State of the Art Report

Enhancing Historical Newspaper Analysis through NLP

ProQuest MDP 2024

Date
December 9, 2024

Student Cohort
Searen Da
Maria Figueiredo
Pedro Lapagesse
Nate Mettke
Nick Schumacher
Hannah Sun

ProQuest Sponsors
John Dillon
Dan Hepp

Faculty Sponsor
Sindhu Kutty

Table of Contents

1. Current Project Status Overview

The 2024 ProQuest MDP cohort is tasked with automating newspaper article segmentation of the Detroit Free Press (DFP). With the majority of entries in the ProQuest database currently being documented only at the page level, digitizing individual articles will improve the searchability of newspaper content, allowing researchers and students to efficiently find articles based on title, content, or author. Automation will also provide users with access to the isolated image and text of each article, facilitating more targeted and efficient content analysis. This initiative is expected to enhance data accessibility and increase user engagement with Clarivate and ProQuest's digital newspaper content.

Manual segmentation as is done currently is both time-intensive and costly, with an estimated expense of approximately 30 cents per page. Therefore, it is crucial to develop an automated pipeline that efficiently processes newspaper pages to produce visual and text-based information for each article. Development details are largely open-ended, allowing the cohort to determine the most suitable technical methods for the pipeline.

Figure 1. Concept of segmented articles

As of December 9, 2024, our finalized model pipeline automatically segments Detroit Free Press front pages at the cost of approximately 15 cents per page. The model performs the following on each DFP page:

- Classifies the title(s) of each article
- Classifies the byline(s)/author(s) of each article
- Classifies the body text lines of each article
- Arranges the lines in each article into reading order (threading)
- Classify and remove artifacts, including any image, caption, ad, index, and masthead.

The output of the model includes the following for each page:

- A JSON representation with positional information for each article
- The concatenated text content for each article
- The newspaper image color-coded by article

Described in detail in *Sponsor Requirements* and *Data Evaluation and Results*, the model meets most of the sponsor-set requirements; if not, it is within an 8% margin of the respective threshold. In addition to this comprehensive report, the team has delivered all code, the complete set of ground truth/manually segmented pages, and a final presentation to ProQuest executives.

2. Project Summary

2.1. Sponsor-Provided Resources

For each newspaper page, ProQuest has provided a JP2 image and an XML file generated through Optical Character Recognition (OCR) of that image. These documents are stored in TDM Studio, ProQuest's text and data mining platform which provides secure access to historical newspapers while enabling researchers to analyze data using a Jupyter interface.

In particular, the XML file organizes the page into a hierarchical structure of TextBlocks, each containing a certain number of TextLines, which represent individual lines of text. Within each TextLine, there are further subdivisions into Strings, which represent individual words.

```
<TextBlock
    xmlns:ns12="http://www.w3.org/1999/xlink" ID="TB.0001b
     .12" HEIGHT="376" WIDTH="648" HPOS="14480" VPOS
     ="3464" ns12:type="simple" language="en">
    <TextLine ID="TB.0001b.12_0" HEIGHT="376.0" WIDTH="648
     .0" HPOS="14480.0" VPOS="3464.0">
        <String ID="TB.0001b.12_0_0" STYLEREFS="TS_10.0_I"
     HEIGHT="376.0" WIDTH="648.0" HPOS="14480.0" VPOS
     ="3464.0" CONTENT="OF" WC="0.998"/>
    </TextLine>
</TextBlock>
```

Figure 2. Example XML structure with content and positional information

Structurally, nearly all TextBlocks are highly inaccurate, spanning multiple or fragments of articles, rendering them unsuitable for precise segmentation. Consequently, our model pipeline focuses on TextLines as the next largest and more reliable element. Each TextLine includes the following positional metadata: width, height, and starting/ending horizontal and vertical positions. The content can be obtained by concatenating the line Strings. Using these heuristics, the model works to accurately group TextLines and reconstruct complete, coherent articles.


The Detroit Free Press

FRANCE COMPLETES MILITARY PLANS FOR RUHR INVASION

STATLER PLANS NEW $9,000,000 DETROIT HOTEL

Death Hovering Over Evelyn Nesbit, Report

ARRESTS SEEN IN PARISH MOB PROBE MONDAY

MICHIGAN MAN WHO IS TIPPED TO SUCCEED FALL

FIND PURCHASE OF ISLE ROYALE IN JOKER CLASS

SENATOR LEADING MOVE TO RECALL TROOPS ON RUHR

65,000 TROOPS READY TO MOVE WITH WEYGAND

SAYS GREESON HELD HIM UP

DEATH PENALTY TALK REVIVED

ORDERS TRIAL FOR MONAGHAN

HAMON WIDOW SEEKS DIVORCE

JUDGE RAPS "FREE LOVE" IN CHICAGO

Is Every Nude Naked? No, Says Censor Cop

THREAT TO THROW ACID DRAWS FINE

Says Whisky May Appear From Hip

APPEALS FAIL; WOMAN MUST DIE FOR CRIME

Hansen Fails In New Plot To Escape Jail

Many Pro-Greeks Hanged by Kemal

Girl-Wife Admits Slaying Husband

Says White House Is Big Fire Trap

Farm Credits Bill Obtains Approval

Figure 3. JP2 image of Detroit Free Press front page on January 7, 1923

2.2. Project Deliverables and Scope

The cohort's aim was to deliver reusable code that operates efficiently within specified speed, memory, and cost constraints. Additionally, we produced evaluation statistics, such as numerical metrics and visualizations, alongside a collection of input/output pairs organized chronologically by year and date. Comprehensive documentation was also provided to our sponsors, detailing the pipeline's design and describing functionality.

Due to the limited project timeline, we chose to optimize pipeline results for a select set of goals rather than attempting to achieve average performance across many. The scope of the project

focuses on identifying titles, authors/bylines, and body text on the front page. Consequently, each page ID (e.g., 1956030101) will correspond to a single page.

For each article, its title, byline, and body text should be grouped together. The merged content should be arranged in reading order, referred to as "threading." Non-article and visual artifacts such as mastheads, photos, indices, and ads should be ignored. Brief previews of articles to be published in the future, lottery information, and photo captions should also be regarded as artifacts.

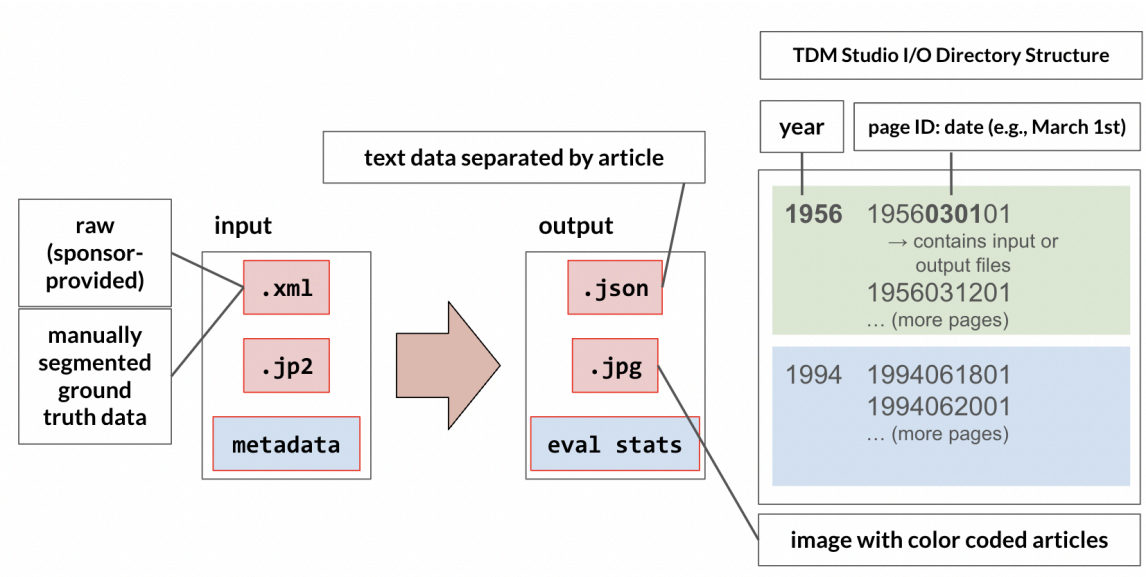


Figure 4. Input and output structure, visualized

Evaluation metrics were developed to assess the model's performance across the aforementioned objectives. To support the evaluation process, we manually segmented a subset of the sponsor-provided XML files to generate ground truth data for both the training and testing sets. Given that TDM Studio only provides access to Detroit Free Press pages from 1923 to 1999, our train and test sets are limited to that range. Nevertheless, we developed a model sufficiently robust and versatile to be applied to pages external to this time period.

