Carnegie Mellon University
Software Engineering Institute

# Leveraging LLMs for Data Coding

*Declan Kutscher, Austin Whisnant*

November 2024



## Introduction

The CERT® Insider Risk team collects information about insider threat incidents to support the research and development of methods, tools, and processes for detecting and mitigating insider incidents. We collect this data primarily from United States federal criminal court cases and code it into a set of specific fields consistent with the IIDES schema [SEI IIDES]. We maintain the encoded data for analysis in a Software Engineering Institute (SEI) Database called Management and Education of the Risk of Insider Threat (MERIT) [SEI 2013].

There are about 50 insider threat court cases per month in the U.S. federal criminal court system. The number of these cases has created a backlog of thousands of insider threat cases awaiting encoding for MERIT. Our goal is to use machine learning (ML) to assist with coding incidents. The court cases consist of unstructured data, including scanned documents, e-filed documents, and webpages. A fully coded case might consist of over 200 fields. The information for these fields could be spread across many different documents, making manual coding tedious. In this paper, we investigate the usefulness and design methodology of applying large language models (LLMs) to improve and automate the process of coding case data. We introduce tools to guide LLMs [Vaswani 2017] to assist in this coding process. We show initial results and lay groundwork for further research in this field in the hopes that the larger scientific community will contribute to solving the problem of coding specific fields from unstructured data.

® CERT is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

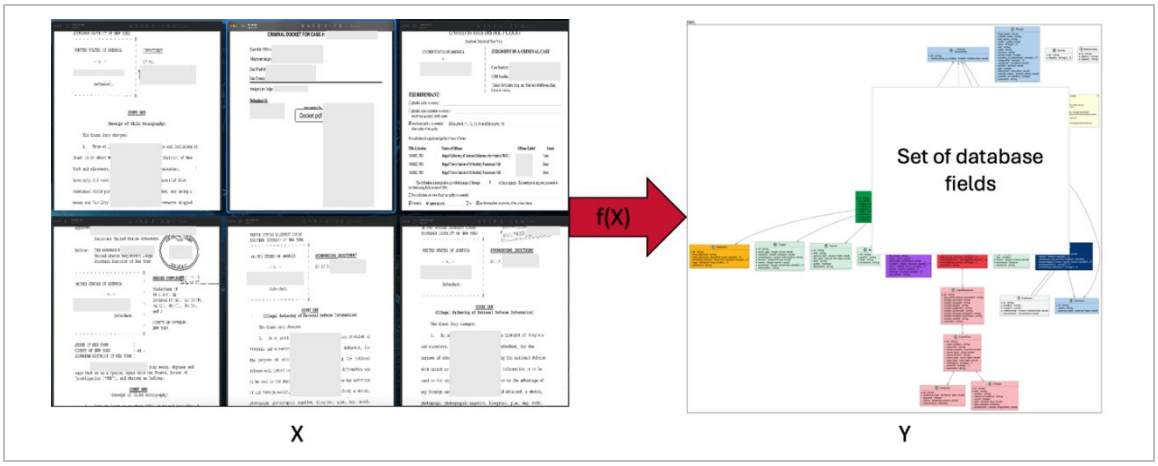


*Figure 1: Our Problem Area*
(We have a corpus of text, X, that contains information about our case, and we would like to find/create/learn a function F that given X can output Y, the set of database fields. Note that the redaction boxes exist only in this paper, not in our source documents.)

## Previous Work

Previous work in extracting information from text focused on using convolutional neural network (CNN) [Le Cun 1990], optical character recognition (OCR), and/or graph neural networks (GNN) [Zhou 2018] to model the text. Many times, this approach was used for extracting key fields from structured text documents that have slight differences among them.

PICK[1] [Yu 2020b] extracts vision features as well as textual features using CNNs and OCR to process and extract key information from structured documents such as invoices, tickets, and receipts. The vision and textual features are used to create a knowledge graph of the document to capture semantic connections across the document. Yu's work introduces a new perspective on key information extraction (KIE) by creating a knowledge graph of the features in a document to model dependencies across key information.

Other key related work is Attend, Copy, Parse [Palm 2018], which uses a CNN and OCR to extract targets from the text and convert them to a desired output format. This model works by extracting and encoding text into a *memory bank*. When given a target as input, the model attends to its memory to find the information and then parses it into the desired format.

[1] PICK stands for **P**rocessing Key **I**nformation Extraction from Documents Using Improved Graph Learning **C**onvolutional Networ**k**s.

These approaches differed from a solution to our problem, since they were designed for highly structured text documents. However, we use some principles of the key information extraction outlined in both approaches for our own tools when utilizing LLMs.

## Initial Testing

Our development process began with a simple question-answer (Q&A) session with models to gauge the difficulty of our task for LLMs. We started with manually cleaned text, and we asked key information-retrieval questions for various fields a coder would need to fill out an insider incident. From our testing, we determined that the ability of models to extract key information was quite good for some prompts, and it retrieved the correct information a high percentage of the time. However, the format was highly volatile, since the output would change drastically between models and prompts. Our initial testing also showed that it was important that the context be concise and clean for accurate retrieval. Given that court records are not at all concise or well structured, developing a method for finding the right context to pass to an LLM was a major hurdle.

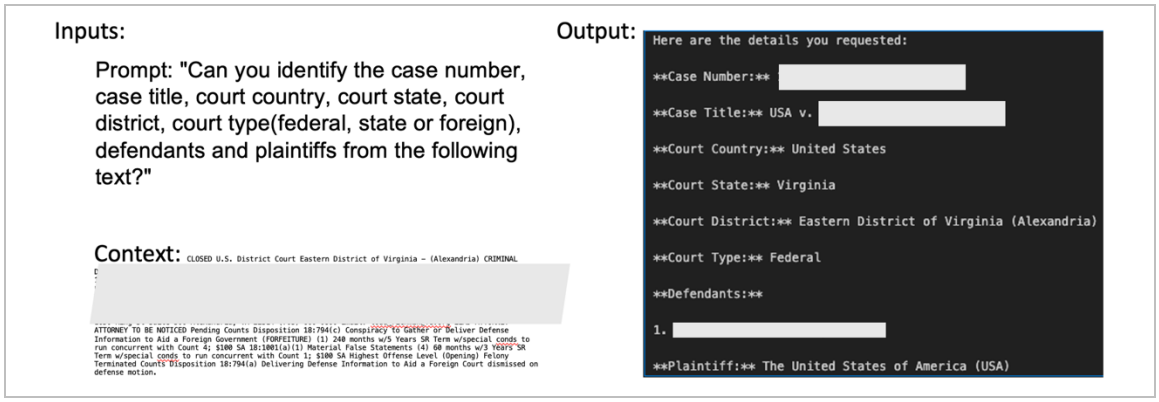


*Figure 2: Example Q&A Input, Context, and Output from an LLM (LLaMA3)*
*(The input queries for the field names and the context provided in the first page of the docket, which is verified to have all the information that should be returned.)*

Given the results of the Q&A, particularly the issue of specifying an appropriate context for our models, our next inclination was to try the new open source vision-language models (VLMs) [Liu 2023, Bordes 2024]. These VLMs were recently released and held promise for helping us solve our problem, since they could cut out one of the longer steps in our process: OCR scanning and text preprocessing.

Most VLMs are designed for object recognition, scene description, and other object-oriented tasks rather than text recognition. The process typically involves downsampling from a high-resolution image. Since encoders are trained for object-oriented tasks, downsampling does not destroy the main features the encoder used to determine the object in images. Even though our portable document format (PDF) files are high resolution, downsampling destroys many of the features of text in an image. The effective receptive field of these encoders is therefore not set on the same granularity of the text in the image of the PDF [Luo 2017, Raghu 2021]. This shortcoming was quite evident in the answers given by

our models. They were able to understand the structure of our document, but they were unable to retrieve exact key information; instead, they tended to generate fake court titles, names, etc.

Figure 3 shows the output of one test. The input prompt is the same as the one depicted in Figure 2: a query for the field names in the object. The image provided is the first page of the court docket. The model gets the country, state, district, and court type correct, but the rest of the information is incorrect, and most of it is not present in the document at all.

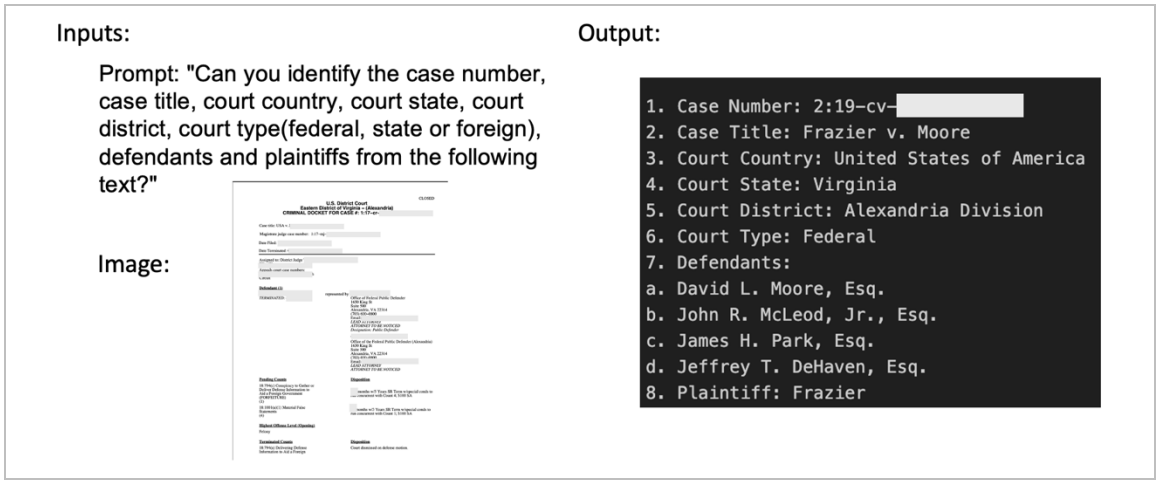


*Figure 3: Example Q&A with LLaVa-LLaMA3 (Liu et al., 2023)*
*(The model gets some output fields correct (i.e., court country, state and district, court type). The case number is incorrect, the title and defendants are incorrect, and they are not present in the document at all.)*

Through our experiments, we found that the state of vision encoders for these documents is not on par with OCR models. Given advancements in vision encoders, we believe this path may become more fruitful. There have been recent advancements in large image modeling that may prove to be useful in this domain [Xu 2024, Ge 2024]. In the short term, the next best option for us is to use current state-of-the-art OCR techniques, specifically Tesseract [Smith 2007], to extract text from our PDF files rather than using VLMs.

