Benchmarking Methods for Digitizing Print Bibliographies

Elizabeth Rodrigues[1], Muhammad Khalid[2], Shayak Nandi[2], Amelia Vrieze[2], and Tianyang Yu[2]

[1] Libraries, Grinnell College, Grinnell, IA, USA
[2] Computer Science, Grinnell College, Grinnell, IA, USA

Abstract

This short paper reports on work in progress to develop a pipeline for the digitization of a two-column format print bibliography as structured data for query. Print bibliographies constitute a major potential source of data for computational literary studies, but only if their digitization in a useful format can be made technically and economically feasible for interested researchers. To this end, the project seeks to benchmark traditional OCR and multimodal LLM text transcription of idiosyncratically formatted text and its conversion to JSON format in the resource-constrained and pedagogically motivated context of a small liberal arts college.

Keywords: multimodal large language models, structured data, dataset creation

1 Introduction

Published bibliographies are a crucial tool for assembling relevant and well-documented corpora for specific areas of literary inquiry. Widespread efforts to digitize books in general have not resulted in masses of bibliographic datasets because simple digitization is not enough to make these data truly usable. Bibliographies need to be transformed into structured data to allow querying and visualization. Because most bibliographies are by definition a niche resource, aimed at more or less exhaustive representation of a small range of texts, unlocking the data potential of print bibliographies may depend on developing workflows that are feasible for small research teams with limited labor, time, and money.

Multimodal LLMs (mLLMs) may have made this pipeline more achievable. With the ability to receive image as well as text input, mLLMs have shown promise in both OCR and OCR post-correction tasks, and they are capable both of turning transcriptions into structured data and generating structured data directly from images. Before humanities researchers engage these tools for data creation widely, though, it should be determined whether the speed, accuracy, and cost are superior to traditional OCR. The goal of this project is to document the creation of a pipeline for the digitization of print bibliographies as structured data usable by a small research team at an undergraduate institution and benchmark its performance compared to traditional OCR. This work-in-progress report shares early findings from benchmarking experiments for OCR and JSON accuracy across Tesseract, OpenAI, and Gemini models.[1]

2 Related Work

The goal of digitizing a print bibliography as structured data brings together the tasks of OCR transcription for historical research purposes and the parsing of transcribed text as structured data.

Reference extraction typically takes a PDF as input for two subtasks: identification of references as distinct from article text and parsing of reference text into structured format. In their review article, Backes et. al provide an overview of freely available approaches and tools for reference extraction, including EXparser, Grobid,Cermine, and AnyStyle. Due to the recency of mLLM approaches, they do

Elizabeth Rodrigues, Muhammad Khalid, Shayak Nandi, Amelia Vrieze, and Tianyang Yu. "Benchmarking Methods for Digitizing Print Bibliographies." In: *Computational Humanities Research 2025*, ed. by Taylor Arnold, Margherita Fantoli, and Ruben Ros. Vol. 3. Anthology of Computers and the Humanities. 2025, 1359–1370. https://doi.org/10.63744/SisvHqHBH67Z.

[1] Code and reproduction data can be found at `https://github.com/GC-20C-Text-Lab/ocr-benchmarking`

not have review results for mLLMs [1]. They find that traditional methods are reportedly effective, but most tools developed for this purpose lack open and well-maintained code.

OCR and information extraction for humanities research using mLLMs is a recent and rapidly expanding area of inquiry with many open questions surrounding performance, prompt optimization, and cost. The relative recency of mLLM availability also means that tools studied and capabilities demonstrated are shifting rapidly. Shi et al. studied Gpt-4Vision for a variety of OCR-related tasks included information extraction and found that performed general basic transcription well for Latin alphabet texts but struggled with multilingual contexts and visual parsing of documents for structured information extraction [9]. Liu et al. find that Gemini (model unspecified, presumably 1.0) and Gpt-4Vision outperform other approaches for structured information extraction on documents from a variety of domains, but they also note that the reasons why these models can perform OCR tasks well remain unclear and therefore difficult to finetune [8].