Certain tasks were ultimately considered to be outside the scope of this project. These include correcting errors in sponsor-provided OCR data, identifying subtitles, and segmenting non-front/interior pages. Additionally, the pipeline did not focus on classifying or associating artifacts, such as photos, captions, and advertisements, with articles.

2.4. Sponsor Requirements

Complete List of User Requirements			
1. TextLine Classification			
Requirement	Priority	Pass/Fail Definition (for both train and test set)	Metric Definition
Title Classification	1	Average page-level recall score is at least 0.95	$(\frac{1}{\#\ pages\ in\ set}) * \sum_{1}^{set\ size} \frac{TP\ title\ lines\ classified\ on\ the\ page}{TP + FN\ title\ lines\ classified\ on\ the\ page} \geq 0.95$
Body Text Classification	1	Average article-level IoU in each time period (1923-1948) (1949-1968) (1969-1999) is at least 0.95	$IoU = \frac{TP\ body\ text\ lines\ classified\ in\ the\ article}{TP+FP+FN\ body\ text\ lines\ classified\ in\ the\ article}$ $(\frac{1}{\#\ articles}) * \sum_{1}^{\#\ articles} IoU\ (time\ period\ x) \geq 0.95$
Artifact Classification	2	Average page-level recall score is at least 0.8	$(\frac{1}{\#\ pages\ in\ set}) * \sum_{1}^{set\ size} \frac{TP\ artifact\ lines\ classified\ on\ the\ page}{TP + FN\ artifact\ lines\ classified\ on\ the\ page} \geq 0.80$
Threading (Line Ordering)	2	Average article-level Kendall's Tau Score is at least 0.8	$\frac{(number\ of\ concordant\ pairs) - (number\ of\ discordant\ pairs)}{(number\ of\ pairs)} \geq 0.8$
Byline Classification	3	Average page-level recall score is at least 0.5	$(\frac{1}{\#\ pages\ in\ set}) * \sum_{1}^{set\ size} \frac{TP\ bylines\ classified\ on\ the\ page}{TP + FN\ bylines\ classified\ on\ the\ page} \geq 0.50$

Complete List of User Requirements		
2. System Computing Benchmark		
Requirement	Priority	Pass/Fail Definition (for both train and test set)
LLM API Cost	1	Maximum 22 cents per page, optimal cost of 6 cents per page
Computing Speed	1	Minimum 10 pages per minute

Figures 5, 6. Sponsor-defined requirement thresholds

Figure 5 describes in detail the requirements thresholds for article segmentation. We selected recall as the metric for title classification, artifact classification, and byline classification because

it allows us to see how many in each category we were able to correctly identify. For body text classification, Intersection over Union (IoU) is preferred as it measures spatial and content overlap. Dividing IoU results across three different time periods, 1923-1948, 1949-1968, and 1969-1999, we evaluate model robusticity across different layouts and styles over time.

We will also evaluate a 2-way IoU, with both the model output and ground truth as reference. Kendall's Tau Score (KTS) is used for evaluating threaded line order similarity because it quantifies the similarity between predicted and true line sequences and focuses on the preservation of relative ordering.

Figure 6 outlines two critical evaluation metrics for the system's computing benchmark, focusing on cost and performance. Since the cost of hosting the pipeline on TDM Studio is negligible, the primary expense comes from paid subscriptions, particularly those needed for prompting OpenAI's large language models with text and images. The LLM API Cost metric emphasizes cost efficiency, requiring a maximum of 22 cents per page while aiming for an optimal target of 6 cents per page to ensure affordability. Meanwhile, the Computing Speed metric sets a minimum requirement of processing 10 pages per minute to guarantee sufficient system throughput and operational efficiency.

Overall, our top priorities center on achieving accurate title and body text classification while maintaining cost efficiency and high processing speed. By focusing on these key objectives, we aim to develop a scalable system that delivers reliable performance, meets budgetary constraints, and processes data efficiently.

3. Standards for Ground Truth Development

3.1. Truthing Overview

This project is non-trivial due to several reasons, one of them being the generation of ground truth data. Producing a usable ground truth set from our inputs requires manually labeling each TextLine in the input XML, in order to match our desired input. For example, we wanted to be able to assign a certain TextLine to the article it belongs to (its Article ID), label a title line as “title” or a byline as “byline,” and give it a “threading” ID to build the correct reading order.

The process of manually reviewing and correcting data line by line to make it suitable for evaluation is labor-intensive and time-consuming, with some files containing over 1,000 TextLines, which might take hours. This makes it difficult to collect a sufficient volume of “truthed” data for comprehensive analysis, which is an essential part of our evaluation process. These limitations highlight the fundamental challenge of preparing data that is both accurate and representative enough for meaningful testing.

During the development phase, we wanted to ensure that intermediate test results would actually provide valuable insight that could help us to improve the model iteratively. Once the model was developed, we also wanted to have reliable measures of accuracy for the output of our system based on a consistent and comprehensive ground truth set. The generation of ground truth data was completed for 100 front pages for final pipeline testing. This includes an 80/20 train/test split and a 70/30 split between no threading and threaded pages. All ground truth files are available in TDM Studio.

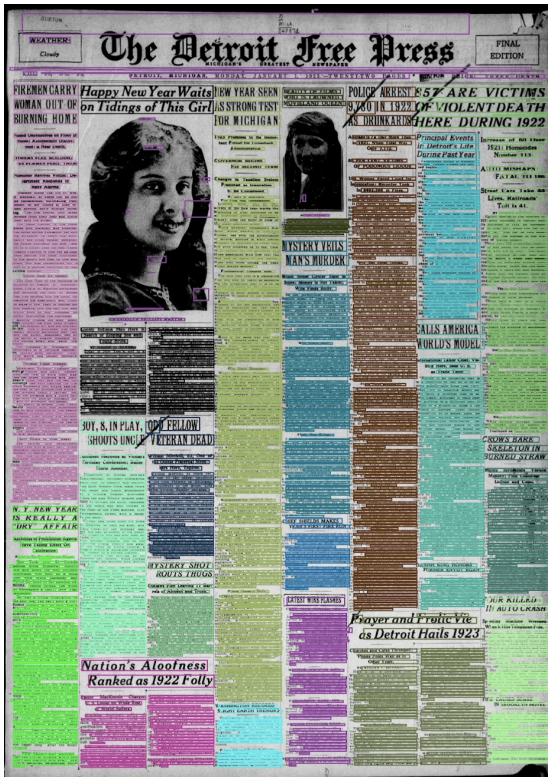


Figure 7. Example visual output of "truthed" front page from January 1, 1923

3.2. Step-by-Step Instructions for Using the Truthing Tool

In order to address these challenges, we developed a truthing tool that streamlines the truthing process. The tool saves valuable time, ensures consistent output among students, and provides an efficient workflow for handling large files.