## Development

Equipped with the results of our initial testing, we focused on two objectives for getting the best output from our LLM case coder:

1. Our models need clean and relevant context to retrieve the correct information.
2. Our models should have consistent structured output to comply with the database schema and facilitate automated case coding.

### Retrieval Augmented Generation

We implemented retrieval-augmented generation (RAG) [Lewis 2020] to automatically retrieve relevant context for our models. RAG is a new development in the LLM space and allows us to attempt to provide relevant context to our models to help them create a viable output. In our case, we want our RAG system to suggest the best text based on the object (e.g., a court case or a person) we are generating. This text should contain the information needed to fill in the fields of the object (e.g., charge, case title, name). To create a RAG system, first you create embeddings and vector representations of the text or chunks of text using an encoder. You can then store these embeddings in a vector database and query the database for the most relevant text or text chunks. In many cases, the user prompt is used as the query prompt. The prompt is embedded, and then the text or text chunks nearest to the prompt's embedding are considered the most relevant. RAG works for many use cases but not for our use case, since the information we provide to our models from the user is only the name of the object we want to encode (e.g., court case or person).

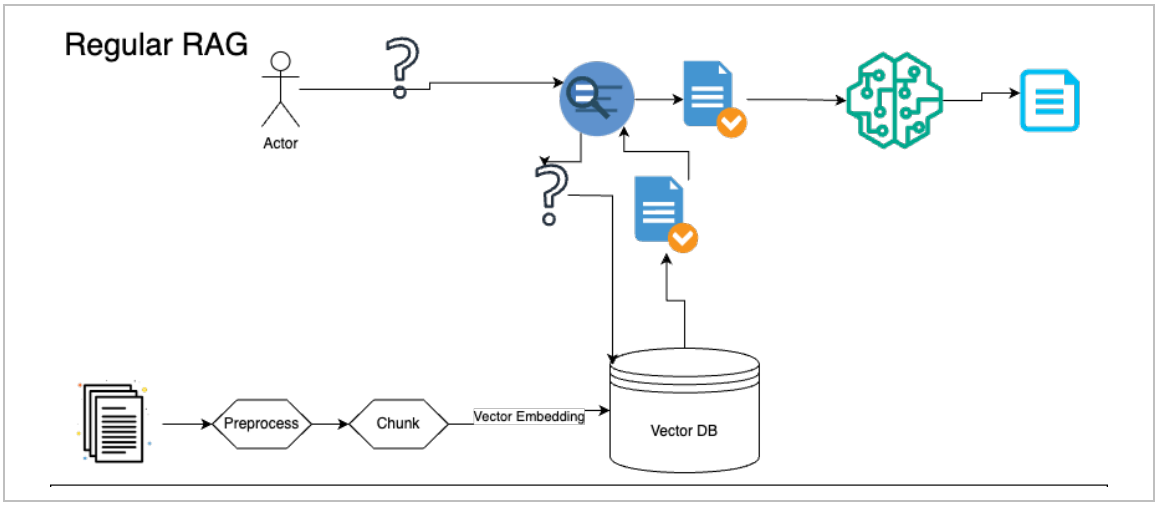


*Figure 4: A RAG System*
*(A user/actor provides a prompt, the prompt is embedded, and the chunks nearest to the prompt in the vector database are retrieved and provided to the model to assist in answering the prompt.)*

Our system, instead, takes the schema of the desired object (i.e., the list of desired fields) and uses the descriptions of the fields to query the vector database. This approach is intended to retrieve the most relevant context for the fields that the model will use to extract information. The descriptions of the fields are used instead of the field names, since the descriptions will contain more keywords related to the requested information. Thus, the embedding will be closer to the desired information and will improve the performance of our models.

For text preprocessing, normal RAG systems tend to apply lemmatization and remove stop words, numbers, and special characters. These preprocessing augmentations create good representations of text in terms of embeddings; however, they destroy the information needed for correctly filling in information that will be used later in other analyses. To resolve this problem, we store two versions of

each chunk of text: (1) the fully preprocessed text used for creating embeddings and (2) the plain text retrieved and used as context for our models.

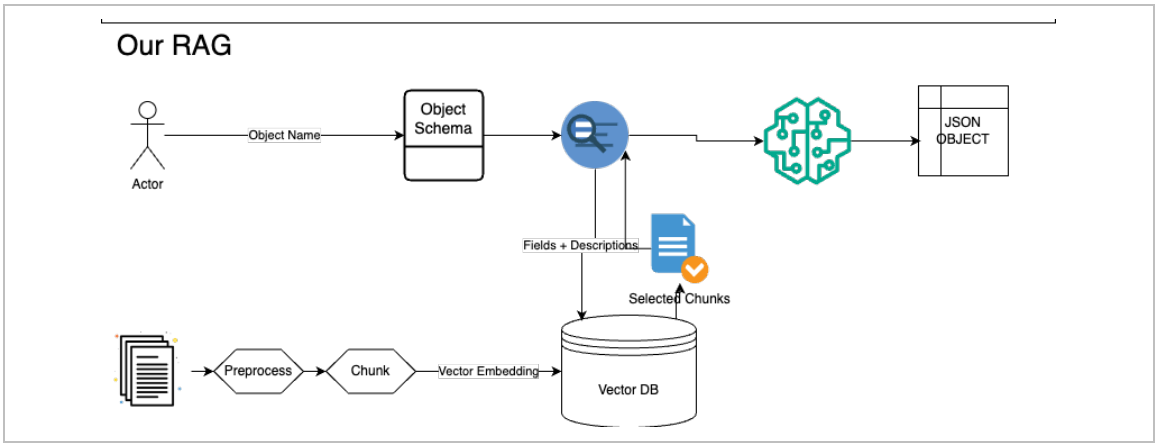


*Figure 5: Our RAG System*
*(A user/actor provides an object name. Then, the fields and their descriptions are embedded. The nearest chunks are then retrieved and used to generate the output JavaScript Object Notation (JSON) object.)*

### Improving Model Output Consistency

To improve the consistency and structure of our output, we first started with prompt engineering. We prompted models to use JSON output formatting and then attempted to turn the model output into our specific database format. This format conversion worked well; however, the models seemed prone to adding unnecessary fields or not formatting the information in the format we wanted. To resolve this problem, we focused on using the JSON schema that we desired in our prompts. This approach led us to discover open source models that were fine-tuned on JSON generation. These models specifically took JSON schemas as input along with the information and generated the output. We found that using these models improved model output a great deal. However, the models performed less well on key information retrieval, which meant we needed to find a middle ground between JSON fine-tuned models and key information retrieval models. One way to approach this middle ground was to use models trained to produce code. Since JSON is an extension of JavaScript, models trained to produce JavaScript and retrieve key information were quite good at performing our task. We found that, much of the time, they were able to understand our task and field schema and produce the correct output formatting.

```
prompt = f"""
        <schema>{schema}</schema>
        Create a {obj_name} object following the schema provided
        Use the following information to fill out the fields:{context}
        """
```

*Figure 6: Example Prompt Used with a JSON Fine-Tuned Model*
*(The schema tags "<schema></schema>" identify the schema the model needs to follow.)*

### Guiding Model Output and FIM Prompting

Even the JSON fine-tuned models still added extra fields and nonsense answers or misidentified persons in the text. To fix this problem, we realized that our models needed more guidance, and we decided to use a "chain-of-thought"-like approach [Wei 2022], which we called "chain of field." This prompting method prompts for each field individually. This approach adds more calls to our process, but it also increases the accuracy of our system. Using chain-of-field prompting, we can use previously LLM- or human-generated answers to populate fields and provide additional guidance and context for generating the objects. Code models also have special prompting methods called fill-in-the-middle (FIM) [Bavarian 2022], which allows you to provide a prefix and suffix in your prompt, and the model then attempts to fill data between them. As an added benefit, using FIM prompting prevents the model from changing the JSON format we require for our schema. By leveraging these tools from code models, we found our best performing models and used them in our final design.

````
Using the context above fill in the json object below ahering to the schema. Fill in the values between the <FIM></FIM> tags.
If a field cannot be determined leave it blank.

```json
{
    "first_name": <FIM></FIM>,
    "last_name": Chen,
    "age":<FIM></FIM>,
    "city": <FIM></FIM>,
    "state": <FIM></FIM>,
    "country":<FIM></FIM>
}
```
````

*Figure 7: Sample Prompt for the Fill-in-the-Middle Prompting Technique for Code-Trained Models (The model fills in between the fill tags, <FIM></FIM>. Other models require other forms for FIM, such as providing prefix and suffix tags. In this toy example, the model was given only the last name of a person, and it can search a document for the rest of the details about that person.)*

### Levels of Autonomy

The system and tools we discuss in this section can exhibit three different types of autonomy:

- full autonomy
- supervised autonomy
- LLM assistant

Which type of autonomy is most suitable depends on the use case. For our case coding, the most suitable type it uses greatly depends on which object we want to create (e.g., court case, person, victim organization).

#### *Fully Autonomous Coding*

We define *full autonomy* as having minimal to no human in the loop. This type of approach is ideal for objects such as the court case and even persons in the case. With the correct context, the model can fill in all the fields, validate the information, and resolve errors. Ideally, these tasks are performed as part

of an ingestion step that happens before a human coder sees the case. Then, when a human coder opens the case, they can correct any inaccuracies.

Pseudo code for this process entails going through every object in the schema and generating information for each field individually. The real system also includes steps where it can inform itself, such as using the defendants in the court case object to generate a person object for each of them. After an object is made, it is validated and then the errors, if any, are resolved.

