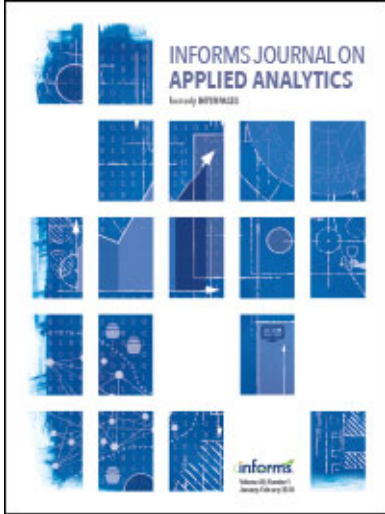




INFORMS Journal on Applied Analytics

Publication details, including instructions for authors and subscription information:
<http://pubsonline.informs.org>

Tutorial: Extracting Unstructured Text Using Large Language Models

Simon Spavound, Oliver Schaer, Panos Markou

To cite this article:

Simon Spavound, Oliver Schaer, Panos Markou (2026) Tutorial: Extracting Unstructured Text Using Large Language Models. INFORMS Journal on Applied Analytics

Published online in Articles in Advance 09 Oct 2026

. <https://doi.org/10.1287/inte.2026.0314>

This work is licensed under a Creative Commons Attribution 4.0 International License. You are free to copy, distribute, transmit and adapt this work, but you must attribute this work as “*Interfaces*. Copyright © 2026 The Author(s). <https://doi.org/10.1287/inte.2026.0314>, used under a Creative Commons Attribution License: <https://creativecommons.org/licenses/by/4.0/>.”

Copyright © 2026 The Author(s)

Please scroll down for article—it is on subsequent pages



With 12,500 members from nearly 90 countries, INFORMS is the largest international association of operations research (O.R.) and analytics professionals and students. INFORMS provides unique networking and learning opportunities for individual professionals, and organizations of all types and sizes, to better understand and use O.R. and analytics tools and methods to transform strategic visions and achieve better outcomes.

For more information on INFORMS, its publications, membership, or meetings visit <http://www.informs.org>

Tutorial: Extracting Unstructured Text Using Large Language Models

Simon Spavound,^{a,*} Oliver Schaer,^{a,*}  Panos Markou^b 

^aLeBow College of Business, Drexel University, Philadelphia, Pennsylvania 19104; ^bDarden School of Business, University of Virginia, Charlottesville, Virginia 22903

*Corresponding author

✉ ss5855@drexel.edu (S. Spavound), os355@drexel.edu (O. Schaer), Markoup@darden.virginia.edu (P. Markou)

 <https://orcid.org/0009-0002-5589-3558> (S. Spavound), <https://orcid.org/0000-0003-1878-8134> (O. Schaer),
<https://orcid.org/0000-0002-4210-281X> (P. Markou)

Received: January 19, 2026

Revised: April 9, 2026; July 28, 2026

Accepted: August 17, 2026

Published Online in Articles in Advance:
October 9, 2026

<https://doi.org/10.1287/inte.2026.0314>

Copyright: © 2026 The Author(s)

 **Open Access Statement:** This work is licensed under a Creative Commons Attribution 4.0 International License. You are free to copy, distribute, transmit and adapt this work, but you must attribute this work as “Interfaces. Copyright © 2026 The Author(s). <https://doi.org/10.1287/inte.2026.0314>, used under a Creative Commons Attribution License: <https://creativecommons.org/licenses/by/4.0/>.”

Abstract. Unstructured text data are valuable but underutilized resources in analytics due to the lack of reliable, scalable, and cost-effective extraction methods. The growing use of large language models (LLMs) is increasing the demand for clean input data from such sources, yet users lack practical guidance on incorporating these models into analytical workflows. We introduce the programmatic building blocks for working with LLMs in analytics pipelines, including API-based deployment, structured outputs, temperature control, multimodal processing, and cost-effective model selection. We then construct an end-to-end pipeline that transforms unstructured PDF documents into structured, analysis-ready JSON data. Using U.S. Food and Drug Administration Advisory Committee transcripts as a running example, we demonstrate a two-stage approach: vision-based text extraction followed by LLM-driven text structuring. Throughout, we emphasize transferable design principles, including task decomposition and parameter control. We show how the resulting structured data enables quantitative analysis that would not be feasible on the raw source documents. Using extracted speaker-statement pairs, we compute speaker-level word counts and sentiment scores, illustrating how these features can characterize stakeholder dynamics in regulatory deliberations. More broadly, the pipeline provides an extensible framework for converting unstructured documents into inputs suitable for forecasting, classification, or decision-making applications. All code is publicly available.

History: This paper was refereed.

Supplemental Material: The online appendix is available at <https://doi.org/10.1287/inte.2026.0314>.

Keywords: LLM • data cleaning • data preparation • natural language processing

Introduction

Recent advances in large language models (LLMs) expand access to natural language processing for analytics practitioners. At the same time, the volume of unstructured text data, such as verbatim meeting transcripts, online product reviews, and patient medical records, has proliferated. Converting these sources into structured, machine-readable formats is a necessary but often burdensome step in analytics pipelines. Existing extraction approaches, such as regular expressions, require extensive manual configuration and are sensitive to document variations. Common artifacts in formats like PDFs, such as inconsistent layouts, embedded headers and footers, and poor digitization quality, further complicate extraction. Consequently, traditional pipelines are difficult to scale, and extraction errors often propagate throughout the analytical workflow, undermining model validity and reproducibility.

LLMs represent a promising development, because they can process unstructured natural language inputs and generate structured, analysis-ready outputs. Despite growing interest in LLMs across various applications (Bray 2025, Li et al. 2025, Cohen et al. 2026), there is a lack of practical guidance on effectively incorporating these models into production-level analytical workflows. Recent evidence indicates that LLMs are highly sensitive to minor input variations (Mohammadi 2024, De Kok 2025), reinforcing the analytics principle that poor-quality inputs inevitably yield unreliable outputs. Although recent methodological work (Schilling-Wilhelmi et al. 2025) formalizes the use of LLMs for text extraction and transcription (Humphries et al. 2025), practitioners still require practical guidance on standardizing these outputs for quantitative analytics and ensuring rigorous validation and responsible integration of LLMs into analytics pipelines (Wasserkrug 2026).

In this tutorial, we present an end-to-end framework for transforming raw, unstructured documents (e.g., PDFs) into structured data formats (e.g., JSON) that can directly support analytical workflows and decision making. Across industries, valuable information relevant to organizational decisions is locked in unstructured text sources such as meeting transcripts (Wu 2024, De Kok 2025), open-text surveys (Bumblauskas et al. 2022), procurement documents (Li et al. 2025), maintenance reports (Wasserkrug et al. 2019), and regulatory filings (Zhu et al. 2025, Zhalechian et al. 2026). We demonstrate our framework using publicly available U.S. Food and Drug Administration (FDA) Advisory Committee transcripts. These documents are verbatim records of expert deliberations that inform regulatory decisions on drug and medical device approvals, safety evaluations, and related conditions of use. They are valuable to multiple stakeholders: regulators seeking to understand how evidence is discussed during committee review, sponsors preparing for future submissions or Advisory Committee meetings, researchers studying regulatory decision making, and members of the public seeking transparency into decisions that affect patient safety and treatment access.

From an analytics perspective, these transcripts exhibit challenges common to real-world text extraction: inconsistent formatting across transcription providers, degraded digitization in older documents, and document lengths that make manual processing impractical. To analyze these deliberations systematically, an analyst must first accurately convert the raw documents into structured data that allows for the identification of speaker turns and associations. These speaker-statement pairs then enable analysts to perform further downstream analyses such as speaker participation and sentiment analysis, as well as other natural language processing (NLP) steps that can serve as features for predicting or explaining advisory vote outcomes.

This tutorial is intended for practitioners who need to extract and analyze information from unstructured text sources as part of their decision-making workflows. We use this FDA Advisory Committee setting as a running example throughout the tutorial to show how an LLM pipeline can be deployed in practice, by first extracting the raw text, then structuring it into JSON, and finally conducting downstream analysis on the structured data, which can yield actionable insights for decision-making. The tutorial is ideal for users who have tried LLMs through chat interfaces and want to apply these models in practice. We assume some familiarity with Python and application programming interfaces (APIs), but no prior expertise with LLM APIs specifically.

We first introduce the fundamental programmatic building blocks of LLMs, their distinct multimodal capabilities, and highlight practical design choices. These choices affect the reliability and reusability of

LLM-based extraction workflows, including how to segment long documents, define structured outputs, and guide model behavior through prompts and schema. We then present a step-by-step guideline for building a comprehensive two-stage data-extraction pipeline that transforms lengthy and irregular meeting transcripts into a structured data format. Furthermore, we provide practical guidance on evaluating extraction quality against ground truth and applying the resulting data to downstream analyses. Each stage includes reproducible code designed for adoption across various document types and analytical settings. In doing so, we provide an extensible starting point for analytics practitioners seeking to incorporate LLM-based document extraction into their own analytical pipelines.¹

Tutorial and LLM Building Blocks

We begin by introducing several “building blocks” toward a programmatic approach to LLMs for analytics tasks. We first discuss why programmatic access is essential for repeatable analytical workflows and outline the main deployment options available. We then address a core practical challenge: LLMs produce variable outputs across runs, both in structure and in content, and we present techniques to enforce consistency in each. Next, we introduce the multimodal capabilities of recent LLMs, which allow them to process images and documents alongside text. Finally, we discuss model selection while managing costs, particularly when designing multistep pipelines. We bring these building blocks together in the following section, where we construct an end-to-end pipeline that transforms unstructured PDF documents into structured, analysis-ready data.

LLM Deployment Options

Most readers will be familiar with the web, desktop, or mobile interfaces from various LLM providers, such as OpenAI’s ChatGPT, Anthropic’s Claude, or Google’s Gemini. Although these interfaces are effective for one-off queries, they are not well suited for automated, repeatable workflows required in data analytics pipelines. Instead, programmatic interactions with LLMs are preferred: sending requests and receiving responses through code rather than a chat window. This lets us control model behavior more precisely, process large volumes of data automatically, and better integrate LLM outputs directly into downstream analytics workflows.

Table 1 describes and compares the main programmatic deployment options for LLMs. In this tutorial, we focus on serverless inference. These services provide access to both proprietary and open-weight models without requiring users to manage infrastructure, making them well suited to rapid prototyping and exploratory data analysis. Although usage costs can scale with

Table 1. Programmatic Deployment Options for LLMs

Feature	Serverless inference	Cloud (self-managed)	Local (on-premise)
Description	Send requests to a provider’s API (e.g., OpenAI, Anthropic); no infrastructure required	Rent GPU instances from a cloud provider (e.g., AWS, Azure) and run models independently	Run models on owned and operated physical hardware
Setup and maintenance	Accessible via API; zero infrastructure management	Requires managing cloud virtual infrastructure, model serving runtimes, and networking	Requires managing physical hardware, power/cooling, OS, and model serving runtimes
Cost model	Pay-per-token; approximately usage-based scaling	Pay for provisioned compute (e.g., GPU instances, storage)	Upfront hardware cost plus operating costs
Data privacy	Sent to third party; relies on vendor privacy terms	Isolated within user-controlled cloud infrastructure	Fully isolated on owned physical infrastructure
Scalability	Automatically managed by the LLM provider	Scales by provisioning additional cloud resources	Limited by physical hardware constraints
Model access	Proprietary frontier models and open-weight models	Open-weight models for self-hosted instances	Open-weight models
Use cases	Rapid prototyping and access to frontier proprietary models	Deployments requiring data control and scaling	Workloads requiring full infrastructure and data control

volume, they offer a low barrier to entry and minimal setup. Importantly, the interaction patterns for calling models are largely consistent across deployment options. The primary differences lie in how models are hosted and accessed. Cloud-based or on-premise deployments may be preferable when greater control over infrastructure, performance, and long-term costs are more salient. Moreover, serverless and some cloud-based deployment modes require transmitting document content to a third-party provider, which may be inappropriate or require additional safeguards. In such settings, organizations may prefer self-managed cloud or local deployments that provide greater control over data access and storage. Therefore, analysts must assess whether source documents contain personally identifiable information (PII), protected health information (PHI), confidential business information, or other regulated data when selecting deployment options.