General workflow*: To refresh the tool and view current truthing progress on the image, input 'u' in the command line and press 'Enter' whenever. When ready, input 'c' into the command line to move on to the next step.*

1. Run a Gaussian Mixture Model (GMM).
    a. Categorizes title and body TextLines based on their height.
    b. Titles tend to have larger and taller TextLines compared to body text.
    c. Note: the GMM is not perfect and can misclassify TextLines.
2. Identify Artifacts based on Proximity to Title.
    a. If no valid title is found for a TextLine within the horizontal or vertical threshold, assign an Article ID of '-1'.
3. Fix Title Lines.
    a. If a line is an artifact, set its Article ID to '-1'.
    b. If a line is body text but not part of a title, set its Article ID to '--' (these lines will no longer display in the current step after refreshing).
    c. Group title lines in the same title by assigning the same Article ID.
4. Fix Body TextLines.
    a. Update the ArticleIDs for any misassigned body text lines.
    b. Remember to update the lines previously set to '--'.
5. Fix Threading Order.
    a. By default, TextLines are grouped based on a similar normalized horizontal position (N-HPOS). Within each group, TextLines are sorted by their vertical position (VPOS) to establish a top-down reading order.
    b. Adjust the line order number for any misassigned line.
    c. The first title TextLine should be the first TextLine in an article.
6. Fix LineType.
    a. By default, lines are initialized as title, artifact, or text based on earlier steps.
    b. Fix any misassigned TextLines.
    c. Manually label bylines.
7. Output a "truthed" version of the sponsor-provided XML.

3.3. Dealing With OCR Issues

The project scope does not involve correcting errors in the provided OCR scan data. Other than misspellings, this includes TextLines spanning multiple articles, or TextLines lacking meaningful text content.

Figure 8 below illustrates each problem and provides a resolution for truthing purposes.

OCR issue	Visual example	Resolution
TextLine spans more than one article (outlined in red)		Assign the TextLine to the article with the largest number of Strings in that TextLine. If multiple articles have equal number of words, assign the smallest ID.
TextLines in artifacts (example is of masthead)		Classify as artifact TextLine (ID -1). The example has done this correctly, as indicated by the purple.
Junk TextLines with fragments of words/letters		Classify as artifact TextLine (Article ID of -1). The example has done this correctly.

Figure 8. Truthing requirements

4. Final Project Pipeline

4.1. Final System Diagram

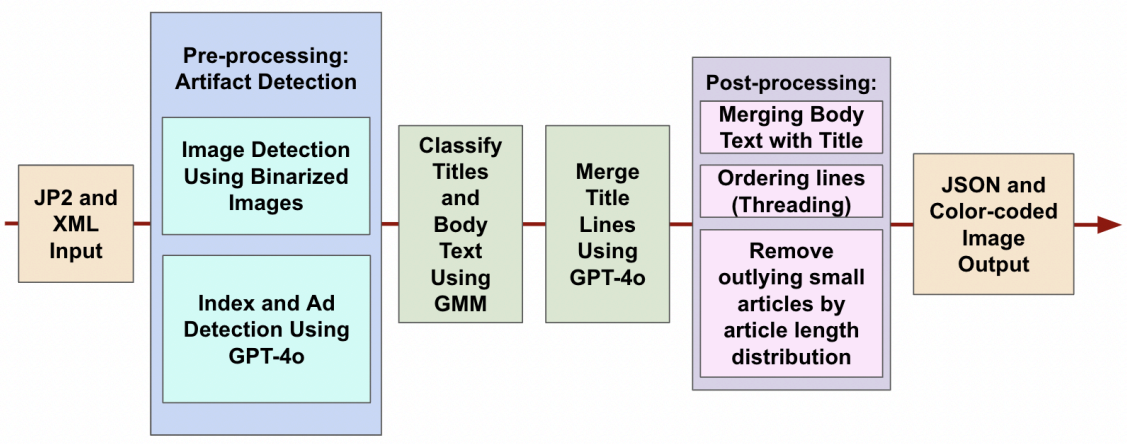
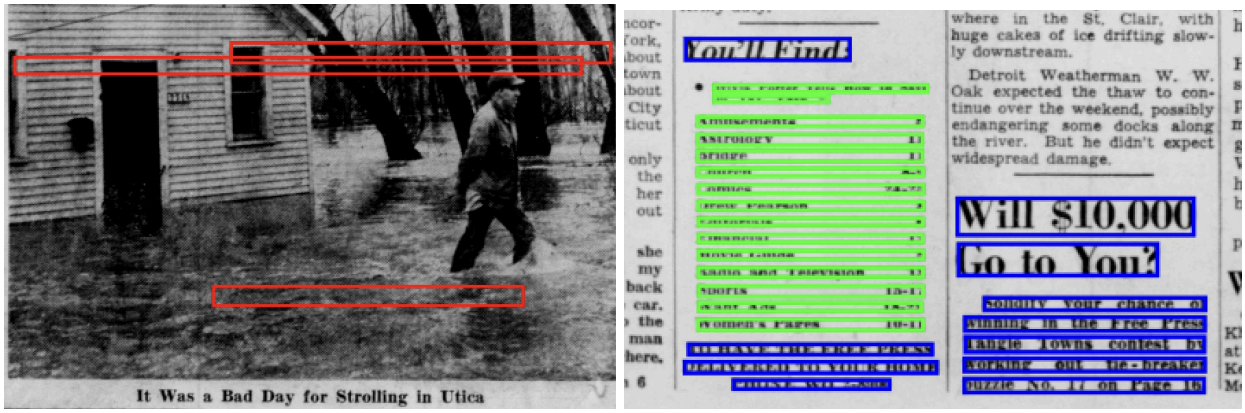


Figure 9. Final model pipeline

1. We first “binarized” the image into black and white, using computer vision to identify regions with high black pixel density. These areas are typically photos and we filtered out any TextLines within them.
2. Indices and advertisements were identified with separate GPT-4o prompts.
3. Article previews found in 1923 front pages were detected using string similarity matching. All text belonging to the “Latest Wire Flashes” article were classified as “ad” artifacts.

Figure 10. “Latest Wire Flashes” TextLines correctly classified as advertisement artifacts

4. The intermediate output image is generated, with classified artifact TextLines as either Red (detected in binarization step), Green (index prompting), or Blue (ad prompting). This step helped us decipher which artifact detection step was being overly aggressive or ineffective.



Figures 11, 12. RGB-color-coded intermediate output of identified artifacts

5. A Gaussian Mixture Model (GMM) was then applied to distinguish between title and body text based on TextLine height. Titles generally have larger font sizes, while body text tends to be smaller in size/height.
6. Prompting GPT-4o with the GMM-labeled title lines, we merged individual title lines belonging to the same article. Prior to this step, the first and second lines in a multi-line title are both correctly classified, but assigned different Article IDs.
7. After title lines were merged, we relied on TextLine horizontal and vertical proximity to associate body text lines with respective titles, fine-tuning distance thresholds based on training data. Our model assumes that a title is positioned at the beginning of an article and thus any body text line should be vertically below its corresponding title.
8. Threading was done using the same logic as described in *Step-by-Step Instructions for Using the Truthing Tool*.
9. We removed outlying small articles by length distribution. Majority of the time, these were artifacts. This step also conveniently removed TextLines in the masthead, as the GMM would classify them into 1-3 line "articles."
10. The final pipeline output, as described in *Current Project Status Overview*, consists of a segmented JSON of TextLines, a JSON with concatenated content for each article on the page, and the image of the page with color-coded articles.

Specific parameters and code logic for the project can be found in the TDM Studio code directory outlined below. The directory structure includes separate folders for inputs, outputs, the final pipeline, and all tested but unused methods. This organization allows for easy navigation and provides clear access to the code responsible for key aspects of the project.