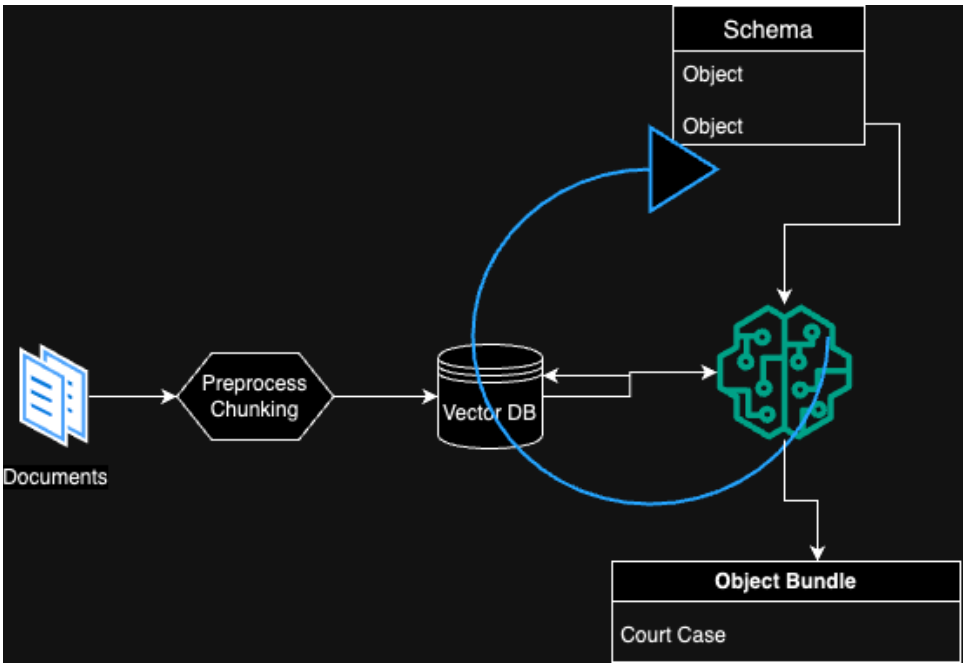


*Figure 8: Fully Autonomous Behavior of Our System*
*(The model iterates over the schema, generating all the objects it can. It can inform itself about certain objects using our tools, such as using the* defendants *field to generate person objects.)*

### *Supervised Autonomous Coding*

The next possible type of autonomy is *supervised autonomy,* where a human in the loop guides the process of coding a case using an LLM. The human tells the model which objects to create and with what information. This role of the human could greatly improve the efficiency of coding a large dataset, since the human in the loop can create objects, fill in information, and then let the model fill out the rest of the object. The model validates and resolves errors in the object before returning it, and then the human can resolve more complex errors or correct inaccurate answers.

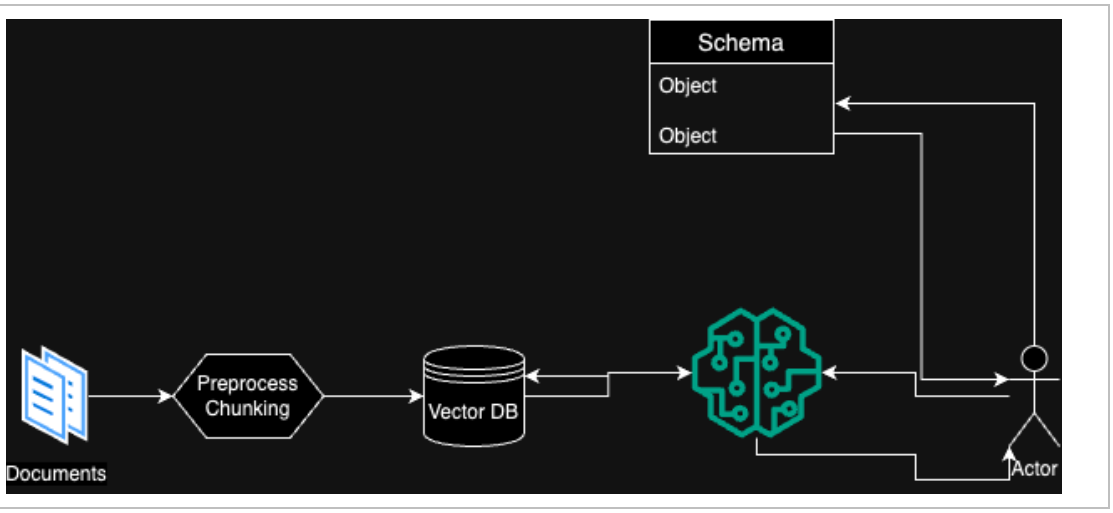


*Figure 9: Semi-Supervised Autonomous Behavior*
*(A human can choose objects to generate and provide partially complete objects for the model to finish.)*

### *LLM Assistant Coding*

The last type of autonomy is Document Q&A, or a *copilot* type of behavior. We use our RAG system to chat with a model about a case instead of filling out objects. This use of RAG may be helpful for very complex objects, such as legal responses and tactics, techniques, and procedures (TTPs). The human can copy and paste text into a chat box when there is a specific paragraph or document they want answers from. This selection allows the most flexibility for the user in deciding what to do with the fields and LLM outputs.

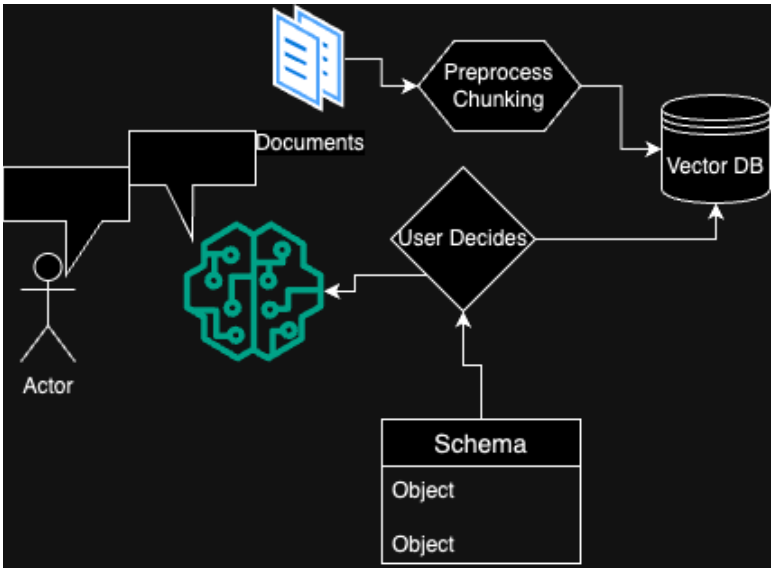


*Figure 10: LLM Assistant Behavior of the System*
*(The user treats the model as a chatbot and can prompt for answers and ask for summaries of the case.)*

### Error Resolution

As is often expected with these types of models, our system still had errors. However, a human in the loop can prompt the model to resolve them in separate calls to the system. This way, the model can focus on resolving the error independently of key information retrieval. Errors that can be easily corrected include those incorrectly implementing the desired schema or not using the correct schema vocabulary. For instance, a model might answer the country field with "United States" instead of the desired "US." Errors can be resolved using the schema directly if the output falls within the dictionary of vocabulary; however, if the error is more complex, we can prompt an LLM to resolve it, which works quite well.

## Experimental Setup

To empirically test the performance of our system, we decided to test different models against a system of regular expressions on two different schema objects: court case (easy) and person (hard).

We tested six models across four different examples of high-profile public cases. Each model used the same configuration file, which included parameters such as context window, temperature, and system prompt. It is possible that each model would have an ideally performing configuration; however, with only four examples on which to base our parameters, these adjustments were outside the scope of our experiment.

The models were benchmarked against a series of custom regular expressions for each field as well as some intuitive heuristics to retrieve fields. For example, a court case number includes the case type in it: *cv* for civil and *cr* for criminal. Creating such regular expressions for each field requires significant engineering for most fields; however, for the easier objects (e.g., court case) it is quite trivial, assuming the documents remain in the same structure (e.g., court case dockets are consistently formatted).

### LLM as a Judge

For a few fields (e.g., case number), judging our results could be done using regular comparators. For more open-ended fields (e.g., names of defendants and plaintiffs), we needed a more adaptive approach. For example, the ground truth of the plaintiff field could be "USA," but other acceptable answers would be "United States," "US," and "United States of America." Similarly, for a person, the ground truth could be "John Adam Doe," and acceptable answers would be "John Doe" or "John A. Doe." Rather than enumerating all acceptable answers for each field, we use an LLM to judge the similarity of the two answers on a scale of 0 to 1, where 1 is the exact ground truth. We chose this score as our metric since it is better able to calculate the correctness of our use case than other string comparison metrics. For example, edit distance [Ristad 1998] would not present consistent scores across different valid answers, since the edit distance between "US" and "United States" is larger than the

edit distance between “United States” and “United Kingdom” even though “US” is much more correct for our use case.

## Results

A speed benchmark was done on an M1 Max MacOS laptop with 32 GB of RAM using Ollama [Ollama 2024]. Using the Codestral 22b [Ai 2024], our process takes around 1 minute per call. Since there is a call for each field, this can take around 36 minutes to code one case autonomously. This could be trivially reduced by better models, hardware, or using proprietary model application programming interfaces (APIs) (e.g., OpenAI’s ChatGPT) [OpenAI 2023]. At around a dozen new cases per week, 36 minutes is perfectly acceptable. If, as expected, these tools are integrated into an application for data coding, a human in the loop would likely reduce the number of unnecessary calls for fields or whole objects that the human knows are not needed for the case.

For our performance experiments, we tested six different models ranging in size from 7B parameter LLaMA architecture [Touvron 2023a] to 9b Gemma2 architecture [Team 2024], and our largest was a 22B mixture of experts [Artetxe 2021]. We used the same model configuration for all models, though there was an opportunity to create the best configuration for each individual model. We decided to use the same configuration across all models to keep the experiment fair.

There are four ground truth examples; we tested two of each example’s objects: an easy court case object and a harder person object. The court case has 9 scorable fields, while the person object has 15 scorable fields.

We allowed each model to go through error resolution once to attempt to resolve errors prior to grading. We tested six different models; however, there are hundreds of others that could apply, and there are new models every day that are bigger, better, and more efficient. We chose these six based on their popularity at the time of our experiment. We benchmarked these models against a series of regular expressions with one static expression per field.