More recent work on tasks similar to the goals of this project have consistently demonstrated promising results for generally trained mLLMs on structured information extraction from page images in a variety of fonts for Latin alphabets. Ghiriti et al. examined Gpt-4Vision for OCR of an early twentieth-century German magazine and found that the mLLM outperformed traditional OCR in general, but on pages with minimal degradation and alignment issues also showed poor results [3]. Kanerva et al. caution that languages other than English may see far worse outcomes [5]. Kim et al. evaluate mLLM transcription with Gpt-4o and Claude Sonnet 3.5 for handwritten texts in a tabular format and Greif et al. (2025) evaluate Gpt-4o and Gemini 2.0 for transcription of directory data in a historical font [4; 7]. Both studies compare mLLM results with that of traditional OCR workflows and find that mLLMs are superior when used with well-devised prompts. Bauder and Jones explored the potential for using mLLMs to perform end-to-end the process of transcribing, structuring, locating full text for, and downloading full text for works in a set of nineteenth-century private library catalogs [2]. They found that mLLMs provided fast and highly accurate structured data transcription but generally trained models could not offer much assistance in corpus assemblage due to limited and inconsistent ability to query the web.

This project contributes to this area of inquiry by further benchmarking LLM performance for structured data transcription for standardized contemporary font with idiosyncratic layout and providing a notebook pipeline using API calls to facilitate bulk processing.

3 Data

Our test case for this pipeline is Louis Kaplan's Bibliography of American Autobiography (1961) [6]. In addition to publication metadata, Kaplan provides a brief subject description and a subject index, where all works related to a range of themes and identities groups (such as "Physicians") are listed by index number (not page number). Some of indexed topics have hundreds of relevant entries, a potential gold mine for a researcher looking to get beyond widely cited texts and develop a much larger corpus.

All print bibliographies will share some common features in layout: they will consist of discrete entries in which the order of information is significant for determining meaning. Several format idiosyncrasies make Kaplan challenging to transform into structured data, either manually or digitally. Transcribing hundreds of relevant entries for a particular topic is made very laborious by the need to determine what page each index number is on. Assuming a researcher has access to a scanner and a high-quality OCR tool, the two-column format means that each page may still need a substantial amount pre-processing or post-correction just for legibility. Traditional OCR would also leave the step of parsing the text into a structured format, which is made difficult by the bespoke citation format that does not use consistent field delimiters and allows omitted fields. While these are the specific challenges presented by Kaplan, a review of other print bibliographies such as Wright's American Fiction, 1774-1850 [11] and Watson et al.'s The New Cambridge Bibliography of English Literature [10] indicates that most if not all print bibliographies will have their own set of idiosyncrasies. A pipeline for transcribing Kaplan may not be generalizable in specifics, but it will be useful in modeling approaches to dealing with such idiosyncrasies.

As a test set, 43 pages of Kaplan were scanned as tiffs using an EPSON 10000XL flatbed scanner at 300dpi in full color setting.

4 Method

4.1 OCR Creation

We transformed tiffs to pngs and performed OCR using pytesseract, a Python wrapper for Google's Tesseract OCR engine. Tesseract was selected because it is a popular, free, and well-documented tool. We used the default configuration to establish a standard baseline, representing typical OCR implementation without fine-tuning. Our resulting plain text output preserved all formatting and text detection errors inherent in traditional OCR processing, including the challenges associated with the two-column layout and marginalia present in the data.

We created additional OCR text files using the mLLMs to generate text directly from an image and, in a separate pass, to correct the corresponding pytesseract file with the alongside, also known as post-correction. These post-correction pages were then manually reviewed and corrected to create textual ground truth.

4.2 OCR Accuracy Metrics

We use character error rate (CER), word error rate (WER), and token sort ratio (TSR) to calculate accuracy. The CER is equal to the Levenshtein distance divided by the character count of the ground truth. The Levenshtein distance is defined as the sum of the number of characters inserted, deleted, and substituted to transform the transcription to the ground truth. Likewise, the WER is the sum of the number of words inserted, deleted, and substituted divided by the total number of words in the ground truth. The TSR is a metric which measures similarity between the ground truth and transcription without respect to ordering.

We report accuracy metrics for both non-normalized and normalized results. For non-normalized results, pre-processing includes (i) replacing line breaks and tabs with spaces, (ii) removing all instances of hyphens followed by spaces, except for numeric intervals where hyphens remain, (iii) replacing multiple spaces with one space, and (iv) removing leading and trailing spaces. These methods will reduce the differences inherent between OCR software and LLMs as well as small differences in transcription that do not affect how the data is understood and may have arisen from human discernment or error when creating the ground truth.

4.3 JSON Creation

To create usable structured JSON output using mLLMs, we needed a way to enforce the structure of the output for benchmarking comparison as well as usability for future database construction. Some mLLMs have built-in features for defining strict guidelines for structured output. For cross-model compatibility, we used a structured output solution called Instructor, a third-party tool for creating and validating structured output from a variety of mLLM providers. We were able to use a single JSON schema for every model, provided in A. Strings have the default value of the empty string ("") and integers have the default value of 0. Because each entry in the original text can exclude any of the fields, a unified way to handle nonexistent information was an important consideration. Almost every entry is missing at least one of the fields (most commonly the maiden name). Assuring the mLLM that it is acceptable for an entry to be missing information and providing guidance of what value to assign to those fields is another way to discourage hallucinations. For cut-off entries at the start or end of pages, whichever information is present on that page is included as its own entry. Ground truth was created by running the mLLM JSON creation process on each page, and then manually correcting it by referencing the original scanned image of the page.