4.2. TDM Studio Directory Tree

```
.
└── /home/ec2-user/SageMaker/
    ├── all_inputs/                          # XML, JP2, and truthed XML for all
    │   ├── 1923/                            # truthed data
    │   │   ├── 1923010101
    │   │   ├── 1923010301
    │   │   └── ...
    │   ├── 1938
    │   └── ...
    ├── all_outputs/
    │   ├── final_ppl_full_set               # training set results on 80 front pages
    │   └── final_ppl_full_test              # test set results on 20 front pages
    ├── FinalPipeline/
    │   ├── artifact_eval_runner             # artifact evaluation
    │   ├── average_cost                     # average cost per page for an output run
    │   ├── byline_eval_runner               # byline evaluation
    │   ├── concatenate_json                 # JSON output of concatenated content
    │   ├── count_truthed_files              # calculate number of truthed files in TDM
    │   ├── e2e_graphs_and_threading         # body text IoU and threading evaluation
    │   ├── final_pipeline                   # main pipeline
    │   ├── final_pipeline_backup            # backup - main pipeline
    │   └── title_eval_runner                # title evaluation
    ├── Getting Started (TDM tutorials)
    ├── Getting Started R (TDM tutorials)
    └── misc/                                # misc: previously experimented methods
        ├── ChatGPT                          # prompt GPT-4o with JP2 + XML
        ├── chatGPT_with_image               # prompt GPT-4o with JP2 only
        ├── data                             # DFP pages pulled directly from database
        ├── DFPBatches                       # specific DFP years we looked at
        ├── DiT                              # experimented pre-trained vision model
        ├── front_pages                      # Version 1 results on DFP front pages
        ├── LLaVA-main                       # experimented pre-trained vision model
        ├── mdp-main                         # Version 0
        ├── mdp-main-v1                      # Version 1, includes truthing tool
        ├── truth_decades                    # final front pages to be truthed
        ├── export_batch                     # export files to be truthed out of TDM
        ├── jp2Viewer                        # view JP2 images
        └── move_front_pages                 # extract front pages from DFPBatches
```

4.3. Use of Large Language Models

While it may seem like a simple solution to feed the entire page into ChatGPT and ask it to segment into individual articles, there are significant challenges. First, the XML file size exceeds the model's input token limit, preventing it from processing the entire content at once. Relying solely on GPT-4o for segmentation is also more costly than manual segmentation. Therefore, we employed a combination of LLM prompting and more traditional machine learning techniques to optimize cost, speed, and accuracy.

We used ChatGPT-4o during two stages in the pipeline. The first stage is the artifact detection in pre-processing. The pipeline sends both the XML data extracted from OCR and the corresponding newspaper image (if needed) to ChatGPT-4o for contextual analysis and classification. The second usage is merging title lines. Newspaper titles often span multiple text lines due to layout constraints. To address this, ChatGPT-4o is tasked with merging adjacent title lines that belong to the same article title, given the image and the python dictionary of the title lines. This merging process ensures that fragmented titles are correctly reconstructed, enabling more accurate merging for the consequent article text contents.

Through extensive experimentation, we identified two key strategies that improved ChatGPT 4.0's performance and consistency. Placing detailed task instructions and examples within the system prompt, rather than the regular prompt, yielded significantly better performance. This setup allowed for clearer task definition and reduced misinterpretation. Then, to ensure a stable and consistent data output, we specified JSON as the required output format for all tasks. This standardized structure improved the stability of the pipeline's downstream processing stages by simplifying data parsing and reducing the need for additional formatting adjustments.

```
    prompt = f"""
You are provided with a Python dictionary and an image of a newspaper. The dictionary contains text lines extracted from
the image, with each line associated with an article ID, the content of the text line, its horizontal position (H-pose),
and its vertical position (V-pose) within the image. The dictionary represents text lines that are potentially part of
articles or titles of articles.

Your task is to analyze the text lines and group them based on their content and positional information (H-pose and V-pose).
Specifically:

Group Identification: Determine if multiple lines belong to the same article based on their content and positional data.
If they do, they should be assigned the same articleID.

Filtering Non-Article Lines: If a line does not belong to any article or if it is not a title of an article, assign it an
articleID of -1. This applies to lines that may be part of artifacts such as headers, weather reports, indexes,
or advertisements.

Take into account the relative position of the lines within the image and any contextual clues in the content that might
indicate whether a line is part of an article, a title, or unrelated text.

Example format:
    {example_output}
    Here is the python dictionary with the lines. Find these lines' ArticleID's:
    {title_lines}
    . \n"""
```

Figure 13. Prompt to GPT-4o to merge title lines in the same article

5. Data Evaluation and Results

5.1. Parameters

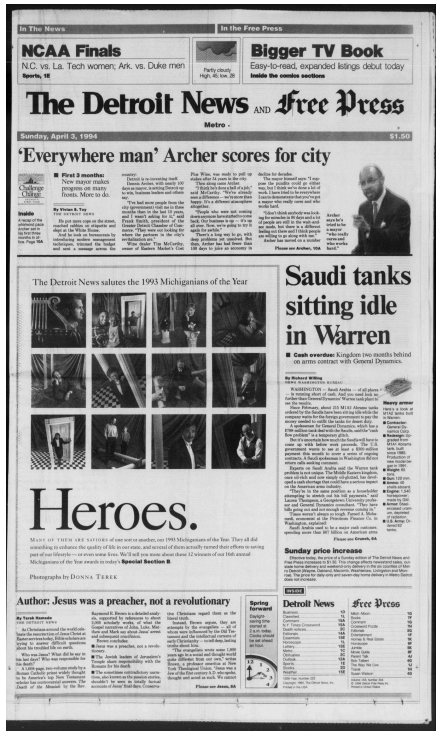
Our final pipeline performance was assessed based on two full pipeline runs on front pages of the Detroit Free Press, spanning from 1923 to 1994. Specifically, the 100 pages were divided 80/20 into the train and test set. The main goal when selecting which front pages to truth was to yield a comprehensive ground truth set that represents multiple decades, as the page formatting highly varies across years. For instance, earlier front pages tend to have smaller fonts, less images, and, consequently, a lot more articles when compared to more recent front pages, which in turn have bigger fonts and prioritize images and other visual information as opposed to cramming many different articles into one page.

Because our pipeline relies on some heuristic-based methods based on some reasonable assumptions about elements like page layout and reading order, it is important that we account for the differences across the years. While our evaluation data encompasses only the specified time period, the diversity of formatting leads us to expect our model to be able to handle these different formats well. Thus, it can be generalizable and widely used across not only different time periods but also different newspapers besides the Detroit Free Press.

Below are some examples of front pages from all of the years we truthed for this project.