For demonstration purposes, we use OpenAI GPT models in a serverless deployment throughout the tutorial. Nevertheless, the principles presented in this tutorial are broadly applicable across LLM providers, and we do not attempt to identify a single “best” model. Accordingly, specific model names, API calls, and parameter settings should be viewed as illustrative implementations of the broader workflow rather than prescriptive recommendations. Instead, we focus on the practical challenges of managing LLM output in analytical workflows, and we provide guidance on enforcing consistency and decomposing tasks into cost-effective, performant pipelines.

Challenges to Consistent Outputs

By default, LLMs often generate varying outputs, even when provided with identical instructions (i.e., “prompts”).

Achieving desired outcomes with LLMs is therefore highly prompt dependent, and outputs can remain sensitive to minor prompt changes (Mohammadi 2024).² This stochastic nature of LLMs produces two forms of instability in their outputs: one in the *structure* of the output and a second in the *content* of the output. One can provide the same prompt multiple times, and the output will often vary substantially, as shown in Figure 1. Even though the answers to the question on the FDA’s key purposes are “similar,” the outputs differ in structure, with varied use of bullet points and section numbers, and in content, in terms of the key purposes and provided explanations.

This poses two main challenges for downstream data analysis. Variations in structure are problematic because most tools and workflows require inputs in a consistent, predefined format, that is, consistent variables, rows, and data types. When the output is organized differently across runs (such as switching between bullet points, numbered lists, or prose), downstream applications might struggle (or even fail) to read in the data. As shown in Figure 1, the first generation presents (most) explanations as prose paragraphs, whereas the third generation organizes them as bullet points under each heading, requiring entirely different parsing logic to extract the same information. Variations in content are also problematic, because they make it difficult to distinguish whether differences in later analysis results stem from the underlying data or from the LLM generating different responses across runs. In Figure 1, the first generation identifies six key purposes of the FDA, whereas the third generation identifies seven with different groupings, meaning any attempt to systematically count or compare purposes across runs would yield inconsistent results.

Downloaded from informs.org by [216.73.216.37] on 09 October 2026, at 16:06 . For personal use only, all rights reserved.



readable consistency. Hence, to address this limitation, LLM providers have introduced native support for structured outputs.³ Rather than relying on the model to *follow* formatting instructions, these approaches *enforce* structure by validating the outputs against a predefined (user-specified) schema. In practice, this typically takes the form of JSON data objects that must conform exactly to a specified schema. JSON is particularly well suited for this purpose: It is both human and machine readable; supports hierarchical, nested data beyond tabular formats such as CSV; integrates

naturally into LLM-based pipelines; and can be imported into analytics software.

Validated structured output is not accessible through the familiar consumer-friendly “chat” interfaces, but only through APIs. These have an initial learning curve but provide two primary advantages for our context. First, APIs enable automated, high-throughput processing of large volumes of data (for example, by looping over multiple prompts) and allow outputs to be integrated directly into downstream data processing pipelines. Second, APIs provide explicit control over model versions and parameters, which is critical for reproducibility. Consumer-facing interfaces may change the underlying model at the LLM provider’s discretion, meaning results can vary between sessions for reasons outside the user’s control. By specifying a fixed model snapshot, we can promote more consistent model behavior over time.

To access the API, we utilize OpenAI’s Python API library. Although this library is designed for OpenAI services, it is also widely accepted as an interface for other commercial providers. We note that these interfaces are currently constantly evolving, and we aim to provide code that, with some modest adaptation, can be used across any of the major LLM providers. As with any other package we discuss in this tutorial, it can be installed via the Python package manager: `pip install openai`. Accessing models this way requires a valid API key and a paid account.⁴ In Figure 2, on

Line 7, we include the same prompt as previously used in Figure 1. Executing Lines 5–8 sends the request to the LLM provider for the specified model snapshot `gpt-5.4-2026-03-05` and stores it in the object `response`, with the generated output printed on Lines 11–22. Note that in Figure 2 the LLM’s output contains many markdown characters (e.g., “`***`” for bold) that will vary each time the model is run, as seen in Figure 1.

Now that the API is sorted, we can turn to solving the structure variation problem. In Figure 3, we first define a desired structure of the output using the Pydantic package (Colvin et al. 2025). Pydantic lets users define a schema using Python classes that act as templates describing what each piece of data should look like. Using Pydantic with LLMs lets us describe required field entries in natural language, which the LLM can then interpret and populate. In this case, we require on Lines 5–11 in Figure 3 that each entry must contain two fields: a short purpose and a corresponding explanation. We also include descriptions for these fields that serve as additional instructions for the model on what each value should contain. We define the overall output structure on Lines 14–17 as a list of these entries. This ensures that the model returns a consistent, machine-readable format. Finally, we modify the prompt to be explicit in how many key purposes we require on Line 22. However, note that this does not guarantee strict adherence to this constraint, because it is not restricted in the Pydantic schema.

Figure 2. (Color online) API Call for the Same Prompt as in Figure 1

```
1 from openai import OpenAI
2
3 client = OpenAI(api_key="your_api_key_here")
4
5 response = client.responses.create(
6     model="gpt-5.4-2026-03-05",
7     input="What are the FDA's key purposes?"
8 )
9
10 print(response.output_text)
11 """
12 The FDA's key purposes are to protect and promote public health. In practice, that
13     ↪ means it:
14
15 1. Ensures safety and effectiveness of:
16     - Human and veterinary drugs
17     - Vaccines and other biologics
18     - Medical devices
19
20 2. Helps ensure the safety of food supply (except most meat, poultry, and some egg
21     ↪ products, which are mainly overseen by USDA).
22
23 [... lines omitted for brevity ...]
24 """
```

Figure 3. (Color online) Adding a Structured Format to the Outputs We Request

```

1  from pydantic import BaseModel, Field
2  from typing import List
3
4  # Define schema for each row
5  class FDAPurposeItem(BaseModel):
6      purpose: str = Field(
7          description="A short phrase summarizing one core purpose of the FDA."
8      )
9      explanation: str = Field(
10         description="A single concise sentence explaining the purpose."
11     )
12
13 # Define overall response schema
14 class FDAPurposes(BaseModel):
15     items: List[FDAPurposeItem] = Field(
16         description="A list of FDA purposes, each with a short label and explanation."
17     )
18
19 # Call API with structured parsing
20 response = client.responses.parse(
21     model="gpt-5.4-2026-03-05",
22     input="What are the FDA's 3 key purposes?",
23     text_format=FDAPurposes
24 )

```

To move from unstructured text generation to structured data outputs, OpenAI's API requires us to replace the `.create` call with `.parse` (Line 20). This allows us to pass the schema directly via the `text_format` argument, enforcing adherence to the specified structure and returning validated outputs. We also use the `typing` package to specify expected data types (e.g., a list of items), enabling Pydantic to validate that the model's response matches the defined schema.

We have now effectively dealt with the issue of structure, as seen from the outputs in Figures 4 and 5. Both prompt generations produced a valid JSON object that precisely adhered to our requested schema, containing three purposes and an explanation. However, across the two sequential runs of the code in Figure 3, we still observe content variations.

Limiting Variations in Content

If the goal is to achieve consistency in outputs, content variability can be explicitly controlled. LLMs generate text stochastically by sampling from a probability distribution over possible next words. Most APIs offer users the ability to adjust a "temperature" parameter. This controls the shape of the distribution, affecting the likelihood of selecting both high- and low-probability words. Lowering the temperature sharpens the distribution, increasing the relative likelihood of high-probability words and reducing variation in the generated output. For example, reducing the temperature from the typical default of around 0.7 to 0.0 produces more deterministic

responses. Conversely, increasing the temperature to 2.0 flattens the distribution, leading to more diverse and variable outputs. Beyond temperature, the available parameters to control content variability differ across providers and model families and continue to evolve. It is important to consult the documentation for a chosen model and view the specific parameter settings used throughout this tutorial as illustrative implementations of this broader principle rather than prescriptive recommendations.

To demonstrate this control, Line 4 in Figure 6 modifies Figure 3 by adding the temperature argument to our request and setting it to 0.0. Figures 7 and 8 display the generated outputs. We see that they continue to follow the output structure we requested, but we now also achieve less variation in content. Both runs generate identical purposes, and the associated explanations exhibit substantially fewer variations compared with the previous examples in Figures 4 and 5.

Modifying the temperature parameter is often assumed to make an LLM deterministic. However, as our example demonstrates, this is not strictly true (Atil et al. 2025). Although the text of the first key purpose is identical across Figures 7 and 8, the second and third purposes still contain variations in content. Although research suggests that lowering the temperature has no impact on applications such as problem solving (Renze 2024), it improves consistency in other tasks, such as code generation (Ouyang et al. 2025). For verbatim text extraction, we observe improvements in consistency

Figure 4. (Color online) Generation 1 of Code in Figure 3

```

1 {
2   "items": [
3     {
4       "purpose": "Protect public health",
5       "explanation": "The FDA helps
        ↳ ensure that foods, drugs, vaccines,
        ↳ medical devices, and other regulated
        ↳ products are safe and effective for
        ↳ consumers."
6     },
7     {
8       "purpose": "Advance public health",
9       "explanation": "The FDA supports
        ↳ innovation and improves access to
        ↳ useful medical products by guiding
        ↳ development and reviewing products
        ↳ based on science."
10    },
11    {
12      "purpose": "Provide accurate
        ↳ information",
13      "explanation": "The FDA gives the
        ↳ public evidence-based information so
        ↳ people can use foods, medicines, and
        ↳ other health-related products safely
        ↳ ."
14    }
15  ]
16 }
```

Figure 5. (Color online) Generation 2 of Code in Figure 3

```

1 {
2   "items": [
3     {
4       "purpose": "Protect public health",
5       "explanation": "The FDA helps
        ↳ ensure that foods, drugs, vaccines,
        ↳ medical devices, and other regulated
        ↳ products are safe and effective for
        ↳ the public."
6     },
7     {
8       "purpose": "Regulate product
        ↳ quality and labeling",
9       "explanation": "The FDA sets and
        ↳ enforces standards for manufacturing,
        ↳ marketing, and labeling so consumers
        ↳ receive accurate information and
        ↳ reliable products."
10    },
11    {
12      "purpose": "Advance innovation and
        ↳ informed use",
13      "explanation": "The FDA supports
        ↳ medical and scientific innovation
        ↳ while providing guidance that helps
        ↳ healthcare professionals and
        ↳ consumers use products appropriately
        ↳ ."
14    }
15  ]
16 }
```

(see Online Appendix). Accordingly, we recommend using the strongest available mechanism for reducing output variability, such as setting the temperature to zero when supported by the selected model.⁵

Leveraging Multimodal LLM Capabilities

Early LLMs were designed to process text exclusively, accepting natural language input and producing natural language output. Since GPT-4, “multimodal” LLMs can also process images, audio, and video while still being prompted in natural language. This is particularly relevant for text extraction tasks: Rather than relying on specialized text recognition software, a multimodal LLM can directly *interpret* what a document page looks like and extract its content in a single

step. Because these LLMs interpret a page’s visual layout and meaning rather than simply recognizing individual characters, they can handle complex document formats with contextual understanding. In some Visual Question Answering benchmarks, multimodal LLMs have been shown to achieve near-human performance (Fu et al. 2025).

