ExStrucTiny: A Benchmark for Schema-Variable Structured Information Extraction from Document Images

Mathieu Sibue   Andres Muñoz Garza   Samuel Mensah   Pranav Shetty
Zhiqiang Ma   Xiaomo Liu   Manuela Veloso
J.P. Morgan AI Research
{name}.{surname}@jpmchase.com

Abstract

Enterprise documents, such as forms and reports, embed critical information for downstream applications like data archiving, automated workflows, and analytics. Although generalist Vision Language Models (VLMs) perform well on established document understanding benchmarks, their ability to conduct holistic, fine-grained structured extraction across diverse document types and flexible schemas is not well studied. Existing Key Entity Extraction (KEE), Relation Extraction (RE), and Visual Question Answering (VQA) datasets are limited by narrow entity ontologies, simple queries, or homogeneous document types—often overlooking the need for adaptable and structured extraction. To address these gaps, we introduce **ExStrucTiny**,[1] a new benchmark dataset for structured Information Extraction (IE) from document images, unifying aspects of KEE, RE, and VQA. Built through a novel pipeline combining manual plus synthetic human-validated samples, ExStrucTiny covers more varied document types and extraction scenarios. We analyze open and closed VLMs on this benchmark, highlighting challenges such as schema adaptation, query under-specification, and answer localization. We hope this work provides a bedrock for improving generalist models for structured IE in visually rich documents.

1 Introduction

Documents are pervasive in business operations, underpinning critical workflows in domains such as finance and healthcare. Forms, receipts, or reports routinely encode essential information that must be extracted to support data storage, automated processes, compliance, and analytics. Manual review is labor-intensive and error-prone, thus motivating research in visually rich document understanding (VRDU), at the crux of natural language processing and computer vision (Cui et al., 2021).

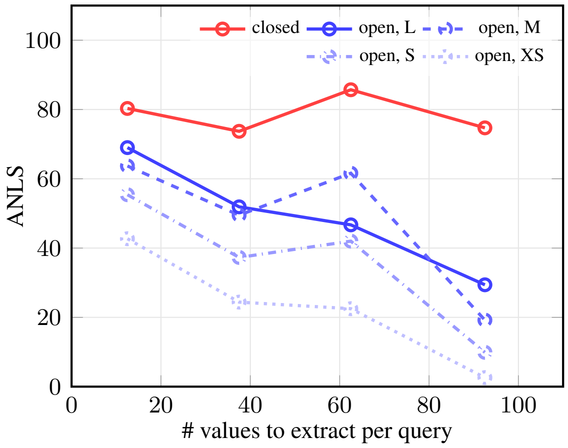


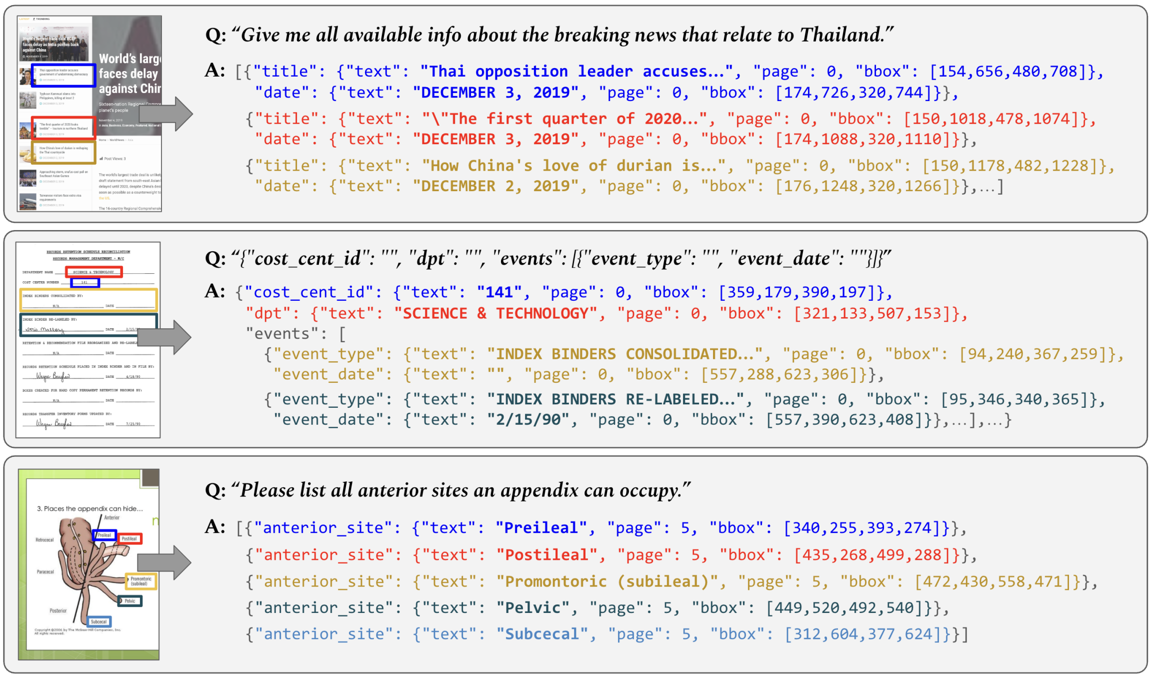
Figure 1: Extraction performance declines for open source VLMs as number of values to extract increases.

In the text-only IE literature, extractive tasks are often categorized by the specificity of their input queries (Qi et al., 2024): closed IE requires a predefined schema to be populated with relevant values, open IE seeks to discover structure without explicit guidance, and on-demand IE addresses underspecified queries that require models to extract implicitly pertinent information (Jiao et al., 2023). This spectrum of query specification has shaped the design and evaluation of text-only IE models.

In the VRDU field, KEE and RE datasets are designed for closed IE, focusing on extracting a predefined set of entities from a single document type using sequence tagging approaches (Park et al., 2019; Jaume et al., 2019; Wang et al., 2025). While realistic in data-rich scenarios, these datasets for training and evaluating specialist models are not meant to assess data-scarce generalization to new domains, entities, and user requirements. On the other hand, VQA datasets in VRDU cover a broader range of entities, but their extractive questions are typically simple single-span closed IE queries (Mathew et al., 2021; Landeghem et al., 2023), which do not mea-

[1] Full dataset available upon request for research only at airdata.requests@jpmorgan.com

Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 5669–5688
March 24-29, 2026 ©2026 Association for Computational Linguistics


3. Places the appendix can hide...
Anterior
Retrocecal
Preileal
Postileal
Paracecal
Promontoric (subileal)
Pelvic
Posterior
Subcecal
Copyright ©2006 by The McGraw-Hill Companies, Inc. All rights reserved.

Q: *“Give me all available info about the breaking news that relate to Thailand.”*

A:
```
[{"title": {"text": "Thai opposition leader accuses…", "page": 0, "bbox": [154,656,480,708]},
  "date": {"text": "DECEMBER 3, 2019", "page": 0, "bbox": [174,726,320,744]}},
 {"title": {"text": "\"The first quarter of 2020…", "page": 0, "bbox": [150,1018,478,1074]},
  "date": {"text": "DECEMBER 3, 2019", "page": 0, "bbox": [174,1088,320,1110]}},
 {"title": {"text": "How China's love of durian is…", "page": 0, "bbox": [150,1178,482,1228]},
  "date": {"text": "DECEMBER 2, 2019", "page": 0, "bbox": [176,1248,320,1266]}},…]
```

Q: *“{"cost_cent_id": "", "dpt": "", "events": [{"event_type": "", "event_date": ""}]}”*

A:
```
{"cost_cent_id": {"text": "141", "page": 0, "bbox": [359,179,390,197]},
 "dpt": {"text": "SCIENCE & TECHNOLOGY", "page": 0, "bbox": [321,133,507,153]},
 "events": [
   {"event_type": {"text": "INDEX BINDERS CONSOLIDATED…", "page": 0, "bbox": [94,240,367,259]},
    "event_date": {"text": "", "page": 0, "bbox": [557,288,623,306]}},
   {"event_type": {"text": "INDEX BINDERS RE-LABELED…", "page": 0, "bbox": [95,346,340,365]},
    "event_date": {"text": "2/15/90", "page": 0, "bbox": [557,390,623,408]}},…],…}
```

Q: *“Please list all anterior sites an appendix can occupy.”*

A:
```
[{"anterior_site": {"text": "Preileal", "page": 5, "bbox": [340,255,393,274]}},
 {"anterior_site": {"text": "Postileal", "page": 5, "bbox": [435,268,499,288]}},
 {"anterior_site": {"text": "Promontoric (subileal)", "page": 5, "bbox": [472,430,558,471]}},
 {"anterior_site": {"text": "Pelvic", "page": 5, "bbox": [449,520,492,540]}},
 {"anterior_site": {"text": "Subcecal", "page": 5, "bbox": [312,604,377,624]}}]
```

Figure 2: Three example query-answer pairs from ExStrucTiny. Top: on-demand IE query in plain text. Middle: closed IE query with schema. Bottom: closed IE query in plain text. Answers are JSON-formatted objects following the same structuring conventions.

sure models' ability to extract and organize multiple pieces of information in structured formats. Finally, while table and chart parsing datasets bear similarities to open IE, most solely concentrate on isolated layout elements and lack the broader document context needed for a holistic evaluation (Choi et al., 2019; Masry et al., 2022; Smock et al., 2022).

The emergence of LLMs and VLMs has enabled more flexible instruction-driven extraction, bridging the modeling gap between closed, open, and on-demand IE (Qi et al., 2024; Wang et al., 2023a; Perot et al., 2024; He et al., 2023). However, existing VRDU datasets do not adequately assess these models' capacity for holistic, fine-grained structured extraction across heterogeneous documents with novel entities and user-specified schemas—especially in zero and few-shot settings.

To address these limitations, we contribute (1) **ExStrucTiny**, a new structured IE benchmark dataset from document images, comprising 304 closed and on-demand IE query-answer pairs and spanning 110 multi-page documents. ExStrucTiny unifies aspects of KEE, RE, and VQA, and is constructed through (2) a novel pipeline combining manual annotations and LLM-generated, human-validated samples across forms, financial reports, slide decks, and webpage screenshots. Additionally, we propose (3) a new evaluation framework for structured IE, and provide (4) a comprehensive performance analysis of open and closed VLMs on ExStrucTiny to highlight persistent challenges with schema adaptation, query underspecification, and answer localization in IE.

2 Related Work

IE Datasets KEE is one of the most studied extraction task in the VRDU landscape, with numerous datasets each targeting a specific type of document, such as receipts (CORD, Park et al. (2019)), invoices (VRDU ad-buy, Wang et al. (2023c)), and forms (FUNSD, Jaume et al. (2019)). These datasets rely on predefined entity ontologies that are closely tied to the targeted domain, and for which values need to be extracted. For example, CORD annotates 24 specific entities in restaurant receipts (e.g., *“menu item name”*). Recent work has explored adapting KEE datasets for generative models by converting their annotations into natural language closed IE queries using either templates (Zmigrod et al., 2024) or LLMs (Giovannini et al., 2025). However, most efforts in this direction produce only single-span extractive questions restricted by the fixed schema of the source datasets, making them fail to capture the complexity and diversity of real-world extractive queries.