4.4 JSON Accuracy

JSON accuracy consists of opening the JSON output as tabular data and reporting the number of matching cells out of all cells. We use three metrics to calculate whether two corresponding cells match. First, the exact match compares whether two cells match exactly after alignment pre-processing is done. Next, the normalized match compares whether two cells match exactly after only alphanumeric, underscore, and whitespace characters are kept and multiple spaces are replaced with single spaces for text cells. Finally, the fuzzy match uses RapidFuzz's Jaro-Winkler algorithm with a match threshold of 0.9 on the non-normalized data.

4.5 Pipeline Architecture

Our processing pipeline builds upon Grief et al.'s Python scripts, adapted to meet our requirements for bibliographic data. Our pipeline starts with PNG images of pages and converts these into two output formats: plain text and structured JSON.

The pipeline processes each page through five distinct methods.[2] Each method (excluding raw OCR) is executed through both Gpt-4o and Gemini-2.5-Flash, which results in 9 total outputs per page per experimental run.

1. Workflow 1 (3 steps):
    - Raw OCR extraction (output in plain text): Pages are processed through pytesseract to generate raw OCR text - without mLLM intervention.
    - OCR post-correction (output in plain text): Raw OCR text and the original PNG are both fed into the mLLM with the OCR output clearly labelled in the prompt.
    - JSON from corrected text (output in JSON): OCR post-corrected text is used as input to generate structured JSON output.
2. Workflow 2 (1 step):
    - Direct image transcription (output in plain text): The original PNG is processed by the mLLM.
3. Workflow 3 (1 step):
    - Direct JSON from image (output in JSON): The original PNG is processed directly to produce structured JSON data.

[2] Given that our workflow involved processing each image through 5 methods with two mLLMs per method in addition to the raw OCR processing step, we were dealing with a significant amount of time dedicated to a single pipeline run with an estimated 30-45 seconds per mLLM API call. To address this, we implemented asynchronous API calls with rate-limiting for all mLLM-related tasks. This optimization reduced processing time by approximately 70 percent for each pipeline execution allowing us to address other issues in the development process and making large-scale benchmarking a feasible goal.

