Prompt Patterns for Structured Data Extraction from Unstructured Text

Max Moundas, Jules White, and Douglas C. Schmidt
Department of Computer Science
Vanderbilt University Nashville, TN, USA
{maximillian.r.moundas, jules.white, d.schmidt}@vanderbilt.edu

***Abstract*—Large language models (LLMs) show promise for extracting structured data from unstructured text due to their ability to identify patterns, keywords (including names, organizations, locations, and topics), and relations in text. In this task, however, LLM performance—specifically their precision, clarity, replicability, and uniform interpretation of results—depends heavily on how users express their prompts, which are instructions they give to LLMs. These performance measures are 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 information from text.**

This paper provides several 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. These combinations enable more effective use of LLMs for text mining and knowledge base construction from unstructured corpora. Moreover, our structured approach to prompt engineering supplies developers with cohesive and flexible templates, facilitating the creation of sophisticated data extraction workflows with more dependable results than *ad hoc* prompting.

***Index Terms*—Large language models (LLMs), Prompt patterns, Prompt engineering, Data Extraction**

I. Introduction

Challenges of extracting structured data from unstructured text. A longstanding challenge in natural language processing is extracting structured data from unstructured text [12], such as extracting product attributes (*e.g.*, price and brand) from product reviews or extracting patient information (*e.g.*, age and medical history) from clinical notes. More detailed examples of this type of extraction include

- *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 challenging because unstructured text lacks explicit demarcations of fields, so the desired information 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 [10].

Traditional rule-based and statistical methods for data extraction rely on hand-engineered features and labeled training data. Recently, large language models (LLMs), such as GPT-4, have shown promise for "few-shot" data extraction [13]. In this context, "few-shot" refers to the ability to perform a task with limited training examples.

Few-shot learning 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 the LLMs with examples and instructions [9]. Until recently, this dependence required users to understand deeply how to interact with LLMs effectively. Unfortunately, this requirement poses a daunting impediment to 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*. Prompt patterns [14] codify best practices for phrasing prompts to maximize extraction accuracy and provide knowledge transfer mechanisms that users can reference to problem-solve with LLMs more effectively [11]. The prompt patterns presented in this paper are reusable artifacts that serve as building blocks users can combine to prompt LLMs to extract desired information and complex nested data from unstructured text.

Prompt engineering [11] refers to the practice of designing and optimizing inputs (prompts) to elicit desired outputs from LLMs. In the context of data extraction, prompt engineering involves crafting instructions that guide LLMs to identify and

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

Table I summarizes the prompt patterns covered in this paper.

TABLE I
SUMMARY OF PROMPT PATTERNS

Pattern Category	Prompt Pattern
Extraction (Section III)	*Semantic Extractor*, *Dynamic Attribute Extractor*, *Pattern Matcher*
Instances Query on Input (Section IV)	*Specify Constraints*
Input Specification (Section V)	*Keyword Trigger Extractor*

Paper organization. The remainder of this paper is organized as follows: Section II gives an overview of structured data extraction using prompt engineering; Section III describes a catalog of prompt patterns for defining extraction formats and transforming input text into target structures, as well as documents various extraction patterns; Section IV discusses prompt patterns for querying input text to filter results; Section V explores patterns for specifying input sources; Section VI introduces a decision tree for selecting the most appropriate extraction pattern based on data characteristics and extraction requirements; Section VII examines chaining extraction patterns can enhance the overall extraction process for complex scenarios; Section VIII reviews related research on prompt engineering and few-shot data extraction methods; Section IX presents concluding remarks that summarize key lessons learned from the prompt patterns and extraction pipelines documented in this paper; and Appendix A provides an overview of our prompt pattern form, which should be familiar to readers acquainted with classic software pattern form [1], [4].

II. Overview of Structured Data Extraction Using Prompt Engineering

A systematic method for effective prompting is needed within the domain of prompt engineering to enable structured data extraction 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 comprehensive catalog of patterns, derived from rigorous research and practical applications over the past 18 months. These patterns serve as modular building blocks for assembling complex data extraction pipelines. By furnishing users with these customizable patterns, we enable sophisticated extraction capabilities, thereby helping to ensure robust and accurate data retrieval across diverse scenarios, such as the following:

- Extracting contact information (*e.g.*, emails or phone numbers) from corporate websites,
- Gathering financial figures from annual reports,
- Parsing patient information from medical records,
- Retrieving citations data 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 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 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 term "where."
- INPUT_SPECIFICATION specifies the source of the data or where the extraction should be performed. For example, it can specify the LLM to only extract data directly following a particular keyword or to only extract information from the last ten lines of the full input text.

This template provides a systematic and 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 template provides a systematic and 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 several ways:

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

For example, in a medical context, the same template can be adapted to extract patient diagnoses, medication dosages, or treatment outcomes by simply adjusting the EXTRACTION_PATTERN and INSTANCES_QUERY_ON_INPUT components. Similarly, in a financial setting, the template can be modified to extract quarterly earnings, stock prices, or market trends by fine-tuning these elements to the specific data requirements.

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 our future work, as outlined in Section IX.

III. Extraction Patterns

This section presents the following three prompt patterns that transform input text into desired target data structures during extraction: *Semantic Extractor*, the *Dynamic Attribute Extractor*, and the *Pattern Matcher*. These patterns enable extracting structured information from unstructured text through a variety of techniques—from flexible semantic parsing to strict pattern matching—providing prompt engineers multiple modular building blocks to meet diverse extraction needs.