Document VQA datasets, such as DocVQA (Mathew et al., 2021), InfographicVQA (Mathew et al., 2022), DUDE (Landeghem et al., 2023),

Dataset	#Docs	#QAs	Len(Doc)	Len(Q)	Len(A)	#Entities per Q	#Values per A	%Q Unansw.	Q-Doc Overlap	$\overline{\lvert \text{Ontology} \rvert}$
CORD	100	100 (2400)	23.6	64.0 (2.7)	24.0 (1.0)	24.0 (1.0)	13.4 (0.6)	100% (67%)	39% (46%)	0.24 (0.01)
Kleister Charity	606	606 (4848)	6824.6	24.0 (3.0)	9.2 (1.2)	8.0 (1.0)	4.6 (0.6)	100% (43%)	48% (69%)	0.01 (0.00)
VRDU Ad-buy	641	641 (8974)	1524.4	25.0 (1.8)	103.5 (7.4)	14.0 (1.0)	65.1 (4.7)	95% (16%)	44% (75%)	0.02 (0.00)
VRDU Registration	1915	1915 (11490)	948.8	13.0 (2.2)	13.4 (2.2)	6.0 (1.0)	4.5 (0.7)	83% (26%)	55% (74%)	0.00 (0.00)
FUNSD	47	47 (32430)	253.9	1652.0 (2.4)	80.3 (0.1)	690.0 (1.0)	13.0 (0.1)	100% (98%)	42% (59%)	14.7 (0.02)
DocVQA	1287	4733	194.9	8.5	2.1	1.0	1.0	0%	56%	0.85
SlideVQA$_{sample}$	218	760	1252.0	12.6	1.7	1.1	1.0	0%	62%	1.00
DUDE$_{val}$	732	1986	1745.7	8.4	3.3	1.1	1.2	13%	61%	1.00
TAT-DQA	277	662	544.6	10.7	7.8	1.0	1.0	0%	67%	1.06
VisualMRC	1392	1472	1017.5	8.8	3.8	1.0	1.0	0%	68%	1.02
ExStrucTiny	110	304	1278.5	18.2	33.5	4.0	8.7	41%	47%	3.46

Table 1: Comparison of ExStrucTiny VS. test splits of KEE (top) and VQA (bottom) datasets across key statistics: number of documents (*#Docs*), number of extractive QA pairs (*#QAs*), average document length (*Len(Doc)*), average query length (*Len(Q)*), average answer length (*Len(A)*), average number of entities requested per query (*#Entities per Q*), average number of extracted values in the answer (*#Values per A*), percentage of queries requesting some missing entities (*%Q Unansw.*), lexical overlap (*Q-Doc Overlap*), normalized ontology size ($\overline{\lvert \textit{Ontology} \rvert}$). As the test split of DUDE does not include answers, we use the validation split instead. See Appendix B for more details.

SlideVQA (Tanaka et al., 2023), and VisualMRC (Tanaka et al., 2021), often include a subset of extractive question-answer pairs akin to closed IE, but with a broader scope than KEE thanks to an ontology-agnostic design. For instance, DocVQA includes a wide variety of questions like *"What is the invoice number?"* or *"Who is the author of the document?"* However, these questions frequently present significant overlap with the language of the documents, which might let models answer correctly by string matching and proximity, rather than true semantic understanding.[2] Furthermore, VQA datasets rarely request multiple related entities in a single question and seldom include unanswerable questions (Landeghem et al., 2023), even though such cases are common in practical applications.

In contrast, ExStrucTiny provides closed and on-demand IE queries—including multi-entity, low-lexical-overlap, and unanswerable queries—across four document types, generated without the constraint of a fixed entity ontology. This enables robust zero-shot evaluation of VLMs for structured IE, directly addressing the limitations of previous KEE and VQA datasets. Table 1 compares ExStrucTiny and the test split of popular KEE and VQA datasets in VRDU across key statistics.

IE Modeling Traditionally, extractive KEE and VQA tasks in VRDU have been tackled using sequence tagging models over OCR tokens, with specialist architectures fine-tuned in data-rich settings for specific domains (Rombach and Fettke, 2025; Xu et al., 2020), though a few early generative exceptions existed (Powalski et al., 2021). These pipeline approaches, usually combining OCR, vision and text encoders, were not designed nor evaluated for zero-shot transfer to new document types, entities, and queries—thus limiting their generalization. Subsequently, vision-only autoregressive models like Donut (Kim et al., 2022) and Pix2Struct (Lee et al., 2023) enabled flexible structured output generation thanks to the use of a decoder, but lacked robust instruction-following capabilities for fulfilling diverse user intents. The more recent post-trained LLMs (Perot et al., 2024; He et al., 2023) and VLMs (Hu et al., 2024; Wang et al., 2024a; Bai et al., 2023; Agrawal et al., 2024) shift the field toward zero-shot structured IE, where models can adapt to novel entities or schemas specified in user queries on-the-fly. However, to our knowledge, no existing VRDU benchmark supports evaluating such generalist models for generic extraction. ExStrucTiny directly addresses this by enabling comprehensive few-shot evaluation of VLMs for structured IE over document images.

Synthetic Data Generation for IE Recent studies have highlighted that LLMs can efficiently generate high-quality instruction following data, when properly prompted (Honovich et al., 2023; Peng et al., 2023; Taori et al., 2023; Wang et al., 2023b). While similar approaches have been successfully applied in text-only IE (Qi et al., 2024; NuMind; Chia et al., 2022), their use in VRDU is still emerging, as seen in works like Giovannini et al. (2025), which uses LLMs to create extractive questions from KEE annotations. However, as noted earlier, these methods inherit limitations from source annotations. Despite this, LLM-based data generation remains a promising avenue for scaling and enriching benchmarks (Kim et al., 2025), and we leverage this strategy to build ExStrucTiny.

[2] As suggested by the high scores achieved on DocVQA, InfographicVQA or ChartQA by small VLMs (Bai et al., 2025)

3 Methodology

3.1 Task Formulation

We mathematically formalize the structured extraction task in ExStrucTiny to highlight its distinction from traditional KEE, RE, and VQA in VRDU. Unlike prior benchmarks, ExStrucTiny requires models to extract and organize multiple pieces of information according to variable user-specified schemas, rather than a fixed ontology of entities. Each query-answer pair $(\mathbf{x}_i, \mathbf{y}_i)$ consists of:

- An input set of d document images $\{I_{i,j}\}_{j=1}^{d}$;
- An input prompt $\mathbf{q}_i$ (string), specifying the entities to extract, either in the form of descriptive text or of a schema to populate;
- A ground truth structured output $\mathbf{y}_i$ (string), which includes the verbatim text values $\{t_{i,k}\}_{k=1}^{n}$ extracted for the requested entities, their page indices $\{p_{i,k}\}_{k=1}^{n}$ ($p_{i,k} \in [\![1, d]\!]$), and bounding boxes $\{b_{i,k}\}_{k=1}^{n}$ ($b_{i,k} \in [\![0, 1000]\!]^4$).

The structured output can be expressed as follows:

$$\mathbf{y}_i = s_i\left(\{t_{i,k}\}_{k=1}^{n}, \{p_{i,k}\}_{k=1}^{n}, \{b_{i,k}\}_{k=1}^{n}\right)$$

where s_i is a function that populates the extracted values, page indices, and bounding boxes into a schema fulfilling the query $\mathbf{q}_i$. We call each $(t_{i,k}, p_{i,k}, b_{i,k})$ an *extraction leaf* of answer $\mathbf{y}_i$.

Given a vision-language model f_θ, the objective is to generate a string $\hat{\mathbf{y}}_i$ that can be parsed into a structured object containing extractions for the entities specified in $\mathbf{q}_i$:

$$\hat{\mathbf{y}}_i = f_\theta\left(\{I_{i,j}\}_{j=1}^{d}, \mathbf{q}_i\right)$$

with $\hat{\mathbf{y}}_i = \hat{s}_i(\{\hat{t}_{i,k}\}_{k=1}^{m}, \{\hat{p}_{i,k}\}_{k=1}^{m}, \{\hat{b}_{i,k}\}_{k=1}^{m})$.

Section 3.5 introduces metrics to evaluate model performance on structure prediction (s_i), text extraction ($\{t_{i,k}\}_{k=1}^{n}$), page localization ($\{p_{i,k}\}_{k=1}^{n}$), and bounding box prediction ($\{b_{i,k}\}_{k=1}^{n}$).

3.2 Small-Scale Manual Data Annotation

To construct an initial subset of ExStrucTiny, we manually annotate query-answer (QA) pairs supporting both closed and on-demand IE—which cover an adequate range of query specificity. More precisely, we include three query types:[3]

- **Closed with plain text:** explicitly requests in plain text the values of specific entities (e.g., *"Extract the name and role of the signers."*)

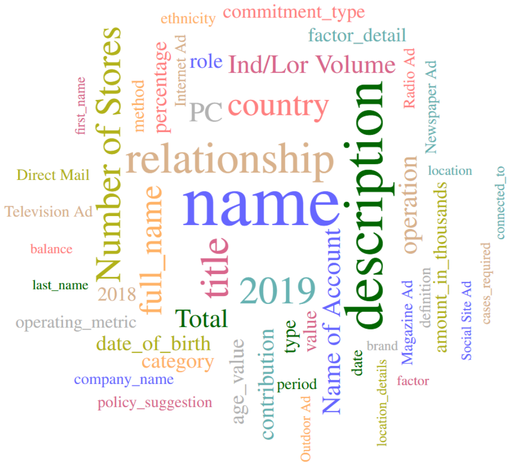


Figure 3: Example entities requested in ExStrucTiny

- **Closed with schema:** provides a JSON-formatted string schema with empty leaves to populate with the values of specific entities (e.g., *'[{"signer name": "", "signer role": ""}]'*)
- **On-demand with plain text:** implicitly asks for the values of child attributes (not known in advance) of parent entities (e.g., *"Extract all details about the signers."*).

We require **all answers to be in JSON format**, regardless of the query type. This enforces structured extraction, enables the representation of relationships between entities, and aligns with common industry practices for API and pipeline communication (where JSON format is prevalent). To minimize ambiguity in output structuring, we provide detailed annotation instructions (Appendix A) ensuring each extraction leaf $(t_{i,k}, p_{i,k}, b_{i,k})$ is represented as a leaf object `{"text":` $t_{i,k}$`,"page":` $p_{i,k}$`,"bbox":` $b_{i,k}$`}` in the JSON output. If an entity is not present in the document, its text, page, and bbox fields are set to `null`; if the value is empty but the entity is present (e.g., a blank form field), only the text value is `""`. For checkboxes, values are standardized as `"Yes"` or `"No"`. Example QA pairs are shown in Figure 2.[4]

To maximize variety in layout, length, and content, we source documents from the test splits of four existing datasets: FUNSD (Jaume et al. (2019), forms), TAT-DQA (Zhu et al. (2022), financial reports), SlideVQA (Tanaka et al. (2023), long slide decks), and VisualMRC (Tanaka et al. (2021), webpage screenshots). Four annotators each select three random documents from each dataset and generate ∼3 QAs of each type per document,

[3] We use "query type" and "QA type" interchangeably.

[4] A closed-with-plain-text counterpart of the middle example could be *"What are the cost cent id, dpt, and events (each with a type and date)?"*

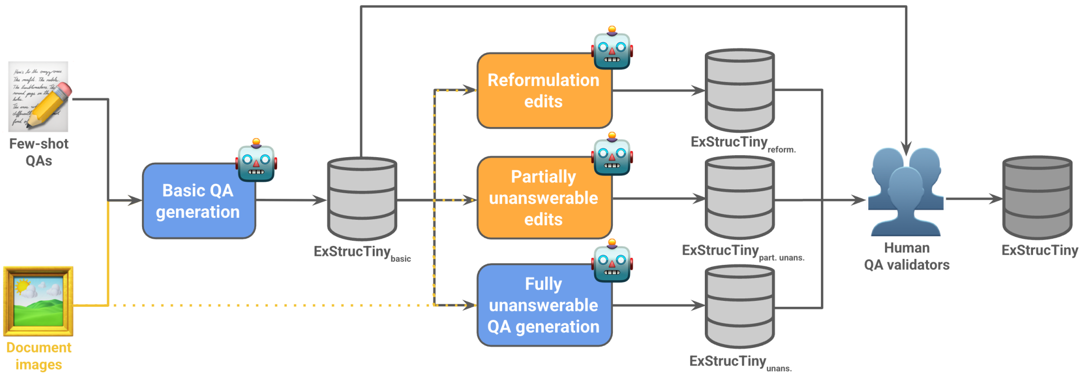


Figure 4: EXSTRUCTINY Synthetic Data Generation Pipeline

following strict guidelines to ensure realism and challenge: queries should (1) request multiple related entities, (2) include some entities likely to be missing—reflecting real-world scenarios—, (3) exhibit limited lexical overlap with document text, (4) target entities with values spanning multiple pages when possible, and (5) incorporate less common layout elements, such as charts and checkboxes. Annotators are also advised not to reuse annotations of the source datasets to avoid repetition and potential test leakage.