To illustrate, we pass an image of a PowerPoint slide from an FDA Advisory Committee Meeting (Figure 9) into gpt-5.4. Because APIs are designed for text transmission, images must first be encoded into a compatible format. Here, we use Base64 encoding, as shown in Figure 10 on Lines 7–8.⁶ This converts the image into a text string that can be directly included in the input argument, as in Line 34.

Figure 6. (Color online) Adding a Temperature Control to Our Structured Prompt Request

```

1 response = client.responses.parse(
2   model="gpt-5.4-2026-03-05",
3   input="What are the FDA's 3 key purposes?",
4   temperature=0.0,
5   text_format=FDAPurposes
6 )
```

In contrast to our previous prompt in Figure 3, the input now contains multiple items. On Line 29 of Figure 10, we define the role as *user*, which specifies the source of the prompt or input. In practice, two roles are commonly used: *user* for task-specific inputs and *system* for instructions governing overall behavior. Because we include both text and images, the input must be structured into typed components *input_text* and *input_image*, respectively. On Line 32, we prompt the model to “Extract and identify the descriptive headings from this slide.” In addition, we update the schema definition (Lines 11–23) to extract headings and structural elements.

The output is now a structured JSON that includes the extracted headline and type identification. For example, as shown in Figure 11, the LLM correctly identified that “Number of Patients” is the “*y*-axis label” of the graph. This simplifies integrating otherwise complex image-processing capabilities into our pipeline. Natural language instructions provide fine control over the extraction process. For example, we could easily rewrite the prompt to only extract the *y*-axis or, vice versa, to ignore it. This simplicity in interacting with multimodal data with essentially the same interface is

another benefit of using LLMs in data analytics pipelines compared with task-specific algorithms.

Cost and Model Considerations

To this point, we have used *gpt-5.4*. However, providers offer a variety of models. These differ in their capabilities and costs, making model selection an important practical consideration when designing a data pipeline. For commercial LLMs accessed via API, costs are determined by the number of “tokens” processed in both the input and output. Rather than operating directly on raw text, LLMs convert text into tokens that correspond to whole words, subword fragments, or punctuation.⁷ Through tokenization, text is converted into a sequence of numbers that the model can process. These tokens serve as the basis for both pricing and technical constraints, such as the model’s context window, that is, the maximum number of tokens it can handle at once.

For example, the image processing task in Figure 10 consumed 1,030 input tokens and 76 output tokens, resulting in a cost of less than \$0.01. Although this is negligible for a single request, costs can scale rapidly for more complex tasks or high-volume workflows. At the time of writing, *gpt-5.4* is priced at \$2.50 per million

Figure 7. (Color online) Generation 1 Using Code in Figure 6

```

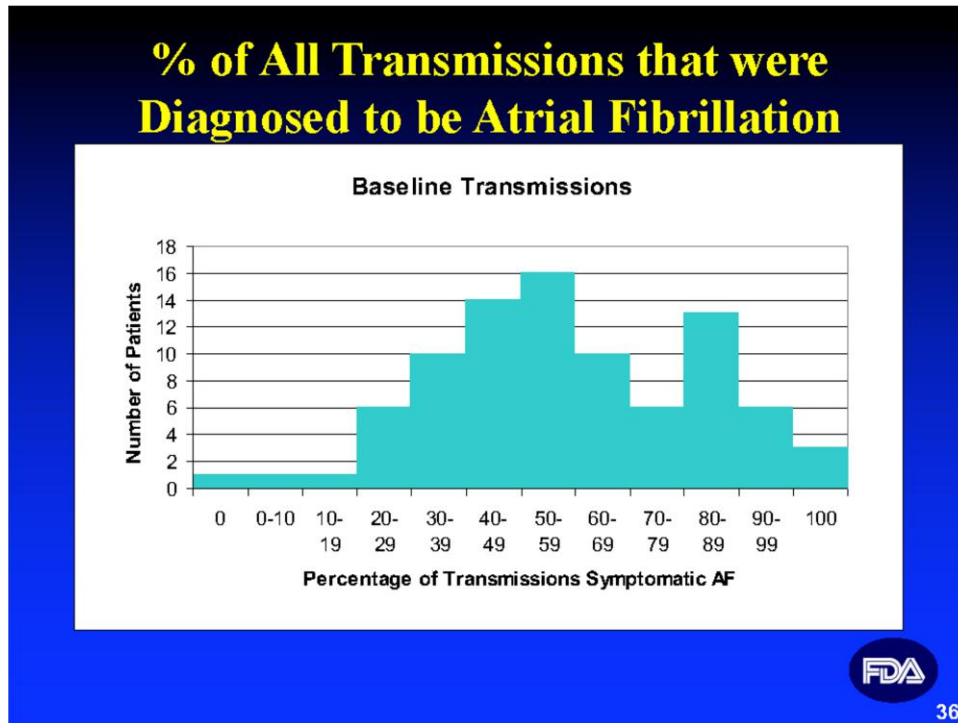
1 {
2   "items": [
3     {
4       "purpose": "Protect public health",
5       "explanation": "The FDA helps
        ↳ ensure that foods, drugs, medical
        ↳ devices, and other regulated products
        ↳ are safe and effective for the
        ↳ public."
6     },
7     {
8       "purpose": "Advance public health",
9       "explanation": "The FDA supports
        ↳ innovation and improves access to
        ↳ safe medical products by helping
        ↳ speed the development and review of
        ↳ new treatments and technologies."
10    },
11    {
12      "purpose": "Provide accurate
        ↳ information",
13      "explanation": "The FDA gives the
        ↳ public science-based information so
        ↳ people can use medicines, foods, and
        ↳ other health-related products safely
        ↳ and effectively."
14    }
15  ]
16 }
```

Figure 8. (Color online) Generation 2 Using Code in Figure 6

```

1 {
2   "items": [
3     {
4       "purpose": "Protect public health",
5       "explanation": "The FDA helps
        ↳ ensure that foods, drugs, medical
        ↳ devices, and other regulated products
        ↳ are safe and effective for the
        ↳ public."
6     },
7     {
8       "purpose": "Advance public health",
9       "explanation": "The FDA supports
        ↳ innovation and improves access to
        ↳ safe and effective medical products
        ↳ through science-based regulation."
10    },
11    {
12      "purpose": "Provide accurate
        ↳ information",
13      "explanation": "The FDA gives the
        ↳ public evidence-based information so
        ↳ people can use medicines, foods, and
        ↳ other products safely."
14    }
15  ]
16 }
```

Figure 9. (Color online) Sample Slide from the December 13, 2000, FDA Advisory Committee Meeting Illustrates the Mixture of Images and Text That Multimodal LLMs Can Process



input tokens and \$15 per million output tokens, with a maximum input (the context window) of up to 1 million tokens, and a maximum output of up to 128,000 tokens.⁸ This asymmetry in pricing leads to an important practical consideration for analytical tasks. Different models not only vary in absolute cost, but also in the relative cost of input versus output tokens. Tasks such as summarization or classification typically involve large inputs and minimal outputs, whereas generative tasks (e.g., report or image generation) often have relatively small inputs but produce substantially larger outputs. As a result, cost-efficient system design should account for whether a task is input-heavy or output-heavy, and optimize accordingly.

In practice, this often means investing additional effort in prompt design or preprocessing inputs to minimize token usage. If a task requires more context than a model can accommodate, it may be necessary to use a larger model with a longer context window, even if increased reasoning capability is not required. Output limits present a different challenge, as the required response length is often difficult to determine in advance.

Decomposing tasks into smaller, more manageable subtasks can improve cost-effectiveness by enabling the use of lower-cost models and mitigating constraints on context window and output length. For example, `gpt-4.1` is approximately 20% and 45% cheaper than `gpt-5.4` for input and output tokens,

respectively, but it has a shorter maximum output of 32,000 tokens, which can constrain certain tasks. Even highly capable models may underperform when required to process large, complex inputs in a single pass, as relevant information can be diluted within long contexts (Liu et al. 2024). As a result, breaking problems into targeted subtasks can improve performance, making model inference both cheaper and more scalable (Belcak et al. 2025). For example, for a summarization task, we can split an original document with 100,000 tokens into five chunks of 20,000 tokens each. This encourages a shift away from relying exclusively on the most capable models, toward designing pipelines that allocate subtasks to appropriately sized models. This strategy of decomposing complex objectives into sequences of smaller model calls is also commonly employed in agentic AI systems.

Finally, for analytics workflows where real-time response is not a primary requirement but requests often need to be made in large numbers, batch processing is also available. Batch processing groups multiple requests into a single job, which is executed asynchronously and in parallel rather than as individual, sequential calls. Most major commercial providers now support batch functionality, offering discounts of around 50% on the same request. However, this approach introduces additional setup complexity and longer turnaround times, with jobs taking up to 24 hours to complete.⁹

Figure 10. (Color online) Prompting GPT to Analyze an Image

```

1  import base64
2  from pathlib import Path
3
4  # Path to your image
5  image_path = Path("3954t1-FDA_Page_37.png")
6
7  with open(image_path, "rb") as f:
8      base64_image = base64.b64encode(f.read()).decode()
9
10 # Define schema for each row
11 class GraphItem(BaseModel):
12     label: str = Field(
13         description="The exact text of the headings as it appears in the graph."
14     )
15     type: str = Field(
16         description="Identifier of the structural element this text represents"
17     )
18
19 # Define overall response schema
20 class GraphLabels(BaseModel):
21     items: List[GraphItem] = Field(
22         description="A list of descriptive headings extracted from the graph."
23     )
24
25 # Call API with structured parsing
26 response = client.responses.parse(
27     model="gpt-5.4-2026-03-05",
28     input=[{
29         "role": "user",
30         "content": [
31             {"type": "input_text",
32              "text": "Extract and identify the descriptive headings from this slide"},
33             {"type": "input_image",
34              "image_url": f"data:image/jpeg;base64,{base64_image}"
35             }
36         ]
37     },
38     text_format=GraphLabels

```

Building a Pipeline for Obtaining Structured Data from Unstructured Text

We now turn to applying the above building blocks and LLM techniques to construct an end-to-end pipeline that transforms unstructured text documents into structured data suitable for downstream analysis. The pipeline, depicted in Figure 12, consists of two main stages. Stage I converts each page of a PDF document into an image and uses a multimodal LLM to extract the raw text, one page at a time. The per-page outputs are then concatenated into a single text file. Stage II divides this text into manageable chunks and passes each through an LLM with a defined data schema, producing speaker-statement pairs in a structured JSON data file.

