured text, LLMs often face challenges in determining which parts of the text are relevant to the extraction task. They may:

- Include extraneous or irrelevant data from the input
- Draw on inappropriate parts of a conversation or document
- Regress to previously addressed topics
- Struggle with maintaining focus on specific aspects of the data
- Process mixed-subject contexts inefficiently

These challenges are particularly acute when dealing with complex documents containing multiple topics or when specific subsets of information need to be extracted while intentionally excluding others.

3.1.3 Solution. The *Specify Constraints* pattern implements a two-part prompt structure:

- A constraints section that explicitly defines the boundaries and conditions for data extraction
- An extraction instruction that specifies the desired structured output format

Key principles associated with applying this pattern include:

- Explicitly state constraints using clear directives (*e.g.*, "Only consider X," "Exclude Y")
- Separate constraints from extraction instructions for clarity
- Define constraints before presenting the extraction task
- Use precise language to delineate scope boundaries

The *Specify Constraints* pattern can be customized by:

- Adjusting constraint specificity
- Combining multiple constraints
- Including exclusion criteria

3.1.4 Structure. The *Specify Constraints* pattern consists of the following three components working together:

- Constraints Declaration:
  - Clearly labeled "Constraints:" section
  - One or more specific constraint statements
  - Optional exclusion criteria
- Extraction Instructions:
  - Desired output format specification
  - Structure definition (typically in braces)
  - Field specifications
- Input Text:
  - The source text for extraction
  - Clearly separated from constraints and instructions

Here's a general example of this type of structure:

> Constraints: [constraint statement]
>
> Extract: [FIELD1], [FIELD2], ...
>
> from the following text:
>
> Input text

3.1.5 Implementation. More detailed examples of implementing the *Specify Constraints* pattern are provided below.

> **Input:**
> Constraints: Only consider renewable energy sources
>
> Extract: TYPE, COST MULTIPLE

[1] The description of *Specify Constraints* in this paper distinctively focuses on its roles in structured data extraction from unstructured text, whereas our earlier work on *Context Conveyor* discussed its broader applications in controlling conversational context within LLM dialogues.

from the following text:

Energy sources differ in costs due to infrastructure and maintenance considerations. Solar energy has upfront costs of around 2.0X compared to fossil fuels, due to solar panel installation. Wind energy comes in at 1.8X, accounting for turbine production and upkeep.

Output:
Solar, 2.0X
Wind, 1.8X

Input:
Constraints: Focus on fruit prices

Extract: FRUIT NAME, COST PER KG

from the following text:

Bananas cost $0.50 per kg, while toy cars are priced at $5 each. Mangoes are priced at $1.20 per kg.

Output:
Bananas, $0.50
Mangoes, $1.20

3.1.6 Consequences. There are several benefits, limitations, and performance implications of applying the *Specify Constraints* pattern, as described below.

- Benefits:
  - Improved extraction accuracy through focused attention
  - Reduced noise in extracted data
  - Better handling of mixed-context documents
  - More precise control over extraction scope
  - Enhanced output relevance
- Limitations:
  - Risk of excluding valuable peripheral data
  - Possible over-narrowing of context
  - Potential reset of previously established context
  - Need for careful balance in constraint specificity
- Performance implications:
  - Generally improved extraction precision
  - More consistent results across similar extractions
  - Better handling of complex documents

3.1.7 Related Patterns.

- *Context Conveyor* pattern, which is useful for managing broader conversational context.

3.1.8 Example Applications. Example applications of the *Specific Constraints* pattern include the following:

- Financial Document Processing
  - Extracting specific transaction types from financial statements
  - Filtering for date ranges or amount thresholds
  - Focusing on particular account categories
- Technical Documentation Analysis:
  - Extracting specifications for specific components
  - Focusing on particular technical parameters
  - Filtering by system or subsystem
- Medical Record Processing:
  - Extracting specific test results
  - Focusing on particular time periods
  - Filtering by condition or treatment type
- Legal Document Analysis:
  - Extracting specific clause types
  - Focusing on particular legal entities
  - Filtering by jurisdiction or date

4 Input Specification Pattern

This section presents the *Keyword Trigger Extractor* pattern, which provides techniques for targeting extraction on relevant sections of input text. This pattern provides users with the means to focus extraction on pertinent data within longer documents or text collections using built-in semantic triggers native to the domain.