Created QAs are then reviewed by a separate annotator to ensure correct extractions as well as consistency with the output formatting and structuring instructions. This process resulted in 102 high-quality QA pairs, spanning a satisfactory range of layouts and extraction challenges.

3.3 Large-Scale Synthetic Data Generation

Manual annotation, while high-quality, does not scale efficiently for cheaply generating diverse and difficult QAs. To address this, we leverage `Gemini-2.5-Flash-Thinking` (Comanici et al., 2025), a competitive large VLM, to synthetically generate QAs and expand our benchmark.

Specifically, for each source dataset, we provide `Gemini` with thorough instructions of our task as well as few-shot examples from the manually-annotated QAs for this dataset (Section 3.2). The model is then iteratively prompted with new document images to generate QAs of each type, while avoiding duplication. We empirically observe that accompanying each shot with a chain-of-thought (CoT) explanation (Wei et al., 2022)—to detail the process behind the creation of each QA given its document—improves adherence to our output structuring guidelines. We also find that using a higher temperature (0.8) and a large reasoning budget (4096 tokens) helps strike a good balance between diversity and quality in the QAs generated.

As illustrated in Figure 4, the initial set of synthetic QAs is subsequently enriched by optionally sending it through different augmentation streams, where the VLM is reprompted to generate new and modified QAs with focused characteristics. On the one hand, we instruct `Gemini` to reformulate entity names in queries and their corresponding answers, thereby reducing lexical overlap with document text. On the other hand, we ask `Gemini` to introduce new entities that are absent from the source document, simulating realistic scenarios where requested information is missing. We also prompt `Gemini` to generate from scratch fully unanswerable QAs, where all requested entities are missing. These enrichments directly support our dataset desiderata by ensuring coverage of many real-world extraction cases.

Finally, to achieve a balanced and representative dataset, we subsample synthetic QAs from each pipeline stage, targeting proportions of 55% basic, 25% reformulated, 15% partially unanswerable, and 5% fully unanswerable QAs. These proportions are configurable hyperparameters of the QA generation pipeline and can be rebalanced to match specific testing objectives in the future.

3.4 Dataset Quality Validation

To guarantee the correctness and consistency of each synthetic QA instance, we conduct a manual validation exercise using a custom-built UI. Three expert validators with prior IE and VQA experience review up to 100 QAs each, focusing on five key areas: clarifying and grounding query text, ensuring extracted text matches the document verbatim, correcting page references, refining bounding boxes, and enforcing output structuring rules. Synthetic

Modification Type	Count	Proportion
ADD		
value_added	1186	23.0%
entity_added	294	5.7%
dict_added	242	4.7%
list_added	183	3.6%
Sub-total	1905	37.0%
DELETE		
value_deleted	400	7.8%
entity_deleted	87	1.7%
dict_deleted	79	1.5%
list_deleted	58	1.1%
Sub-total	624	12.1%
REPLACE		
value_change	2465	47.8%
query_type_change	115	2.2%
entity_renamed	44	0.9%
Sub-total	2624	50.9%
Total Modifications	**5153**	**100.0%**

Table 2: Hierarchical breakdown of edits done during the data validation exercise, by type and category.

Statistic	Manual	Synthetic
# documents	12	98
Avg. # QAs / doc	8.5	2.1
Avg. annot. + valid. time	50 min	16 min
# QAs	102	202
Closed w/ plain text	51	68
Closed w/ schema	25	68
On-demand	26	66

Table 3: Final manual vs. synthetic data comparison.

QAs were prioritized for review based on entity rarity, layout complexity, and number of entities (see Algorithm 1 in Appendix C).

Rather than rejecting borderline cases, validators were tasked with repairing errors in query text, extracted text, page references, bounding boxes, and answer schema whenever possible. QAs were discarded only when errors were irreparable, such as ungrounded questions or non-trivial malformed JSONs; in practice, only 2 samples were rejected.

After validation, we analyze the types and frequency of modifications done by validators. Overall, our budget allowed to validate 202 synthetic QAs kept in ExStrucTiny, with each requiring on average 25.5 edits. Table 2 categorizes edits into three main classes—additions, deletions, and replacements—with 11 sub-classes. Value changes (regarding the text, page, and bbox fields of the extraction leaves) and additions account for respectively 47.8% and 23% of all modifications, underscoring the importance of human intervention.

To further verify dataset quality, we perform cross-validation of edits among validators; we observe a high agreement rate of 96.2% at the extraction leaf level, indicating strong instruction adherence and robust dataset quality (more in App. C).

After combining manually-annotated and VLM-generated human-validated instances, ExStrucTiny totals 304 QAs over 110 documents, with an acceptably balanced proportion of closed with plain text, closed with schema, and on-demand QAs (Table 3). Figure 3 highlights the top 50 entities included in it and illustrates the variety of topics covered. Finally, Table 1 shows that, despite fewer documents than many KEE and VQA datasets in VRDU, ExStrucTiny exhibits lower query-document lexical overlap, more entities requested per query, and a reasonable rate of partial unanswerability. These characteristics pose unique challenges to VLMs for structured IE, which we analyze in detail in Section 4 with benchmark results covering multiple models.

3.5 Evaluation Metrics for Structured IE

To evaluate model predictions on ExStrucTiny, we assess performance across four dimensions for each query-answer pair $(\mathbf{x}_i, \mathbf{y}_i)$: structure prediction (s_i), text extraction $(\{t_{i,k}\}_{k=1}^{n})$, page localization $(\{p_{i,k}\}_{k=1}^{n})$, and bounding box prediction $(\{b_{i,k}\}_{k=1}^{n})$. The main challenge is that, despite detailed output structuring guidelines, VLMs may produce valid yet differently structured outputs, with possible variations in key names, extracted values, and schema organization. Indeed, multiple JSON structures may convey the same information, so exact string matching or JSON object equality is insufficient for a fair evaluation.

To address this, we propose a semantic mapping approach to align the extraction leaves $(\{t_{i,k}\}_{k=1}^{n}, \{p_{i,k}\}_{k=1}^{n}, \{b_{i,k}\}_{k=1}^{n})$ from the ground truth with those from the predicted output $(\{\hat{t}_{i,k}\}_{k=1}^{m}, \{\hat{p}_{i,k}\}_{k=1}^{m}, \{\hat{b}_{i,k}\}_{k=1}^{m})$. Specifically, we flatten both the ground truth and predicted structured objects—representing each leaf node by its path from the root (full entity name). These flattened objects are then input to a text-only LLM, which is few-shot prompted to generate a mapping from ground truth keys to predicted keys, account-

Model	Reason.	Map. R	Map. P	Map. F1	Total time (s)
`Llama-3.1-8b`	✗	0.960	0.784	0.863	**560**
`Phi-4-reasoning-14B`	✓	<u>0.967</u>	<u>0.924</u>	<u>0.945</u>	7741
`gpt-oss-20b`	✓	**0.996**	**0.956**	**0.976**	<u>1552</u>

Table 4: Recall, precision, and F1 for schema mappers.

ing for paraphrasing and structural differences. Unmatched ground truth keys are mapped to `null`, indicating missing extractions.

We benchmark three small text-only LLMs (Dubey et al., 2024; Abdin et al., 2025; Agarwal et al., 2025) for this mapping task essential to the metric computation, finding that reasoning models yield the best precision and recall (Table 4). More precisely, `gpt-oss-20b` provides near perfect performance for a reasonable runtime. More details on this comparison of text-only LLMs for mapping ground truth to predicted schemas are in Appendix D. We use `gpt-oss-20b` with a temperature of 0.95, top-K=10, and medium reasoning effort as our schema mapper in the rest of the paper. We recommend this setup to readers for consistency.

Once the mapping is established by the text-only LLM, we compute standard metrics to measure how well the predicted extractions cover the ground truth ones: recall, precision, and F1 for matched entities. Then, for each matched extraction leaf, we compute ANLS (Biten et al., 2019) for text similarity; page accuracy for localization; Intersection over Union (IoU, Everingham et al. (2010)) and normalized proximity (Nourbakhsh et al., 2025b) for bounding box overlap. If a ground truth extraction leaf is not matched, we assign zero scores for ANLS, page accuracy, IoU, and proximity. Additionally, we use normalized tree-edit distance (Zhang and Shasha, 1989) to compare the overall structure of predicted and ground truth objects.

This evaluation framework enables fair and comprehensive assessment of VLMs on ExStrucTiny—accounting for the inherent diversity in structured output generation.

4 Experiments

4.1 Settings

We benchmark closed (Comanici et al., 2025) and open VLMs (Bai et al., 2025; Kamath et al., 2025; Agrawal et al., 2024; Jiang et al., 2023; Du et al., 2025) of various sizes and architectures from multiple providers on ExStrucTiny. We use vLLM (Kwon et al., 2023) for inference on four NVIDIA L40S GPUs with a temperature of 0.2 for all open models. Evaluation uses metrics from Section 3.5, ensuring consistent, thorough comparison. Each VLM receives three shots from ExStrucTiny (one per supported query type) to illustrate our detailed task instructions.

4.2 Text Extraction Results

Closed Models Have an Edge Results in Table 5 show a substantial performance gap between closed-source and open-source models across all document classes. The highest performing closed model outscores the best open model by 18+ points.

Open Models Highlight the Benefit of Size We observe a clear positive correlation between performance and model size across all benchmarked models. This relationship is particularly evident within individual model families. For example, the `Qwen2.5-VL` family shows an increase in performance from 38.5 points with the 3B parameter model to 61.4 points with the 72B parameter model. Similarly, the `Gemma-3` family starts at 25 points with the 3B version and reaches 52.9 points with the 27B parameter version.

On-demand and Schema Struggles Table 5 reveals that, across model sizes, performance on closed-with-schema and on-demand QAs is consistently lower than on plain-text closed QAs. The difficulty of schema queries can be attributed to their length—they request approximately three times as many entities as plain-text queries. On-demand queries are even more challenging due to their underspecification, requiring models to interpret the query in-depth, resolve implicit references to entities or concepts, and align these with the image to produce the appropriate extractions. Consequently, on-demand queries are the most difficult overall, with the majority of models recording their lowest scores on these tasks across document types.