A. The Semantic Extractor Pattern

1) Intent & Context: 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 that rely on explicit rules or patterns, this approach uses a descriptive framework to identify and capture desired data implicit in the target structured specification. This pattern 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) Motivation: Conventional data extraction methods often require explicit rule definitions, regular expressions, or manual tagging, which can be time-consuming, tedious, and error-prone. The *Semantic Extractor* pattern enables users to describe the kinds of information they're looking to extract in natural language and in the format it should be extracted. By incorporating semantic descriptions into the structured format, data extraction becomes intuitive and adaptive. This approach not only saves time but also leverages the language understanding capabilities of LLMs to handle variations in input text where traditional extraction methods may fail.

To implement the *Semantic Extractor* pattern successfully, consider the following key ideas, applying those that are relevant to your specific extraction task:

- Describe a partially or fully formed example of the target data structure.
- Use semantic descriptions within the target data structure to guide the extraction of specific data points.

These ideas are not mutually exclusive and can be combined as needed to optimize the extraction process for your particular use case.

3) Example Implementation:

> Please 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."

The extraction prompt provides a clear template for the target data, which consists of two primary components:

1) The **name** of the car.
2) The **identifier** or the VIN number.

This template incorporates semantic descriptions within the target data structure to guide the extraction process. For instance, "the name of the car" indicates that the LLM should look for the specific name or model of the car and map it to the name attribute. Similarly, "the VIN number" guides the LLM on what to extract for the identifier attribute.

4) Example Implementation:

> **Prompt**: Extract from the text: { name: 'the name of the musician', instrument: 'the primary instrument they play' } from: "Ludwig van Beethoven, a maestro primarily known for his prowess with the piano, whose compositions have stood the test of time."
> **Prompt**: Extract: { brand: 'brand of the phone', feature: 'its standout feature' } from: "The latest 'TechStar' phone boasts a revolutionary '3D touch' feature that has tech enthusiasts buzzing."

5) Consequences: The *Semantic Extractor* pattern leverages the capabilities of LLMs to understand and process natural language so it can handle variations in input, such as synonyms or rephrased sentences, without requiring precise query terms. For example:

> Original text: "The EcoSprint, identified by VIN 12345ABCDE, is making waves in the auto industry."
>
> Variation 1: "With its unique identifier 12345ABCDE, the EcoSprint sedan is turning heads."
>
> Variation 2: "Automotive enthusiasts are excited about the EcoSprint, a new car with serial number 12345ABCDE."

In both these variations, the pattern should successfully extract the car name and identifier, demonstrating its flexibility. However, the flexibility of *Semantic Extractor* is primarily a consequence of using LLMs rather than a unique feature of the pattern itself, *i.e.*, the pattern's strength lies in its structured means of guiding an LLM's extraction process.

The effectiveness of the *Semantic Extractor* pattern depends on how well an LLM can interpret the provided semantic descriptions. In turn, this capability relies on the quality and training of the LLM itself. A balance must be struck between providing enough detail to guide the extraction and not over-specifying to the point where it over-constrains the LLM, potentially missing relevant data variations.

While *Semantic Extractor* reduces the need for explicit rule definitions, it requires clear and unambiguous semantic descriptors. Vague or ambiguous descriptors may lead to incorrect or incomplete data extractions. Frequent testing and iterative refinement of prompts are therefore essential for reliable outcomes.

To illustrate how this pattern differs from alternative extraction methods, consider the following example:

> Review: 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.
>
> Extract the name and VIN number of the car in JSON format.
>
> Example: Input: Tesla just released an upgraded version of its Cybertruck model. The VIN for this model is ABC123.
>
> Output: {name: "Cybertruck", identifier: "ABC123"}

While this alternative approach can be effective, the Semantic Extractor pattern offers the following advantages:

- It provides a clear structure for the desired output, reducing ambiguity,
- The semantic descriptions offer context about what each field represents, potentially improving accuracy, and
- It can be more easily adapted to extract different or additional information without changing the core prompt structure.

The adaptability of the Semantic Extractor pattern is particularly noteworthy. By maintaining a consistent structure while allowing for flexible semantic descriptions, users can easily modify extraction targets without altering the underlying prompt framework. For instance, consider the following adaptation:

> Original: { name: "the name of the car", identifier: "the VIN number" }
> Modified: { name: "the name of the car", identifier: "the VIN number", manufacturer: "the company that made the car", year: "the year the car was released" }

This modification expands the extraction scope to include additional fields without changing the core prompt structure. Such adaptability is challenging with rule-based systems or regular expressions, which often require significant rewrites to accommodate new extraction targets. The Semantic Extractor's flexibility stems from its reliance on natural language understanding rather than rigid syntactic rules, allowing it to interpret and extract new semantic descriptions with minimal adjustments.

Alternative patterns, such as *Dynamic Attribute Extractor* or *Pattern Matcher*, may be more suitable when precision and handling complex data structures are paramount. These patterns rely on stricter templates and conditions, providing more control over the extracted data's format and accuracy in cases where semantics alone may be insufficient for effective extraction.

B. The Dynamic Attribute Extractor Pattern

1) Intent & Context: The *Dynamic Attribute Extractor* pattern is designed to extract and refine 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, ensuring accurate and comprehensive data capture while allowing for customizable constraints.

2) Motivation: In the diverse landscape of unstructured text, entities are often described with varying attributes, which can lead to inconsistency and loss of information when using rigid, predefined data extraction methods. For example, a product description might include unique features not captured by a fixed extraction template, resulting in incomplete data representation. Alternative methods of data extraction might achieve uniformity, but at the expense of valuable data since they cannot handle the natural variation and richness of attributes that actual instances may possess.