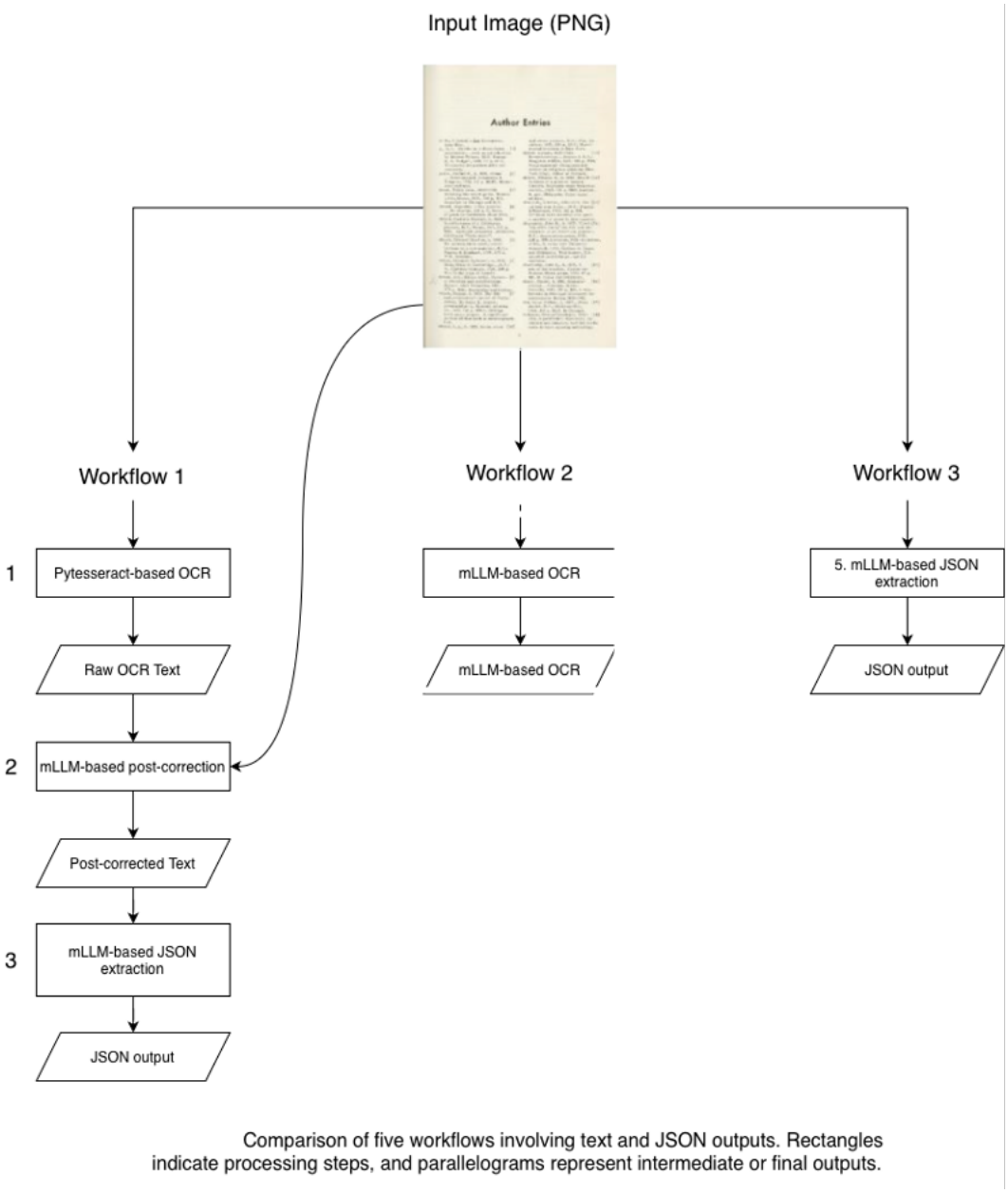

Figure 1: Text processing pipeline

4.6 Model Selection

Our selection criteria was two-fold. First, we needed to ensure computational efficiency for large-scale processing which required us to use models capable of handling substantial document volumes without compromising on compute and time-related limitations. Secondly, we sought state-of-the-art multimodal capabilities to minimize potential bottlenecks during correction to ensure that models could effectively utilize both visual and textual data sources.

Based on these requirements, we selected Gpt-4o (OpenAI) and Gemini-2.5-Flash (Google) to process the images and correct our OCR output. Both models had sufficient context windows for lengthy bibliographic entries and supported asynchronous processing which proved to significantly reduce the time it took to run our benchmarks. They also supported structured JSON output through the instructor library, a key component in one of our benchmarks.

4.7 Prompt Engineering

Prior researchers stress the importance of careful prompt engineering in their paper. Initial prompt testing was done with the AI Studio version of Gemini 2.5 Pro (temperature = 0). Gemini 2.5 Pro required minimum prompting to behave how we wanted it, with very few noticeable errors. The only extra instructions were to put the index numbers at the end of entries, put each entry on a separate line, and to not make up information that did not appear on the page.

Unfortunately, the API access to this model was not free, so we reverted to Gemini 2.5 Flash for the rest of our testing. While continuing to test, we noticed that on pages with handwritten text the mLLM was able to transcribe those notes as well, so the line "Ignore handwritten marginalia when transcribing" was added to the end. The mLLM would sometimes avoid transcribing the headings and page numbers, possibly because they were not "entries." Therefore, "Transcribe all text that appears on the page" was added within the prompt. Next, testing needed to be expanded to include GPT-4o as well. Unexpectedly, GPT-4o did not respond well to some aspects of the prompt that improved the behavior in Gemini. Asking the API to provide the output in a txt file occasionally caused it to insist that it did not have this functionality. The line about handwritten marginalia caused it to sometimes add a message at the top that it could only transcribe typed text. At this point, we combined aspects of our current prompt with a prompt that GPT-4o had responded well to, in hopes of creating a prompt that was acceptable to both models. The biggest change was providing the model a "role" to play, which we can hypothesize reduced direct messages to the user being inserted at the beginning of output because a "text correction assistant" would not need to speak to a user. The role also provides a justification for transcribing a book which reduced refusals of the task. Finally, a separate prompt was created for the direct-from-image transcription. Prompts are included in Appendix B.

To fully maximize the behavior of either model, separate prompts would be used that tailor to their differences, but in order to fairly benchmark their performance compared to one another a universal prompt was needed.