Open Models are Affected by Extraction Length Analysis of Figure 1 shows closed-source models maintain stable extraction quality even when the QAs expect 50+ extracted values, while open-source models' performance declines as extraction count increases. It is difficult to attribute this advantage to any specific factor, since little is known about closed models' training data, size, or training recipe. The improved robustness may result from progress across any or all of these dimensions (e.g.,

	Forms			Financial Reports			Slide Decks			Web Pages					
Model	text	schema	on-dem.	text	schema	on-dem.	text	schema	on-dem.	text	schema	on-dem.	**Manual**	**Synthetic**	**Avg.**
Qwen2.5-VL-3B	61.4	48.3	47.1	50.4	56.2	21.5	55.5	40.4	32.4	46.5	21.9	24.2	35.0	40.1	38.6
Gemma-3-4B	64.8	48.3	25.3	37.3	46.6	12.6	33.9	27.3	9.9	25.8	16.6	6.0	24.3	25.3	25.0
Qwen2.5-VL-7B	74.7	60.4	58.0	68.3	63.1	32.4	64.0	61.5	33.1	45.6	35.9	26.8	45.3	50.4	48.8
Gemma-3-12B	69.3	68.0	66.2	70.5	59.8	36.4	69.7	49.3	24.2	46.4	21.2	19.5	40.7	45.8	44.3
Pixtral-12B	70.5	60.6	74.9	70.6	47.1	40.4	68.0	53.1	42.9	8.9	4.8	5.4	37.0	44.5	42.3
Kimi-VL-A3B-16B	67.3	64.5	67.9	77.6	51.8	29.5	49.3	42.1	40.3	41.1	10.5	15.5	32.4	44.8	41.0
Qwen2.5-VL-32B$_{text}$	73.0	57.0	39.0	78.1	72.9	**57.3**	55.1	50.0	28.7	61.3	50.7	29.9	48.1	50.7	49.9
Qwen2.5-VL-32B	67.0	**79.3**	66.8	67.1	66.4	39.6	45.0	69.8	37.4	49.3	48.3	38.5	40.6	61.3	55.1
Gemma-3-27B	**75.7**	70.5	70.6	70.6	61.8	52.7	**83.7**	64.3	36.5	50.0	24.6	21.6	50.0	54.1	52.9
Mistral-Small-3.2-24B	75.0	78.5	80.2	**88.9**	**78.5**	41.5	83.5	**85.7**	**47.2**	18.9	12.7	7.4	45.0	60.3	55.8
Qwen2.5-VL-72B-FP8	74.5	70.9	**80.3**	83.0	70.2	56.9	78.7	63.5	42.4	**67.1**	**53.2**	**39.6**	**58.8**	**62.5**	**61.4**
Gemini-2.5-Flash	82.1	**86.9**	**85.9**	88.7	80.9	55.8	78.2	**86.2**	71.5	68.1	65.7	43.6	67.8	76.1	73.6
Gemini-2.5-Pro	**85.3**	83.2	75.0	**91.0**	**82.5**	**80.0**	**91.5**	84.4	**80.4**	**74.7**	**72.0**	**63.0**	**81.2**	**78.8**	**79.5**

Table 5: Model performance on text extraction (ANLS) across document types and query types.

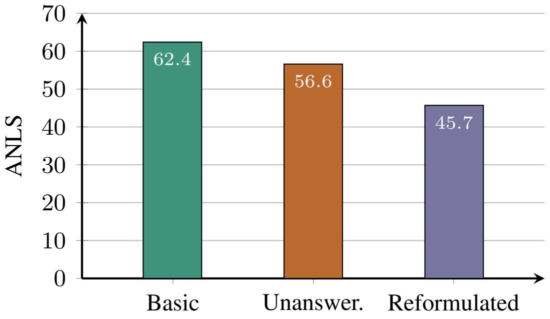


Figure 5: Extraction performance breakdown on synthetic QAs: basic vs. fully & partially unanswerable vs. reformulated (averaged across VLMs).

Table 5 shows model size alone can have substantial impact within model families).

Manual vs. Synthetic Data Table 5 also indicates a modest performance gap between manually generated data and synthetic data. Models score on average 13.6% lower on manually-authored QAs, suggesting they remain slightly more difficult than synthetic QAs. Crucially, there is no evidence of bias favoring Gemini-2.5-Flash, despite its role in drafting the synthetic data. This may be explained by the rigorous manual editing and validation performed on all Gemini-generated entries and detailed in Section 3.4. Finally, neither split is saturated: the best model reaches 81.2% ANLS on the manual subset and 78.8% on the synthetic one.

Reformulations and Unanswerable Requests in Synthetic Data Figure 5 breaks down extraction performance by synthetic category: basic queries; fully and partially unanswerable ones; reformulated ones (see pipeline in Figure 4). VLMs seem to struggle to correctly identify when a requested entity is missing, as shown by a lower score on fully and partially unanswerable synthetic queries compared to basic ones. Similarly, ANLS drops by nearly 27% on reformulated queries relative to basic ones, highlighting sensitivity to synonymy and reduced lexical overlap with source documents. These results underscore the value of augmenting initial Gemini generations through multi-stage edits to increase both the realism and the difficulty of ExStrucTiny.

Text-only Baseline To quantify the contribution of visual information in ExStrucTiny, we construct a text-only baseline by applying OCR to each document and evaluating Qwen2.5-VL-32B using only the text tokens extracted. Compared with the image-only configuration, the text-only variant yields a ∼10% lower ANLS overall—indicating visual cues materially improve structured IE performance in this benchmark (especially for on-demand QAs).

4.3 Answer Localization Results

While ANLS measures textual extraction quality, we also assess grounding (Nourbakhsh et al., 2025a; T.y.s.s et al., 2025) via localization metrics: bounding-box intersection over union (IoU), normalized box proximity, and page accuracy (see Section 3.5). Table 6 reveals both open and closed-source models struggle with localization (maximum IoU of 14.4%) and page identification (best accuracy of 84.3%). Moreover, we observe that correct text predictions do not reliably entail correct spatial grounding, indicating a calibration gap between extraction and evidence localization. This difficulty may reflect the continuous, high-variance nature of bounding-box prediction and sensitivity to layout scale and spatial relations, which remain challenging for current VLMs. These limitations warrant caution in industry applications where precise localization of evidence is required, even when textual extractions are accurate.

Model	Page acc.	Bbox IoU	Bbox prox.
`Qwen2.5-VL-3B`	14.8	1.6	20.2
`Gemma-3-4B`	15.7	0.9	16.0
`Qwen2.5-VL-7B`	24.9	3.1	27.1
`Gemma-3-12B`	41.9	4.1	37.3
`Pixtral-12B`	39.9	2.5	38.4
`Kimi-VL-A3B-16B`	22.7	1.2	2.4
`Qwen2.5-VL-32B`	57.2	**11.2**	51.7
`Gemma-3-27B`	48.7	4.3	43.7
`Mistral-Small-3.2-24B`	**60.7**	5.4	53.2
`Qwen2.5-VL-72B-FP8`	59.6	10.3	**54.5**
`Gemini-2.5-Flash`	74.2	13.4	71.9
`Gemini-2.5-Pro`	**84.3**	**14.4**	**74.8**

Table 6: Model performance on page accuracy, bounding box IoU, and bounding box proximity.

Model	Valid Python	Valid Ext.	Map. R	Map. P	Map. F1	Tree Sim.
`Qwen2.5-VL-3B`	93.4	37.2	65.1	68.5	62.0	57.6
`Gemma-3-4B`	96.3	65.9	59.6	62.4	57.0	56.3
`Qwen2.5-VL-7B`	98.3	53.5	72.3	77.1	71.3	69.4
`Gemma-3-12B`	**100.0**	88.2	72.6	70.9	67.5	72.0
`Pixtral-12B`	99.3	94.4	69.9	73.8	68.2	67.0
`Kimi-VL-A3B-16B`	99.0	5.1	68.3	68.9	63.4	57.0
`Qwen2.5-VL-32B`	98.0	95.1	73.7	75.2	71.9	67.0
`Gemma-3-27B`	98.7	90.6	76.8	72.8	71.3	**77.1**
`Mistral-Small-3.2-24B`	99.3	**99.4**	79.7	77.3	75.0	70.0
`Qwen2.5-VL-72B-FP8`	99.0	88.8	**80.5**	**83.8**	**79.5**	72.3
`Gemini-2.5-Flash`	98.0	98.6	87.3	81.5	81.5	72.6
`Gemini-2.5-Pro`	**100.0**	**99.8**	**88.1**	**82.8**	**83.1**	**78.3**

Table 7: Model perf. on format and extraction leaf validity, mapping recall/precision/F1, and tree edit similarity.

4.4 Output Structure Results

Larger Models Yield More Valid Outputs To ensure structured data quality, we check if (1) model outputs are parsable in Python and if (2) JSON-formatted leaves follow our custom extraction rules. Table 7 shows all models generate Python-executable structures, but only those with at least 12B parameters produce enough valid extraction leaves. `Kimi-VL-A3B-16B` is an exception, with only 5.1% valid extraction leaves.

Closed Models Recall More Entities Table 7 also presents schema mapping results (see Section 3.5) for each model's output. Closed VLMs extract more requested entities—`Gemini-2.5-Pro` and `Gemini-2.5-Flash` achieving recall scores of 88.1% and 87.3%. For open-source models, recall seems to improve with scale, suggesting that further increasing open VLM size may help cover more expected extractions and narrow the gap.

Structural Prediction Quality Tree edit similarity measures how closely the structure of a predicted output resembles that of the ground truth JSON answer. Table 7 shows both closed VLMs and certain large open models achieve high alignment with the ground truth. While a closed model achieves the highest scores, several open models also perform strongly, with `Gemma-3-27B` and `Qwen2.5-VL-72B` among the top performers.

4.5 Answer Context Breakdown

Table 8 in Appendix E shows that models consistently struggle in chart and free-text contexts, with 11 out of 12 models recording their lowest scores in one of these settings. Lower performance on charts was expected: charts are less structured and include visual attributes (e.g., colors, shapes) that can act as distractors, and chart parsing and understanding remain challenging for current VLMs (Wang et al., 2024b). As for free text, this might be due to certain queries targeting values embedded within dense pages containing substantial non-answer content and plausible near-matches, increasing the risk of false positives and hindering extraction.

5 Conclusion

We introduce ExStrucTiny, a benchmark dataset designed to evaluate structured IE from visually rich documents using diverse, challenging queries—including closed, schema-based, and on-demand formats—across varied document types and contexts. By expecting JSON-formatted outputs and including multi-entity, cross-page, and unanswerable queries, ExStrucTiny addresses key limitations of existing benchmarks and better reflects real-world extraction needs. Our experiments show that closed-source VLMs currently lead in recall and robustness, while larger models generally produce more valid outputs. However, all models face difficulties with intricate queries and answer localization, especially in less structured contexts like charts and free text. These findings highlight persistent gaps in current model capabilities. We hope this work spurs further research into more adaptable, robust, and semantically grounded extraction models for real-world applications.

Limitations

While ExStrucTiny provides a rigorous testbed for structured IE, certain limitations remain. First, our use of ANLS for text evaluation is suboptimal for certain data types, such as numbers and dates, where character-based similarity may not accurately reflect extraction quality (Nourbakhsh et al., 2025b); future work should explore alternative metrics tailored to these cases (26% of all values annotated). Second, ExStrucTiny is currently English-only and does not assess IE performance on documents in other languages. Finally, the schema mapping stage essential to our metric computation relies on a text-only LLM to align ground truth and predicted extraction leaves. Although effective, this approach is slower than programmatic solutions and may not yield perfect mappings. Addressing these limitations in future work will further strengthen this benchmark and its utility for the broader document intelligence research.

Disclaimer

This paper was prepared for informational purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

Marah I Abdin, Sahaj Agarwal, Ahmed Awadallah, Vidhisha Balachandran, Harkirat S. Behl, Lingjiao Chen, Gustavo de Rosa, Suriya Gunasekar, Mojan Javaheripi, Neel Joshi, Piero Kauffmann, Yash Lara, Caio César Teodoro Mendes, Arindam Mitra, Besmira Nushi, Dimitris Papailiopoulos, Olli Saarikivi, Shital Shah, Vaishnavi Shrivastava, and 4 others. 2025. Phi-4-reasoning technical report. *CoRR*, abs/2504.21318.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sébastien Bubeck, Che Chang, Kai Chen, and 105 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *CoRR*, abs/2508.10925.

Pravesh Agrawal, Szymon Antoniak, Emma Bou Hanna, Baptiste Bout, Devendra Singh Chaplot, Jessica Chudnovsky, Diogo Costa, Baudouin De Monicault, Saurabh Garg, Théophile Gervet, Soham Ghosh, Amélie Héliou, Paul Jacob, Albert Q. Jiang, Kartik Khandelwal, Timothée Lacroix, Guillaume Lample, Diego de Las Casas, Thibaut Lavril, and 23 others. 2024. Pixtral 12b. *CoRR*, abs/2410.07073.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *Preprint*, arXiv:2308.12966.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Ming-Hsuan Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025. Qwen2.5-vl technical report. *CoRR*, abs/2502.13923.

Ali Furkan Biten, Rubèn Tito, Andrés Mafla, Lluís Gómez i Bigorda, Marçal Rusiñol, C. V. Jawahar, Ernest Valveny, and Dimosthenis Karatzas. 2019. Scene text visual question answering. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 4290–4300. IEEE.