The *Dynamic Attribute Extractor* pattern overcomes these limitations by offering a solution that begins with a broad extraction framework that adjusts based on the specific attributes found in the text, while also allowing for the specification of constraints. This facilitates a more nuanced data extraction process that can handle diversity and ensure the complete attribute set is represented in the structured data output, subject to defined constraints.

3) Key Ideas: To implement the *Dynamic Attribute Extractor* pattern successfully, consider the following key ideas, applying those that are relevant to your specific extraction task:

- Start with a definition of the target structured data format with semantic descriptions embedded.
- Include an open-ended section describing what additional attributes should be discovered and extracted.
- Specify constraints defining limits or requirements on attribute selection, such as "[all instances must have the same attributes]" or "[each instance can have different attributes]".
- Consider an initial discovery process to identify relevant attributes before final extraction.

These ideas are not mutually exclusive and can be combined as needed to optimize the extraction process for your particular use case.

The *Dynamic Attribute Extractor* pattern's unique strength lies in its ability to adapt to varying attribute sets while maintaining specified constraints. This flexibility allows for comprehensive data capture across diverse datasets, distinguishing it from more rigid extraction methods.

4) Example Implementation:

> Please extract:
>
> { name: "the name of the car", ...attributes related to the car [all instances should have the same attributes]... }
>
> from the following text:
>
> "The 'EcoSprint' sedan stood out for its V6 engine, allowing it to go from 0 to 60 mph in just 5.4 seconds. Braking performance is also top-notch, with the sedan coming to a complete stop from 60 mph in just 115 feet. The cabin has the EcoSprint's premium 10-speaker setup. Safety hasn't been compromised either, as the EcoSprint comes equipped with adaptive cruise control, blind-spot monitoring, and automated emergency braking.
>
> The 'TrailRider' SUV, draped in its unique 'Desert Sand' hue, was hard to miss. Its engine, a robust V8, ensures that the vehicle can handle both city roads and off-road trails with ease. With a 0-60 time of 6.2 seconds, the TrailRider packs a punch in terms of raw power. The braking performance is equally commendable, enabling the SUV to decelerate from 60 mph to a standstill in about 120 feet. Audio enthusiasts riding inside the TrailRider will be treated to a booming 12-speaker setup, perfect for long journeys and adventures. To top it all off, safety features such as lane-keeping assist, 360-degree cameras, and a collision warning system ensure the TrailRider provides a secure environment for its passengers."

Similar to the *Semantic Extractor* pattern, the *Dynamic Attribute Extractor* pattern provides a structure for the target name, but it empowers the LLM to determine additional relevant target data. From this example, we would expect to see the following additional components in the output generated by the LLM:

1) The **engine** in the car.
2) The **0-60** time performed by the car.
3) The **braking performance** of the car.
4) The **speakers** installed in the car.
5) The **safety features** in the car.

5) Example Implementation:

> **Prompt:** Extract from the text: { name: 'the name of the gadget', ...attributes related to the gadget [each instance can have different attributes]... } from: "The 'TechMaster Pro' laptop boasts 16GB RAM and a sleek design. In contrast, the 'UltraView' tablet impresses with its 12-inch display."
>
> **Prompt:** Extract { brand: 'brand of the beverage', ...attributes related to the beverage [all instances should have at least one common attribute]... } from: "The 'Quench' water comes in a 500ml bottle, while 'EnerBoost' energy drink offers a caffeine punch and comes in a 250ml can."

While the ellipses (...) are used in our example to denote the open-ended attribute section, alternative syntactic elements could be employed. The key is to provide a clear structure that guides the LLM in understanding where dynamic attribute extraction should occur. Other potential syntactic approaches could include:

- Using specific delimiters (*e.g.*, ### or $$$),

- Employing a structured prompt language (*e.g.*, SudoLang), and
- Utilizing natural language instructions (*e.g.*, "Extract any additional relevant attributes").

The choice of syntax should prioritize clarity and consistency within the prompt engineering ecosystem being used.

6) Consequences: The *Dynamic Attribute Extractor* offers significant advantages in terms of adaptability and the ability to capture a rich, diverse dataset. It enables an LLM to parse text and extract desired data, even if the prompt did not explicitly state the listed components. This pattern differs from a simple JSON extraction prompt in its ability to specify constraints and adapt to varying attribute sets. While a standard JSON output would typically use the same attributes for all instances, *Dynamic Attribute Extractor* allows for flexibility when needed. This flexibility is particularly useful when dealing with diverse data sets where attributes may vary significantly between instances.

However, the flexibility of *Dynamic Attribute Extractor* can lead to challenges in maintaining consistency across extracted data and ensuring all relevant attributes are captured. The effectiveness of this pattern relies heavily on well-crafted constraints and iterative refinement by experienced prompt engineers. It requires a balance between allowing for dynamic discovery and providing sufficient guidance to the LLM.

One potential limitation of the *Dynamic Attribute Extractor* is the risk of overlooking important attributes. To mitigate this, consider implementing a two-step process:

1) **Initial discovery** – Use the LLM to analyze the text and identify potentially relevant attribute.
2) **Specific extraction** – Based on the discovered attributes, create a more targeted extraction prompt that explicitly lists the attributes to be extracted.

This approach combines the adaptability of dynamic extraction with the precision of specified attributes, resulting in more comprehensive and accurate data extraction.

C. The Pattern Matcher Pattern

1) Intent & Context: The *Pattern Matcher* pattern precisely extracts structured data from unstructured text by searching for instances that exactly match a predefined template, which is similar to applying a regular expression (regex) within an LLM context. It is best suited for extraction tasks where the data structure is rigidly defined upfront and unlikely to change. Precision is therefore favored over flexibility.