Figures 14, 15, 16. Example DFP front pages from 1923, 1938, and 1949

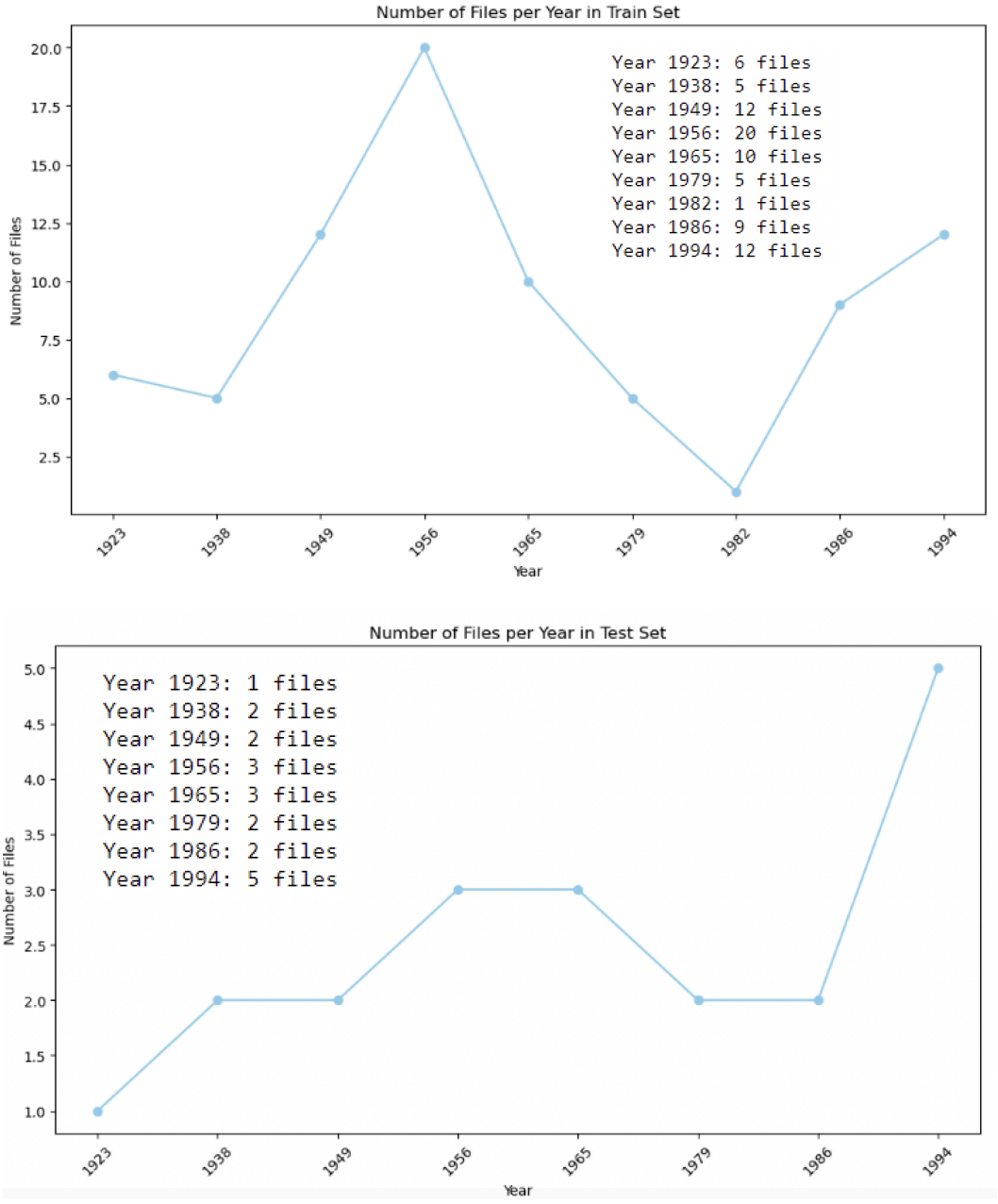
Figures 17, 18, 19. Example DFP front pages from 1956, 1965, and 1979



Figures 20, 21. Example DFP front pages from 1986 and 1994

We then use this comprehensive evaluation set to avoid fine-tuning our model to specific decades, ensuring that the methods can be generalized across time.

Represented in **Figure 22** and **Figure 23** below is the distribution of truthed pages per publication year in the train and test set.



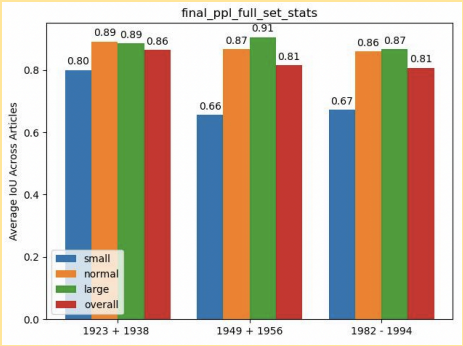
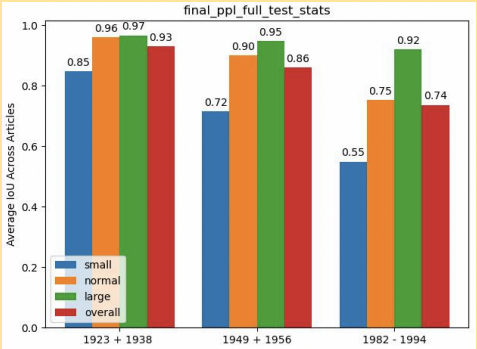
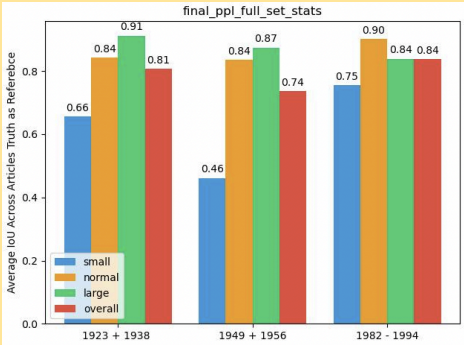
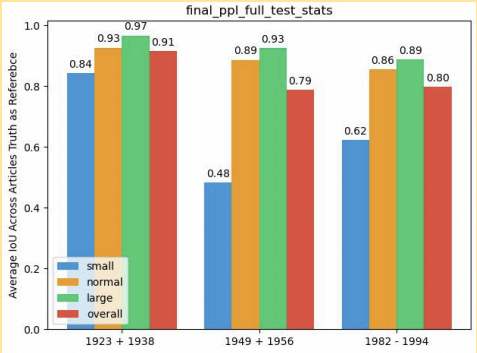
Figures 22, 23. Distribution of evaluation pages based on publication year

5.2. Training and Test Set Results

The table below contains the results for each of our requirements from the full run of the evaluation set with the final segmentation pipeline, as well as whether the requirement met the specific threshold or not.

By keeping these sets separate, we can ensure that the evaluation accurately reflects the model's ability to perform on new inputs rather than simply "memorizing" the training data. This separation helps prevent overfitting and provides a reliable measure of the pipeline's effectiveness in meeting the specified requirements.

Colors: **Yellow** - sponsor-defined threshold not met, **Green** - sponsor-defined threshold met

Requirement	**Training Set Results (80)**	**Test Set Results (20)**
Title Classification Recall: 0.95	Recall: 0.8799	Recall: 0.9071
Body Text Classification (Model Output as Reference) IoU: 0.95		
Body Text Classification (Ground Truth as Reference) IoU: 0.95		
Artifact Classification Recall: 0.80	Recall: 0.8020	Recall: 0.7798
Threading	340 articles/27 pages threaded	69 articles/6 pages threaded

KTS: 0.80	KTS: 0.9241	KTS: 0.9066
Byline Classification Recall: 0.50	61 pages with bylines Recall: 0.659359	14 pages with bylines Recall: 0.615847
Cost < $0.22/page	$0.16 per page	$0.14 per page
Speed > 10 pages/minute	Multithreading allows 12 front pages to be processed in 1 minute	

5.3. Limitations of Model Pipeline

The noticeable decrease in performance for small articles in the end-to-end body text IoU graphs could be attributed to several factors. One reason is that small articles are often less prominent on the front page and may feature smaller titles. Since the Gaussian Mixture Model (GMM) used to classify titles relies on the height of the TextLines, these smaller titles often do not meet the criteria for classification. Additionally, small articles have less body text or fewer well-defined features, making it harder for the model to differentiate them from non-text artifacts or background noise.