Yew Ken Chia, Lidong Bing, Soujanya Poria, and Luo Si. 2022. Relationprompt: Leveraging prompts to generate synthetic data for zero-shot relation triplet extraction. In *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 45–57. Association for Computational Linguistics.

Jinho Choi, Sanghun Jung, Deok Gun Park, Jaegul Choo, and Niklas Elmqvist. 2019. Visualizing for the non-visual: Enabling the visually impaired to use visualization. *Comput. Graph. Forum*, 38(3):249–260.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit S. Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, Luke Marris, Sam Petulla, Colin Gaffney, Asaf Aharoni, Nathan Lintz, Tiago Cardal Pais, Henrik Jacobsson, Idan Szpektor, Nan-Jiang Jiang, and 81 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *CoRR*, abs/2507.06261.

Lei Cui, Yiheng Xu, Tengchao Lv, and Furu Wei. 2021. Document ai: Benchmarks, models and applications. *Preprint*, arXiv:2111.08609.

Angang Du, Bohong Yin, Bowei Xing, Bowen Qu, Bowen Wang, Cheng Chen, Chenlin Zhang, Chenzhuang Du, Chu Wei, Congcong Wang, Dehao Zhang, Dikang Du, Dongliang Wang, Enming Yuan, Enzhe Lu, Fang Li, Flood Sung, Guangda Wei, Guokun Lai, and 73 others. 2025. Kimi-vl technical report. *CoRR*, abs/2504.07491.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, and 82 others. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. 2010. The pascal visual object classes (VOC) challenge. *Int. J. Comput. Vis.*, 88(2):303–338.

Simone Giovannini, Fabio Coppini, Andrea Gemelli, and Simone Marinai. 2025. Boundingdocs: a unified dataset for document question answering with spatial annotations. *CoRR*, abs/2501.03403.

Jiabang He, Lei Wang, Yi Hu, Ning Liu, Hui Liu, Xing Xu, and Heng Tao Shen. 2023. Icl-d3ie: In-context learning with diverse demonstrations updating for document information extraction. *Preprint*, arXiv:2303.05063.

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2023. Unnatural instructions: Tuning language models with (almost) no human labor. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 14409–14428. Association for Computational Linguistics.

Anwen Hu, Haiyang Xu, Liang Zhang, Jiabo Ye, Ming Yan, Ji Zhang, Qin Jin, Fei Huang, and Jingren Zhou. 2024. mplug-docowl2: High-resolution compressing for ocr-free multi-page document understanding. *Preprint*, arXiv:2409.03420.

Guillaume Jaume, Hazim Kemal Ekenel, and Jean-Philippe Thiran. 2019. Funsd: A dataset for form understanding in noisy scanned documents. *Preprint*, arXiv:1905.13538.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

Yizhu Jiao, Ming Zhong, Sha Li, Ruining Zhao, Siru Ouyang, Heng Ji, and Jiawei Han. 2023. Instruct and extract: Instruction tuning for on-demand information extraction. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 10030–10051. Association for Computational Linguistics.

Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean-Bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, and 191 others. 2025. Gemma 3 technical report. *CoRR*, abs/2503.19786.

Geewook Kim, Teakgyu Hong, Moonbin Yim, Jeongyeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han, and Seunghyun Park. 2022. Ocr-free document understanding transformer. *Preprint*, arXiv:2111.15664.

Seungone Kim, Juyoung Suk, Xiang Yue, Vijay Viswanathan, Seongyun Lee, Yizhong Wang, Kiril Gashteovski, Carolin Lawrence, Sean Welleck, and Graham Neubig. 2025. Evaluating language models as synthetic data generators. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 6385–6403. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM.

Jordy Van Landeghem, Rubén Tito, ukasz Borchmann, Micha Pietruszka, Pawe Józiak, Rafa Powalski, Dawid Jurkiewicz, Mickaël Coustaty, Bertrand Ackaert, Ernest Valveny, Matthew Blaschko, Sien Moens, and Tomasz Stanisawek. 2023. Document understanding dataset and evaluation (dude). *Preprint*, arXiv:2305.08455.

Kenton Lee, Mandar Joshi, Iulia Raluca Turc, Hexiang Hu, Fangyu Liu, Julian Martin Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. 2023. Pix2struct: Screenshot parsing as pretraining for visual language understanding. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 18893–18912. PMLR.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq R. Joty, and Enamul Hoque. 2022. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 2263–2279. Association for Computational Linguistics.

Minesh Mathew, Viraj Bagal, Rubèn Tito, Dimosthenis Karatzas, Ernest Valveny, and C. V. Jawahar. 2022. Infographicvqa. In *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022, Waikoloa, HI, USA, January 3-8, 2022*, pages 2582–2591. IEEE.

Minesh Mathew, Dimosthenis Karatzas, and C. V. Jawahar. 2021. Docvqa: A dataset for vqa on document images. *Preprint*, arXiv:2007.00398.

Armineh Nourbakhsh, Siddharth Parekh, Pranav Shetty, Zhao Jin, Sameena Shah, and Carolyn Rose. 2025a. Where is this coming from? Making groundedness count in the evaluation of Document VQA models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 5326–5346, Albuquerque, New Mexico. Association for Computational Linguistics.

Armineh Nourbakhsh, Siddharth Parekh, Pranav Shetty, Zhao Jin, Sameena Shah, and Carolyn P. Rosé. 2025b. Where is this coming from? Making groundedness count in the evaluation of Document VQA models. In *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pages 5326–5346. Association for Computational Linguistics.

NuMind. Nuextract llm.

Seunghyun Park, Seung Shin, Bado Lee, Junyeop Lee, Jaeheung Surh, Minjoon Seo, and Hwalsuk Lee. 2019. Cord: A consolidated receipt dataset for post-ocr parsing.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with GPT-4. *CoRR*, abs/2304.03277.

Vincent Perot, Kai Kang, Florian Luisier, Guolong Su, Xiaoyu Sun, Ramya Sree Boppana, Zilong Wang, Zifeng Wang, Jiaqi Mu, Hao Zhang, Chen-Yu Lee, and Nan Hua. 2024. LMDX: Language model-based document information extraction and localization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15140–15168, Bangkok, Thailand. Association for Computational Linguistics.

Rafal Powalski, Lukasz Borchmann, Dawid Jurkiewicz, Tomasz Dwojak, Michal Pietruszka, and Gabriela Palka. 2021. Going full-tilt boogie on document understanding with text-image-layout transformer. In *16th International Conference on Document Analysis and Recognition, ICDAR 2021, Lausanne, Switzerland, September 5-10, 2021, Proceedings, Part II*, volume 12822 of *Lecture Notes in Computer Science*, pages 732–747. Springer.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. 2024. ADELIE: Aligning large language models on information extraction. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7371–7387, Miami, Florida, USA. Association for Computational Linguistics.

Alexander Michael Rombach and Peter Fettke. 2025. Deep learning based key information extraction from business documents: Systematic literature review. *ACM Comput. Surv.*, 58(2).

Brandon Smock, Rohith Pesala, and Robin Abraham. 2022. Pubtables-1m: Towards comprehensive table extraction from unstructured documents. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 4624–4632. IEEE.

Ryota Tanaka, Kyosuke Nishida, Kosuke Nishida, Taku Hasegawa, Itsumi Saito, and Kuniko Saito. 2023. Slidevqa: A dataset for document visual question answering on multiple images. *Preprint*, arXiv:2301.04883.

Ryota Tanaka, Kyosuke Nishida, and Sen Yoshida. 2021. Visualmrc: Machine reading comprehension on document images. *Preprint*, arXiv:2101.11272.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://crfm.stanford.edu/2023/03/13/alpaca.html`.

Santosh T.y.s.s, Youssef Tarek Elkhayat, Oana Ichim, Pranav Shetty, Dongsheng Wang, Zhiqiang Ma, Armineh Nourbakhsh, and Xiaomo Liu. 2025. CoCoLex: Confidence-guided copy-based decoding for grounded legal text generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vienna, Austria. Association for Computational Linguistics.

Dongsheng Wang, Natraj Raman, Mathieu Sibue, Zhiqiang Ma, Petr Babkin, Simerjot Kaur, Yulong Pei, Armineh Nourbakhsh, and Xiaomo Liu. 2024a. DocLLM: A layout-aware generative language model for multimodal document understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8529–8548, Bangkok, Thailand. Association for Computational Linguistics.

Dongsheng Wang, Ran Zmigrod, Mathieu J. Sibue, Yulong Pei, Petr Babkin, Ivan Brugere, Xiaomo Liu, Nacho Navarro, Antony Papadimitriou, William Watson, Zhiqiang Ma, Armineh Nourbakhsh, and Sameena Shah. 2025. BuDDIE: A business document dataset for multi-task information extraction. In *Proceedings of the Joint Workshop of the 9th Financial Technology and Natural Language Processing (FinNLP), the 6th Financial Narrative Processing (FNP), and the 1st Workshop on Large Language Models for Finance and Legal (LLMFinLegal)*, pages 35–47, Abu Dhabi, UAE. Association for Computational Linguistics.

Xiao Wang, Weikang Zhou, Can Zu, Han Xia, Tianze Chen, Yuansen Zhang, Rui Zheng, Junjie Ye, Qi Zhang, Tao Gui, Jihua Kang, Jingsheng Yang, Siyuan Li, and Chunsai Du. 2023a. Instructuie: Multi-task instruction tuning for unified information extraction. *Preprint*, arXiv:2304.08085.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023b. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13484–13508. Association for Computational Linguistics.

Zilong Wang, Yichao Zhou, Wei Wei, Chen-Yu Lee, and Sandeep Tata. 2023c. VRDU: A benchmark for visually-rich document understanding. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, pages 5184–5193. ACM.

Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, Alexis Chevalier, Sanjeev Arora, and Danqi Chen. 2024b. Charxiv: Charting gaps in realistic chart understanding in multimodal llms. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. 2020. Layoutlm: Pre-training of text and layout for document image understanding. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 1192–1200. ACM.

Kaizhong Zhang and Dennis E. Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262.

Fengbin Zhu, Wenqiang Lei, Fuli Feng, Chao Wang, Haozhou Zhang, and Tat-Seng Chua. 2022. Towards complex document understanding by discrete reasoning. In *MM '22: The 30th ACM International Conference on Multimedia, Lisboa, Portugal, October 10 - 14, 2022*, pages 4857–4866. ACM.

Ran Zmigrod, Pranav Shetty, Mathieu Sibue, Zhiqiang Ma, Armineh Nourbakhsh, Xiaomo Liu, and Manuela Veloso. 2024. "What is the value of templates?" Rethinking Document Information Extraction Datasets for LLMs". In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 13162–13185. Association for Computational Linguistics.