### Court Case Object Tests

In Figure 11, we can see that Codestral [Ai 2024] and LLaMA3.1 [Dubey 2024] perform quite well at coding court case objects, meeting or exceeding regular expressions on some examples, while others

fall short. We can also see that all the models performed quite poorly on Example 3, and from here on, we decided to omit that example so that it does not interfere with representative performance.[2]

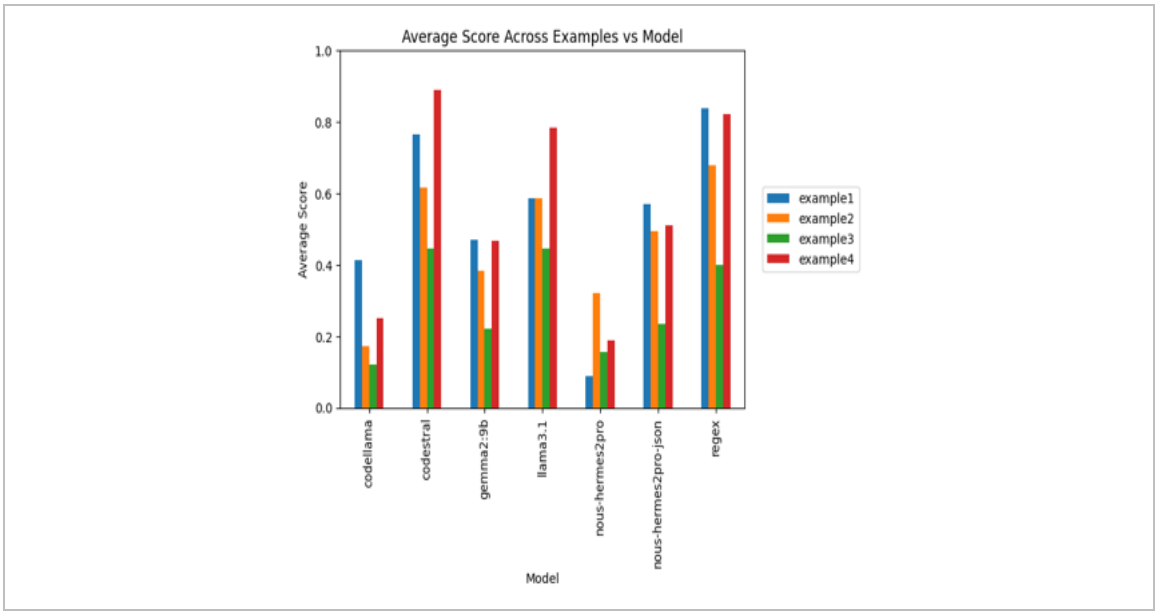


*Figure 11: Average Score per Model vs. Example on Court Case Objects*

In Figure 12, we can see that the easiest fields for the models to process are the case number, title, court country, state, and district. While the most difficult fields are court type, case type, defendant, and plaintiff. We think the reason for this is that the easier fields are *key information retrieval* while the other fields are *reasoning* fields. The reasoning fields are ones that do not state the answer in the document and may require the model to observe the information in the document and decide what is the appropriate answer. For instance, for case type, you may consider the charges as well as the plaintiff to determine if it is a civil or criminal case.

[2] Example 3 was a civil case, which is quite rare in our data.

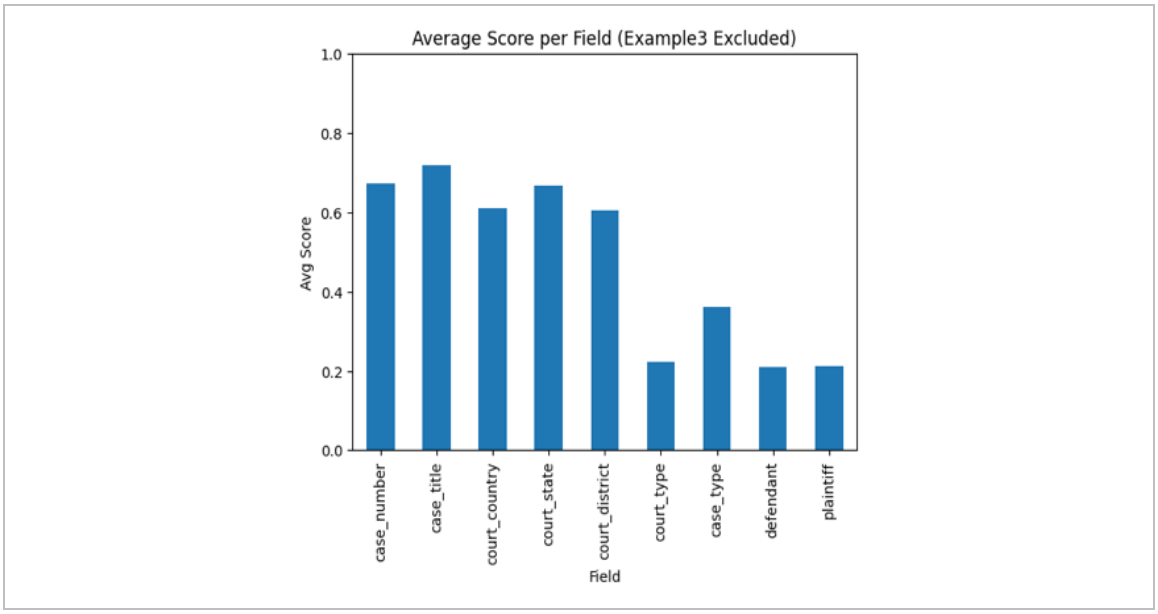


*Figure 12: Average Score per Field of a Court Case Object (Excluding Example 3)*

With regular expressions, however, there are some hints in the docket text that can be exploited. For instance, we can get the case type from the case number, and the court type can be based on the district of the court. Models may also be able to learn these tricks through in-context learning or few-shot learning.

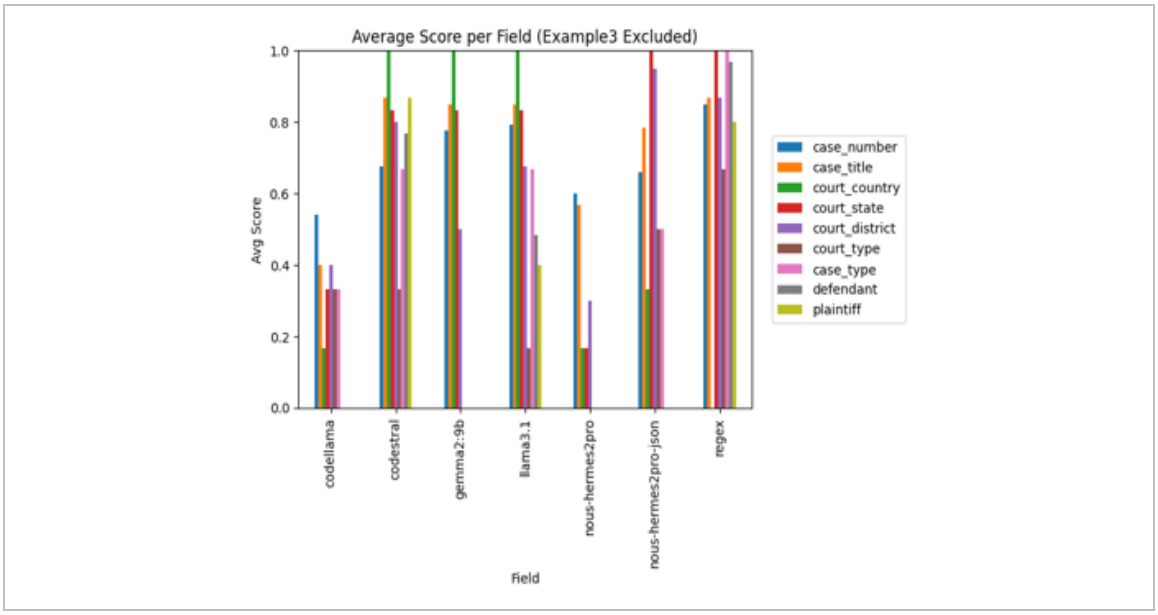


*Figure 13: Average Score per Field of a Court Case Object for Each Model*

Figure 13 shows which models are perhaps better at *key information retrieval* versus *reasoning* fields; however, the general trend is that the newer models perform best. Code Llama is the LLaMA2

architecture [Touvron 2023b], and Nous Hermes 2 Pro [Nous Research 2001] is a Mistral [Jiang 2023] model that is fine-tuned on a custom synthetic dataset. These older architectures were trained on older datasets and show the drastic improvement in the general ability of open source models over time.

We can also see regular expression performance against the models on certain fields. These results help us determine which fields to grab with regular expressions in a production system and which fields would be best done with an LLM model.

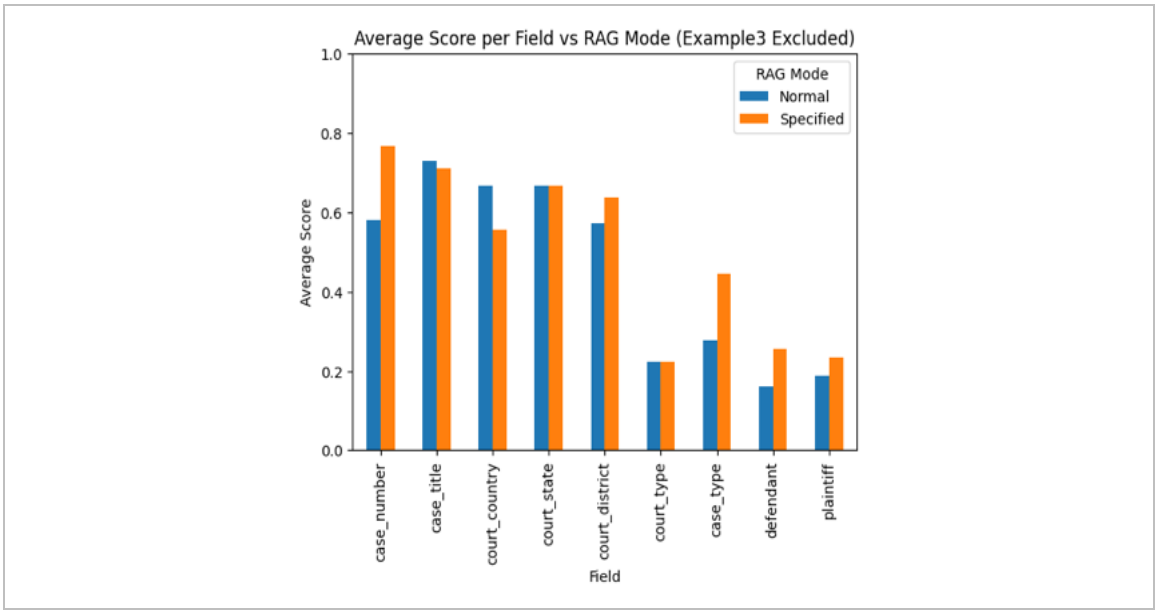


*Figure 14: Average Score per Field in a Court Case Object vs. RAG Mode (Excluding Example 3)*

The average performance of the models across the fields based on the RAG mode, specified or normal, is shown in Figure 14. We can see that, for five fields, specifying the RAG mode to select from one document improved performance slightly, and for some (e.g., case number and case type), it improved a lot. For other fields, the RAG mode had less of an effect or a negative effect but not by a significant amount. This result may be due to randomness (in some cases) or our small number of examples.