The significant reduction in performance for more recent years, particularly in the 1994 pages, can be attributed to the evolving layout style of these pages. The earliest version of our model pipeline was trained on 1923 pages, which had a rigid and columnar structure that made it easier to classify and segment articles. However, pages from 1994 feature much more dynamic and variable layouts that change from day to day. These pages often contain more complex arrangements of text and images, with articles appearing in non-traditional formats and varying in size and positioning. As a result, the pipeline, which was optimized for the more predictable, structured layouts of earlier pages, struggles to adapt to the more diverse and flexible designs found in later years. This shift necessitates a more robust and adaptable model capable of handling such variability, which our current pipeline was not fully equipped to address.

6. Reflection and Looking Ahead

One of the key takeaways is the importance of modularity in the pipeline. Setting up a modular structure allowed the team to experiment with different tasks and functionalities while enabling flexibility for adjustments as needed. This adaptability also extended to team collaboration, where members were reassigned to different tasks to meet shifting project demands. However, we encountered challenges in time management, particularly with the preparation of ground truth

data. The effort required was greater than anticipated, and we could have benefited from developing the truthing tool earlier in the project timeline. Moreover, the team's full focus on front pages meant that less attention was given to interior pages, which hindered our ability to explore jump line detection and article continuation across pages like we intended at first.

Moving forward, more exploration and development in these areas, alongside improvements in OCR scans and LLM integration, will enhance the pipeline's performance and applicability. Appendix B describes some ML, NLP, and computer vision techniques that we explored throughout the year, but did not find promising given our scope. This does not mean that they are not applicable at all to this project. If we had more time to dive deeper into the models and potentially fine-tune them, we believe that they could have been very valuable to the development.

Furthermore, we believe that as AI models become better, faster, and more accurate, the possibilities for this project also will grow. Gaining access to the OpenAI API made a huge difference in the way we approached most of the problems we had tried to tackle in the first semester, and most attempts yielded much better results. Our pipeline currently relies on ChatGPT-4o, which is highly valuable for tasks where we ask our API to retrieve information based on the image scan and XML data, or a combination of methods that use both sources of information.

However, there are some clear limitations when it comes to more advanced tasks that the 4o model might not be as suited for as of now, such as “correcting” a poor OCR scan or working solely with the image to retrieve titles for example. We found that the model often generates erroneous text that does not actually belong to the image. When thinking about accuracy and the goals of our project, accidentally adding made-up text to an article is a risk that we are not willing to take with the current iteration of ChatGPT. In the future, with better AI models, this type of task might potentially become not only less risky, but very promising.

Appendix A: Past Pipeline Iterations

Version 1

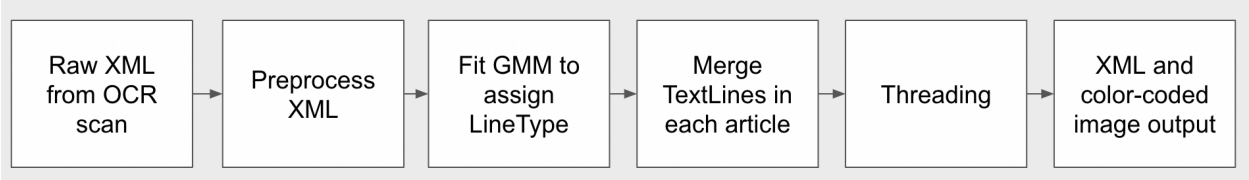


Figure 24. Version 1 Pipeline

One of the initial discoveries was that the positional information, such as the TextLine height and its horizontal position (HPOS) and vertical position (VPOS) attributes, was the most reliable input for the model, as it was highly accurate even when the actual text was often misspelled or incomplete due to OCR errors. By leveraging these spatial features, a GMM could effectively distinguish between larger titles and smaller body content. Overwhelmingly, the title is the first line of an article, so body content lines could simply be merged to the nearest title above them.

Even though our pipeline evolved as the project progressed, the basic logic below of relying on accurate positional information and using spatial features to classify and merge text lines remained consistent throughout. This foundational approach proved fairly robust across different time periods and newspaper layouts.

1. Add LineType and ArticleID attributes to each TextLine in the raw XML. Default value for LineType is ‘text’, as most TextLines on a page are body content.
2. Use a Gaussian Mixture Model (GMM) to classify TextLine elements into either title or text. Update the LineType attribute of classified titles to “title.”
3. Group title lines by spatial proximity using the HPOS and VPOS attributes.
4. Assign a unique ArticleID to each merged title group.
5. Merge body content lines based on spatial proximity to titles and label them with the same ArticleID.
6. Organize TextLine elements within each article by: 1) Grouping based on similar normalized HPOS and 2) Sorting by VPOS to establish a top-down reading order.
7. Generate output visualizations with each ArticleID mapped to a unique RGB color. The Version 1 pipeline did not classify artifacts, so all ArticleID values should be non-negative.

Version 2

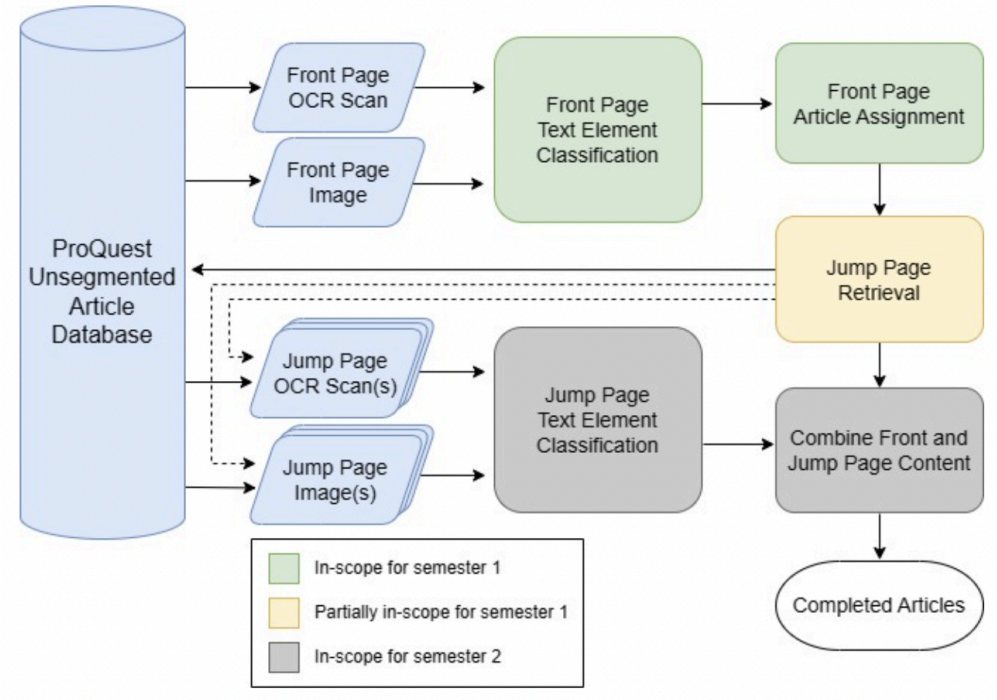


Figure 25. Version 2 Pipeline

Figure 26. Jump line example from 1979

While developing the second pipeline iteration, segmenting interior pages was still in scope. We wanted to locate front page articles that continued on subsequent pages. The logic for this task was based on identifying jump lines, which typically appear as the last line of a front page article as in **Figure 26**. By detecting these jump lines, we could locate the file ID of the corresponding interior page and link the continuation of the article. However, accurately identifying jump lines proved challenging, as they are often the same size or even smaller than body content lines. Combined with existing OCR errors, similarity in size and appearance made it difficult for the model to distinguish jump lines from regular body content consistently, adding complexity to the segmentation process. Ultimately, we decided not to tackle interior page segmentation, as our work on front pages was still imperfect. Our focus shifted toward developing a pipeline that works reliably for a few targeted areas, such as front-page title and body content, rather than attempting to handle multiple tasks with only average performance. This approach allowed us to prioritize quality and robustness over scope.