A Data Generation Instructions

```
# GUIDELINES
Below are guidelines to follow in order to generate query-answer pairs based on the images of an input document
    :
## Query:
- The "query" field of your output elements is a string.
- Queries can pertain to one of three types of extractive queries (for now, fully open information extraction
    queries where entire documents are requested to be parsed in a structured format are out of scope):
(a) **Closed IE with plain text:** the query directly mentions in a sentence/in the form of plain text the
    names of the entities to extract (ex: "what is the SSN of the signers?" => [{"signer SSN": "123456"}]). **
    These entity names mentioned in the query should not always perfectly overlap the vocabulary/language used
     in the document, to create some difficulty in the questions and force some semantic understanding of the
    entity names**! (Ex: if the document contains "study name", it might be better to ask "Give me the topic
    of the report" instead of something very straightforward and template-based such as "What is the name of
    the study?").
(b) **Closed IE with schema:** the query directly mentions in the form of a schema the names of the entities to
     extract (ex: "{"signer SSN": ""}" => {"signer SSN": "123456"}). **These entity names mentioned in the
    query should not always perfectly overlap the vocabulary/language used in the document, to create some
    difficulty in the questions and force some semantic understanding of the entity names**! (Ex: if the
    document contains "study name", it might be better to request "{"report_topic": ""}" instead of something
    very straightforward like "{"study name": ""}").
(c) **On-demand IE:** the query does not directly mention the name of all the entities to extract but instead
    mentions a *parent entity name* to guide the extraction (ex: "parse all the dependents' information" =>
    [{"name": "Leo", "age": "12"}]). In this case, the answer will have to propose a schema somewhat inspired
    by the layout/structure of the document and populate it with values of children entities related to the
    parent entity requested in the query. Note that on-demand queries should remain relatively **vague** about
     the exact children entities that should appear in the answer (ex: "Extract all available information
    regarding ..."; **you should not** further specify this query by adding "..., including childrenAttribute1
    , childrenAttribute2, and childrenAttribute3", as these are not supposed to be known in advance). It
    should be up to the model tasked with answering the on-demand queries (not you) to uncover what the query
    exactly wants based on the parent-children implicit relationships between the entities presented in the
    document (ex: a person --parent entity-- can have a name, a phone number, an address, a role, etc. --
    children attributes--).
- Note: you can also create queries about information not present in the document, in which case the answer
    should contain a corresponding None value for the "text" field (see below) or should just be an empty list
    .
- Text queries can also be simple declarative sentences (ex: "Extract all information about ..."), besides
    questions.
- Try to create queries about many relevant/connected entities at the same time, and avoid lexical overlap with
     the document, as requested above.
## Answer:
- The "answer" field of your output elements should be a list of nested dictionaries that contain all the user-
    requested extractions expressed in the query, whenever the query is not a schema. If the query is an empty
     schema, then the answer object should just adhere to this schema and populate it with extraction
    dictionaries (see below).
- Extractions that compose the answer should be returned in the order of (1) appearance of the entities
    mentioned in the query, and (2) for multiple extractions of the same underlying type of entity, in order
    of appearance in the document when considering it in reading order (left to right, top to bottom).
- Dictionaries within the "answer" field can either be sub-dictionaries or sub-lists for hierarchical entities
    as well as for line items/templated items (ex: [{"name": "Leo", hobbies: [{"name": "guitar", "duration":
    "2h"}]}]), or they can be **extraction dictionaries** containing verbatim text that directly comes from
    the document tokens. The schema of the extraction dictionaries is: {"text": str, "page": int, "bbox": [int
    ], "answer_context": Enum({"free_text", "templated_items", "individual_key_value", "chart", "image", "
    checkbox"})}. The extraction dictionaries are the final leaf nodes of the answer structure; they contain
    the actual text values extracted from the document. For the sake of clarity, we will replace these
    extraction dictionaries with their extraction text values in our examples of the "# GUIDELINES" section.
- A line item or templated item can be defined as a set of fields all referring to the same underlying parent
    entity (ex: if someone named X with role Y signed the document at date Z, then the corresponding line item
    /templated item is composed of the following child entities: {"name": X, "role": Y, "date": Z}). For
    example, if you generate a closed IE query that asks for the age and name of all the signers of a form,
    the answer could have the following structure: [{"age": W, "name": X}, {"age": Y, "name": Z}, ...]. In
    contrast, if the closed IE query had been "extract the document ID as well as the amount requested in the
    form", it does not really make sense to present the answer as [{"document ID": X, "amount requested": Y}]
    since "document ID" and "amount requested" are not directly related. Thus, the semantic relatedness of
    entities should influence the output structure (e.g., whether they are grouped in the same parent
    dictionary or not).
- The keys of the dictionaries that compose the answer (but are not "extraction dictionaries") should be, in
    order of preference (as long as a meaningful key name is proposed):
(a) the exact schema key names for closed IE user queries that directly mention a schema to populate
(b) strings that minimally overlap the user-mentioned entity names (for plain text closed queries; ex: "what is
     the SSN of the signers?", "[{"signer SSN": "123456"}]")
(c) strings coming from the document tokens (for on-demand/open queries; ex: "parse all the dependents
    information" => [{"name": "Leo", "age": "12"}]; also for certain plain text closed queries if it is hard
    to overlap an appropriate key name with the user query)
(d) strings you suggest based on your understanding of the values being extracted and your internal knowledge (
    for on-demand/open queries; e.g., when the string value to extract is floating text and not an actual key-
    value field from the doc, where the key name could be repeated verbatim; also relevant for certain closed
    queries if it is hard to overlap an appropriate key name with the user query).
- When you identify that the query requests line items/templated items in the answer, there is no need to name
    the keys of each item of the answer using XXX_1, XXX_2, XXX_3, etc; the **numeral ordering is useless**.
    For instance, [{"edit_1": "add"}, {"edit_2": "remove"}, {"edit_3": "modify"}] should instead be formatted
    as [{"edit": "add"}, {"edit": "remove"}, {"edit": "modify"}]; the numbers appended to key names are
    useless if the keys are returned in reading order of the doc.
- Null values VS. non-found values in the document: for closed queries (both plain text and schema-based), if
    one of the requested keys cannot be found, you should yield this extraction dictionary for its
    corresponding leaf node in the output structure: {"text": None, "page": None, "bbox": None}. If all
    requested keys cannot be found in the document, then you should return an empty answer list. In contrast,
    if one of the child entities of the parent entity requested comes from a key-value field with an empty/non
    -filled value in the doc (e.g., "N/A"), you should yield an extraction dictionary like the following for
    its corresponding leaf node: {"text": "", "page": int, "bbox": [int], "answer_context": Enum({"free_text",
     "templated_items", "individual_key_value", "chart", "image", "checkbox"})}}. In short, **N/As (empty/non-
    filled values) should be extracted as "" text values**. However, not finding requested info in a doc
    should yield {"text": None, "page": None, "bbox": None} extraction dictionaries or an empty list.
- Checkboxes: checkboxes can be seen as key-values too; user-requested checkboxes that are unchecked should
    have "No" as a value, and checked ones should have "Yes".
- For on-demand/open queries, two extraction dictionaries about the same underlying entity should not be merged
     together if they dont appear as one single line item/templated item in the document (ex: [{"name": "Leo S
    ", "age": "14"}, {"name": "Leo S", "grade": "5"}] stays like that if they are present as two different
    line items/templated items of the same document).
## query_type:
- string field (enum); can only be one of "closed_with_plain_text", "closed_with_schema", "on_demand".
```

B Dataset Comparison Metrics

We compare ExStrucTiny with popular KEE and VQA datasets using a common set of QA-level statistics. Because KEE and VQA use different annotation formats, we harmonize them into the same view before computing metrics in Table 1.

KEE recasting For a KEE test set of size n having an entity ontology of size k, we report statistics for two different recasting scenarios:

- **Document-level queries:** one query per document which always requests the same k entities at once; total #QAs $= n$. Example: *"Extract the post town, postcode, and street line."*
- **Entity-level queries:** k queries per document, i.e. one per entity of the ontology (as in Hu et al. (2024); Wang et al. (2024a)); total #QAs $= nk$. Example: *"Extract the post town."* + *"Extract the postcode."* + *"Extract the street line."*.

Results for KEE datasets in Table 1 show non-parenthesized values for the document-level recasting scenario and parenthesized values for the entity-level recasting scenario.

VQA adaptation We retain only *extractive* QAs for the analysis of VQA test sets, i.e., QAs with all annotated answers appearing in the document text tokens. To estimate the number of entities referenced per question, a text-only model (`gpt-oss-20b`) parses entity mentions from the query. Example: *"What is the invoice number?"* $\rightarrow$ {*"invoice number"*};
"What are the subtotal and tax amounts?" $\rightarrow$ {*"subtotal"*, *"tax"*}.

Statistics We define below the metrics computed:

- **Number of documents (*#Docs*):** Count of documents in the test split.
- **Number of extractive QA pairs (*#QAs*):** Count of question-answer pairs for which the answer values can be found verbatim in the document.
- **Average document length (*Len(Doc)*):** Average word count per document.
- **Average query length (*Len(Q)*):** Average word count per query.
- **Average answer length (*Len(A)*):** Average word count per answer (when aggregating all extracted values annotated for it).
- **Average count of entities requested per query (*#Entities per Q*):** Average number of entity named in each query; obtained by parsing the query with an LLM for VQA datasets.
- **Average number of extracted values in the answer (*#Values per A*):** Average count of answer values annotated for each QA.
- **Percentage of queries requesting some missing entities (*%Q Unansw.*):** Measures how often documents lack values for at least one of the requested entities from the query.
- **Lexical overlap (*Q-Doc Overlap*):** Proportion of overlapping tokens between the query and the document text.
- **Normalized ontology size ($\overline{|\textit{Ontology}|}$):** Ratio of the number of unique entities in the ontology to the total number of QA pairs in the test split.

Interpretation KEE datasets, when transformed with document-level queries, yield few QAs, small ontologies, and near-ubiquitous partially unanswerables (almost no document contains values for every single entity of the ontology)—all indicating limited query diversity and unrealistic coverage. When transformed with entity-level queries, KEE datasets have very short queries, single-entity requests, few answers, high query-document overlap, and an extremely low normalized ontology size due to the large number of QAs. VQA datasets instead show normalized ontology sizes near 1 (each question targets a distinct entity) but at the cost of high lexical overlap and predominantly single-entity, single-value queries. Only DUDE contains a notable fraction of fully unanswerable QAs. In contrast, ExStrucTiny offers a middle ground: longer multi-entity queries, many answer values expected per query, high entity diversity, a realistic proportion of queries with missing entities, and comparatively lower lexical overlap.

C Dataset Annotation Details

Small-Scale Manual Data Annotation For the manual split of ExStrucTiny, the validation process involved an initial review after each human annotator (four in total) had completed approximately half of their QAs—review during which a validator provided feedback and suggested adjustments based on the annotation instructions. Annotators then revised their work accordingly and completed the remaining QAs. Finally, the validator conducted a comprehensive review of all manually-annotated QAs to address any remaining issues. The validator was one of the main authors of the annotation instructions and reviewed all 102 manually generated QAs to ensure consistency.

Algorithm 1: Target-Guided Importance Sampling in Feature Space

```
Input: Dataset $\mathcal{D} = \{x_i\}_{i=1}^N$ with $x_i \in \mathbb{R}^k$ (metadata vector of QA $i$); desiderata $\mathcal{S} = \{(d_j, w_j)\}_{j=1}^k$ with
    $d_j \in$ {maximize, minimize}; number of targets $M$; number of samples $n_{\text{pick}}$; $d(\cdot,\cdot)$ a distance on $[0,1]^k$ (e.g.,
    Mahalanobis in percentile space with regularization).
Output: Subsampled set $\mathcal{D}_{\text{sub}} \subset \mathcal{D}$.
Step 1: Normalize features between 0 and 1;
Compute all normalized features $R \in [0,1]^{N \times k}$ for $\mathcal{D}$ (e.g., percentile-normalization to downweight outliers);
Step 2: Sample target vectors;
for $m \leftarrow 1$ to $M$ do
    for $j \leftarrow 1$ to $k$ do
        if $d_j = maximize$ then
            $t_{m,j} \sim \text{Beta}(\alpha_j, 1)$ where $\alpha_j$ increases with $w_j$;
        else
            $t_{m,j} \sim \text{Beta}(1, \beta_j)$ where $\beta_j$ increases with $w_j$;
Step 3: Nearest-neighbor assignment (without replacement);
$\mathcal{I} \leftarrow \{1, \dots, N\}$, $\mathcal{D}_{\text{sub}} \leftarrow \emptyset$;
for $m \leftarrow 1$ to $M$ do
    if $|\mathcal{D}_{sub}| = n_{pick}$ then
        break;
    $i^* \leftarrow \arg\min_{i \in \mathcal{I}} d(R_i, t_m)$;
    Add $x_{i^*}$ to $\mathcal{D}_{\text{sub}}$ and remove $i^*$ from $\mathcal{I}$;
return $\mathcal{D}_{sub}$;
```

Large-Scale Synthetic Data Generation We generate nearly 3,000 synthetic QAs across 350 test documents from the four source datasets. We prompt `Gemini-2.5-Flash Thinking` to propose nine QAs per document (three per QA type, matching the human annotation protocol) with the instructions A. Then, we implement programmatic fixes to clean the QAs generated, such as JSON format repairs and bounding box reconciliation based on the OCR-processed source documents. Due to resource constraints, we could manually validate only 202 out of the nearly 3,000 synthetic QAs. We prioritize QAs for manual validation using the target-guided importance sampling algorithm detailed in Algorithm 1. For this, we first pre-define metadata-level desiderata (e.g., maximizing the number of requested entities or pages in the answer) for the QAs. We then sample prototypical vectors from distributions biased toward these desired characteristics and select synthetic QAs whose metadata are closest to these prototypes. This approach prioritizes QAs with target properties while avoiding overrepresentation of outliers.