This result is interesting because it shows that humans having some supervision in the process can help with coding fields. However, it could be that the best chunk of information is the same in both modes, but the requested answer was incorrectly retrieved in the normal mode.

### Person Object Tests

We next tested a person object, which is quite difficult compared to the court case object. The information can be spread across many different documents. It was not possible to use regular expressions on this object, since the information does not follow regular patterns like it does for a court case object. It is for this reason that we are experimenting with LLMs.

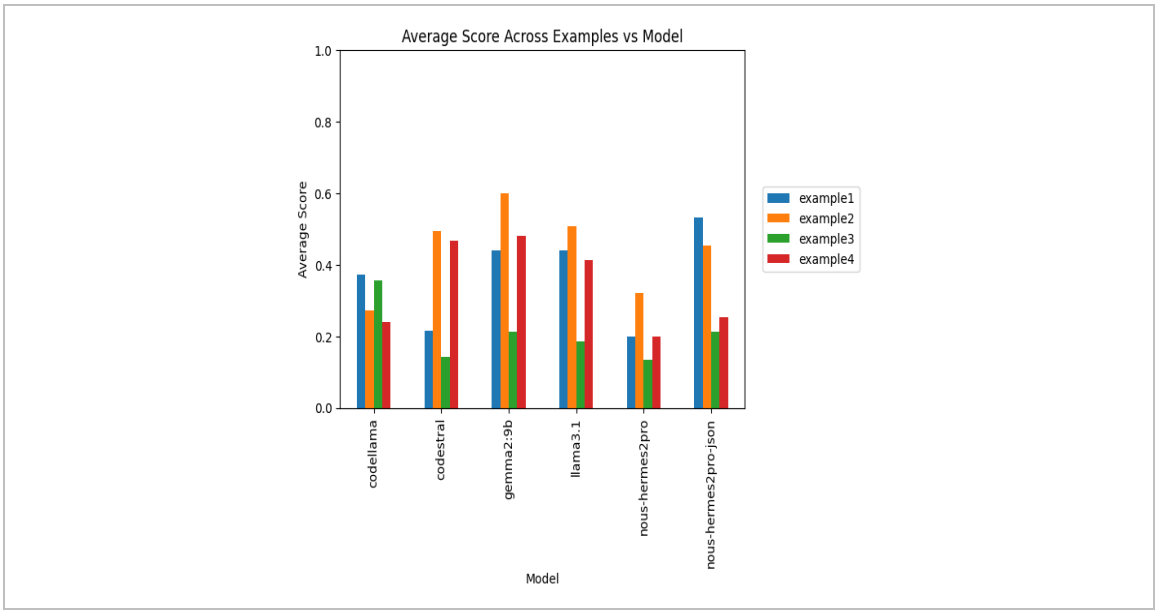


*Figure 15: Average Score per Model vs. Example on Person Objects*

We can see that the models performed worse on this object than the court case object (Figure 15), but they performed quite well on a few fields, which indicates that this object is best done in tandem with a human (Figure 16). The model can fill in the fields it can, and the human can correct and find the rest of the fields.

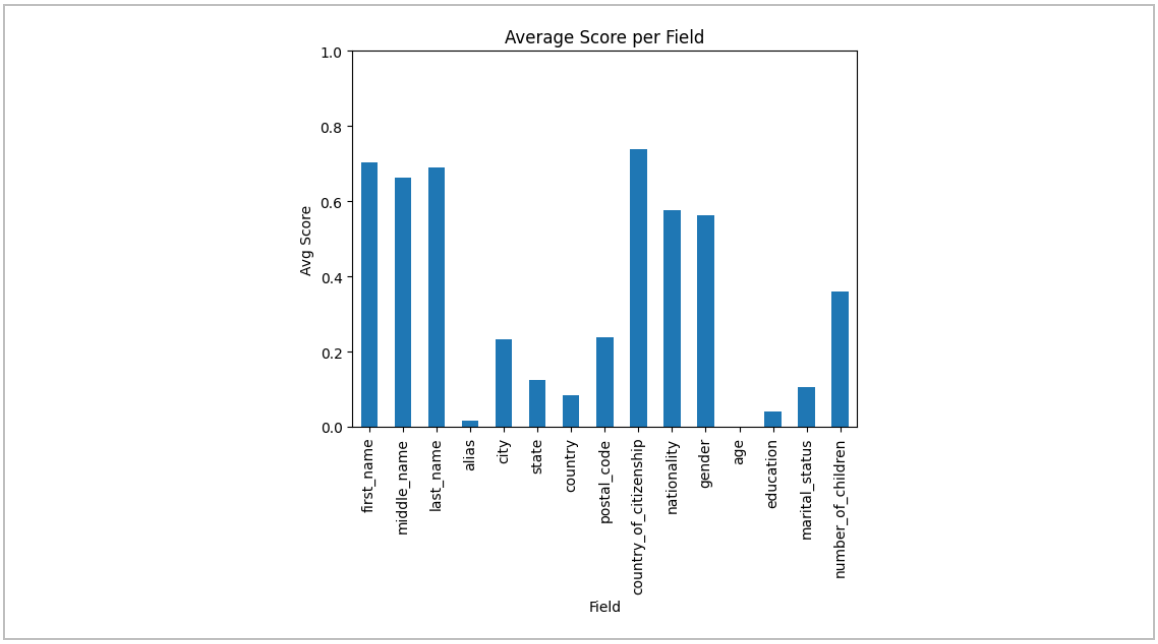


*Figure 16: Average Score per Field of Person Object Excluding Example 3*

Some court documents do not have all the information requested by the schema; therefore, neither a human nor an LLM could identify information such as the city or postal code of the person.

### Common Errors

Common mistakes that models make include repeating the prompt or schema as an answer, anonymizing people's names, and mistaking people for different roles. Repeating the prompt is common when the context fills up or the model is not designed to have a long context window. Our prompts, including the context and schema, were around 10,000 tokens, which means, for models with smaller context windows, errors might occur. This type of error can be corrected by using larger model context sizes, which is an increasing trend in new open source models— with many current models scaling from 4k, 8k, 32k, and now 128k token windows over the last few years [Ding 2024].

Anonymization can occur due to a training or licensing issue. Some models are trained to be censored and "safe." This training can mean that the models are prohibited from doing certain actions or working within certain contexts. Models can then refuse to use people's names or refuse to generate answers that would be considered harmful. One example case had a charge that involved child sexual abuse material, and models tend to avoid this sort of information. Models also tend to mistake attorneys or "interested parties" on the docket as defendants or plaintiffs [Glukhov 2023]. This mistake was quite common throughout the development process. It shows that unstructured text is quite difficult to parse because of (1) the many escape characters and (2) a lack of understanding the surrounding text. For example, the model may "understand" that a defendant should be a person/entity, so it retrieves a name of a person, but it retrieves a nearby name that is incorrect.

## Conclusion

In this paper, we presented some tools and design methodologies to address the problem of guiding LLMs to create structured outputs from unstructured text for the purpose of coding incident cases in a large database. We also presented our implementation of these tools and three different types of autonomy. Our approach shows good performance and potential compared to regular expressions and is a good starting point for future researchers to tackle this problem in this domain and others.

### Next Steps

The next steps in this project are at the forefront of LLM development. RAG is currently one of the hottest topics of research due to its performance boost for LLMs and its potential to replace search engines. Recently, Microsoft released Graph RAG [Edge 2024], which utilizes knowledge graphs instead of embedding spaces. Also, open source models are improving over time; a better open source model may be released that can either run faster or offer greater performance. In less than a year, open source models reached greater than human abilities on the first set of multitask language benchmarks,

and new benchmarks had to be made. Currently, the top open source models are averaging 46% across all tasks on the newest benchmark leaderboard but, it is only a matter of time before they reach the human score as they did previously [Open LLM 2024].

Within the system itself, there is a lot of room for improvement in the packaging and tooling that we have started to make. For instance, using a framework such as LangChain [Pandya 2023] or DSPy [Soylu 2024], we can create cleaner code and classes as well as utilize all their tools for monitoring, analyzing, optimizing, and training our models. Once our existing MERIT database is fully converted into an IIDES conformant schema [SEI IIDES], we will have ~1,500 incidents, each with 10-12 objects to use as examples for fine-tuning (Figure 17).

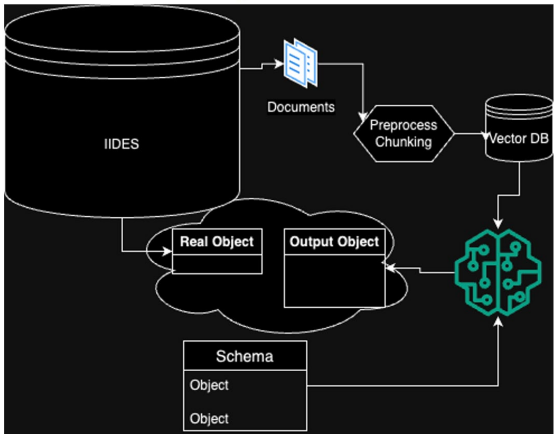


*Figure 17: Example of Fine-Tuning a Model for this Task Using the IIDES Database*

Overall, this task is quite complex and especially unique. A lot of today's LLM research focuses on agents that perform small tasks to complete a larger task and have some way of organizing and deciding what tasks to do. The agents are provided with specific tools and functions to assist in completing their task. Our problem area could be seen as quite agent-like in nature, and it would be interesting to see where future research goes in terms of the ability to complete complex agent-like tasks, since this future research could help improve our insider incident data collection.