4.1 The Keyword Trigger Extractor Pattern

4.1.1 Summary. The *Keyword Trigger Extractor* pattern identifies and extracts structured data that consistently appears after predefined cue words or phrases in unstructured text corpora. The pattern leverages semantic markers within text to efficiently locate and extract relevant data without scanning entire documents.

4.1.2 Problem. In many real-world scenarios, valuable structured data is embedded within unstructured text, often preceded by consistent keyword markers. The challenges include:

- Processing large volumes of text efficiently without reviewing entire documents
- Ensuring accurate extraction of specific data fields
- Handling variations in how data is presented after keywords
- Maintaining extraction accuracy across different document formats
- Managing the relationship between trigger words and their associated data
- Dealing with potential inconsistencies in keyword usage or formatting

The *Keyword Trigger Extractor* pattern addresses these challenges while providing reliable data extraction that can scale across large text corpora.

4.1.3 Solution. The *Keyword Trigger Extractor* pattern implements a targeted extraction approach via the following key components:

- Identification of specific keywords or phrases that reliably precede target data
- Definition of the expected structured data format to be extracted
- Specification of the relationship between the trigger and the target data
- Clear instructions for handling the extraction process

4.1.4 Structure. The *Keyword Trigger Extractor* pattern consists of the following four components working together:

- Trigger Definition
  - Specific keywords or phrases that flag relevant data
  - Clear specification of the trigger format
- Data Structure Template
  - Field definitions for extracted data
  - Expected format and relationships between fields
- Extraction Rules
  - Guidelines for identifying relevant data after triggers
  - Specifications for handling multiple fields or complex data structures
- Context Management
  - Rules for maintaining relationship between trigger and target data
  - Handling of surrounding text and potential interference

Here's a general example of this type of structure:

Following the keyword "TRIGGER", extract:

FIELD_NAME: FIELD_DESCRIPTION

from:

INPUT_TEXT

4.1.5 Implementation. More detailed examples of implementing the *Keyword Trigger Extractor* pattern are provided below.

Input:
Following the keyword "Born:"

Extract:

birthDate: date of birth,
birthPlace: place of birth

from the following text:

Born: 5th July 1980 in New York

Output:
birthDate: 5th July 1980,
birthPlace: New York

Input:
Following the keyword "Price:"

Extract:

cost: amount in USD

from the following text:

The item details are as follows: Price: $20

Output:
cost: 20

Input:
Following the keyword "Address:"

Extract:

street: street name,
city: city name,
postal: postal code

from the following text:

Address: 123 Maple St, Springfield, 12345

Output:
street: 123 Maple St
city: Springfield
postal: 12345

4.1.6 Consequences. There are several benefits and limitations of applying the *Keyword Trigger Extractor* pattern, as described below.

- Benefits:
  - Efficient identification of specific fields of interest
  - Enhanced precision through use of reliable semantic markers
  - Scalable approach for large text corpora
  - Built-in semantic guidance through domain-specific triggers
- Limitations:
  - Dependency on consistent keyword usage
  - Potential missed data if keywords vary
  - Limited to data appearing immediately after triggers
  - May overlook data introduced by alternative keywords
  - Reduced effectiveness with inconsistent text formatting

4.1.7 Related Patterns.

- *Pattern Matcher* pattern, which defines an alternative approach for highly structured documents.
- *Semantic Extractor* pattern, which defines an alternative for less structured text with varied keywords.