Several deliberate design choices improve the pipeline's performance. These design choices are independent of any specific LLM provider or API and illustrate

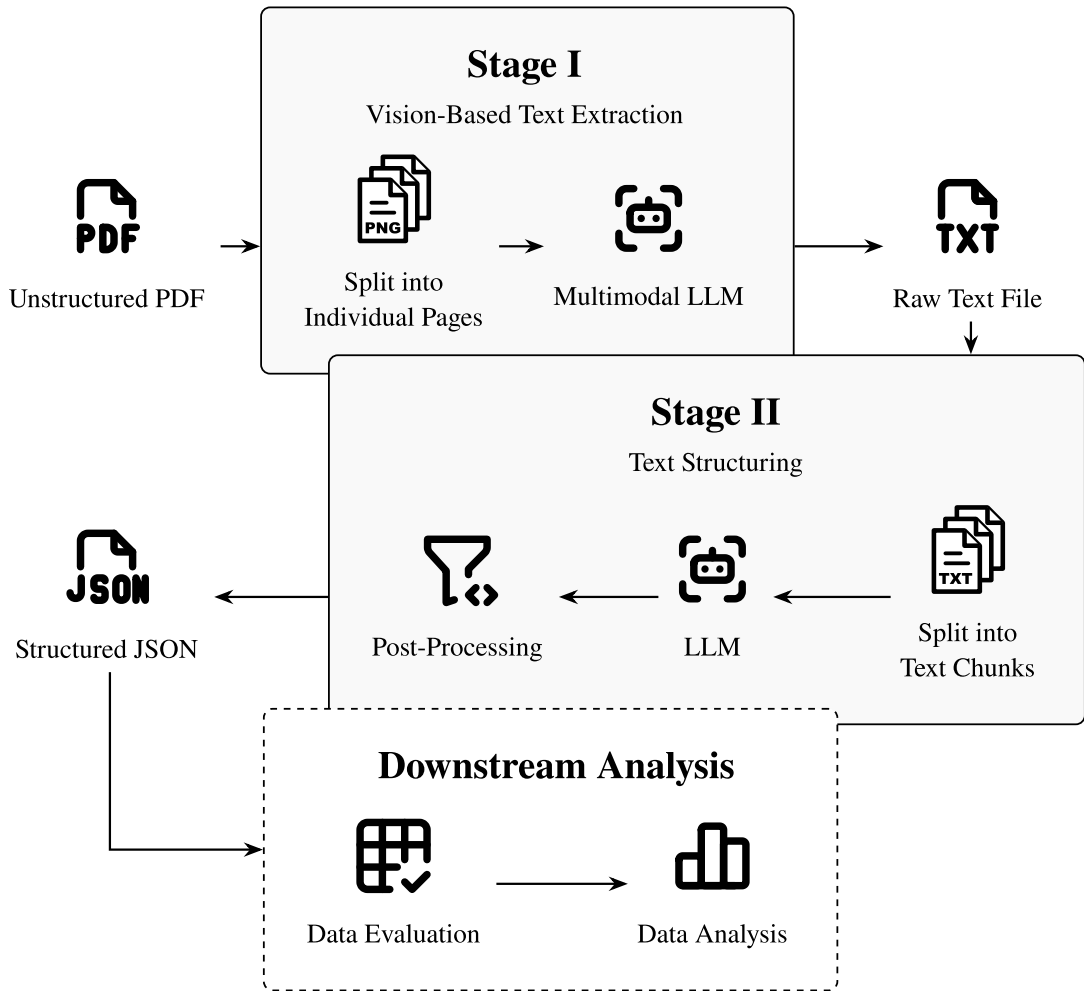
transferable principles for building reliable analytics pipelines. Specifically, separating text extraction (Stage I) from data structuring (Stage II) allows each stage to focus on a single task, thereby improving output accuracy. Processing documents in smaller pieces (individual PDF pages/images in Stage I, and chunks of text in Stage II) enables the use of lower-cost models while staying within context window limits. Deferring edge cases and “clean up” to a deterministic postprocessing step avoids overcomplicating the LLM prompts and minimizes the chance of further errors.

Such a modular, LLM-based processing pipeline offers several key advantages. First, it is highly flexible with respect to input format, accommodating documents with or without embedded text layers, as well as scanned or photographed documents. Second, by decomposing documents into smaller chunks, the pipeline is inherently

Figure 11. Structured Output from Image Analysis

```
1 {
2   "items": [
3     {
4       "label": "% of All Transmissions that were Diagnosed to be Atrial Fibrillation",
5       "type": "title"
6     },
7     {
8       "label": "Baseline Transmissions",
9       "type": "subtitle"
10    },
11    {
12      "label": "Number of Patients",
13      "type": "y-axis label"
14    },
15    {
16      "label": "Percentage of Transmissions Symptomatic AF",
17      "type": "x-axis label"
18    }
19  ]
20 }
```

Figure 12. Extraction Pipeline Converts Unstructured PDF Documents into Structured JSON Through a Two-Stage Process: Vision-Based Text Extraction Using a Multimodal LLM Followed by LLM-Based Text Structuring



scalable, enabling efficient processing of long and complex documents. This chunking strategy also improves accuracy, as models operate on more focused inputs, and it enhances cost-effectiveness by allowing the use of smaller, less expensive models. Third, prompting in natural language further increases flexibility, allowing the LLM to generalize across diverse document formats without requiring task-specific rules. Additionally, the pipeline is easily extendable through postprocessing steps, where deterministic or NLP-based methods can refine outputs further. Finally, by structuring results in JSON, the pipeline produces a standardized, machine-readable format that seamlessly supports a wide range of downstream analytical tasks.

To ground the demonstration, we use verbatim transcripts from U.S. FDA Advisory Committee meetings, which are publicly available. Advisory Committee deliberations directly inform FDA regulatory decisions on drug approvals and safety evaluations, carrying significant implications for public health policy. Accurate records of these discussions are therefore valuable to researchers, regulators, and drug and medical device sponsors alike. However, these PDF transcripts pose several challenges common to real-world text extraction tasks. Their layouts and digitization quality vary considerably over time, with differences in headers, footers,

and line-numbering formats across transcription providers (see Figure 13, (a) and (b), for extracts from two documents). These differences make it difficult to develop a single rule-based extraction approach. Moreover, their document length (often 200+ pages) renders cost-effective processing challenging.

Stage I: Vision-Based Text Extraction

Stage I of the pipeline first converts a PDF into a series of high-resolution PNG images, before turning them over to a multimodal LLM to extract the text. In Figure 14, Lines 6–9 convert each page of a PDF into a separate PNG image using the `convert_from_path` function from the `pdf2image` library. This function accesses the PDF, extracts each page as a PNG image, and saves each image to a specified output folder. An example image is then encoded in Lines 12–14 using the same `Base64` function as in Figure 10. This encoding converts the image into a text string that can be included directly in the API request (in Line 37), allowing the page image, prompt, and model settings to be passed together in a single, self-contained call. (For simplicity, the code in Figure 14 focuses on the single page shown in Figure 13(a); however, the same procedure applies to all pages via simple iteration.)

Figure 13. Sample Pages Illustrating Variation in Formatting and Structure Across the FDA Advisory Committee Documents

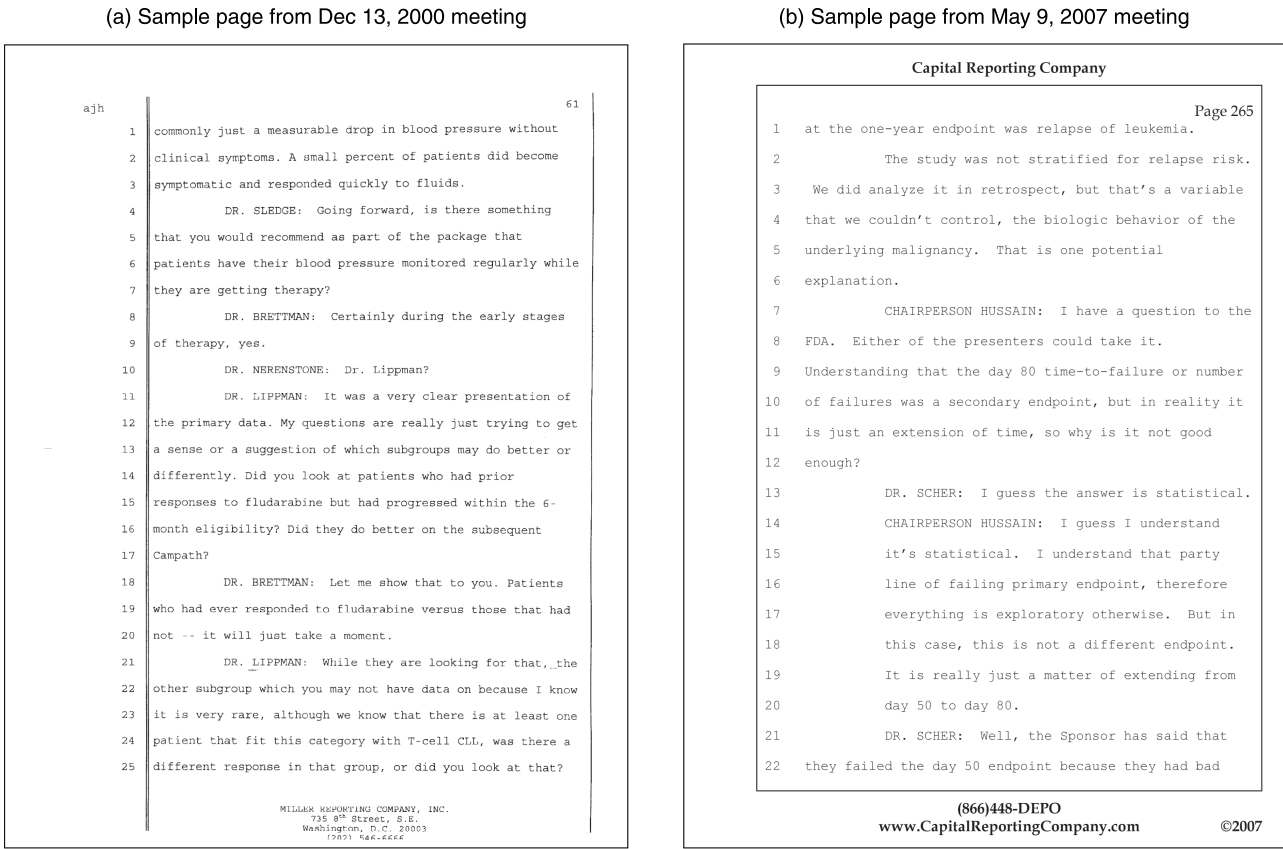


Figure 14. (Color online) Code to Convert Image and Prompt LLM to Extract Raw Text

```
1 from pdf2image import convert_from_path
2 from pathlib import Path
3 from openai import OpenAI
4 import base64
5
6 pdf = Path("example.pdf") # input PDF
7 out = Path("example") # output folder
8 out.mkdir(parents=True, exist_ok=True) # create folder if it does not exist
9 images = convert_from_path(pdf, output_folder=out, fmt="png", output_file=pdf.stem)
10
11 # We load and encode one single page into base64
12 image_path = Path("example/example0001-061.png")
13 with open(image_path, "rb") as f:
14     base64_image = base64.b64encode(f.read()).decode()
15
16 # Initialize OpenAI client
17 client = OpenAI(api_key="your_api_key_here")
18
19 # Image extraction prompt
20 prompt = """You are a professional OCR engine. Transcribe the text EXACTLY as written.
21 Do not correct spelling or grammar errors.
22 If text is blurry use context to infer the word.
23 Ignore headers footers as well as line and page numbers.
24 """
25
26 # Image extraction call
27 response = client.responses.create(
28     model="gpt-4.1-mini-2025-04-14",
29     temperature = 0.0,
30     input=[
31         {
32             "role": "user",
33             "content": [
34                 { "type": "input_text", "text": prompt},
35                 {
36                     "type": "input_image",
37                     "image_url": f"data:image/jpeg;base64,{base64_image}",
38                 },
39             ],
40         },
41     ],
42 )
43
44 # store output for further processing
45 raw_text = response.output_text
```

Next, each encoded page image is passed to the LLM along with a short extraction prompt specifying exactly how the text should be transcribed. The user prompt on Lines 20–24 defines the model’s extraction task at the page level. The prompt includes several instructions. It instructs the model to behave like an optical character recognition (OCR) engine and transcribe the text exactly as written, making verbatim fidelity the central objective. It then reinforces this objective by explicitly prohibiting spelling and grammar correction, because even “helpful” edits would reduce fidelity to the source document. A

further instruction specifies how the model should behave when the page quality is degraded: rather than omitting uncertain text or introducing arbitrary noise, it should use the surrounding visual and textual context to infer the most likely word. Finally, the model is instructed to ignore headers, footers, and line or page numbers, allowing its visual attention to focus on the substantive transcript content rather than recurring layout artifacts.