# Appendix: Supplemental Materials, Court Case Objects, and Person Objects

## Supplemental Materials

- Introduction to LLMs by Mark Riedl [Riedl 2023]
- Introduction to RAG by Piyush Thakur [Thakur 2023]

## Court Case Object

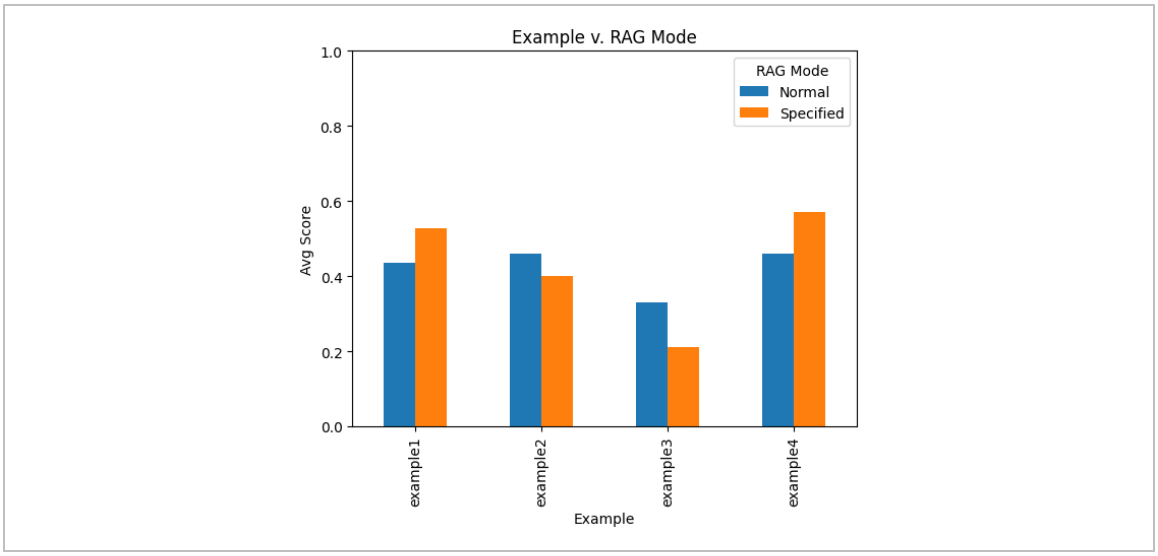


*Figure 18: Average Score per Example Incident Case vs. RAG Mode*

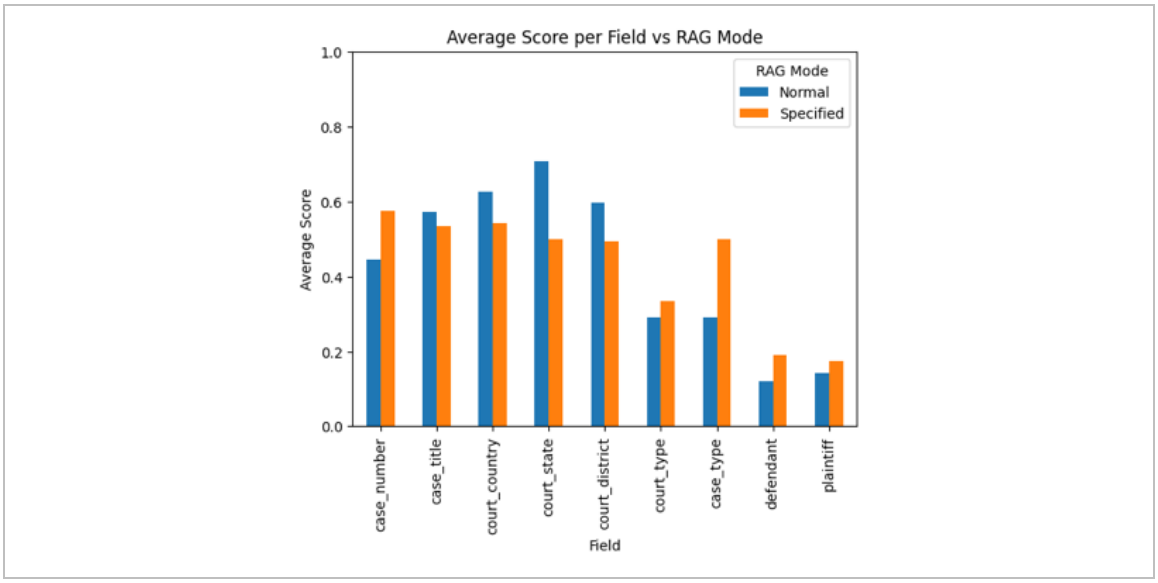


*Figure 19: Average Score per Field in a Court Case Object vs. RAG Mode*

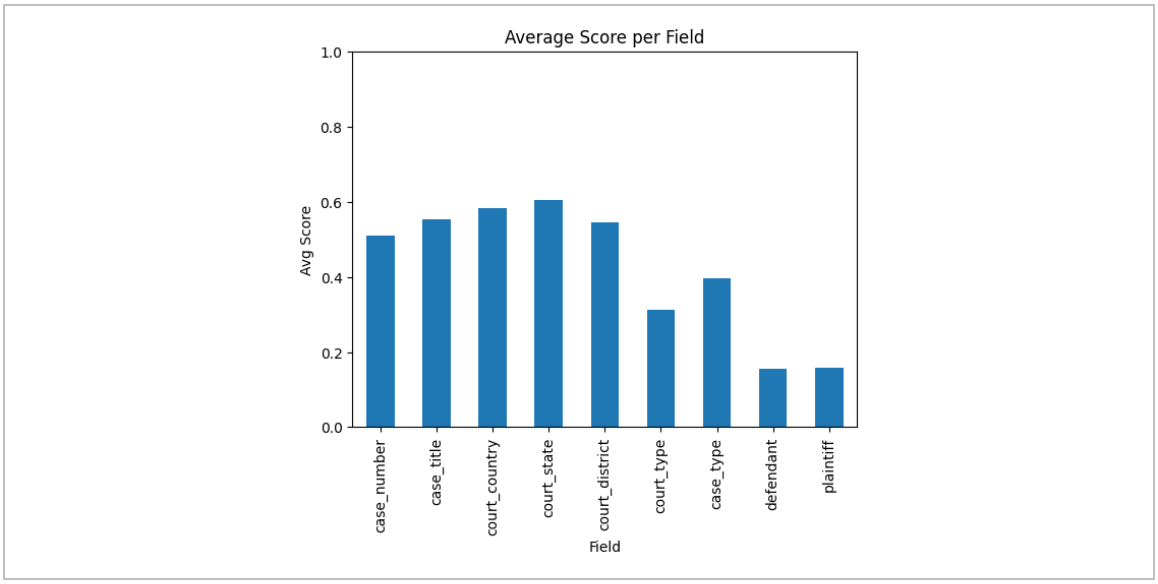


*Figure 20: Average Score per Field of a Court Case Object*

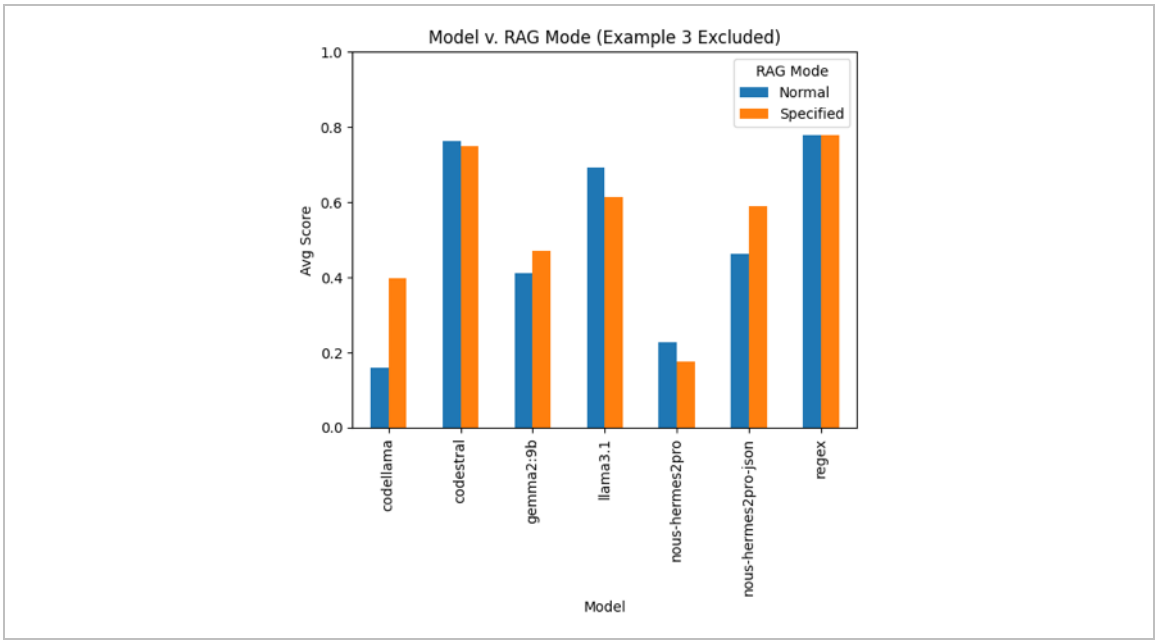


*Figure 21: Average Score per Model vs. RAG Mode (Excluding Example 3)*