Version 3

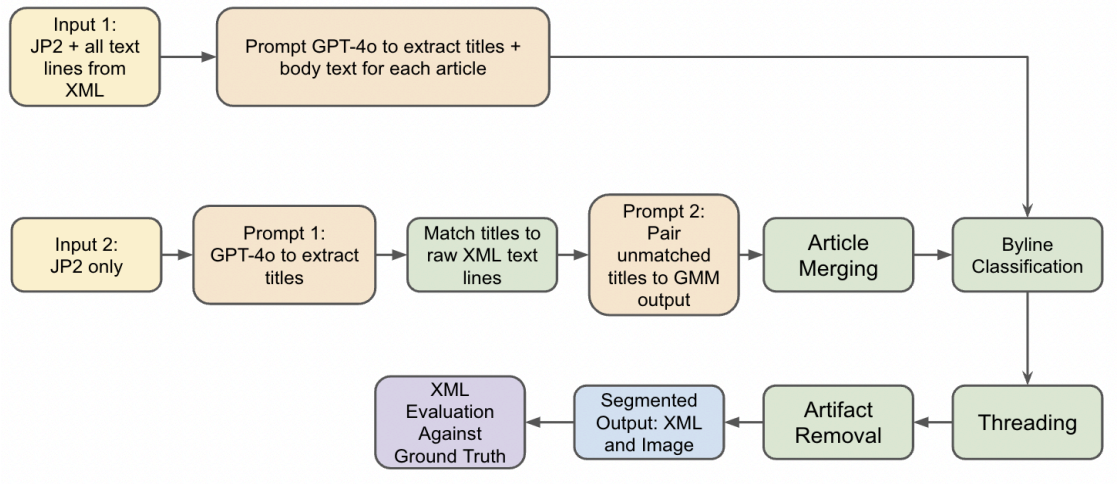


Figure 27. Version 3 Pipeline

Version 3 of the pipeline introduced significant changes. Access to OpenAI API endpoints via ProQuest's proxy allowed us to integrate GPT-4o into the pipeline. GPT-4o had several advantages: it often outperformed traditional machine learning techniques in classification accuracy and provided flexibility in designing creative prompts. However, it also came with drawbacks, such as its "black box" nature, making it difficult to understand the reasoning behind incorrect classifications. Additionally, it introduced non-zero costs into the pipeline, necessitating optimization, and was slow to process without multithreading.

We experimented with an LLM-only approach and a hybrid strategy combining GPT-4o with a GMM. In the LLM-only approach, GPT-4o processed both the JP2 image and its corresponding XML file. Supplementing the XML data allowed the LLM to retrieve ID, content, and positional information to visualize outputs. However, this significantly increased the token costs. To address these challenges, the JP2-only route implemented a more cost-efficient two-step process. The first GPT-4o call generated a list of titles from the JP2 image, which were matched to raw XML data using exact string matching. Unmatched titles were processed in a second GPT-4o call and paired with any similar GMM output. The hybrid approach eliminated prompting with the large XML file. As a result, costs were reduced by one half compared to the LLM-only method.

The major downside of both input combinations was the inherent attention limit of LLMs, which caused them to overlook information further down the newspaper page when processing large XML files or base-64 encoded images. Upon evaluation, Version 3 did not show a significant performance improvement over the GMM-only output. However, artifact detection still required LLMs for their ability to identify complex patterns and contextual cues, such as distinguishing

between subtle variations in syntax or font styles indicative of non-article elements. By classifying most article TextLines with non-LLM methods, we could strategically allocate cost-intensive LLM steps to artifact detection, which brought us to the final project pipeline.

Appendix B: Other Techniques Explored

Document Image Transformer (DiT)

DiT (Document Image Transformer) is a self-supervised transformer model designed for Document AI tasks. It processes document images using a vision transformer (ViT) architecture, capturing both visual and structural information from text images. DiT was trained on large-scale unlabeled datasets and learns generalized document representations without human annotations, making it effective for tasks like document classification and layout analysis. We tried to use DiT to segment pages of the newspaper into its elements (i.e. paragraphs of text, titles, headline, index, ads, and images). Our reasoning behind this usage is that if we could break a newspaper image down into its elements, we could piece these elements together to form our articles.

The specific model we chose for testing was trained by Microsoft's foundational model research team and fine-tuned on PubLayNet, a large dataset for document layout analysis. We tested both provided "large" models: one uses Mask R-CNN as the detection algorithm and the other Cascade R-CNN[1]. These models segment document elements using masking and classify them into one of: Title, Figure, Text, or Table as shown in **Figure 28**.

Figure 28. DiT example segmentation on PubLayNet

[1] Models available at https://github.com/microsoft/unilm/tree/master/dit

However, we found this model to be ineffective at segmenting our newspapers. We show a representative example in **Figure 29 (left)** below. When attempting to have DiT segment the entire page at once, all tested DiT models segmented the newspaper by claiming that the entire newspaper was a single "figure." We believe that this behavior is caused by the difference in text size between PubLayNet, which the DiT models were fine-tuned on, and our input newspaper pages. While PubLayNet consisted of mostly academic articles and textbook pages which contained few columns per page with not many paragraphs in each column and large figures, our newspapers could contain many long columns on a page with both large and small images.

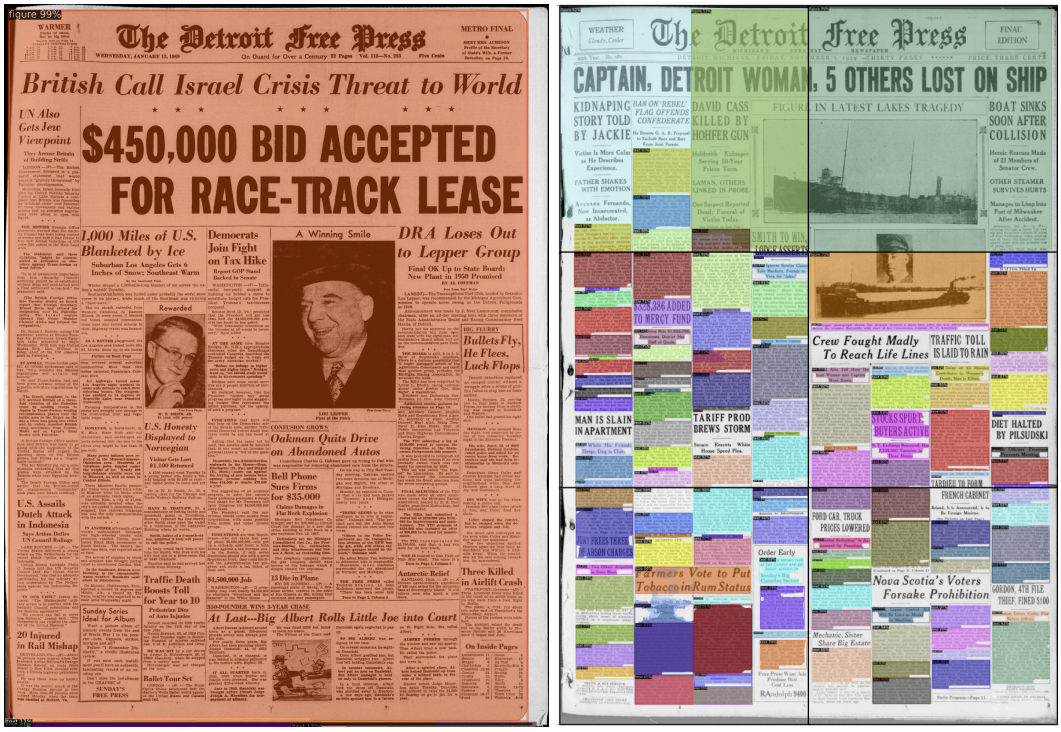


Figure 29. DiT segmentation output on entire newspaper (left) and 6 newspaper sections (right)

In an attempt to align the text size of the newspapers with the training data from PubLayNet, we decided to split the newspaper into sections and prompt DiT with the sections instead of the full newspaper. While this was successful at generating better results (shown in **Figure 29 (right)**), these results were not consistent and not strong enough to build the pipeline around.