LLMs allow us to define this extraction behavior directly in natural language. Rather than programmatically attempting to specify where footers occur on the

page, the LLM is left to “reason” about what a footer is and to deal with it accordingly. Depending on the use case, the same basic prompt on Lines 20–24 can be easily adapted to preserve headers, include annotations, or target other document types, without requiring document-specific, rule-based preprocessing.

Rather than combining visual transcription with additional formatting or structuring instructions, we keep the task at the level of page-by-page raw text extraction. This improves the reliability of the vision-based extraction step and, by reducing the problem to a single page and task at a time, allows the user to leverage a smaller multimodal model such as gpt-4.1-mini. Compared with gpt-5.4, this reduces costs by approximately 84% and 89% in input and output tokens, respectively. We execute this prompt through the client.responses.create function for each PNG image, with the temperature set to 0.0 to minimize variability as shown on Line 29.

The resulting output is high-quality, raw text, which we store as raw_text on Line 45. To see this, we can compare Figures 15 and 16. Figure 15 demonstrates the text obtained by directly extracting the PDF’s underlying embedded text layer—that is, the text one might obtain by programmatically reading the PDF rather than analyzing the page visually with an LLM. In poorly digitized documents, as in our example, this embedded text layer can be degraded: visual characters may not match their underlying encodings, words may be fragmented across arbitrary line breaks, and layout artifacts such as line numbers or marginal text may be mixed into the extracted content.

In contrast, Figure 16 demonstrates that the LLM-based approach successfully follows our instructions and recovers the substantive transcript text from the rendered page itself. In particular, it has successfully removed nonessential elements such as line numbers, page numbers, and idiosyncratic marginalia (e.g., “ajh”) while preserving domain-specific terminology (e.g., “T-cell CLL”), brand names (e.g., “Campath”), and proper names. The extracted text closely matches the original document while reducing token volume by approximately 30%, thereby lowering the input costs for Stage II (see the Online Appendix for more details on the costs to execute the pipeline).

These results are achieved without the need for document-specific, rule-based preprocessing pipelines, which would otherwise require substantial development effort and often fail to generalize across different page layouts. This flexibility allows a much wider range of unstructured documents to be processed effectively, improving the quality of data available for analysis. This vision-based approach is more expensive than processing the PDF’s built-in text layer directly. However, in our experience, the cleaner and more consistent text it produces more than compensates for these extra costs

Figure 15. (Color online) Native PDF Text Layer

```
1  ajh      # [ <-- Probable initials ]
2  1        # [ <-- Line numbers in PDF ]
3  2
4  3
5  4
6  5
7  6
8  [... lines omitted for brevity ...]
9  13
10 14
11 15
12 16
13 17
14 18
15 19
16 20
17 21
18 22
19 23
20 24
21 25
22 61      # [ <-- Page Number ]
23 commonly
24 just
25 a measurable
26 drop
27 in
28 blood
29 pressure
30 [... lines omitted for brevity ...]
31 you
32 look
33 at
34 that?
35 MILLER # [ <-- Transcriber address ]
36 REPORTING
37 COMPANY,
38 INC.
39 735
40 gfh Street, # [ <-- Unknown word ]
41 S.E.
42 Washington,
43 D.C.
44 20003
45 13n3\ # [ <-- Unknown unicode ]
46 c;dC-CCCC # [ <-- Extra characters ]
```

in downstream analytical tasks, as illustrated by Figures 15 and 16.

Stage II: Text Structuring

The cleaned raw text from Stage I forms the input to Stage II. This second stage will transform the text into a structured JSON for downstream analysis. This requires two considerations.

The first consideration is to define the target data structure. For meeting transcripts such as in our case, a natural representation is to store the dialogue as a

Figure 16. (Color online) LLM Vision-Based Text Extraction

```

1  commonly just a measurable drop in blood
   ↳ pressure without clinical symptoms. A
   ↳ small percent of patients did become
   ↳ symptomatic and responded quickly to
   ↳ fluids.
2
3  DR. SLEDGE:   Going forward, is there
   ↳ something that you would recommend as
   ↳ part of the package that patients
   ↳ have their blood pressure monitored
   ↳ regularly while they are getting
   ↳ therapy?
4
5  DR. BRETTMAN: Certainly during the
   ↳ early stages of therapy, yes.
6
7  DR. NERENSTONE: Dr. Lippman?
8
9  DR. LIPPMAN:   It was a very clear
   ↳ presentation of the primary data. My
   ↳ questions are really just trying to
   ↳ get a sense or a suggestion of which
   ↳ subgroups may do better or
   ↳ differently. Did you look at patients
   ↳ who had prior responses to
   ↳ fludarabine but had progressed within
   ↳ the 6-month eligibility? Did they do
   ↳ better on the subsequent Campath?
10
11 DR. BRETTMAN:  Let me show that to you.
   ↳ Patients who had ever responded to
   ↳ fludarabine versus those that had not
   ↳ -- it will just take a moment.
12
13 DR. LIPPMAN:   While they are looking for
   ↳ that, the other subgroup which you
   ↳ may not have data on because I know
   ↳ it is very rare, although we know
   ↳ that there is at least one patient
   ↳ that fit this category with T-cell
   ↳ CLL, was there a different response
   ↳ in that group, or did you look at
   ↳ that?
    
```

sequence of speaker-statement entries: Each entry contains one field identifying the speaker and a second field containing that speaker's spoken statement. This structure mirrors the way transcripts are organized in the source documents and converts the dialogue into a machine-readable format that can be validated, filtered, and analyzed programmatically. In our case, this representation is useful because it allows us to isolate individual speakers, compare contributions across participants, and prepare the transcript for subsequent quantitative analysis. This approach is not limited to this format, however; the same pipeline can be adapted

to other output structures, provided that the desired schema can be specified in Pydantic.

Figure 17 presents the Pydantic schema adapted for our use case involving meeting transcripts. The schema defines an array of entries, each with two required properties: speaker and statement. As with the earlier demonstration, the schema requires us to specify the desired entry (Line 5) and the overall response structure (Line 16). Beyond defining data types, the schema serves as a secondary layer of instructions through the description field. Specifically, we restate that the speaker property must be in upper-case, reiterate the naming rule for the unidentified speaker turn (Line 9), and that the statement property must capture the text verbatim while normalizing whitespace (Line 12). Finally, to strictly enforce schema adherence without adding additional fields, we include an extra `model_config` argument on Lines 6 and 17.

The second consideration is the original document size. In our example, the PDF transcripts often exceed 200 pages. Passing the text from an entire document in a single request would require input and output exceeding 100,000 tokens, necessitating using models with large context windows, which are typically significantly more expensive. Moreover, prior work suggests that model performance can degrade when processing very long inputs (Liu et al. 2024). To address this, we will split the text into smaller units, or “chunks,” akin to what we did for the pages in Stage I. Chunking reduces the token requirements per request, allowing users to use smaller, more cost-effective models, while also improving the reliability of structured outputs.

However, chunking introduces a tradeoff: Smaller chunks improve cost and reliability, but each request must resend the prompt, which counts toward the input token limit. This overhead is a greater concern here, because the structuring prompt in Stage II is substantially longer than in Stage I (i.e., 50 versus 230 tokens), as we will see shortly. For example, on a 350-page transcript (~87,500 tokens), page-by-page parsing would add approximately 80,500 tokens in prompt overhead alone, effectively doubling our overall input token costs. For this demonstration, we provide code to process a single chunk containing the text fragment extracted in Stage I and shown in Figure 16. However, the accompanying GitHub repository includes a version that uses a chunk size of 5,000 tokens to process the entire transcript iteratively. Each text chunk is then passed to the LLM, and the outputs are combined to generate a single structured JSON.

Our system prompt on Line 2 in Figure 18 is zero-shot, meaning that the prompt includes task instructions but no examples. This approach is appropriate for the present use case as verbatim text extraction requires relatively limited interpretation, and the zero-shot configuration performed well in our evaluation. The

Figure 17. (Color online) Structured Format for Returning Speaker-Statement Pairs from the Raw Text

```

1 from pydantic import ConfigDict, BaseModel, Field
2 from typing import List
3
4 # Pydantic class for speaker-statement pairs
5 class TranscriptEntry(BaseModel):
6     model_config = ConfigDict(extra="forbid")
7
8     speaker: str = Field(
9         description="The speaker's name in UPPERCASE. Use 'UNKNOWN' if the text starts
10         ↳ without a name. Do not extract names from headers or rosters."
11     )
12     statement: str = Field(
13         description="The spoken text verbatim, including all bracketed [...] or
14         ↳ parenthetical (...) notes like [Laughter.], (Applause.). Remove all line breaks
15         ↳ (\\n) so it is a single line."
16     )
17
18 # Pydantic schema class
19 class TranscriptResponse(BaseModel):
20     model_config = ConfigDict(extra="forbid")
21
22     entries: List[TranscriptEntry] = Field(
23         description="A list of transcript entries."
24     )

```

system prompt on Line 2 establishes a “Prime Directive” that prioritizes word fidelity and whitespace normalization. Although more elaborate prompts may provide benefits by allowing the LLM to correct typos or grammar, doing so carries the risk of unintended alterations to the original text; we recommend performing this in a separate postprocessing step, if required. The prompt also includes rules for ignoring metadata or Stage I artifacts, and guidelines for speaker identification. The latter is essential for handling “orphaned” text, that is, dialogue that lacks a speaker name because the speaker-turn began in the preceding text chunk. The LLM labels these instances as “UNKNOWN,” which we will resolve later. Considering such potential failure modes when dealing with LLMs is vital for good performance in analytics tasks.

Having access to clean raw text also makes Stage II suitable for many other analytics use cases beyond verbatim transcription. In many applications, analysts may not need to extract every word from a document, but instead need to identify specific decision-relevant quantities. For example, in procurement settings, extracting purchase quantities, supplier information, or prices from unstructured documents can support downstream sourcing and planning decisions (Li et al. 2025). This can be accommodated by modifying the prompt in Figure 18 and replacing the speaker-statement schema in Figure 17 with a schema tailored to the desired fields. Extraction tasks that involve complex field definitions or recurrent edge cases may benefit from alternative

prompt strategies, such as few-shot prompting (Brown et al. 2020), in which representative examples are included in the prompt to demonstrate the desired extraction behavior. This can be particularly helpful when fields are ambiguous or when similar text should be treated differently depending on context. Moreover, retrieval-augmented generation (RAG) may be appropriate when the desired output depends on information beyond the current document, or when extracted information needs to be validated or standardized against a larger document collection, such as product catalogs or business records. In these settings, the model first retrieves the relevant documents and uses them to inform its response before generating an output (Lewis et al. 2020). Finally, for organizations repeatedly performing the same highly specialized extraction task at scale, fine-tuning may further improve extraction performance and, in some cases, reduce costs by enabling task-specialized models, but it requires substantially more development effort (Lu et al. 2025). The principles of task decomposition and structured outputs outlined here remain applicable regardless of the prompting strategy employed.