4.1.8 Example Applications. Example applications of the *Keyword Trigger Extractor* pattern include the following:

- Customer Record Processing
  - Extracting personal information from forms
  - Analyzing customer feedback
- Research Document Analysis
  - Extracting key findings from papers
  - Processing methodology sections
  - Analyzing results sections
- Legal Document Processing

  - Extracting dates and deadlines
  - Processing party information
  - Analyzing contract terms
- Medical Record Analysis
  - Extracting patient information
  - Processing diagnosis details
  - Analyzing treatment plans

5 Chaining Structured Data Extraction Patterns

While the five individual extraction patterns presented above offer powerful capabilities for structured data extraction, combining multiple patterns can yield even more precise and comprehensive results. This section explores the concept of chaining extraction patterns into *Pattern sequences*, which are ordered groups of patterns applied to create a particular architecture or design in response to a specific situation [1].

The pattern sequences presented below demonstrate how the synergistic use of multiple patterns can enhance the overall extraction process. These combinations build upon the strengths of individual patterns to address complex extraction scenarios that individual patterns in isolation may struggle to handle effectively.

5.1 Technical Specification Extraction Chain Pattern Sequence

5.1.1 Summary. The *Technical Specification Extraction Chain* pattern sequence combines three complementary patterns (*Keyword Trigger Extractor* → *Pattern Matcher* → *Semantic Extractor*) to systematically process technical documentation and extract standardized component specifications. This pattern sequence transforms unstructured technical documents into structured, validated component data by leveraging each pattern's strengths in sequence.

5.1.2 Problem. Technical documentation often contains critical component specifications embedded within larger documents, presenting the following challenges:

- Specifications are scattered throughout lengthy documents
- Component identifiers follow strict formatting rules requiring validation
- Specification details vary in structure but must conform to standardized outputs
- Manual extraction is time-consuming and error-prone
- Data relationships between components must be maintained
- Technical specifications require precise extraction without interpretation errors

Additional forces that complicate the solution include the following:

- Technical documents often mix specifications with marketing or descriptive content
- Serial numbers and component identifiers must exactly match prescribed formats
- Component specifications may include both required and optional attributes
- Cross-referencing between components may be necessary
- Data validation requirements vary by field type

5.1.3 Solution. The *Technical Specification Extraction Chain* pattern sequence implements a three-stage extraction process:

- Keyword Trigger Stage:
  - Identifies specification sections using consistent keyword markers
  - Isolates relevant text blocks for further processing
  - Maintains document structure and relationships
- Pattern Matching Stage:
  - Validates and extracts standardized identifiers
  - Ensures compliance with formatting requirements
  - Creates anchor points for detailed extraction
- Semantic Extraction Stage:
  - Processes specification details for validated components
  - Structures data according to predefined schemas
  - Maintains relationships between components and specifications

Key principles:

- Each stage builds on the previous stage's output
- Validation occurs at multiple levels
- Data relationships are preserved throughout the chain
- Extraction becomes increasingly specific and structured

5.1.4 Structure. The *Technical Specification Extraction Chain* pattern sequence consists of the following three interconnected components:

- Keyword Trigger:
  - Trigger Definition
  - Section Boundary Rules
  - Content Block Identification
- Pattern Matcher:
  - Serial Number Format
  - Validation Rules
  - Match Collection Logic
- Semantic Extractor:
  - Field Definitions
    - component_id
    - voltage
    - current
    - temp_range
  - Relationship Mappings
  - Output Schema

5.1.5 Implementation. An example of implementing the *Technical Specification Extraction Chain* pattern sequence is provided below.

Input:
Following the keyword "Technical Specifications:"

Extract all instances of the pattern:

```
[A-Z]{3}-[0-9]{3}-[0-9]{3}-[0-9]{3}
```

THEN Extract:

component_id: the component identifier (e.g., A-123, B-456)

voltage: the voltage rating of the component
current: the current draw specification
temp_range: the operating temperature range

from the following text:

The new hardware lineup includes several innovations.
Technical Specifications:
Component A-123
- Voltage Rating: 12V DC
- Current Draw: 500mA
- Operating Temp: -20°C to +85°C
- Serial Number: ABC-123-456-789

Other features include...
Technical Specifications:
Component B-456
- Voltage Rating: 24V DC
- Current Draw: 750mA
- Operating Temp: -30°C to +90°C
- Serial Number: XYZ-789-012-345