After the manual validation exercise, we estimate the inter-validator agreement rate: a meta-validator reviewed a sample of 60 synthetic QAs (stratified by source dataset) out of the <202 previously validated by the other validators. For each extraction leaf, the meta-validator noted any disagreements with prior validation edits (by comparing pre- and post-validation) or identified missed changes. Agreement was calculated as 1 minus the ratio of total disagreements to total possible relevant changes across QAs. Due to resource constraints, we did not re-validate all synthetic QAs from scratch; however, the high inter-validator agreement of 96.2% indicated no pressing need for full revalidation.

D Evaluation Metric for Structured IE

Synthetic Dataset for Mapping Evaluation To assess the feasibility of using an LLM "judge" for schema mapping in the metric computation process, we construct a controlled evaluation dataset for this intermediate task. We first curate few-shot exemplars that cover common schema variations between ground-truth and predicted structured answers (differences in hierarchy, key names, textual values, page indices, and bounding-box coordinates). Using these exemplars, we prepare two prompt templates for `Gemini-2.5-Flash`: (i) a transformation prompt that modifies keys, values, and permutes array orders; and (ii) a perturbation prompt that inserts or deletes key-value pairs with probability 0.5. The resulting synthetic dataset provides systematic deviations to stress-test the mapping procedure under realistic differences from ground truth. It comprises 37 examples, all of which were manually reviewed.

Schema Mapping Instructions We share the prompt fed to the text-only LLM mapper below:

~~~~
# TASK OVERVIEW
Your are a helpful assistant that needs to match the keys of a ground truth dictionary with the keys of a prediction dictionary.
After describing the general guidelines for this task, I will show you a few examples of the matching operation your are supposed to perform; then, I will provide you with a new pair of ground truth and prediction dictionaries whose keys you will need to match.
Your output, which needs to be wrapped between "```python" and "```", will be used to evaluate the predictions of a model trained to extract structured information from documents; you will thus need to be precise in your mapping.

# GENERAL GUIDELINES
## Input characteristics:
You will receive **one ground truth dictionary** and **one prediction dictionary**:
    - **They contain values extracted from a document, tagged with semantic key names**.
    - All their keys and values are strings.
    - Both do NOT contain nested objects.
    - Each of these dictionaries was obtained by flattening an initial dictionary that contained nested objects; the key names that you will see in each of the two input dictionaries indicate the prior structure of the initial dictionaries.
        Example: the key name "0.person.age" indicates that the associated value corresponds to the age of person nb 0. If we also see the key name "0.person.address" in the same input dictionary, then it indicates that the value associated to this other key name corresponds to the address of that same person nb 0.
In addition to these two dictionaries, you will also receive the extractive query that these two dictionaries try to answer. The query is just a string.
## Matching you have to perform:
For each key-value pair of the ground truth input dictionary, you need to match the considered ground truth key-value pair to a key-value pair from the prediction dictionary. Some more instructions:
    - To match key-value pairs between the ground truth dictionary and the prediction dictionary, you should **mainly rely on the VALUE of the key-value pairs**.
    - You can rely a bit on the underlying meaning of the key strings.
    - Do not forget to pay attention to the numbers delimited by "." in the key names: they should help you identify to which parent object each children attribute belongs to.
    - **In most cases**, you should try to match one key-value pair from the ground truth dictionary to one key-value pair from the prediction dictionary. A key-value pair from the ground truth dictionary may be matched to multiple key-value pairs from the prediction dictionary iff the string value of that ground truth key can be reconstructed by concatenating/combining the string values of these multiple prediction keys.
        Example: the key "0.full name" from the ground truth dictionary {"0.full name": "John Doe", ...} should be matched to the two keys ("persons.0.First_Name", "persons.0.Last_Name") from the prediction dictionary {"persons.0.First_Name": "John", "persons.0.Last_Name": "Doe", ...}.
    - Conversely, a prediction key-value pair should **normally** be used to match only one ground truth key-value pair in your output. The only case where a prediction key-value pair may appear multiple times in your output mapping is when its string value can be reconstructed by concatenating/combining the string values of multiple ground truth key-value pairs.
        Example: the key "0.first name" from the ground truth dictionary {"0.first name": "John", "0.last name": "Doe", ...} should be matched to the key "persons.0.Full Name" from the prediction dictionary {"persons.0.Full Name": "John Doe", ...}; the key "0.last_name" from the same ground truth dictionary should also be matched to the key "persons.0.Full Name" from that prediction dictionary.
    - As most matches between the ground truth and the prediction dictionary will be one-to-one, you should consider the overall effect of each individual match on the rest of the mapping. Do not focus only on greedily making the best local matches for a few ground truth keys at the expense of others (indeed, most prediction keys will not be eligible for reuse). Instead, proceed holistically to return a mapping where all ground truth keys are matched as well as possible.
    - Try to match as many key-value pairs from the ground truth dictionary to key-value pairs from the prediction dictionary. However, **if a key-value pair from the ground truth dictionary has truly no equivalent(s) in the prediction dictionary, please LEAVE IT UNMATCHED** and put None in the corresponding map (e.g., if "person.age" from the ground truth dictionary cannot be matched to a key-value pair from the prediction dictionary, then your output dictionary should contain {..., "person.age": None}).
## Output format expected
Your final output should be a python dictionary wrapped between "```python" and "```" that maps all the string key names of the ground truth dictionary to string key names from the prediction dictionary. Since it can happen that a ground truth key is matched to multiple prediction keys, all the values of your output python dictionary should be **tuples of key names from the ground truth dictionary** (most will be singletons). If no match with a prediction key-value pair is found, the value should be None.

# FEW SHOT EXAMPLES
Here are a few examples of what you are supposed to do given a ground truth dictionary, a prediction dictionary, and a query --along with explanations of why the proposed fuzzy mapping is the way it is.
## Example nb 1:
### Ground truth dictionary:
```python
{'0.start_date': '01/01/94', '0.end_date': '12/31/94', '0.TOTAL COST': '$210,910', '1.start_date': '01/01/95', '1.end_date': '12/31/95', '1.TOTAL COST': '$212,481', '2.start_date': '01/01/96', '2.end_date': '12/31/96', '2.TOTAL COST': '$220,416'}
```
### Prediction dictionary:
```python
{'slot1.expenses': '212481', 'slot1.period start': '01-01-95', 'slot1.period end': '12-31-95', 'slot2.dt_begin': '01-01-94', 'slot2.dt_final': '12-31-94', 'slot3.price': '220416' 'slot3.start': '01 01 96', 'slot3.end': '12 31 96'}
```
### Corresponding extractive query:
"What was the cost over time of the project?"
### Proposed output fuzzy mapping:
```python
{'0.start_date': ('slot2.dt_begin',), '0.end_date': ('slot2.dt_final',), '0.TOTAL COST': None, '1.start_date': ('slot1.period start',), '1.end_date': ('slot1.period end',), '1.TOTAL COST': ('slot1.expenses',), '2.start_date': ('slot3.start',), '2.end_date': ('slot3.end',), '2.TOTAL COST': ('slot3.price',)
}
```
~~~~

~~~~
Explanation (just so that you understand some important details): here, we could NOT match '0.TOTAL COST'
      to any key from the prediction dictionary because, although item 0 from the ground truth does match
      slot2 from the prediction dictionary, no sub-key of slot2 provides a cost value. If slot2 had had a
      sub-key entitled "cost" or something along these lines, even with a None or "" or "N/A" value, we
     could have matched "0.TOTAL COST" to "slot2.cost".
## Example nb 2:
### Ground truth dictionary:
```python
{'0.percentage_of_prime_working_age_women_discovering_car_manufacturers_through_magazines': '17%'}
```
### Prediction dictionary:
```python
{'0.value': '21', '0.unit': '%'}
```
### Corresponding extractive query:
"What percentage of prime working-age women discover car manufacturers through magazines?"
### Proposed fuzzy mapping:
```python
{'0.percentage_of_prime_working_age_women_discovering_car_manufacturers_through_magazines': ('0.value',
      '0.unit')}
```
Explanation (just so that you understand some important details): here, even though the values are
     different for the single key of the ground truth and for the keys of the prediction dictionaries
     ("17%" VS. "21" + "%"), these values seem to refer to the same underlying entity --based on how the
     keys are named and on the fact that we are requesting **a single** type of percentage in the query.
     Indeed, here, even though the extracted values in the prediction dictionary are incorrect ("21" +
     "%" VS. "17%"), the prediction does seem to have attempted to extract the "percentage of prime
     working-age women discover car manufacturers through magazines". It is just that it called it "0.
     value" and "0.unit" instead of "0.
     percentage_of_prime_working_age_women_discovering_car_manufacturers_through_magazines". Also, as you
      can see, the prediction dictionary split the requested percentage value into two key-value pairs
     ({"0.value": "21", "0.unit": "%"}), whereas the ground truth dictionary combined both into the a
     single key-value pair. We thus had to match the single key-value pair from the ground truth to the
     two key-value pairs from the prediction dictionary. Had the situation been reversed (i.e., two more
     granular key-value pairs in the ground truth corresponding to one coarser key-value pair in the
     prediction), we would have had to match each key-value pairs from the ground truth dictionary to the
      same prediction key-value pair, exceptionally.
## Example nb 3:
### Ground truth dictionary:
```python
{'0.name': 'John Doe Alpha', '0.position': 'President & CEO', '0.dept': None, '0.phone': '212-555-0101',
     '0.email': 'alpha@doe-bureau.example.com', '1.name': 'John Doe Beta', '1.position': 'EVP', '1.dept':
      None, '1.phone': '212-555-0102', '1.email': 'beta@doe-bureau.example.com', '2.name': 'John Doe
     Gamma', '2.position': 'SVP', '2.dept': 'Strategic Sales Insights', '2.phone': '212-555-0103', '2.
     email': 'gamma@doe-bureau.example.com', '3.name': 'John Doe Delta', '3.position': 'VP', '3.dept': '
     Strategic Insights', '3.phone': '212-555-0104', '3.email': 'delta@doe-bureau.example.com', '4.name':
      'John Doe Epsilon', '4.position': 'Sr Director', '4.dept': 'Strategic Research Insights', '4.phone
     ': '212-555-0105', '4.email': 'epsilon@doe-bureau.example.com'}
```
### Prediction dictionary:
```python
{'authors.0.name': 'John Doe Alpha', 'authors.0.title': 'President & CEO', 'authors.0.email': 'alpha@doe-
     advertising.example.com', 'authors.1.name': 'John Doe Betta', 'authors.1.title': 'EVP', 'authors.1.
     email': 'beta@doe-advertising.example.com', 'authors.2.name': 'John Doe Gamme', 'authors.2.title': '
     SVP Strategic Sales Insights', 'authors.2.email': 'gamma@doe-advertising.example.com', 'authors.3.
     name': 'John Doe Epsilon', 'authors.3.title': 'Sr Director Strategic Research Insights', 'authors.3.
     email': 'epsilon@doe-advertising.example.com', 'authors.4.name': 'John Doe Delta', 'authors.4.title
     ': 'VP Strategic Insights', 'authors.4.email': 'delta@doe-advertising.example.com'}
```
### Corresponding extractive query:
"Who are the authors of the slides? Give all information about them."
### Proposed fuzzy mapping:
```python
{'0.name': ('authors.0.name',), '0.position': ('authors.0.title',), '0.dept': None, '0.phone': None, '0.
     email': ('authors.0.email',), '1.name': ('authors.1.name',), '1.position': ('authors.1.title',), '1.
     dept': None, '1.phone': None, '1.email': ('authors.1.email',), '2.name': ('authors.2.name',), '2.
     position': ('authors.2.title',), '2.dept': ('authors.2.title',), '2.phone': None, '2.email': ('
     authors.2.email',), '3.name': ('authors.4.name',), '3.position': ('authors.4.title',), '3.dept': ('
     authors.4.title',), '3.phone': None, '3.email': ('authors.4.email',), '4.name': ('authors.3.name',),
      '4.position': ('authors.3.title',), '4.dept': ('authors.3.title',), '4.phone': None, '4.email': ('
     authors.3.email',)}
```
Explanation (just so that you understand some important details): here, even though the values extracted
     in the predictions are not all perfectly accurate compared to the ground truth overall, it is still
     easy to map ground truth keys to prediction ones based on their names/structure and how similar
     their values are. It is also clear that the prediction dictionary forgot all the phone number
     extractions, and that the prediction key-value pairs corresponding to "John Doe Delta" have index
     "3" VS. index "4" in the ground truth dictionary (this really underlines that, even if the key names
      are very similar/share the same index between gt and pred, they are not necessarily the best match:
      their corresponding gt and pred values should help you determine which gt and pred keys go together
     ). Finally, we note that the ground truth dictionary was more granular regarding "position" and "
     dept" information (separate key-value pairs for each, per author), whereas the prediction dictionary
      grouped both of these pieces of information into the same key-value pair "title" when department
     information is available, for each author. Thus, the "authors.2.title", "authors.3.title", and "
     authors.4.title" prediction keys can be used twice in the output dictionary, since their values
     contain both "position" and "dept" information. As for "authors.0.title" and "authors.1.title",
     since their values do not contain any department information, the ground truth keys "0.dept" and "1.
     dept" should not be matched to them (as opposed to "0.position" and "1.position"), hence their None
     values in the output dictionary.

# YOUR TURN
Now, you will be provided with a new ground truth dictionary, a new prediction dictionary, and a new
     extractive query. After some reasoning, you will need to propose a fuzzy mapping between the key
     names of the ground truth dictionary and of the prediction dictionary. Remember: your final fuzzy
     mapping should be wrapped between "```python" and "```".
## Ground truth dictionary:
```python
~~~~

If the LLM mapper does not include all ground truth keys in its output mapping, we re-prompt it in the conversation to map the missing ground truth key-value pairs to predicted ones. The same correction process is applied if the LLM mapper matches a ground truth key to a key that cannot be found among the predicted key-value pairs. Finally, we club ground truth keys in the output mapping in order to align the two schemas to the same level of granularity.

E Text Extraction Results: Breakdown by Answer Context

Model	Free Text	Key-Value Field	Templated Items	Chart	Checkbox
Qwen2.5-VL-3B	39.9	52.6	43.6	28.3	38.4
Gemma-3-4B	29.2	51.8	23.5	16.4	36.0
Qwen2.5-VL-7B	48.2	59.8	55.3	38.3	58.3
Gemma-3-12B	37.6	66.8	51.0	26.7	62.4
Pixtral-12B	31.1	55.1	53.1	37.3	31.5
Kimi-VL-A3B-16B	37.1	63.8	47.7	26.5	35.8
Qwen2.5-VL-32B	56.2	**77.2**	58.0	40.5	57.6
Gemma-3-27B	38.2	76.4	65.4	38.5	29.7
Mistral-Small-3.2-24B	37.1	75.1	**69.6**	**45.9**	72.3
Qwen2.5-VL-32B-FP8	**58.3**	76.2	69.4	39.6	**81.2**
Gemini-2.5-Flash	69.5	91.1	78.1	69.0	84.5
Gemini-2.5-Pro	**77.4**	**92.2**	**85.5**	**76.9**	**86.8**

Table 8: Model performance on free text, key-value field, templated items (which include table rows), chart, and checkbox formats.

F VLM Benchmarking Prompt

The prompt used to test VLMs on ExStrucTiny can be found in the next page. The three shots (one per QA type, based on FUNSD source documents for brevity) from the manually-annotated subset of ExStrucTiny are removed for space considerations.