2) Motivation: In many production datasets, certain types of structured data reliably follow fixed formats, including phone numbers, postal codes, social security numbers, etc. The rigid conventions in these formats make them ideal candidates for extraction via the *Pattern Matching* prompt pattern. Since the structure is predictable, users can define a template that specifies the precise expected pattern, such as "[3 digits]-[3 digits]-[4 digits]" for phone numbers.

While traditional tools like Python, sed, awk, and/or perl scripts can handle such pattern matching tasks efficiently , implementing this pattern with an LLM can be beneficial in certain scenarios. For instance, when working with a larger LLM-based system where maintaining a consistent workflow is crucial or when the pattern matching is part of a more complex language understanding task that leverages an LLM's capabilities.

3) Key Ideas: To implement the *Pattern Matcher* pattern successfully, the following key ideas should be followed, applying those that are relevant to your specific extraction task:

- Define a clear search pattern that matches the expected structure of the target data,
- Use delimiters, boundaries, and formatting clues to narrow the search,
- Allow for multiple instances of the pattern,
- Match against the pattern definition exactly with no interpretation, and
- Consider whether the task is better suited for traditional pattern matching tools or if LLM integration provides additional value.

These ideas are not mutually exclusive and can be combined as needed to optimize the extraction process for your particular use case.

4) Example Implementation:

> Please 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."

This extraction prompt provides a precise pattern that matches the expected structure of the target data order IDs. The pattern consists of the text "Order ID: " followed by an 8 digit number enclosed in brackets. By defining this strict template, an LLM can scan the input text and extract any substrings that exactly match the pattern, without needing to interpret the context.

The explanatory text "for future reference" in the example implementation above provides a clue that an order ID is likely to follow. When applied to the sample input text, the pattern matches "Order ID: 12345678" precisely. This precision allows an LLM to extract the structured order ID based on the predefined pattern format.

5) Example Implementation:

> **Prompt:** Extract all instances of the social security number pattern: "[3 digits]-[2 digits]-[4 digits]" from the following text: "Both John's and Jane's social security numbers, 123-45-6789 and 987-65-4321, were written down on the form used to purchase their new car."
>
> **Prompt:** Extract all instances of the email address pattern: "[a-z0-9._%+-]@[a-z0-9].[a-z]" from: "If you are struggling with this material and need

help before the exam, email me your questions at 'yourprofessor@myuniversity.edu'."

Prompt: Extract all instances of the phone number pattern: "([3 digits]) [3 digits]-[4 digits]" from: "You can reach the help desk at 123-456-7890 anytime from 9am-5pm Monday through Friday."

6) Consequences: The *Pattern Matcher* pattern offers a reliable solution for data extraction tasks when the specific structure or format of the target data is predetermined. One of its primary benefits is the high precision in extracting data, particularly when the pattern is defined accurately in advance. This precise nature ensures that only the most relevant data is extracted, eliminating the possibility of errors due to misinterpretation.

However, the rigidity of *Pattern Matcher* comes with its own set of limitations. The pattern is notably brittle, meaning any deviation from the predefined pattern—even if slight—will cause the matcher to overlook relevant data. Users must therefore have a thorough understanding of the structure and format of the data they're dealing with before attempting extraction via this pattern. This limitation contrasts with more flexible data extraction patterns, such as Dynamic Attribute Extraction, that understand and extract data based on its context rather than an exact pattern.

Utilizing the *Pattern Matcher* across varied data sources may yield incomplete results when the data lacks uniformity. For example, phone numbers could appear in multiple formats, such as "(123) 456-7890," "123-456-7890," or "123.456.7890." If the Pattern Matcher is configured to recognize only one specific format, discrepancies such as typographical errors or format variations can result in relevant information being overlooked. Consequently, the effectiveness of this pattern depends on the predictability and standardization of data presentation within the targeted dataset.

While this pattern demonstrates the capability of LLMs to perform regex-like operations, it may not always be the most efficient solution. In many cases, preprocessing text using traditional text-processing tools, such as Python, sed, awk, or perl scripts, might be more appropriate. The decision to implement *Pattern Matcher* with an LLM should be based on the specific requirements of the task at hand, the overall system architecture, and whether an LLM's additional capabilities provide significant advantages over conventional pattern matching tools.

IV. Instances Query on Input Patterns

This section presents the *Specify Constraints* prompt pattern that was first introduced in previous research [14] under the name *Context Conveyor*).[1] While its broader usage includes managing conversational flow and topic relevance, here we emphasize the application of *Specify Constraints* specifically to structured data extraction from unstructured sources.

[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.

A. The Specify Constraints Pattern

1) Intent & Context: The *Specify Constraints* pattern refines the attention of LLMs during data extraction tasks by allowing users to explicitly state constraints within which LLMs should operate. By clearly defining the constraints to apply, users can better direct an LLM to focus on specific aspects of the data, thereby reducing the likelihood of irrelevant data extraction and improving response coherence.

2) Motivation: When LLMs are tasked with extracting structured data they benefit greatly from understanding the specific constraints under which they should operate. Clear constraints help prevent the inclusion of extraneous information and ensure that extracted data aligns with user intentions. The *Specify Constraints* pattern mitigates the challenges associated with free-flowing, unstructured text, where LLMs might otherwise draw on inappropriate parts of a conversation or document—by allowing users to delineate the relevant content scope clearly. When constraints are stated explicitly, users can maintain a focus that is pertinent to the ongoing data extraction, avoiding the LLM's potential regression to previously addressed or unrelated topics.