5 Results

Gpt-4o consistently outperformed Gemini 2.5 Flash for unstructured OCR accuracy, with all differences except for token sort ratio between the two mLLMs statistically significant at the $p<.05$ level. Both mLLMs outperformed Pytesseract significantly. Two considerations, however, are that we used a paid version of Gpt-4o in order to have API access and that the prompt we ended up using for both models may have slightly favored Gpt-4o due to it being more likely to refuse tasks, leading us to tailor the prompt to avoid them. The cost of using the paid model for the 5822 pages sent in the course of developing and then running experiments using the pipeline totaled USD 35.41, or a little more than half a cent per page.

Method	Text Processing	Metric	gemini-2.5-flash	gpt-4o	pytesseract
LLM trans.	Not normalized	CER (%)	5.39	2.40	9.94
		TSR (%)	98.63	98.60	97.02
		WER (%)	8.67	4.68	15.54
	Normalized	CER (%)	4.26	1.88	7.72
		TSR (%)	99.30	99.20	98.11
		WER (%)	5.22	2.34	11.39
LLM OCR post-corr.	Not normalized	CER (%)	5.21	1.74	9.94
		TSR (%)	98.47	98.65	97.02
		WER (%)	7.52	3.99	15.54
	Normalized	CER (%)	4.03	1.38	7.72
		TSR	99.14	99.27	98.11
		WER (%)	5.14	1.41	11.39

Table 1: Transcription accuracy overview by method and model.

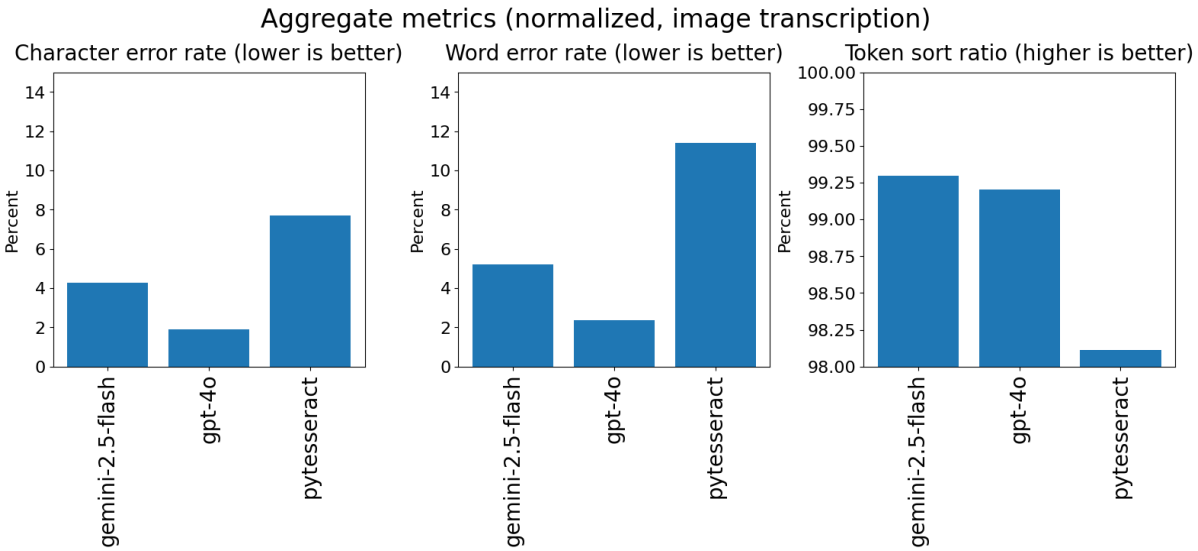

Figure 2: Aggregate metrics for normalized text, transcribed from image

For JSON creation directly from images, however, Gemini 2.5 Flash consistently outperformed Gpt-4o, with a 94 percent fuzzy match rate. Gemini also performed slightly better creating JSON from images than from text. There was not a statistically significant difference between the models for JSON creation from post-corrected text. We were able to use the Gemini 2.5 Flash API at no cost during this experiment; however, we began to hit rate limits when we attempted multiple runs of the entire experiment in one day.

Much analysis remains to be done on error patterns in the JSON and their effect on data usability, but as a preliminary overview we examined the granular match rates for each field. The lowest match rates were for the “library” field, for which values are drawn from a set of abbreviations specific to this work, likely making them more of a challenge for the model to predict (see Figure 3 and Figure 4).

Method	Text Processing	Metric	gemini-2.5-flash	gpt-4o
Image to JSON	Fuzzy matching	Matches (%)	94.24	86.78
	Not normalized	Matches (%)	89.48	80.77
	Normalized	Matches (%)	93.46	85.40
Post-corrected text to JSON	Fuzzy matching	Matches (%)	92.34	88.91
	Not normalized	Matches (%)	88.41	84.02
	Normalized	Matches (%)	91.77	88.26

Table 2: JSON accuracy overview by method and model.