To execute these instructions, the API call shown in Figure 18 combines the `system_prompt`, the `raw_text` chunk generated in Figure 14, and the structured-output schema defined by `TranscriptResponse`. The resulting structured data are shown in Figure 19. As in Stage I, we set the model temperature to 0.0 to minimize content variation. Splitting the task into

Figure 18. (Color online) LLM Prompt and API Call to Structure Text

```
1 # Text structuring prompt
2 system_prompt = """You are an expert transcript cleaner. You will receive a raw text
   ↳ chunk. Your goal is to extract the dialogue into a structured format.
3
4 **PRIME DIRECTIVE - WORD FIDELITY VS. FORMAT FLEXIBILITY:**
5 1. **DO NOT** change words, fix grammar, remove non-verbal markers (e.g., [Laughter
   ↳ .], (Applause.)) or autocomplete sentences.
6 2. **DO** fix whitespace. You must remove line breaks within a sentence and normalize
   ↳ multiple spaces into a single space.
7
8 Follow these processing rules:
9
10 1. **Scope & Exclusions:**
11   - **Ignore Metadata:** Do not extract meeting rosters, attendee lists, or headers.
   ↳ Start extracting only when the actual dialogue begins.
12   - **Ignore Artifacts:** Remove page numbers, file paths, and margin line numbers.
13
14 2. **Speaker Identification:**
15   - **Standard Speech:** Extract the speaker's name exactly as written and convert to
   ↳ UPPERCASE.
16   - **Orphaned Speech:** If the chunk starts with sentences/dialogue but has no
   ↳ speaker name attached, label the speaker as: UNKNOWN."""
17
18 # text structuring call
19 response = client.responses.parse(
20     model="gpt-4.1-mini-2025-04-14",
21     temperature=0.0,
22     input=[
23         {"role": "system", "content": system_prompt}, # Prompt instructions
24         {"role": "user", "content": raw_text}          # Section of raw text
25     ],
26     text_format=TranscriptResponse
27 )
```

smaller chunks also reduces the required context and output-token limits, allowing us to use a smaller, less expensive model. However, model selection remains a specific design choice, and empirical testing is required to determine the appropriate tradeoff between cost, speed, and accuracy for any specific use case.

Despite the careful prompt design and structured output constraints above, LLM-based extraction is not error-free. Although such errors are typically infrequent, they are systematic enough to require explicit postprocessing handling. However, this postprocessing is not limited to correcting extraction errors. Once the transcript is converted into a structured format, a wide range of additional transformations becomes straightforward to implement. These may include filtering out specific speakers, standardizing grammar or punctuation, removing transcription artifacts such as [sic], or translating the text into another language. The exact set of operations depends on the intended downstream analysis and may be implemented using deterministic rules or additional inference-based methods. All of these steps, such as filtering, are now simpler to execute as they can be applied directly to our structured data.

Although it is technically possible to perform many of these transformations within a single LLM call, doing so introduces unnecessary complexity. Recall our design principle in splitting tasks, in that the LLM is primarily responsible for handling variability in the input and converting unstructured text into a consistent, structured representation. Subsequent transformations are then applied to this structured output using programmatic postprocessing. In other words, LLMs handle nondeterministic tasks—such as interpreting inconsistent formatting or identifying speaker boundaries—whereas deterministic operations are best handled explicitly outside the model. This separation results in a more modular, interpretable, and maintainable system.

A simple example illustrates this design. In Figure 19, the first entry is labeled with the speaker “UNKNOWN.” This occurs when a chunk begins midstatement or when a speaker’s contribution spans a chunk boundary, preventing the model from confidently assigning a speaker label. In a postprocessing step, we replace each UNKNOWN label with the most recently identified speaker. (For brevity, we omit the implementation and refer the reader to the associated GitHub repository.) This

Figure 19. (Color online) Output of Stage II: Structured JSON Outputs of the Processed Text from Figure 16

```

1 {
2   "entries": [
3     {
4       "speaker": "UNKNOWN",
5       "statement": "commonly just a measurable drop in blood pressure without clinical
6       ↪ symptoms. A small percent of patients did become symptomatic and responded
7       ↪ quickly to fluids."
8     },
9     {
10      "speaker": "DR. SLEDGE",
11      "statement": "Going forward, is there something that you would recommend as part
12      ↪ of the package that patients have their blood pressure monitored regularly while
13      ↪ they are getting therapy?"
14    },
15    {
16      "speaker": "DR. BRETTMAN",
17      "statement": "Certainly during the early stages of therapy, yes."
18    },
19    {
20      "speaker": "DR. NERENSTONE",
21      "statement": "Dr. Lippman?"
22    },
23    {
24      "speaker": "DR. LIPPMAN",
25      "statement": "It was a very clear presentation of the primary data. My questions
26      ↪ are really just trying to get a sense or a suggestion of which subgroups may do
27      ↪ better or differently. Did you look at patients who had prior responses to
28      ↪ fludarabine but had progressed within the 6-month eligibility? Did they do better
29      ↪ on the subsequent Campath?"
30    },
31    {
32      "speaker": "DR. BRETTMAN",
33      "statement": "Let me show that to you. Patients who had ever responded to
34      ↪ fludarabine versus those that had not -- it will just take a moment."
35    },
36    {
37      "speaker": "DR. LIPPMAN",
38      "statement": "While they are looking for that, the other subgroup which you may
39      ↪ not have data on because I know it is very rare, although we know that there is at
40      ↪ least one patient that fit this category with T-cell CLL, was there a different
41      ↪ response in that group, or did you look at that?"
42    }
43  ]
44 }

```

deterministic correction resolves the ambiguity without increasing prompt complexity or repeated LLM calls. By separating extraction from refinement, the pipeline ensures that structured outputs remain both consistent and extensible.

Considerations for Downstream Analysis

The structured JSON output from the pipeline provides a clean and flexible foundation for downstream analysis, enabling further transformations and analytics without reprocessing the original text. In the following,

we discuss how to evaluate the quality of LLM extraction and illustrate how such structured data can facilitate downstream analytical tasks in practice.

Assessing LLM Extraction Accuracy

LLM-based extraction requires careful validation before implementation into any decision-making process (Wasserkrug 2026). For verbatim text extraction, we are particularly interested in text extraction accuracy and correct speaker identification. To exemplify this concern, Figure 20 shows malformed output produced by Stage II when directly passed a degraded PDF text

Figure 20. (Color online) Malformed Text Processed from Text Extract Layer

```
1 "speaker": "DR. SLEDGE",
2 "statement": "Second question, could
  ↳ you give me a oetter sense of _w,hat
  ↳ the hypotensive events meant, and the
  ↳ -. severity of the hypotension that
  ↳ was observed, and was any of this
  ↳ anaphylactoid?"
```

layer. Note the mistaken character changes and additions (e.g., “oetter” instead of “better” and “w,hat” instead of “what”). Figure 21 shows the same passage correctly processed with the extracted text from our vision-based Stage I. Beyond infrequent character-level errors, we found LLMs to occasionally hallucinate by correcting typos in the source document or dropping short speaker statements. Finding and correcting these patterns via effective evaluation procedures is essential for designing targeted postprocessing steps or prompt adjustments.

Several ways exist to assess text extraction accuracy (see Neudecker et al. (2021) for an overview). The most straightforward approach requires two components: an appropriate accuracy metric and a human-verified ground truth to compare against. Although there are several OCR benchmarks that help gauge and compare the algorithm’s general capabilities (Fu et al. 2025, Yang et al. 2025), they are often not domain specific (Beyene and Dancy 2026). This is particularly problematic for extraction tasks with additional structural complexity, as in our meeting transcripts, which include speaker turns. Therefore, we highly recommend creating a domain-specific ground truth from the extraction source document. In addition, such a ground truth directly enables us to build rules or train algorithms for a postprocessing pipeline to handle any systematic errors if required (Nguyen et al. 2021).

To establish a domain-specific ground truth, one approach is to synthetically generate digital typeset source documents and mimic the degradation and noise by manually printing and scanning them (Kanungo and Haralick 1999). However, our FDA transcripts span 18 years and vary considerably in formatting and image

Figure 21. (Color online) Correct Text Processed from Vision Extraction

```
1 "speaker": "DR. SLEDGE",
2 "statement": "Second question, could
  ↳ you give me a better sense of what
  ↳ the hypotensive events meant, and the
  ↳ severity of the hypotension that was
  ↳ observed, and was any of this
  ↳ anaphylactoid?"
```

quality, making a synthetic generation approach impractical. Because exhaustive transcription of large text documents is rarely feasible, we instead recommend verifying a representative sample that captures the structural diversity. As an example, we sampled across multiple documents rather than exhaustively verifying a single source. Specifically, we manually extracted 25 continuous pages from 10 randomly selected transcripts, yielding a 250-page evaluation benchmark. This sampling strategy ensures coverage of distinct meeting phases with varying speaker-turn patterns, such as short turnovers in the introduction and multipage-long speaker statements in the sponsor presentation phase.

After establishing ground truth, we can measure the pipeline’s text accuracy. Although various error measures exist (see Neudecker et al. (2021) for a comprehensive survey), verbatim text fidelity is typically evaluated using character error rate (CER) or its counterpart, character accuracy ($1 - \text{CER}$). CER calculates the minimum number of character insertions, substitutions, and deletions required to transform the pipeline output into the ground truth. The error is standardized by the number of characters in the ground truth, making the metric sensitive to sentence length; that is, the same number of errors leads to a much higher error for a short sentence than for a much longer one. Therefore, computing a cumulative error rate across the entire document can provide a more stable comparison. In addition to character accuracy, entity extraction tasks such as speaker attribution are more appropriately evaluated using a metric like the F_1 -score, because it measures classification performance independently of document length. In the Online Appendix, we present a detailed benchmarking and cost analysis. Evaluated against the 250-page ground truth, our pipeline achieves a character accuracy and F_1 -score of 0.99925 and 0.99927, respectively. Scaling the pipeline to process all 1,922 pages across the 10 transcripts incurred a total processing cost of \$3.50.

Finally, for meaning-dependent tasks such as topic modeling, semantic similarity scores like the BERT-Score compare contextual word embeddings to assess whether the output preserves the original text’s meaning even when paraphrased (Zhang et al. 2019). Domain-specific variants can further improve this evaluation by capturing terminology and nuances particular to specialized fields. For example, BioBERT (Lee et al. 2019), trained on biomedical corpora such as research articles and clinical notes, could be used to evaluate extraction accuracy in contexts like FDA Advisory Committee transcripts where domain-specific terminology is prevalent.