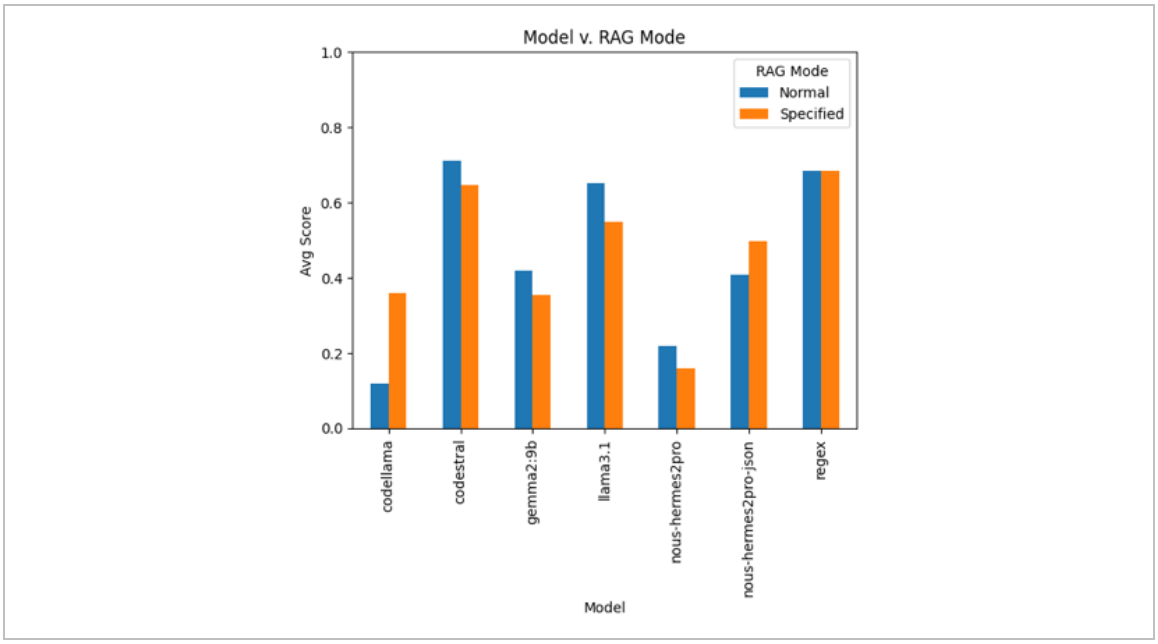


*Figure 22: Average Score per Model vs. RAG Mode*

## Person Object

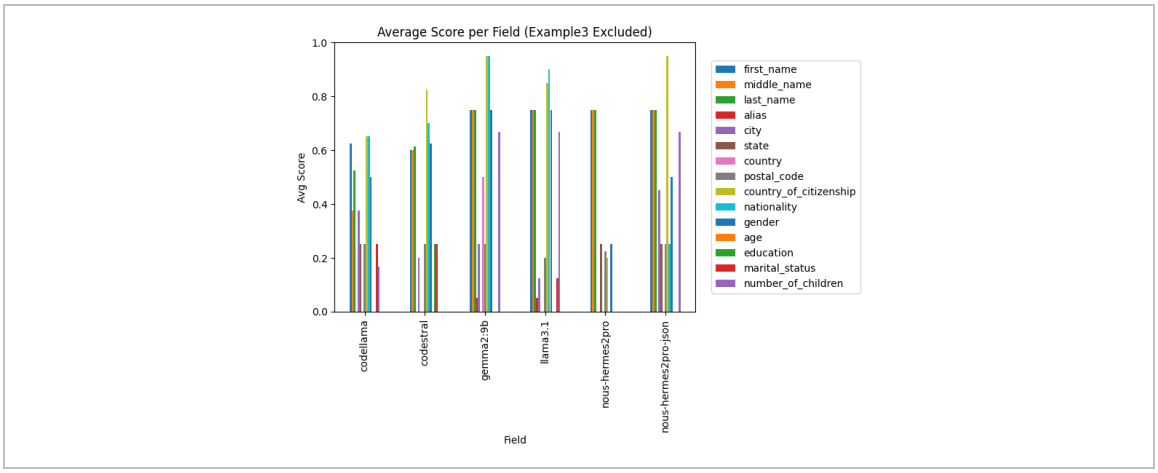


*Figure 23: Average Score per Field of a Person Object for Each Mode (Excluding Example 3)*

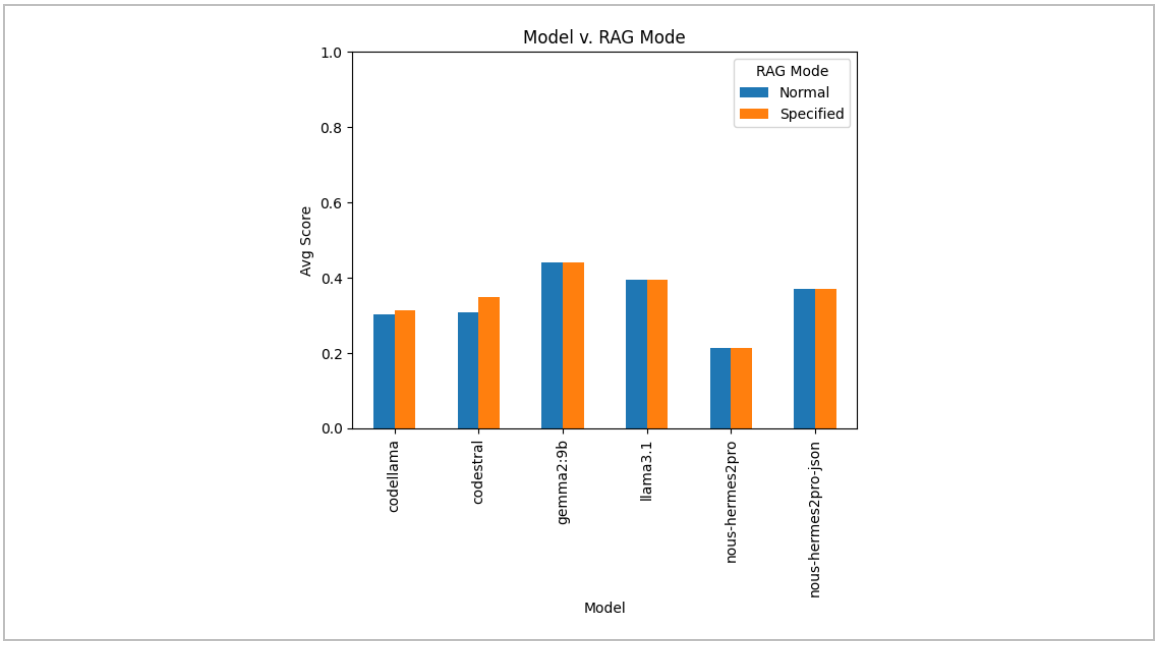


*Figure 24: Average Score per Model vs. RAG Mode*

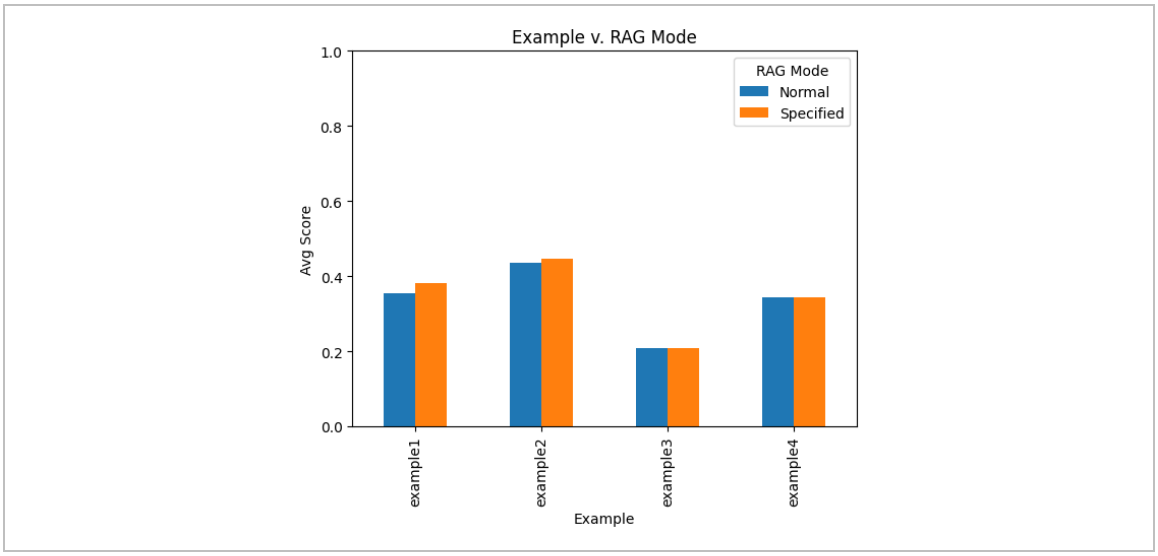


*Figure 25: Average Score per Example vs. RAG Mode*

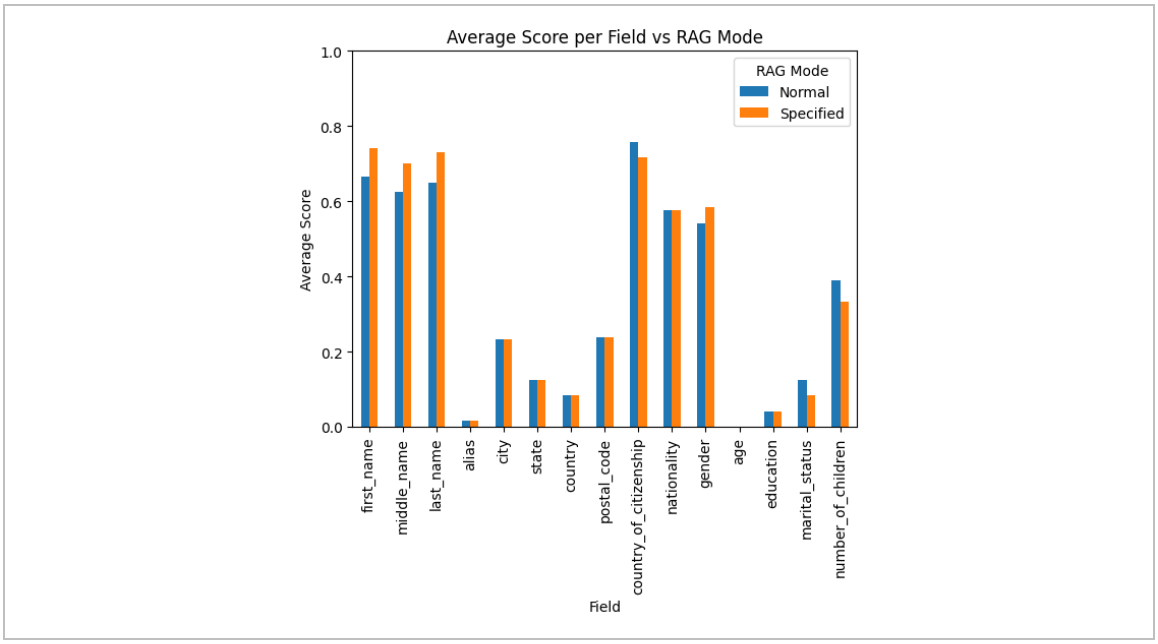


*Figure 26: Average Score per Field of a Person Object vs. RAG Mode*


*Figure 27: Example Improvement That Having the Correct Context Can Give in Terms of Retrieving the Correct Information from the Context*