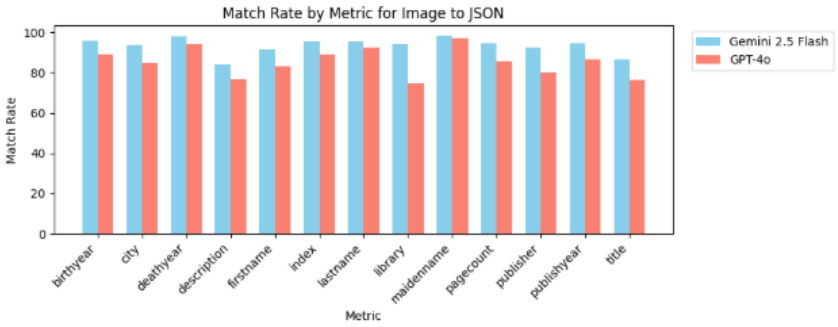

Figure 3: Field error rates for image to JSON transcription

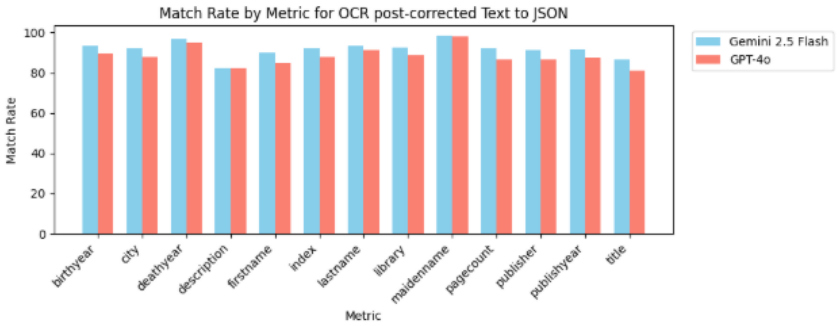

Figure 4: Field error rates for OCR post-corrected text to JSON transcription

Two broad categories of error that are already noticeable in the text to JSON process are incomplete entries due to page splits, incomplete entries due to the model either skipping text or not recognizing that text should belong to a field (see Figure 5).

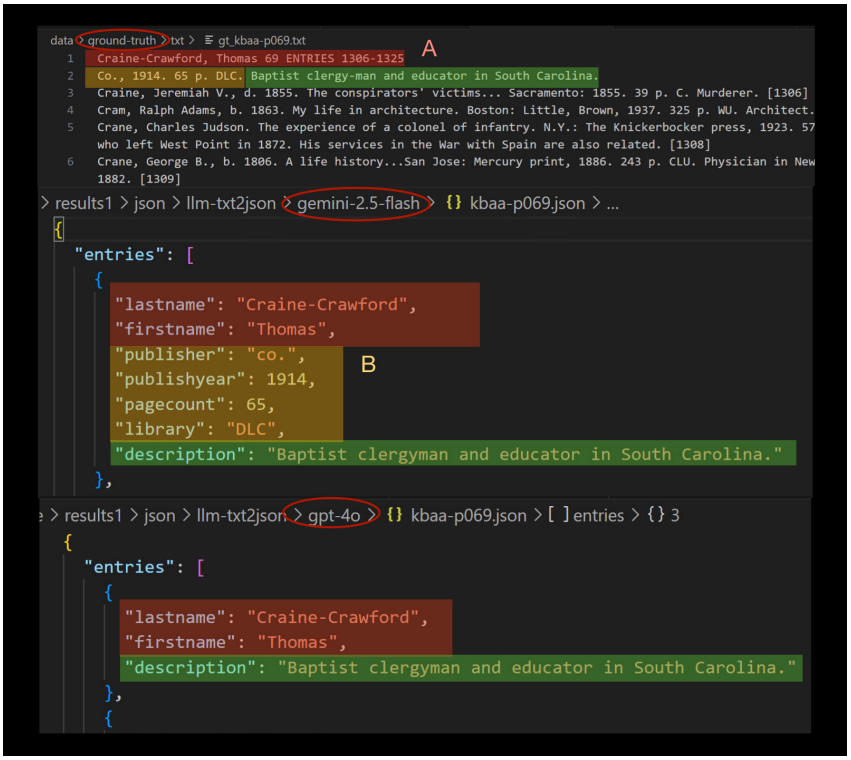

Figure 5: A: Both models misinterpreted the page heading as the name of the author due to the first entry continuing from the previous page. B: Since the name of the publisher is cut off on the original page this publisher field, while strange, is correct.