Output:
Component 1:
- component_id: A-123
- voltage: 12V DC
- current: 500mA
- temp_range: -20°C to +85°C

Component 2:
- component_id: B-456
- voltage: 24V DC
- current: 750mA
- temp_range: -30°C to +90°C

5.2 Medical Record Medication Extraction Chain Pattern Sequence

5.2.1 Summary. The *Medical Record Medication Extraction Chain* pattern sequence combines three complementary patterns (*Specify Constraints → Adaptive Attribute Extractor → Pattern Matcher*) to systematically extract and validate medication information from medical records. This pattern sequence transforms unstructured medical documentation into structured, standardized medication data while maintaining accuracy and completeness of prescription details.

5.2.2 Problem. Medical records contain critical medication information embedded within extensive patient documentation, presenting several challenges: Core Challenges:

- Medication information is mixed with other clinical data
- Prescription details vary in structure and completeness
- Dosage formats must conform to strict standards
- Multiple medications per patient require relationship maintenance
- Historical prescriptions must be distinguished from current ones

Complicating Factors:

- Inconsistent formatting across different healthcare providers
- Mix of standard and custom dosing instructions
- Various measurement units and frequencies
- Complex medication schedules
- Presence of conditional prescriptions
- Requirements for tracking prescription changes
- Need to maintain patient privacy while processing data

5.2.3 Solution. The *Medical Record Medication Extraction Chain* pattern sequence implements a three-stage extraction process:

- Constraint Specification Stage:
  - Defines clear boundaries for medication-related content
  - Excludes irrelevant medical information
  - Establishes context for extraction
- Adaptive Attribute Stage:
  - Extracts varying medication details
  - Captures all relevant prescription attributes
  - Maintains relationships between medications
- Pattern Matching Stage:
  - Validates dosage formats
  - Ensures standardization of measurements
  - Verifies prescription completeness

Key principles associated with applying this pattern include:

- Progressive refinement of extracted data
- Maintenance of medication relationships
- Validation at multiple stages
- Flexible attribute handling
- Strict format enforcement for critical fields

5.2.4 Structure. The *Medical Record Medication Extraction Chain* pattern sequence consists of the following three interconnected components:

- Constraints:
  - Scope Definition:
    - Include: prescribed medications, dosages
    - Exclude: patient history, vitals, diagnostics
  - Context Rules:
    - Current vs. historical prescriptions
    - Active vs. discontinued medications
- Adaptive Attributes:
  - Required Fields:
    - Medication name
    - Dosage
    - Frequency
  - Optional Fields:
    - Method of administration
    - Duration
    - Refills
    - Special instructions
- Pattern Matcher:
  - Dosage Format: [0-9]+mg
  - Frequency Patterns
  - Duration Formats

Data Flow Structure: Full Medical Record → Medication Sections → Detailed Prescriptions → Validated Dosages

5.2.5 Implementation. An example of implementing the *Medical Record Medication Extraction Chain* pattern sequence is provided below.

Input:
Constraints: Only consider sections related to prescribed medications and dosages

Exclude: patient history, vitals, and diagnostic information

THEN Extract:

medication_name: name of the prescribed medication,
...attributes related to medication prescriptions [must include dosage, frequency]...

FINALLY Extract all instances of the dosage pattern:

```
[0-9]+mg
```

from the following text:

Patient Record - Jane Doe
Blood Pressure: 120/80
Prescribed Medications:
- Lisinopril 10mg
Dosage: 1 tablet
Frequency: Daily
Method: Oral
Duration: 30 days
Refills: 2

Notes: Patient reports mild dizziness
Prescribed Medications:
- Metformin 500mg
Dosage: 1 tablet
Frequency: Twice daily
Method: Oral
Duration: 90 days
Side effects: Monitor for GI issues
Refills: 3

Output:
1. Medication Name: Lisinopril
- Dosage: 10mg
- Frequency: Daily

2. Medication Name: Metformin
- Dosage: 500mg
- Frequency: Twice daily

Extracted Dosage Patterns
- 10mg
- 500mg

5.3 Considerations When Chaining Patterns into Pattern Sequences

Chaining structured data extraction patterns into pattern sequences allows a more nuanced approach to data extraction, leveraging the strengths of multiple patterns to overcome individual limitations and providing the following benefits:

- Increased precision and accuracy in complex extraction scenarios,
- Enhanced flexibility in handling diverse data structures,
- Improved ability to extract contextually relevant data, and
- More efficient processing of large volumes of unstructured text.