3) Key Ideas: To implement the *Specify Constraints* pattern successfully, the following key ideas should be followed, applying those that are relevant to your specific extraction task:

- Provide explicit constraints regarding the input text, such as "Only consider X," "Exclude Y," etc.
- Separate constraints from the extraction task instructions for clarity.

These ideas are not mutually exclusive and can be combined as needed to optimize the extraction process for your particular use case.

4) Example Implementation:

Constraints: Only consider renewable energy sources

Extract: { TYPE, COST_MULTIPLE }

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. Hydroelectric and geothermal energies stand at 1.6X and 2.1X, respectively, owing to dam construction and drilling expenses. Coal, a predominant fossil fuel source, generally costs 1.0X, but it carries significant environmental implications."

This prompt provides explicit constraints to focus only on certain parts of the context, which, in this case, is renewable energy sources. This constraint shapes the context considered by an LLM when extracting the requested "TYPE, COST_MULTIPLE" structured data. Without it, an LLM might extract data on all energy types mentioned, including the non-renewable ones.

The example input text contains details on both renewable and non-renewable sources. However, the LLM only extracts data for the renewable types based on the specified constraint. For instance, it will extract

- Solar, 2.0X
- Wind, 1.8X
- Hydroelectric, 1.6X
- Geothermal, 2.1X

but it will ignore fossil fuels and coal.

5) Example Implementation:

Prompt: Constraints: Focus on fruit prices

Extract: { FRUIT_NAME, COST_PER_KG } from the following: "Bananas cost $0.50 per kg, while toy cars are priced at $5 each. Mangoes are priced at $1.20 per kg."

Prompt: Constraints: Prioritize historical figures

Extract: { PERSON_NAME, ERA } from the text: "Einstein, who was prominent in the 20th century, had a profound impact on physics. Apples are a great source of fiber."
Prompt: Constraints: Only consider the first sentence

Extract: { BRAND, SPEED } from the text: "Toyota's new model runs at 120mph. In unrelated news, the weather is set to be sunny tomorrow.;;
Prompt: Constraints:

- Only consider renewable energy sources
- Exclude energy sources with a cost multiple above 1.8X

Extract: { TYPE, COST_MULTIPLE } from the 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. Hydroelectric and geothermal energies stand at 1.6X and 2.1X, respectively, owing to dam construction and drilling expenses. Coal, a predominant fossil fuel source, generally costs 1.0X, but it carries significant environmental implications."

6) Consequences: The *Specify Constraints* pattern can significantly enhance the relevance and accuracy of data extracted by an LLM, particularly in complex or subject-mixed contexts. By instructing the LLM to consider only the parts of the text aligned with the current analytical goal, users can minimize noise and improve the overall quality of the structured data output.

However, this precision may come with potential limitations. For example, excessive narrowing of context can omit insightful information that are relevant to broader analytical objectives, though may seem outside the immediate scope. Likewise, insufficient context management can lead to vague or overgeneralized data extraction that does not meet the desired level of specificity. It is therefore critical to articulate the context with sufficient granularity to guide an LLM effectively while remaining inclusive of information that could offer value. This balance is vital to ensure that structured data extracted from unstructured text is both accurate and comprehensive.

Moreover, the *Specific Constraints* pattern may inadvertently reset or obscure context that has been previously established, thereby impacting the LLM's output in unintended ways, as discussed in [14]. To mitigate this risk, consider the continuity of context and the repercussions of context shifts carefully. Confirmations or summaries potentially explaining the impact of these shifts could be integrated to ensure informed decision-making concerning context manipulation.

V. Input Specification Patterns

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

A. The Keyword Trigger Extractor Pattern

1) Intent & Context: The *Keyword Trigger Extractor* pattern provides a straightforward way to identify and extract structured data that consistently appears after predefined cue words or phrases in unstructured text corpora. This pattern is particularly useful when dealing with large volumes of text where certain types of information are consistently introduced using fixed keywords. *Keyword Trigger Extractor* allows rapid filtering and data gathering by honing in on specific textual cues already present in the source material. Rather than scanning entire documents, this pattern leverages the existing semantics of introductions already in the text itself.

2) Motivation: In many domains, textual data contains important details, such as names, dates, and figures, that are reliably preceded by certain keywords or labels that flag their significance. For instance, customer records may consistently contain fields like "Name:", "Date of Birth:", "Address:", etc. Research papers often introduce key information using standard keywords like "Abstract", "Introduction", "Results", and so on.

By specifying these cues as triggers, an LLM can rapidly locate and pull structured data following them with a high degree of accuracy. These cues avoid the complexity of applying generic rules or patterns that may inadvertently extract incorrect information. Domain-specific cues also provide built-in semantic guidance for extraction.

In long or complex documents, the selective power of key trigger words makes focused data extraction possible without laboriously reviewing entire texts. This capability enables the rapid construction of structured data sets by targeting only the most relevant information needed. Reducing manual effort also makes it feasible to process larger volumes of text data.

3) Key Ideas: To implement the *Keyword Trigger Extractor* pattern successfully, the following key ideas should be followed, applying those that are relevant to your specific extraction task:

- Identify a keyword or set of keywords that precede the target data in the text.
- Define the structured data type or format you expect to extract following the keyword.

These ideas are not mutually exclusive and can be combined as needed to optimize the extraction process for your particular use case.

4) Example Implementation:

Following the keyword "Born:", extract:

{ birthDate: 'date of birth', birthPlace: 'place of birth' }

from:

"...Born: 5th July 1980 in New York..."

This prompt provides the keyword trigger "Born:" that consistently precedes the target data to extract, which in this case are the date and place of birth details. It also specifies the structured format to capture this data as two fields: "birthDate:" and "birthPlace:". The explanatory text in the target format indicates what values should go in each field.