6 Next Steps and Open Questions

Our experiment was focused on answering questions about the overall feasibility of this approach, as opposed to perfecting the process. Our images were intentionally scanned at high resolutions and were usually just under 15MB, which would require additional processing to be usable with Claude, which sets a 5MB upper limit for image upload. Scanning the images in black and white and scanning at a lower resolution may be a possible way to decrease the number of tokens required to resolve requests, but more testing would be required to see if there is a tradeoff where lower resolutions eventually produce worse outcomes.

The biggest issue to resolve before this pipeline would be ready for production use would be to address entries split between pages. For a structured dataset of the entire text to be "complete," a method of combining cut-off entries from the beginning and ends of adjacent pages would need to be devised.

Ultimately, we want to be able to use the transcribed JSON object to populate a database for query and visualization. We are currently exploring the use of a model context protocol (MCP) architecture for a streamlined, low/no-code environment for structured data extraction, database creation, and query.

Acknowledgements

The authors are grateful for the support of the Grinnell College Summer Mentored Advanced Project program.

References

[1] Backes, Tobias, Iurshina, Anastasiia, Shahid, Muhammad Ahsan, and Mayr, Philipp. "Comparing free reference extraction pipelines". en. In: *International Journal on Digital Libraries 25*, no. 4 (Dec. 2024), pp. 841–853. DOI: https://doi.org/10.1007/s00799-024-00404-6.

[2] Bauder, Julia and Jones, Chris. “Using Generative AI to Turn 19th- Century Library Catalogues into Data: Applications and Limitations”. In: *Library Catalogues as Data: Research, Practice and Usage*, ed. by Melissa Terras Paul Gooding and Sarah Ames. Facet Publishing, 2025, pp. 167–184.

[3] Ghiriti, Alex, Göderle, Wolfgang, and Kern, Roman. “Exploring the Capabilities of GPT4-Vision as OCR Engine”. In: *Linking Theory and Practice of Digital Libraries: 28th International Conference on Theory and Practice of Digital Libraries, TPDL 2024, Ljubljana, Slovenia, September 24–27, 2024, Proceedings, Part II*. Berlin, Heidelberg: Springer-Verlag, Sept. 2024, pp. 3–12. DOI: https://doi.org/10.1007/978-3-031-72440-4_1.

[4] Greif, Gavin, Griesshaber, Niclas, and Greif, Robin. “Multimodal LLMs for OCR, OCR Post-Correction, and Named Entity Recognition in Historical Documents”. 2025. arXiv: 2504.00414 [cs.CL]. URL: https://arxiv.org/abs/2504.00414.

[5] Kanerva, Jenna, Ledins, Cassandra, Käpyaho, Siiri, and Ginter, Filip. “OCR Error Post-Correction with LLMs in Historical Documents: No Free Lunches”. arXiv:2502.01205 [cs]. Feb. 2025. DOI: 10.48550/arXiv.2502.01205.

[6] Kaplan, Louis. *A Bibliography of American Autobiographies*. Madison: University of Wisconsin Press, 1961.

[7] Kim, Seorin, Baudru, Julien, Ryckbosch, Wouter, Bersini, Hugues, and Ginis, Vincent. “Early evidence of how LLMs outperform traditional systems on OCR/HTR tasks for historical records”. arXiv:2501.11623 [cs]. Jan. 2025. DOI: 10.48550/arXiv.2501.11623.

[8] Liu, Yuliang, Li, Zhang, Huang, Mingxin, Yang, Biao, Yu, Wenwen, Li, Chunyuan, Yin, Xucheng, Liu, Cheng-lin, Jin, Lianwen, and Bai, Xiang. “OCRBench: On the Hidden Mystery of OCR in Large Multimodal Models”. In: *Science China Information Sciences 67*, no. 12 (2024). arXiv:2305.07895 [cs]. DOI: https://doi.org/10.1007/s11432-024-4235-6.

[9] Shi, Yongxin, Peng, Dezhi, Liao, Wenhui, Lin, Zening, Chen, Xinhong, Liu, Chongyu, Zhang, Yuyi, and Jin, Lianwen. “Exploring OCR Capabilities of GPT-4V(ision): A Quantitative and In-depth Evaluation”. arXiv:2310.16809 [cs]. Oct. 2023. DOI: https://doi.org/10.48550/arXiv.2310.16809.