Data Analysis

Now that the unstructured text is in a structured data format defined by consistent JSON keys, we can

analyze the JSON directly or, if preferred, easily flatten the data into any two-dimensional data format, such as a CSV, or place it into a database. To briefly illustrate how this structured format supports downstream processing of the extracted transcript, we present two examples: a structured speaker-participation analysis via speaker labels and word counts and a sentiment analysis on the unstructured individual statements. In the context of FDA Advisory Committee discussions, insights into who speaks, what they say, and how they say it can provide valuable information for FDA reviewers deciding whether to approve a drug or device application; product sponsors deciding how to design clinical trials and research pipelines; and the broader public seeking transparency into how regulatory decisions that affect patient safety and treatment access are reached. For brevity, we omit the required code here and refer to the accompanying GitHub repository.

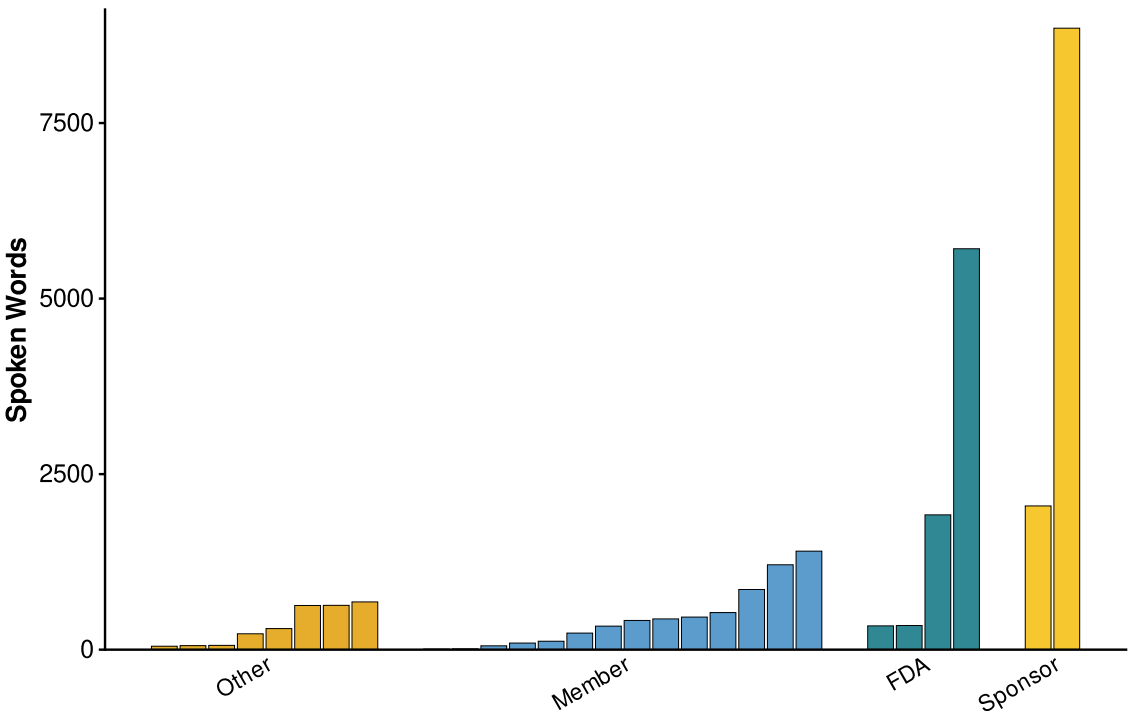
Structured Analysis: Speaker Participation. To demonstrate the utility of the structured output, we first evaluate speaker participation during a meeting by mapping the extracted text to external metadata. Using the structured JSON, we aggregate the total number of spoken words by grouping the records by the `speaker` field. We then augment these data by matching each unique speaker name to their official role during the

meeting, for example, committee member, consultant, or FDA personnel. As shown in Figure 22, the resulting distribution reveals a highly right-skewed participation pattern, where a single sponsor representative and an FDA official account for the majority of spoken words, contrasted by a long tail of participants with minimal contributions. This structured representation allows analysts to directly understand meeting dynamics and identify key actors before conducting more granular text analysis.

Unstructured Analysis: Speaker Sentiment. Because Advisory Committee meetings influence critical regulatory outcomes, evaluating the tone of the discussion can provide additional context regarding the decision-making process (Hansen et al. 2018, Markou and Chan 2026). Unlike the preceding word-count analysis, which operates on the speaker identification, sentiment analysis processes the unstructured text within the `statement` field. Because our JSON schema explicitly pairs each unstructured text block with a specific `speaker` key, these statements can be programmatically passed to a text-processing pipeline and aggregated by individual speakers or stakeholder role (e.g., FDA representatives, sponsors, committee members, or patient representatives).

To demonstrate this capability, we apply the widely accessible `vaderSentiment` package (Hutto and

Figure 22. (Color online) Number of Spoken Words Varies Substantially Across Speakers, Demonstrating That Participation in the Meeting Was Concentrated Among a Subset of Participants



Note. Each column represents individual speakers, the vertical axis reports the number of words attributed to each speaker, and the color encodes their respective roles.

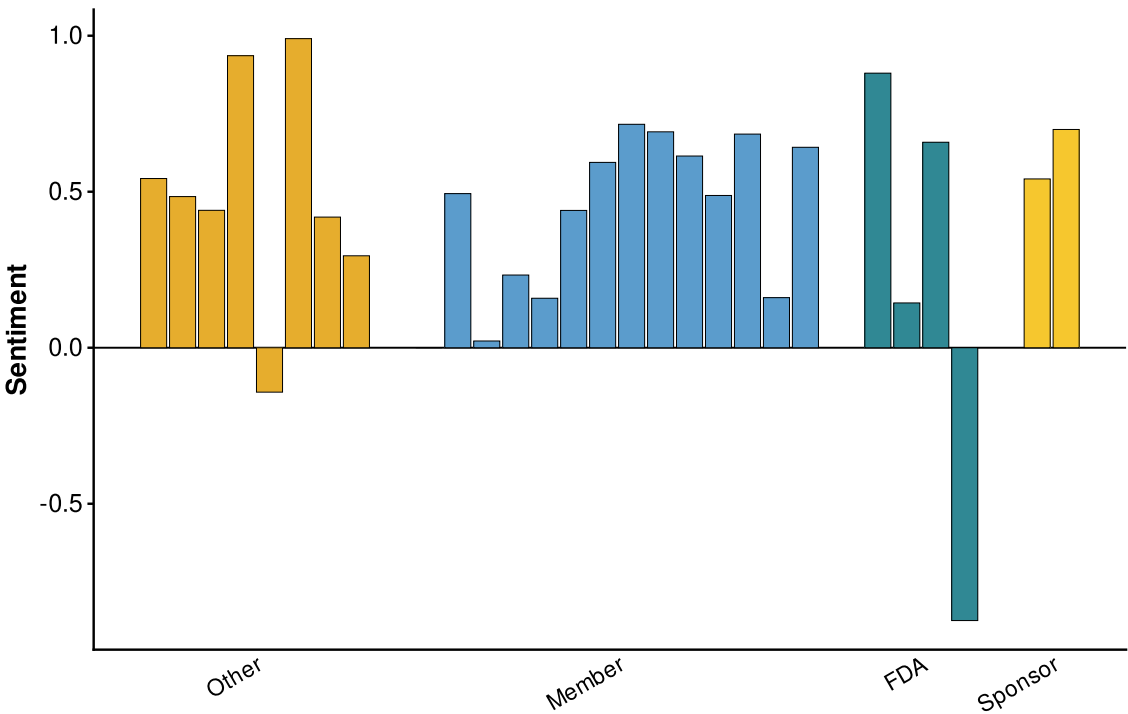
Gilbert 2014) to the extracted statements and average the scores by speaker. As a generic, rule-based lexicon, Valence Aware Dictionary and Entiment Reasoner (VADER) provides a highly reproducible baseline for our pipeline, although it lacks domain adaptation for complex regulatory environments. Because our focus is on the data-extraction phase, we present these scores solely as a demonstration of downstream analysis rather than a validated sentiment measure. Applications supporting decision-critical analyses should validate the chosen sentiment methodology against appropriate ground truth.

In the example shown in Figure 23, the speaker-level scores illustrate how the extracted structured data can generate decision-relevant insights by identifying which participants expressed relatively more positive or critical language during the deliberation. A sponsor preparing for a future Advisory Committee meeting could use these results to identify speakers expressing relatively negative sentiment and immediately retrieve their associated statements to understand the scientific questions, safety concerns, or evidentiary issues that motivated those views. Likewise, FDA reviewers could quickly identify stakeholder groups expressing greater concern and prioritize the corresponding discussion for closer review during the deliberative process. Speaker-level sentiment and other transcript-derived features can also serve as direct inputs to predictive models. Aggregating sentiment by stakeholder role allows

analysts to quantify how critically committee members assessed the sponsor’s evidence relative to the FDA’s reviewers. Similarly, speaker-level sentiment may serve as one predictor of the Advisory Committee voting outcomes, which the FDA follows in the vast majority of cases. Markou and Chan (2026) demonstrate that deliberation patterns in these meetings are predictive of voting outcomes, and Hansen et al. (2018) apply similar text-based analyses to Federal Reserve deliberations to study how discussion dynamics shape monetary policy decisions.

Such analyses provide a quantitative summary of discussions previously accessible only through manual reading of lengthy transcripts. Although we illustrate this capability using sentiment analysis, the same structured speaker-statement representation can support many other downstream analytical workflows. More broadly, structured data extraction of unstructured text enables many forms of downstream analysis beyond the example considered here. Many institutional records are distributed as PDFs, and historical archives may exist only as scanned or printed documents that must first be digitized before analysis can occur. Examples include transcripts of customer feedback sessions or focus groups, records of telephone conversations, discussions of government decision-making bodies, or collections of written feedback and survey responses stored in difficult-to-analyze formats. Converting these

Figure 23. (Color online) Speaker-Level Sentiment Variation Across Meeting Participants, Revealing Differences That Are Obscured When the Transcript Is Analyzed Only in Aggregate



Note. Each column represents individual speakers, the vertical axis reports their average sentiment score, and the color encodes their respective participant groups.

materials into structured data makes them amenable to modern analytics workflows, enabling practitioners to apply quantitative methods to previously unstructured sources of information.

Concluding Remarks

The substantial time traditionally required for data extraction and cleaning has long been a bottleneck in analytical workflows. As LLMs are increasingly adopted across a wide range of tasks beyond data preparation itself, they create a growing demand for clean, well-structured inputs; resources that have historically been difficult and costly to obtain. At the same time, these models offer a practical means to address this challenge, providing new tools to automate and scale the transformation of unstructured data. In this tutorial, we present an end-to-end pipeline that transforms raw, unstructured PDFs into structured data and, ultimately, into features suitable for downstream analysis and decision making. Central to this approach are two key capabilities: structured outputs, which ensure consistent output formats, and multimodal processing, which allows models to interpret complex visual document layouts.

Through our specific application to FDA documents, we have demonstrated how such a document-extraction pipeline can achieve high accuracy, outlined practical approaches for evaluating its performance, and presented a use case for downstream data analysis. We emphasized the importance of programmatic interaction with LLMs via APIs, enabling tighter control over outputs and facilitating integration into broader analytical workflows. We also highlighted the benefits and motivation for decomposing tasks into smaller, more manageable components, allowing the effective use of lower-cost models without sacrificing overall performance.

Text extraction is not only a standalone data preparation task; it is a foundational component of analytics workflows that are rapidly adopting Generative AI. For example, RAG systems combine LLMs with external document retrieval, ensuring model responses are well grounded in relevant information. These systems depend on clean, well-structured source material that can be indexed, searched, and retrieved reliably. Looking forward, extraction pipelines are also likely to become core components of agentic systems, in which LLM-based agents retrieve documents, extract relevant fields, validate outputs, and pass structured information to downstream optimization, forecasting, or decision models.