When applied to the sample input, the assistant locates the "Born:" trigger word and extracts the structured data following it as

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

The date and location details are inserted into the appropriate fields based on the semantics provided in the target specification, without needing additional context.

Prompt: Following the keyword "Price:", extract: { cost: 'amount in USD' } from: "The item details are as follows: Price: $20..."

Prompt: Following the keyword "Address:", extract: { street: 'street name', city: 'city name', postal: 'postal code' } from: "...Address: 123 Maple St, Springfield, 12345..."

5) Consequences: The *Keyword Trigger Extractor* pattern efficiently identifies specific fields of interest, which is beneficial in domains with well-formatted text. By leveraging keywords that consistently flag the introduction of relevant data, this pattern enhances the precision and accuracy of data extraction tasks. In practice, *Keyword Trigger Extractor* turns specific cues within large texts into reliable anchors for structured data capture, aiding in the systematic organization and retrieval of information.

The *Keyword Trigger Extractor* pattern enables efficient extraction by honing in on relevant data using predetermined keyword cues already present in the text itself. The trigger provides built-in context that guides the LLM. By reducing the search space via domain-specific triggers, this technique enables rapid data gathering from large corpora without extensive manual effort.

Although the *Keyword Trigger Extractor* pattern offers precise extraction based on predefined cues, there are issues that must be considered. The extraction's accuracy depends heavily on the consistency of the keyword's use in the text. The extractor may miss out on data if the keyword varies, or if different keywords are used for the same type of information. This pattern may thus not capture data that is far removed or separated from the keyword, so it's crucial to ensure target data appear immediately after the keyword. Over-reliance on specific keywords can lead to oversights, especially if data can be introduced by multiple keywords or in varying contexts.

VI. Selecting the Appropriate Extraction Pattern

Selecting the most suitable extraction pattern is crucial to achieve optimal results in data extraction tasks. This section presents a decision tree to guide the selection process, considering the nature of the data, its presentation, and specific extraction requirements. This decision tree builds upon all five patterns discussed in earlier sections: *Semantic Extractor*, *Dynamic Attribute Extractor*, *Pattern Matcher*, *Specify Constraints*, and *Keyword Trigger Extractor*.

A. Decision Tree for Pattern Selection

Figure 1 shows the process of selecting the most appropriate extraction pattern based on key characteristics of the data and extraction requirements. To use this decision tree, start at the top left box and follow the path based on the answers to each question. The leaf nodes indicate the recommended pattern for a specific scenario.

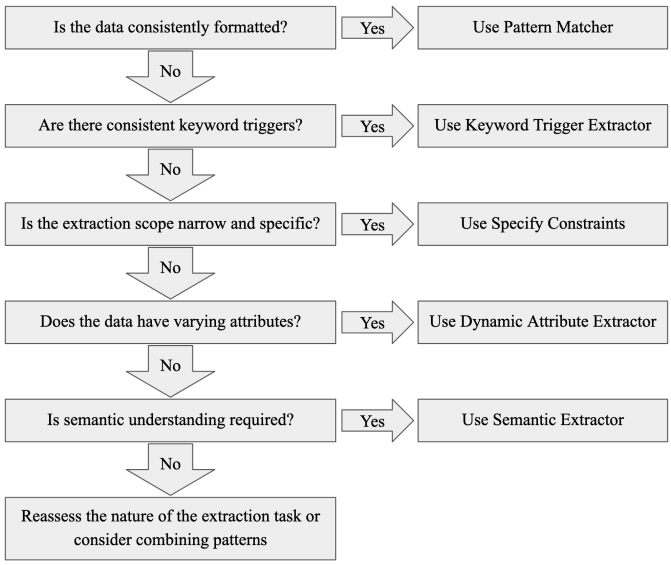

Fig. 1. Decision Tree for Selecting and Applying Extraction Patterns

B. Decision Points Explained

The following list elaborates on the key decision points presented in the decision tree (Figure 1). Each item corresponds to

a node in the tree, providing further explanation of the criteria used to navigate the pattern selection process:

1) **Is the data consistently formatted?** If the data follows a strict, predictable format (*e.g.*, phone numbers, dates), the *Pattern Matcher* may be the most efficient choice.
2) **Are there consistent keyword triggers?** When specific keywords reliably precede the target data, the *Keyword Trigger Extractor* can effectively locate and extract the information.
3) **Is the extraction scope narrow and specific?** If the extraction task requires focusing on particular aspects within a broader context, the *Specify Constraints* pattern helps filter the relevant information.
4) **Does the data have varying attributes?** For cases where the attributes of the target data may differ between instances, the *Dynamic Attribute Extractor* offers the necessary flexibility.
5) **Is semantic understanding required?** When the extraction task demands comprehension of context and meaning, the *Semantic Extractor* is the most suitable choice.