~~~~
# OVERVIEW
You are a helpful assistant required to answer **extractive** queries based on document images that the user will
     provide.
You should always return your final answer in a structured object format presented as a list of (potentially-
     nested) dictionaries or a populated schema adhering to that of the input query.
NOTE: extractive means that the answer must be contained in the document; in the rest of the instructions, "IE"
     stands for "Information Extraction".
# GUIDELINES
Below are guidelines to follow in order to correctly answer queries based on the images of an input document (i.e
     ., extract the right information, using the right output structure):
## Input query:
- The input "query" you will receive is a string.
- Queries can be formulated either in plain text or as "empty" structured object schemas that you need to populate
      with extracted values.
- Queries can pertain to one of three types of extractive queries:
    (a) **Closed IE with plain text:** the query directly mentions in a sentence/in the form of plain text the
         names of the entities to extract (ex: "what is the SSN of the signers?" => [{"signer SSN": "123456"}])
    (b) **Closed IE with schema:** the query directly mentions in the form of a schema the names of the entities
         to extract (ex: "{"signer SSN": ""}" => {"signer SSN": "123456"})
    (c) **On-demand IE:** the query does not directly mention the name of all the entities to extract but instead
         mentions a *parent entity name* to guide the extraction (ex: "parse all the dependents information" =>
         [{"name": "Leo", "age": "12"}]). In this case, your answer will have to propose a schema somewhat
         inspired by the layout/structure of the document and populate it with extracted values of children
         entities related to the parent entity requested in the query.
- Queries can be simple declarative sentences (ex: "Extract all information about ..."), besides questions.
## Your output answer:
- Your answer should be a list of nested dictionaries that contain (inside extraction dictionaries) all the user-
     requested extractions expressed in the query, **whenever the query is not a schema**. When the query is an
     empty schema, then the answer object should just adhere to this schema and populate it with extraction
     dictionaries (see below).
- The extractions that compose your answer should be returned in order of (1) appearance of the entities mentioned
      in the query, and (2) for multiple extractions of the same underlying type of entity, in order of appearance
      in the document when considering it in reading order (left to right, top to bottom).
- Dictionaries within your structured answer can either be nested dictionaries/sub-lists when extracting
     hierarchical entities or line items/templated items (ex: [{"name": "Leo", hobbies: [{"name": "guitar", "
     duration": "2h"}]}]), OR **extraction dictionaries** containing verbatim text directly extracted from the
     document tokens. The schema of the extraction dictionaries that will be the leaf nodes of your output
     structure is: {"text": str, "page": int, "bbox": [int]}. The extraction dictionaries are the final/leaf nodes
      of the structured output that contain the actual text values you extract from the document. For the sake of
     clarity, we replace these extraction dictionaries with the text values in our examples of the # GUIDELINES
     section.
- Within an **Extraction dictionary** (i.e., a leaf node of your output structure), the page field should be an
     integer referring to the index of the page where the extracted text value came from. The text "### New doc
     image nb i" will provide page indices in the user prompt (0-indexed).
- Within an **Extraction dictionary**, the bounding box coordinates (see "bbox" field) should be normalized
     between 0 and 1000 and should correspond to the bounding box of the rectangle containing the extracted text.
- A line item or templated item can be defined as a set of fields all referring to the same underlying parent
     entity (ex: if someone named X with role Y signed the document at date Z, then the line item/templated item
     is composed of the following child entities: {"name": X, "role": Y, "date": Z}). This is particularly
     important for correctly structuring your answer object. For example, if you come across a closed IE query
     that asks for the age and name of all the signers of a form, and that all of this information is presented in
      a templated manner, your answer should have the following structure: [{"age": W, "name": X}, {"age": Y, "
     name": Z}, ...]. In contrast, if the closed IE query had been "extract the document ID as well as the amount
     requested in the form", it does not really make sense to present the answer as [{"document ID": X, "amount
     requested": Y}] since "document ID" and "amount requested" are not directly related (both semantically and
     from a layout perspective).
- The key names of the dictionaries that compose your answer (but that are not leaf "extraction dictionaries")
     should be, in order of preference (as long as a meaningful key name is proposed):
    (a) the actual schema key names for closed IE user queries that directly mention a schema to populate (you
         just need to adhere to the input schema and fill it with extraction dictionaries)
    (b) strings that minimally overlap the user-mentioned entity names (for plain text closed queries; ex: "what
         is the SSN of the signers?", "[{"signer SSN": "123456"}]")
    (c) strings coming from the document tokens (for on-demand queries; ex: "parse all the dependents information"
          => [{"name": "Leo", "age": "12"}]; also for certain closed queries if it is hard to overlap an
         appropriate key name with the user query)
    (d) strings you suggest based on your understanding of the values being extracted and your internal knowledge
         (for on-demand queries; ex: when the string to extract is floating text, not an actual key-value where
         the key name could be repeated; also for certain plain text closed queries if it is hard to overlap an
         appropriate key name with the user query).
- When you identify that the query requests line items/templated items in the answer, there is no need to name the
      keys of each item of the answer using XXX_1, XXX_2, XXX_3, etc; the **numeral ordering is useless** (ex: [{"
     edit_1": "add"}, {"edit_2": "remove"}, {"edit_3": "modify"}] should instead be formatted as [{"edit": "add"},
      {"edit": "remove"}, {"edit": "modify"}]; the numbers appended to key names are useless if the keys are
     returned in reading order of the doc).
- Null values VS. non-found values in the document: for closed queries (both plain text and schema-based), if one
     of the requested keys cannot be found, you should yield this extraction dictionary for its corresponding leaf
      node in the output structure: {"text": None, "page": None, "bbox": None}. If all requested keys cannot be
     found in the document, then you should return an empty answer list. In contrast, if one of the child entities
      of the parent entity requested comes from a key-value field with an empty/non-filled value in the doc (e.g.,
      "N/A"), you should yield an extraction dictionary like the following for its corresponding leaf node: {"text
     ": "", "page": int, "bbox": [int]}. In short, **N/As (empty/non-filled values) should be extracted as "" text
      values**. However, not finding requested info in a doc should yield {"text": None, "page": None, "bbox":
     None} extraction dictionaries or an empty list.
- Checkboxes: checkboxes can be seen as key-values too; user-requested checkboxes that are unchecked should have "
     No" as text value, and checked ones should have "Yes".
- For on-demand queries, two extraction dictionaries about the same underlying entity should not be merged
     together if they dont appear as one single line item/templated item in the document (ex: [{"name": "Leo S", "
     age": "14"}, {"name": "Leo S", "grade": "5"}] stays like that if they are present as two different line items
     /templated items of the same document).
- For closed queries where the user inputs an empty schema that you are expected to populate, your structured
     answer should indeed populate the user schema but it should have **extraction dictionaries** as leaf nodes,
     still. In other words, do not only populate input user schemas with text values: populate them with
     extraction dictionaries/leaf nodes that contain "text", "page", and "bbox" key-value pairs.
- Please wrap your structured answer containing the extraction dictionaries between "```python" and "```".
# FEW SHOT EXAMPLE nb ...
Below are document images + an example query-answer pair to show you more concretely the type of answer you should
      be generating given an input query.
## Document images: [...]
## Input query for this document: "[...]"
## Output answer to the above query: ```python[...]```
# YOUR TURN NOW
Below are images of a new document. Please provide a structured answer to the query that will be given at the end,
      by taking into account both the initial guidelines and the example(s) provided above, if any. Make sure to
     wrap your structured answer between "```python```" and "```".
## New document images: [...]
## Input query for this new document: "[...]"
## Your output answer to the above query for the new document:
~~~~