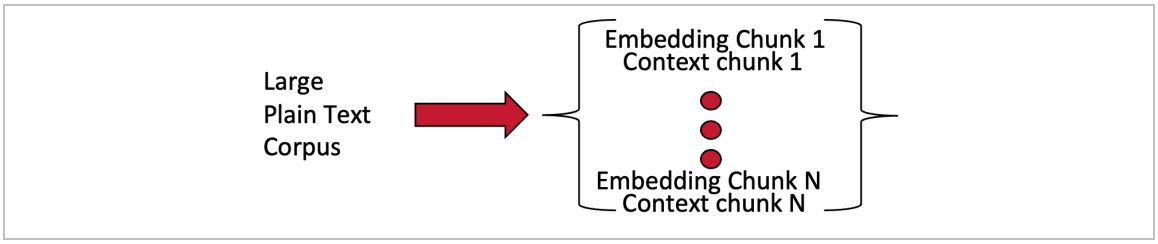


*Figure 28: What Our Preprocessing Steps Produce*
*(For each text chunk, the process results in a (1) context chunk that has minimal to no destructive augmentations and an (2) embedding chunk that has destructive augmentations. The embedding chunk is put into the encoder to create an embedding, and then it is stored in the vector database, while the context chunk is returned from the vector database and used by the LLM to answer the prompt.)*

# References


### [Ai 2024]
Ai, M. Codestral: Hello, World! *Mistral AI Website*. May 29, 2024. https://mistral.ai/news/codestral/

### [Artetxe 2021]
Artetxe, M., et al. Efficient Large Scale Language Modeling with Mixtures of Experts. *ArXiv Website*. October 26, 2022. https://arxiv.org/abs/2112.10684

### [Bavarian 2022]
Bavarian, M.; Jun, H.; Tezak, N.; Schulman, J.; McLeavey, C.; Tworek, J.; & Chen, M. Efficient Training of Language Models to Fill in the Middle. *ArXiv Website*. July 28, 2022. https://arxiv.org/abs/2207.14255

### [Bordes 2024]
Bordes, F. et al. An Introduction to Vision-Language Modeling. *ArXiv Website*. May 27, 2024. https://arxiv.org/abs/2405.17247

**[Ding 2024]**
Ding, Y.; Zhang, L. L.; Zhang, C.; Xu, Y.; Shang, N.; Xu, J.; Yang, F.; & Yang, M. LongRoPE: Extending LLM Context Window Beyond 2 Million Tokens. *ArXiv Website*. https://arxiv.org/abs/2402.13753

**[Dubey 2024]**
Dubey, A. et al. The Llama 3 Herd of Models. *ArXiv Website*. August 15, 2024. https://arxiv.org/abs/2407.21783

**[Edge 2024]**
Edge, D.; Trinh, H.; Cheng, N.; Bradley, J.; Chao, A.; Mody, A.; Truitt, S.; & Larson, J. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *ArXiv Website*. April 14, 2024. https://arxiv.org/abs/2404.16130

**[Ge 2024]**
Ge, C.; Cheng, S.; Wang, Z.; Yuan, J.; Gao, Y.; Song, J.; Song, S.; Huang, G.; & Zheng, B. ConvLLaVA: Hierarchical Backbones as Visual Encoder for Large Multimodal Models. *ArXiv Website*. May 24, 2024. https://arxiv.org/abs/2405.15738

**[Glukhov 2023]**
Glukhov, D.; Shumailov, I.; Gal, Y.; Papernot, N.; & Papyan, V. LLM Censorship: A Machine Learning Challenge or a Computer Security Problem? *ArXiv Website*. July 20, 2023. https://arxiv.org/abs/2307.10719

**[Jiang 2023]**
Jiang, A. Q. et al. Mistral 7B. *ArXiv Website*. October 10, 2023. https://arxiv.org/abs/2310.06825

**[Le Cun 1990]**
Le Cun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; & Jackel, L. D. Handwritten Digit Recognition with a Back-Propagation Network. Pages 396-404. In Proceedings, Conference on Neural Information Processing Systems (NeurIPS). June 1990. https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf

**[Lewis 2020]**
Lewis, P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *ArXiv Website*. May 22, 2020. https://arxiv.org/abs/2005.11401

**[Liu 2023]**
Liu, H.; Li, C.; Wu, Q.; & Lee, Y. J. Visual Instruction Tuning. *ArXiv Website*. April 17, 2023. https://arxiv.org/abs/2304.08485

**[Luo 2017]**
Luo, W.; Li, Y.; Urtasun, R.; & Zemel, R. Understanding the Effective Receptive Field in Deep Convolutional Neural Networks. *ArXiv Website*. January 25, 2017. https://arxiv.org/abs/1701.04128

**[Nous Research 2001]**
Nous Research. Hermes-2-Pro-Mistral-7B. · *Hugging Face Website*. April 15, 2001. https://hugging-face.co/NousResearch/Hermes-2-Pro-Mistral-7B

**[Ollama 2024]**
Ollama. ollama/ollama. *GitHub Website*. August 5, 2024 [accessed]. https://github.com/ollama/ollama

**[Open LLM 2024]**
Open LLM. Open LLM Leaderboard. *Hugging Face Website*. November 11, 2024 [accessed]. https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard

**[OpenAI 2023]**
OpenAI et al. GPT-4 Technical Report. *ArXiv Website*. March 15, 2023. https://arxiv.org/abs/2303.08774

**[Palm 2018]**
Palm, Rasmus Berg; Laws, Florian; & Winther, Ole. Attend, Copy, Parse—End-to-End Information Extraction from Documents. *ArXiv* Website. December 2018. https://doi.org/10.48550/arXiv.1812.07248

**[Pandya 2023]**
Pandya, K. & Holia, M. Automating Customer Service Using LangChain: Building Custom Open-Source GPT Chatbot for Organizations. *ArXiv Website*. October 9, 2023. https://arxiv.org/abs/2310.05421

**[Raghu 2021]**
Raghu, M.; Unterthiner, T.; Kornblith, S.; Zhang, C.; & Dosovitskiy, A. Do Vision Transformers See Like Convolutional Neural Networks? *ArXiv Website*. August 19, 2021. https://arxiv.org/abs/2108.08810

**[Riedl 2023]**
Riedl, M. A Very Gentle Introduction to Large Language Models Without the Hype. *Medium Website*. April 13, 2023. https://mark-riedl.medium.com/a-very-gentle-introduction-to-large-language-models-without-the-hype-5f67941fa59e

**[Ristad 1998]**
Ristad, E. & Yianilos, P. Learning String-Edit Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* Volume 20. Issue 5. May 1998. Pages 522–532. https://doi.org/10.1109/34.682181

**[SEI 2013]**
Software Engineering Institute. Analyzing Insider Threat Data in the MERIT Database [blog post]. *SEI Blog*. October 17, 2013. https://insights.sei.cmu.edu/blog/analyzing-insider-threat-data-in-the-merit-database/

**[SEI IIDES]**
Todo after IIDES is publicly published

**[Smith 2007]**
Smith, R. An Overview of the Tesseract OCR Engine. Pages 629-633. In *Proceedings of the International Conference on Document Analysis and Recognition*. September 2007. https://doi.org/10.1109/icdar.2007.4376991

**[Soylu 2024]**
Soylu, D.; Potts, C.; & Khattab, O. Fine-Tuning and Prompt Optimization: Two Great Steps that Work Better Together. *ArXiv Website*. July 15, 2024. https://arxiv.org/abs/2407.10930

**[Team 2024]**
Team, G. et al. Gemma 2: Improving Open Language Models at a Practical Size. *ArXiv Website*. July 31, 2024. https://arxiv.org/abs/2408.00118

**[Thakur 2023]**
Thakur, P. A Gentle Introduction to Retrieval Augmented Generation (RAG). *Weights and Biases Website*. November 2023. https://wandb.ai/cosmo3769/RAG/reports/A-Gentle-Introduction-to-Retrieval-Augmented-Generation-RAG---Vmlldzo1MjM4Mjk1

**[Touvron 2023a]**
Touvron, H. et al. LLaMA: Open and Efficient Foundation Language Models. *ArXiv Website*. February 27, 2023. https://arxiv.org/abs/2302.13971

**[Touvron 2023b]**
Touvron, H. et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. *ArXiv Website*. July 18, 2023. https://arxiv.org/abs/2307.09288

**[Vaswani 2017]**
Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; & Polosukhin, I. Attention Is All You Need. *ArXiv Website*. June 12, 2017. https://arxiv.org/abs/1706.03762

**[Wei 2022]**
Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.; Le, Q.; & Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *ArXiv Website*. January 28, 2022. https://arxiv.org/abs/2201.11903

**[Xu 2024]**
Xu, R.; Yao, Y.; Guo, Z.; Cui, J.; Ni, Z.; Ge, C.; Chua, T. S.; Liu, Z.; Sun, M.; & Huang, G. LLaVA-UHD: an LMM Perceiving Any Aspect Ratio and High-Resolution Images. *ArXiv Website*. March 18, 2024. https://arxiv.org/abs/2403.11703

**[Yu 2020a]**
Yu, W.; Lu, N.; Qi, X.; Gong, P.; & Xiao, R. PICK: Processing Key Information Extraction from Documents using Improved Graph Learning-Convolutional Networks. *ArXiv Website*. April 16, 2020. https://arxiv.org/abs/2004.07464

**[Yu 2020b]**
Yu, W.; Lu, N.; Qi, X.; Gong, P.; & Xiao, R. PICK: Processing Key Information Extraction from Documents using Improved Graph Learning-Convolutional Networks. *ArXiv Website*. April 16, 2020. https://arxiv.org/abs/2004.07464

**[Zhou 2018]**
Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; & Sun, M. Graph Neural Networks: A Review of Methods and Applications. *ArXiv Website*. December 20, 2018. https://arxiv.org/abs/1812.08434

## Legal Markings

## Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

**Phone**: 412/268.5800 | 888.201.4479
**Web**: www.sei.cmu.edu
**Email**: info@sei.cmu.edu