[10] Watson, George, Pickles, J. D, Willison, Ian Roy, and Bateson, Frederick Wilse. *The New Cambridge Bibliography of English Literature*. Cambridge: Cambridge University Press, 1969.

[11] Wright, Lyle. *American Fiction, 1774-1850; A Contribution Toward a Bibliography*. California: Ward Ritchie Press, 1939.

A Entry Schema Using Instructor

```
class Entry(BaseModel):

lastname: str = Field(default="", description="The author\'s last
   name. In the original entries, names are written in Last Name,
   First Name format. If the name is followed by \"(pseud.)\", this
   indicates that the name is a pseudonym, but it should still be
   stored in this field. If this information is not found, assign
   the empty string to this field.",)

firstname: str = Field(default="", description="The author\'s first
   name(s). In the original entries, names are written in Last Name,
    First Name format. If this information is not found, assign the
   empty string to this field.",)
```

```
maidenname: str = Field(default="", description="The author\'s
    maiden name. If they have one, it will be found in parentheses
    after the rest of the name. If this information is not found,
    assign the empty string to this field.",)

birthyear: int = Field(default=0, description="The year the author
    was born. In the original entries, it will be written either as b
    .YEAR or YEAR-YEAR where the first year is the birth year. If
    this information is not found, assign 0 to this field.",)

deathyear: int = Field(default=0, description="The year the author
    died. In the original entries, it will be written as YEAR-YEAR
    where the second year is the death year. If this information is
    not found, assign 0 to this field.",)

title: str = Field(default="", description="The title of the book.
    In the original entries, it should appear after the author's name
     and birth/death dates. If this information is not found, assign
    the empty string to this field.",)

city: str = Field(default="", description="The city the book was
    published in. In the original entries, it is often followed by a
    colon. It may occasionally take the format CITY, STATE. It is
    acceptable to include the state in this field. In entries that
    include the words \"No imprint\", this information is often
    unavailable. If this information is not found, assign the empty
    string to this field.",)

publisher: str = Field(default="", description="The name of the
    publisher which published the book. In the original entries, it
    often follows a colon. In entries that include the words \"No
    imprint\", this information is often unavailable. If this
    information is not found, assign the empty string to this field
    .",)

publishyear: int = Field(default=0,description="The year the book
    was published. Usually separated from the name of the publisher
    by a comma, although occasionally a period was used instead. In
    entries that include the words \"No imprint\", this information
    is often unavailable. If this information is not found, assign 0
    to this field.",)

pagecount: int = Field(default=0, description="The number of pages
    in the book. Found in the format NUMBER p. If this information is
     not found, assign 0 to this field.",)

library: str = Field(default="", description="The abbreviated name
    of the library that the book was found in. In the original
    entries, this information is found after the page count. If this
    information is not found, assign the empty string to this field
```

```
    .",)

description: str = Field(default="", description="A description of
    the book, or the author\'s occupation. After the library field,
    the description consists of whatever is left of the entry (unless
     the index number is treated as occurring at the end of the entry
    ). In entries where the author uses a pseudonym, the information
    after \"(pseud.)\" should be included in this field. If this
    information is not found, assign the empty string to this field
    .",)

index: int = Field(default=0, description="The index number of the
    bibliography entry. In the original entries, this information is
    enclosed in square brackets. Entries that redirect elsewhere due
    to the author using a pseudonym do not have an index number. If
    this information is not found, assign 0 to this field.",)

\# Container so that multiple entries are contained in a single JSON
     object

class Entries(BaseModel):

entries: List[Entry]
```

B Prompt Engineering

B.1 Initial prompt

Please correct the Tesseract transcription of this bibliography page using the scanned image of the page and provide the output in a txt file. Where index numbers in square brackets appear, they should be moved to the end of the entry. Put each entry on a separate line. Only transcribe text that appears on the page and do not attempt to predict missing information or complete cut off entries.

B.2 Improved prompt

You are a text correction assistant. Your task is to clean up and correct errors from raw OCR output. The text may contain misrecognized characters, broken words, or incorrect formatting. Carefully read the provided OCR output, compare it to the original image, and produce a corrected version that is as faithful to the original content as possible. Only correct obvious OCR errors, and do not attempt to complete cut-off entries or predict missing information. Put each entry on a separate line. When an entry has an index number in square brackets, place it at the end of the entry. Input (Raw OCR Text):

B.3 Direct from image prompt

Your task is to transcribe this image of a historical bibliography page as faithfully as possible. Only transcribe typed text that appears on the page and do not attempt to predict missing information or complete cut off entries. Put each entry on a separate line. When an entry has an index number in square brackets, place it at the end of the entry.