C. Application Examples

To demonstrate the application of the decision tree shown in Figure 1, consider the following five examples:

- **Example 1: Extracting phone numbers from a customer database**
  Decision path: Consistently formatted? Yes → apply *Pattern Matcher*.
  Rationale: Phone numbers typically follow a consistent format, making *Pattern Matcher* the most efficient choice for accurate extraction.
- **Example 2: Extracting product details from e-commerce listings**
  Decision path: Consistently formatted? No → Consistent keyword triggers? Yes → apply *Keyword Trigger Extractor*.
  Rationale: E-commerce listings often use consistent keywords (*e.g.*, "Price:", "Description:") to introduce product details, making *Keyword Trigger Extractor* suitable.
- **Example 3: Extracting specific energy data from a comprehensive report**
  Decision path: Consistently formatted? No → Consistent keyword triggers? No → Narrow extraction scope? Yes → apply*Specify Constraints*.
  Rationale: When focusing on specific energy data within a broader report, *Specify Constraints* helps narrow the extraction scope to relevant sections.
- **Example 4: Extracting car details from diverse automotive reviews**
  Decision path: Consistently formatted? No → Consistent keyword triggers? No → Narrow extraction scope? No → Varying attributes? Yes → apply *Dynamic Attribute Extractor*.
  Rationale: Car reviews may contain varying attributes for different models, making *Dynamic Attribute Extractor* the most flexible choice.
- **Example 5: Extracting key points from legal documents**
  Decision path: Consistently formatted? No → Consistent keyword triggers? No → Narrow extraction scope? No → Varying attributes? No → Semantic understanding required? Yes → apply *Semantic Extractor*.
  Rationale: Legal documents require understanding of context and meaning, making *Semantic Extractor* the most appropriate choice.

D. Limitations

Although the decision tree shown in Figure 1 provides a structured approach to selecting an extraction pattern it does incur several limitations. For example, real-world scenarios may not always fit neatly into the categories presented above. Likewise, some scenarios may benefit from combining patterns, such as using *Specify Constraints* to focus on relevant sections before applying *Dynamic Attribute Extractor*. The effectiveness of each pattern can vary depending on the specific implementation and the capabilities of the underlying LLM.

VII. Chaining Extraction Patterns

While the 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 either

- **Pattern compounds**, which capture recurring pairs of patterns that can be treated as a single distinct pattern to addresses a design problem [1], or
- **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 compounds and 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.

A. Benefits of Chaining Patterns into Pattern Sequences

Chaining extraction patterns into pattern sequences allows for 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 information, and
- More efficient processing of large volumes of unstructured text.

For instance, combining the *Specify Constraints* pattern with the *Semantic Extractor* pattern into a pattern compound can effectively filter large datasets before applying more computationally intensive semantic analysis, resulting in both improved accuracy and efficiency.

B. Common Chaining Strategies

The following examples showcase common chaining strategies and their applications in various extraction scenarios. These strategies demonstrate how different patterns can be combined to address specific extraction challenges.

1) Pattern Compound: Combine the Specify Constraints Pattern with the Semantic Extractor pattern:

- **Intent & Context:** This pattern compount extracts specific structured data from a large, diverse dataset by first narrowing the focus to relevant information and then applying semantic extraction.
- **Motivation:** Use this pattern compound when dealing with comprehensive reports or databases where only a subset of information is relevant, and that information requires semantic understanding to extract accurately.
- **Example Implementation:**

  FIRST, apply constraints: Only consider renewable energy sources

  THEN, extract: { energy_type: "the type of renewable energy", cost_factor: "the cost multiple compared to fossil fuels", primary_challenge: "the main obstacle for implementation" }

  From the following text: [Insert comprehensive energy report text here]

2) Pattern Compound: Combine the Keyword Trigger Extractor Pattern with the Dynamic Attribute Extractor Pattern:

- **Intent & Context:** This pattern compound identifies specific sections within a document using keywords, then extracts varied attributes from each section.
- **Motivation:** This combination is ideal for processing documents with multiple distinct sections, each containing different sets of attributes, such as product catalogs or multi-topic reports.
- **Example Implementation:**

  FIRST, identify product descriptions starting with "Product:".

  THEN, for each identified product, extract: { name: "the name of the product", ...attributes related to the product [each product can have different attributes]... }

  from the text following "Product:" up to the next "Product:" or end of text.

3) Pattern Compound: Combine the Specify Constraints pattern with the Pattern Matcher Pattern:

- **Intent & Context:** This pattern compound extracts specific formatted data within a larger context by first applying constraints and then matching precise patterns.
- **Motivation:** This combination is effective for extracting structured data with known formats from large datasets, particularly when only a subset of the formatted data is relevant.
- **Example Implementation:**

  FIRST, apply constraints: Only consider phone numbers starting with area codes 5xx

  THEN, extract all instances of the pattern: "([5][0-9]2)-[0-9]3-[0-9]4"

  from the following customer database: [Insert customer database text here]

4) Pattern Sequence: Comprehensive Data Extraction and Analysis:

- **Intent & Context:** This pattern sequence demonstrates the chaining of more than two patterns to perform a comprehensive data extraction and analysis task. It combines the *Specify Constraints*, *Keyword Trigger Extractor*, *Semantic Extractor*, and *Dynamic Attribute Extractor* patterns to process complex, multi-faceted documents.
- **Example Implementation:**

  FIRST, apply constraints: Only consider sections related to renewable energy projects.

  THEN, identify project descriptions starting with the keyword "Project:"

  NEXT, for each identified project, extract: name: "the name of the project", type: "the type of renewable energy"

  FINALLY, extract: ...attributes related to the project [each project can have different attributes]...

  From the following text:

  "Our company is involved in various energy initiatives.

  Project: SolarTech
  SolarTech is our flagship solar energy project located in Arizona. It spans 500 acres and has a capacity of 100MW. The project employs cutting-edge photovoltaic technology and is expected to offset 150,000 tons of CO2 annually.

  Project: WindForce
  WindForce is an offshore wind farm off the coast of Maine. It consists of 50 turbines, each capable of generating 6MW. The project faced initial challenges due to environmental concerns but has

since gained community support.

We are also considering expanding into nuclear energy, but no concrete plans have been made yet.

Project: HydroFlow
HydroFlow is a run-of-river hydroelectric project in Oregon. It has a capacity of 20MW and utilizes the natural flow of the river to generate electricity. The project has been praised for its minimal environmental impact and fish-friendly design."