However, chaining structured data extraction patterns together also requires careful consideration of several factors to ensure optimal performance and reliability. The order in which these chained patterns are applied is crucial, as the sequence in which patterns are applied can impact the extraction results significantly. Using keywords such as "FIRST" or "THEN" in the prompt can effectively inform the order in which the patterns are used, providing clear guidance to the LLM.

Complexity management is equally important since it is essential to ensure that the chained process remains comprehensible and maintainable, especially as the number of chained patterns increases. This process may involve documenting the purpose and function of each pattern in the chain, as well as the rationale behind their ordering. Moreover, robust validation is critical to verify that the chained patterns produce the desired outcomes across various input scenarios. This validation involves comprehensive testing with diverse datasets to ensure the reliability and accuracy of the extraction process.

6 Related Work

This section reviews the evolution of structured data extraction methods, encompassing traditional rule-based and statistical approaches, as well as emerging few-shot learning techniques and prompt engineering strategies that leverage LLMs.

6.1 Traditional Rule-based and Statistical Methods

Extracting structured data from unstructured corpora has traditionally been performed using rule-based systems and statistical models. Rule-based methods, such as the SystemT project [4] and regex-based extraction methods [3], rely on hand-crafted patterns and heuristics to parse and extract data. These methods benefit from high precision in well-understood domains but suffer from a lack of flexibility and scalability, often requiring intensive manual effort for rule crafting and maintenance.

Statistical approaches leverage a variety of machine learning techniques, including *Conditional Random Fields* (CRFs) [7] and deep learning models, that are trained on large annotated datasets. Techniques like *Named Entity Recognition* (NER) [8, 11] showcase

the effectiveness of statistical methods in identifying entity boundaries within the text. These models can generalize better than rule-based systems, especially when sufficient training data is available, but at the cost of expensive data annotation and limited cross-domain applicability.

6.2 Emerging Few-Shot Learning and Prompt Engineering Methods

The advent of large pre-trained LLMs, such as BERT and GPT-3 (and its successors in the GP4-4 family of models), marks a paradigm shift in structured data extraction strategies [14]. These methods apply few-shot learning capabilities inherent to LLMs and harness prompt engineering to guide models in performing tasks with only a few examples provided. Prompt engineering stands at the forefront of current research to enhance the performance of LLMs in data extraction with less reliance on extensive supervised training.

While prompt engineering demonstrates promising results in a diverse range of tasks and domains, designing and implementing effective prompts remains a nuanced challenge. Our contribution builds upon these advancements by introducing prompt patterns tailored for structured data extraction, driving towards a more systematic and modular approach. By providing users with a catalog of reusable prompt templates, as well as pattern compounds and sequences, our work integrates smoothly into the evolving framework of prompt engineering, offering a standardized and scalable solution for extracting knowledge from unstructured text [12].

7 Concluding Remarks

This paper introduces a systematic method for constructing structured data extraction prompts using modular, reusable prompt patterns. Our modular prompt framework and reusable patterns enable the development of robust extraction pipelines that retrieve data from unstructured corpora and textual artifacts. The key lessons learned we have gleaned thus far from our research on prompt patterns are summarized below:

- *These prompt patterns enable robust and customizable extraction pipelines for unstructured text.* Our exploration of patterns like *Semantic Extractor*, *Adaptive Attribute Extractor*, and *Specify Constraints* demonstrates the flexibility of prompts for customizing extraction to particular use cases. Likewise, adhering to a systematic paradigm ensures uniformity in execution across LLMs.
- *These prompt patterns offer a new approach to structured data extraction* that is oriented to handling inconsistent unstructured text. Unlike traditional rule-based approaches, which depend on pre-defined extraction rules that often fail when encountering deviations in text structure, extraction based on prompt patterns leverages the interpretive capabilities of LLMs. This ability to interpret and respond to varied textual inputs makes them a versatile option for retrieving data from text that does not adhere to a consistent format.
- *Our work represents an important step toward democratizing these technologies* by empowering expert and novice users alike to translate textual data into actionable insights. By refining best practices for prompt engineering, we aim to unlock the capabilities of LLMs to extract knowledge from text corpora via user-friendly, customizable extraction tools that balance human intuitions with AI capabilities [6].

Looking ahead, we plan to expand this initial catalog of five prompt patterns by incorporating new formats, data types, and domains. Our future work will focus on rigorously testing these patterns across different LLM architectures to better understand their generalizability. In particular, we plan to replicate these prompt patterns systematically with other widely-used LLMs, such as Gemini, Claude, and DeepSeek, to validate their broad applicability and identify any model-specific variations in effectiveness, as discussed in Appendix A. This replication work will help determine whether these patterns truly represent LLM-agnostic best practices for structured data extraction.

We also aim to explore a hybrid approach where LLMs analyze unstructured text to generate regular expressions (regexes). These regexes can then be used for traditional data extraction methods. This generated regex could potentially be fed back into an LLM for validation and refinement, creating a more robust and efficient extraction pipeline.

Our long-term goal remains to enable natural language interfaces to LLMs that mimic human-computer interaction. Just as SQL provides declarative, structured queries for relational data, our prompt framework allows users to describe extraction tasks at a high level for unstructured text [9].

Acknowledgments

We thank our PLoP shepherd, Michael Weiss, for insightful feedback on earlier drafts that greatly improved the quality and clarity of our work. We also acknowledge how the Advanced Data Analysis capability in OpenAI's ChatGPT-4 helped refine our prose and maintain consistency throughout the paper, as well as generating the input and output examples used to showcase our prompt patterns. While the core concepts, analyses, and pattern structures are our own, ChatGPT-4 provided valuable support in enhancing the paper's overall coherence, readability, and demonstration of pattern applications through generated examples.

A Applicability of Prompt Patterns to Multiple LLMs

While our prompt patterns provide a systematic framework, the output and performance of extraction tasks may vary depending on the specific LLM used. In particular, LLMs have varying capabilities, knowledge bases, and response characteristics. Nevertheless, the patterns presented here provide a structured foundation that can be applied across different LLMs to improve extraction results.

We designed the prompt patterns presented in this paper to apply across different LLMs. To showcase their utility and practicality, these patterns have been primarily tested using OpenAI's GPT-4. GPT-4's advanced natural language understanding capabilities make it an ideal candidate for demonstrating the effectiveness of our structured data extraction methods.

Other LLMs offer built-in capabilities for returning structured data formats. These features are typically provider-specific, however, and may not be available across all models. Our research thus provides a model-agnostic approach to structured data extraction that can be implemented regardless of whether a particular LLM offers native structured output support. Nevertheless, the quality and

consistency of extraction outputs may vary depending on the specific LLM being used, as different models have varying capabilities in understanding and processing natural language instructions.

We encourage readers to apply these prompt patterns with their favorite LLMs, such as more recent versions of ChatGPT, Gemini, Claude, DeepSeek, or other LLMs available to them. Experimentation with different LLMs is encouraged since it provides further insights into the generalizability and adaptability of prompt patterns across different LLM architectures and training paradigms. Please let us know if you find wildly different results compared to what we report here with ChatGPT-4.