Taken together, this tutorial provides a functional and extensible framework for deploying LLM-based extraction pipelines in real-world settings. We anticipate future work will build on this foundation to provide access to a wide range of document types,

including financial reports, medical records, and supply chain documentation. By reducing the time and cost of unlocking information from unstructured sources, such approaches have the potential to significantly expand access to data and improve decision making across domains.

Acknowledgments

The authors thank Nora Gaudion for invaluable research assistance in this work. The authors used OpenAI's ChatGPT, Anthropic's Claude, and Google's Gemini to assist with spelling, text editing, and coding. The authors thoroughly reviewed all suggestions and take full responsibility for the publication's content.

Endnotes

¹ The accompanying code is maintained on GitHub (<https://github.com/sos-analytics/ijaa-llm-tutorial>) and permanently archived on Zenodo (<https://doi.org/10.5281/zenodo.22013691>).

² For this reason, optimal prompt styles vary between providers and are constantly evolving, and regularly consulting provider guidelines is prudent. Official prompt guidance is provided by OpenAI (<https://platform.openai.com/docs/guides/prompt-engineering>); Anthropic (<https://platform.claude.com/docs/en/build-with-claude/prompt-engineering>); and Google (<https://ai.google.dev/gemini-api/docs/prompting-strategies>). Additionally, OpenAI provides a prompt optimization feature (<https://platform.openai.com/chat/edit?models=gpt-4.1&optimize=true>).

³ See OpenAI's structured output overview (<https://developers.openai.com/api/docs/guides/structured-outputs>).

⁴ For instructions on key generation and the billing process, visit OpenAI's official documentation (<https://platform.openai.com/docs/quickstart>).

⁵ Some recent reasoning models provide only limited support for temperature adjustment. For OpenAI models GPT-5 and newer, only two temperature settings are supported (zero and one) when `reasoning_effort=none`. Although these settings seem undocumented, setting temperature = 0 yields more consistent outputs (see Online Appendix).

⁶ Images can also be transmitted via URL or uploaded directly to LLM providers.

⁷ OpenAI provides a free tokenizer tool, which allows users to measure how many tokens a prompt will use (<https://platform.openai.com/tokenizer>).

⁸ Last accessed on August 19, 2026. For current pricing, visit OpenAI (<https://developers.openai.com/api/docs/pricing>).

⁹ For implementation details, see the OpenAI Batch API documentation (<https://platform.openai.com/docs/api-reference/batch>).

References

- Atil B, Aykent S, Chittams A, Fu L, Passonneau RJ, Radcliffe E, Rajagopal GR, et al. (2025) Non-determinism of "deterministic" LLM system settings in hosted environments. Akter M, Chowdhury T, Eger S, Leiter C, Opitz J, Çano E, eds. *Proc. 5th Workshop Evaluation Comparison NLP Systems* (Association for Computational Linguistics, Stroudsburg, PA), 135–148.
- Belcak P, Heinrich G, Diao S, Fu Y, Dong X, Muralidharan S, Lin YC, Molchanov P (2025) Small language models are the future of agentic AI. Preprint, submitted June 2, <https://arxiv.org/abs/2506.02153>.

- Beyene FS, Dancy CL (2026) *A Survey of OCR Evaluation Methods and Metrics and the Invisibility of Historical Documents* (Association for Computing Machinery, New York).
- Bray RL (2025) A tutorial on teaching data analytics with generative ai. *INFORMS J. Appl. Anal.* 55(4):319–343.
- Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, et al. (2020) Language models are few-shot learners. Larochelle H, Ranzato M, Hadsell R, Balcan M, Lin H, eds. *Advances in Neural Information Processing Systems*, vol. 33 (Curran Associates Inc., Red Hook, NY), 1877–1901.
- Bumblauskas D, Igou A, Kalghatgi S, Wetzel C (2022) Public policy and broader applications for the use of text analytics during pandemics. *INFORMS J. Appl. Anal.* 52(6):568–581.
- Cohen MC, Dai T, Perakis G, Agrawal N, Allon G, Boute RN, Cachon GP, et al. (2026) OM Forum—Supply chain management in the ai era: A vision statement from the operations management community. *Manufacturing Service Oper. Management* 28(3):687–705.
- Colvin S, Jolibois E, Ramezani H, Garcia Badaracco A, Dorsey T, Montague D, Matveenko S, et al. (2025) Pydantic validation. <https://docs.pydantic.dev/latest/>.
- De Kok T (2025) ChatGPT for textual analysis? How to use generative LLMs in accounting research. *Management Sci.* 71(9):7888–7906.
- Fu L, Kuang Z, Song J, Huang M, Yang B, Li Y, Zhu L, et al. (2025) Ocrbench v2: An improved benchmark for evaluating large multimodal models on visual text localization and reasoning. Belgrave D, Zhang C, Lin H, Pascanu R, Koniusz P, Ghassemi M, Chen N, eds. *Advances in Neural Information Processing Systems*, vol. 38 (Curran Associates Inc, Red Hook, NY), 107930–107981.
- Hansen S, McMahon M, Prat A (2018) Transparency and deliberation within the FOMC: A computational linguistics approach. *Quart. J. Econom.* 133(2):801–870.
- Humphries M, Leddy LC, Downton Q, Legace M, McConnell J, Murray I, Spence E (2025) Unlocking the archives: Using large language models to transcribe handwritten historical documents. *Historical Methods: J. Quant. Interdisciplinary History* 58(3):175–193.
- Hutto C, Gilbert E (2014) Vader: A parsimonious rule-based model for sentiment analysis of social media text. *Proc. Internat. AAAI Conf. Web Soc. Media* 8(1):216–225.
- Kanungo T, Haralick R (1999) An automatic closed-loop methodology for generating character groundtruth for scanned documents. *IEEE Trans. Pattern Anal. Machine Intelligence* 21(2): 179–183.
- Lee J, Yoon W, Kim S, Kim D, Kim S, So CH, Kang J (2019) Biobert: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics* 36(4):1234–1240.
- Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, Küttler H, et al. (2020) Retrieval-augmented generation for knowledge-intensive nlp tasks. Larochelle H, Ranzato M, Hadsell R, Balcan M, Lin H, eds. *Advances in Neural Information Processing Systems*, vol. 33 (Curran Associates Inc., Red Hook, NY), 9459–9474.
- Li X, Culmone V, De Reyck B, Yoo OS (2025) Automating procurement practices using artificial intelligence. *INFORMS J. Appl. Anal.* 55(3):195–223.
- Liu NF, Lin K, Hewitt J, Paranjape A, Bevilacqua M, Petroni F, Liang P (2024) Lost in the middle: How language models use long contexts. *Trans. Assoc. Comput. Linguistics* 12:157–173.
- Lu W, Luu RK, Buehler MJ (2025) Fine-tuning large language models for domain adaptation: Exploration of training strategies, scaling, model merging and synergistic capabilities. *NPJ Comput. Materials* 11(1):84.
- Markou P, Chan TH (2026) Incentivizing information exchange within groups: The role of voting protocols in us food and drug administration advisory committees. *Management Sci.* 72(7):5723–5743.
- Mohammadi B (2024) Explaining large language models decisions using Shapley values. Preprint, submitted March 29, <https://arxiv.org/abs/2404.01332>.
- Neudecker C, Baierer K, Gerber M, Clausner C, Antonacopoulos A, Pletschacher S (2021) A survey of ocr evaluation tools and metrics. *Proc. 6th Internat. Workshop Historical Document Imaging Processing* (Association for Computing Machinery, New York), 13–18.
- Nguyen TTH, Jatowt A, Coustaty M, Doucet A (2021) Survey of post-ocr processing approaches. *ACM Comput. Surv.* 54(6):1–37.
- Ouyang S, Zhang JM, Harman M, Wang M (2025) An empirical study of the non-determinism of chatgpt in code generation. *ACM Trans. Software Engrg. Methodology* 34(2):1–28.
- Renze M (2024) The effect of sampling temperature on problem solving in large language models. Al-Onaizan Y, Bansal M, Chen YN, eds. *Proc. Findings Assoc. Comput. Linguistics: EMNLP 2024* (Association for Computational Linguistics, Stroudsburg, PA), 7346–7356.
- Schilling-Wilhelmi M, Ríos-García M, Shabih S, Gil MV, Miret S, Koch CT, Márquez JA, et al. (2025) From text to insight: Large language models for chemical data extraction. *Chemical Soc. Rev.* 54(3):1125–1150.
- Wasserkrug S (2026) Editorial statement—Harnessing the power of large language models responsibly in applied analytics. *INFORMS J. Appl. Anal.* 56(4):v–vi.
- Wasserkrug S, Krüger M, A Feldman Y, Shindin E, Zeltyn S (2019) What's wrong with my dishwasher: Advanced analytics improve the diagnostic process for miele technicians. *INFORMS J. Appl. Anal.* 49(5):384–396.
- Wu DA (2024) Text-based measure of supply chain risk exposure. *Management Sci.* 70(7):4781–4801.
- Yang Z, Tang J, Li Z, Wang P, Wan J, Zhong H, Liu X, et al. (2025) CC-OCR: A comprehensive and challenging OCR benchmark for evaluating large multimodal models in literacy. *Proc. IEEE/CVF Internat. Conf. Comput. Vision (IEEE/CVF, Los Alamitos, CA)*, 21744–21754.
- Young RJ, Gillins B, Matthews AM (2025) When models can't follow: Testing instruction adherence across 256 llms. Preprint, submitted October 18, <https://arxiv.org/abs/2510.18892>.
- Zhalechian M, Saghaian S, Robles O (2026) Harmonizing safety and speed: A human-algorithm approach to enhance the FDA's medical device clearance policy. *Management Sci.*, ePub ahead of print August 28, <https://doi.org/10.1287/mnsc.2024.06477>.
- Zhang T, Kishore V, Wu F, Weinberger KQ, Artzi Y (2019) Bertscore: Evaluating text generation with BERT. Preprint, submitted April 21, <https://arxiv.org/abs/1904.09675>.
- Zhu Y, Sen S, Everhart A, Karaca-Mandic P (2025) A deep learning approach for predicting FDA's 510(k) medical device recalls using device citation relationships. *Inform. Systems Res.* 0(0):1–24. preprint.

Simon Spavound is an assistant clinical professor in the Department of Decision Sciences and MIS at Drexel University's LeBow College of Business. He holds a PhD in economics from Lancaster University. He was head of data science at Peak, where he applied forecasting, optimization, and machine learning to business problems in the retail and manufacturing sectors. His research interests include time series forecasting, machine learning, and the trustworthy application of artificial intelligence in organizational decision making.

Oliver Schaer is an assistant professor in the Department of Decision Sciences and MIS at Drexel University's LeBow College of Business. He received his PhD in management science from Lancaster University. His research develops predictive decision tools

that integrate unstructured data, time-series econometrics, diffusion modeling, and natural language processing to address uncertainty in product lifecycle management and sustainable operations.

Panos Markou is an assistant professor of business administration in the Technology and Operations Management area at the

University of Virginia Darden School of Business. His research focuses on innovation and technology management, spanning topics such as research and development project selection and portfolio decisions, new product development, pharmaceutical and regulated product development, and how organizations evaluate, deliberate on, and learn from innovation decisions.