- **Key Ideas:** This pattern sequence combines the following four patterns:
  1) *Specify Constraints*, which focuses the extraction on renewable energy projects,
  2) *Keyword Trigger Extractor*, which identifies relevant sections starting with "Project:",
  3) *Semantic Extractor*, which extracts specific structured data (name and type) for each project, and
  4) *Dynamic Attribute Extractor*, which captures additional attributes that may vary between projects.

  Chaining these patterns together enables a comprehensive extraction that is both focused and flexible, capturing essential information about each renewable energy project while adapting to the varying details provided for each.

C. Considerations for Effective Chaining

Chaining extraction patterns into pattern compounds and pattern sequences requires careful consideration of several factors to ensure optimal performance and reliability. The order of application is crucial, as the sequence in which patterns are applied can significantly impact the extraction results. 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.

VIII. Related Work

A. Traditional Rule-based and Statistical Methods

Historically, the extraction of structured data from unstructured corpora has been performed using rule-based systems and statistical models. Rule-based methods, such as the SystemT project [3] and regex-based extraction methods [2], 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) [6] and deep learning models, that are trained on large annotated datasets. Techniques like *Named Entity Recognition* (NER) [7], [10] 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.

B. Emerging Few-Shot Learning and Prompt Engineering Methods

The advent of large pre-trained language models, such as BERT, GPT-3, and its successors, marks a paradigm shift in structured data extraction strategies [13]. 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 seamlessly integrates into the evolving framework of prompt engineering, offering a standardized and scalable solution for extracting knowledge from unstructured text [11].

IX. Concluding Remarks

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

- **Modular prompt architecture with reusable patterns enables robust and customizable extraction pipelines for unstructured text.** Our exploration of patterns like *Semantic Extractor*, *Dynamic 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 systems.
- **The prompt engineering patterns explained in this paper offer a new approach to structured data extraction, particularly adept at handling inconsistent unstructured text.** Unlike traditional rule-based

approaches, which depend on pre-defined extraction rules and 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 prompt patterns a versatile option for retrieving information from text that does not adhere to a consistent format.

- **Our work represents an important step toward democratizing these technologies to empower 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.

Looking ahead, we plan to expand this initial pattern catalog by incorporating new formats, data types, and domains. Our goal is to enable natural language interfaces to LLMs that mimic human-computer interaction. Just as the *Structured Query Language* (SQL) provides declarative, structured queries for relational data, our prompt architecture allows users to describe extraction tasks at a high level for unstructured text [8].

By continuing to refine best practices for prompt engineering, we can unlock the capabilities of LLMs to extract knowledge from text corpora. Our work represents an important step toward customizable, user-friendly extraction tools that balance human intuitions with AI capabilities [5]. The democratization of these technologies will empower experts and novice users alike to translate textual data into actionable insights.

ACKNOWLEDGMENTS

We thank our PLoP shepherd, Michael Weiss, for his insightful feedback on earlier drafts that substantially improved the quality and clarity of our work. We also acknowledge the use of OpenAI's ChatGPT-4, specifically its Advanced Data Analysis capability, in refining the language and maintaining consistency throughout the document. While the core concepts and analyses are the authors' own, ChatGPT-4 provided valuable support in enhancing the paper's overall coherence and readability.

APPENDIX A
OVERVIEW OF PROMPT PATTERN FORM

The prompt patterns presented in this paper are documented using a form similar to classic software patterns [4], with analogous versions of the **Name**, **Classification**, **Intent**, **Motivation**, **Structure**, **Example Implementation**, and **Consequences** sections in classic pattern form. Each section of our prompt pattern form is outlined briefly below:

- **Name:** A unique identifier for referring to the prompt pattern.
- **Intent & Context:** Summarizes the goals and rationale for the pattern.
- **Motivation:** Describes the situations where applying the pattern is relevant.
- **Key Ideas:** Outlines the key ideas and information (often phrased as instructions or rules) that must be conveyed to an LLM to achieve the desired capabilities.
- **Example Implementation:** Provides a sample implementation showing how the pattern can be instantiated in any LLM.
- **Consequences:** Discusses the benefits and potential drawbacks of applying the pattern.

Our prompt pattern form intentionally elides the **Known Uses** section. 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 makes it hard to present our proposed patterns in the classic 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 [14].

The prompt patterns outlined in this paper have been designed with the intent to be generally applicable across different LLMs. To illustrate 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. We encourage readers to apply these prompt patterns with their favorite LLMs, such as ChatGPT-4, or other advanced models available to them. Experimentation with different LLMs is not only welcome but encouraged, as it can provide further insights into the generalizability and adaptability of prompt patterns across different LLM architectures and training paradigms.

REFERENCES

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

[2] Chia-Hui Chang, Mohammed Kayed, Moheb R. Girgis, and Khaled F. Shaalan. A survey of web information extraction systems. *IEEE transactions on knowledge and data engineering*, 18(10):1411–1428, 2006.

[3] Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Frederick Reiss, and Shivakumar Vaithyanathan. Systemt: an algebraic approach to declarative information extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 128–137, 2010.

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

[5] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. *arXiv preprint arXiv:1704.08760*, 2017.

[6] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning, ICML*, volume 1, 2001.

[7] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[8] Fei Li and Hosagrahar V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, 2014.

[9] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*, 2021.

[10] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[11] Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–7, 2021.

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

[13] Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. Constrained language models yield few-shot semantic parsers. *arXiv preprint arXiv:2104.08768*, 2021.

[14] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. A prompt pattern catalog to enhance prompt engineering with chatgpt, 2023.