B Overview of Prompt Pattern Form

The prompt patterns presented in this paper are documented using a variation of the classic POSA pattern format [2] to address the unique requirements of prompt engineering. Each pattern is described through the following sections:

- **Name**. A unique identifier for referring to the prompt pattern.
- **Summary**. A description of what the pattern does and the specific structured data extraction problem it solves.
- **Problem**. A detailed description of the data extraction challenge being addressed, including the context in which it occurs, the forces that make it hard to solve, and the constraints that must be considered.
- **Solution**. A precise description of how to apply the pattern, including the prompt structure, its key components, required elements and their relationships, and any variations or customization points.
- **Structure**. A detailed breakdown of the pattern's key components, their organization, and how they work together to achieve the desired outcome.
- **Implementation**. Concrete examples showing sample prompts, example inputs and outputs, common variations, and implementation considerations.
- **Consequences**. A balanced discussion of the benefits, limitations, performance implications, and impact on extraction quality and reliability.
- **Related Patterns**. References to complementary patterns, alternative approaches, and common pattern combinations or sequences.
- **Example Applications**. Real-world scenarios demonstrating effective pattern applications, including specific use cases and domain applications.

Our prompt pattern form intentionally omits the **Known Uses**, **Dynamics**, and **See Also** sections that are typically found in classic POSA pattern forms. Unlike classic software patterns, which are well-documented and accessible through open-source repositories across the Internet, prompts for LLMs are generally not codified or systematically available to the public. This lack of codification and availability poses challenges in presenting our proposed patterns entirely in the conventional pattern form.

The use of prompts is still an emerging practice, and there is no centralized repository or widely recognized standard that documents their use in a systematic way. As a result, our work aims to fill this gap by identifying, formalizing, and sharing these prompt patterns based on our extensive prompt engineering experience. While we understand that this approach does not align fully with the classic pattern forms, we hope to contribute to the foundation upon which future prompt patterns can be more formally validated and documented.

The template of a prompt pattern often begins with a conversational scoping statement, such as "for the following data extraction task" or "when processing the input text," that sets the context for the LLM to focus on structured data extraction. The prompt then provides a series of statements conveying the specific extraction capabilities the LLM should exhibit, typically phrased as rules, guidelines, or instructions. These rules may include conditional logic indicating when certain extraction techniques should be applied, such as "if the text contains numerical data, extract it in the following format." By codifying best practices into reusable templates, prompt patterns enable more reliable means for instructing LLMs to extract structured data that meets quality goals, conforms to specified formats, and adheres to extraction principles [15].

References

[1] Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt. 2007. *Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages*. Wiley and Sons, New York.

[2] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 2001. *Pattern-oriented software architecture: a system of patterns. Volume 1*. John Wiley & Sons, Chichester, England.

[3] Chia-Hui Chang, Mohammed Kayed, Moheb R. Girgis, and Khaled F. Shaalan. 2006. A Survey of Web Information Extraction Systems. *IEEE transactions on knowledge and data engineering* 18, 10 (2006), 1411–1428.

[4] Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Frederick Reiss, and Shivakumar Vaithyanathan. 2010. SystemT: An Algebraic Approach to Declarative Information Extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Stroudsburg, PA, 128–137.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.

[6] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a Neural Semantic Parser from User Feedback. arXiv:1704.08760 [cs.CL]

[7] John Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*. Morgan Kaufmann, Williamstown, MA, USA, 282–289.

[8] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, 260–270. https://doi.org/10.18653/v1/N16-1030

[9] Fei Li and Hosagrahar V. Jagadish. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *Proceedings of the VLDB Endowment* 8, 1 (2014), 73–84.

[10] Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing Continuous Prompts for Generation. arXiv:2101.00190 [cs.CL]

[11] David Nadeau and Satoshi Sekine. 2007. A Survey of Named Entity Recognition and Classification. *Lingvisticae Investigationes* 30, 1 (2007), 3–26.

[12] Laria Reynolds and Kyle McDonell. 2021. Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. arXiv:2102.07350 [cs.CL]

[13] Sunita Sarawagi. 2008. Information Extraction. *Foundations and Trends® in Databases* 1, 3 (2008), 261–377.

[14] Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained Language Models Yield Few-Shot Semantic Parsers. arXiv:2104.08768 [cs.CL]

[15] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. In *Proceedings of the 30th Pattern Languages of Programming (PLoP) conference*. The Hillside Group, Allerton Park, IL, 24.