One other strategy we tested was to eliminate noise that may affect DiT from being able to segment the newspaper by instead reconstructing the newspaper from the OCR scan and using the reconstructed newspaper as input. To reconstruct the newspaper from the OCR scan, we printed all text picked up by the OCR scan of the newspaper image onto a new document. The

text was printed in the font size derived from the size of the textline that the text was contained in and positioned according to the x and y position attributes assigned to each text element in the XML data. We show a reconstructed newspaper image in **Figure 30**.

Figure 30. A newspaper image (left) with its reconstructed image representation (right)

While our reconstruction process appeared to eliminate the noise in our newspaper image, we did not see significant improvement in the output from DiT. The output of DiT on our reconstructed newspaper image is shown in **Figure 31** below.

Figure 31. A reconstructed newspaper image input section (left) with the output of DiT (right)

A key problem with this method is deciding how to split the newspaper. The simplest choice that comes to mind is splitting the newspaper down the middle, vertically or horizontally. This method encounters the problem of potentially splitting a paragraph into two across multiple sections. Further, attempting to split down the gap between columns is also difficult since there is not always a clear line that separates the newspaper into two. Without consistent segmentation results or labelled fine-tuning data to improve the consistency of the model in segmenting small text, we decided to not use DiT in our final model pipeline, instead opting for other visual machine learning model usage.

Topic Modeling

After identifying and labeling regions of a document, we can use topic modeling to group them by their content themes to form complete articles. In this section, we examine topic modeling with contextual embeddings which improves performance on large datasets as opposed to techniques without embedding input.

In Bidirectional Encoder Representations from Transformers (BERTopic)[2], embeddings are generated through a process that leverages the BERT neural language model. Text data is initially tokenized into fundamental units, including words, subwords, and special tokens like `[CLS]` for text context representation and `[SEP]` for sentence delineation. This tokenization enables BERTopic to handle a diverse vocabulary efficiently. BERT then processes these tokens

[2] https://maartengr.github.io/BERTopic/index.html

bidirectionally, considering the context provided by adjacent tokens to produce high-dimensional vector embeddings that capture the semantic meanings and nuances of each token.

UMAP (Uniform Manifold Approximation and Projection) is utilized to reduce the dimensionality of embeddings by projecting the high-dimensional vector space into a lower-dimensional space. These embeddings, representing tokens' usage within their textual context, are clustered using HDBSCAN to identify semantically similar groups to form topics. The model further refines these topics with a class-based TF-IDF (Term Frequency-Inverse Document Frequency) method, emphasizing the most representative words. **Figure 32** provides a visual representation of the BERTopic pipeline.

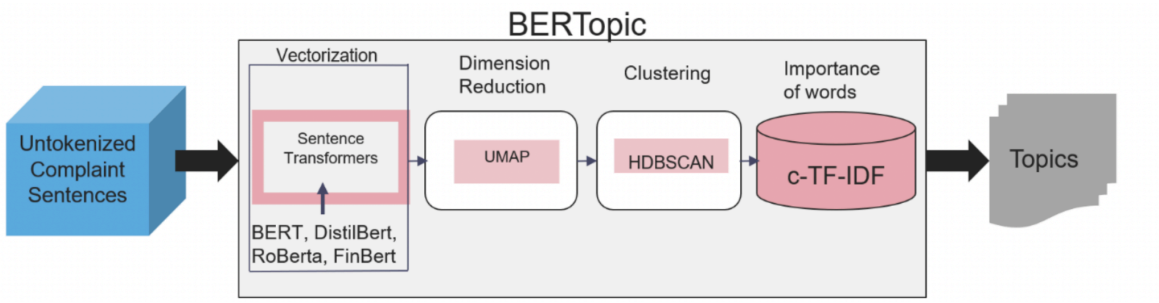


Figure 32. Research Methodology, BERTopic

BERTopic was applied to a dataset of tweets collected before and after the COVID-19 lockdown in April 2020 (Kellert and Zaman, 2022). The objective was to identify shifts in the context and associations of words related to COVID-19, reflecting the impact of the pandemic on language. The analysis showed that terms such as "quarantine," "lockdown," and "social distancing" became more frequent and were used in new contexts after the lockdown, indicating changes in their meanings. By examining how these words were used in various topics before and after the lockdown, it was clear that the pandemic changed their related themes.

BERTopic's ability to identify and cluster thematic content based on embeddings makes it a valuable tool for segmenting and merging labeled fragments of an article. Leveraging context helps us differentiate the meaning of polysemous words based on the current article topic.

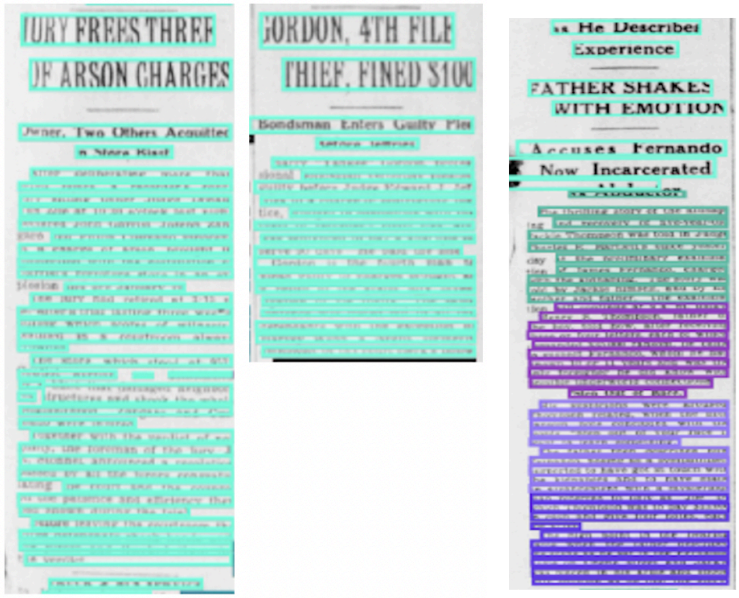


Figure 33. Incorrect topic classifications by BERTopic

Upon testing BERTopic, we discovered that a single article is often assigned multiple topics, and multiple articles can share the same topic, leading to overlapping classifications (see **Figure 33**). Additionally, the use of BERTopic for topic modeling requires a sufficiently large text corpus to generate meaningful topics. For older newspapers with a greater number of articles and text lines, BERTopic performs well, producing more accurate and diverse topics. However, with smaller datasets, such as those from newer newspapers or those with fewer articles, BERTopic significantly reduces the number of topics, often limiting them to just 2-3, which impacts the granularity and relevance of the topics extracted.

Tesseract (OCR)

Tesseract is an open-source OCR engine that extracts text from images and scanned documents[3]. We attempted to use Tesseract as a preprocessing tool to filter out artifact TextLines from the raw XML data. To do this, we located TextLines that contained less than two words and performed OCR using Tesseract on the region that contains the TextLine. If Tesseract finds text within the TextLine boundaries, we keep the TextLine; otherwise we remove it from the OCR data.

Using this method had mixed results. While we were able to reduce the number of artifact TextLines in the OCR data, we also removed many TextLines containing real content because neither OCR scan found English words within the TextLine boundaries. These false positive

[3] https://github.com/tesseract-ocr/tesseract

error TextLine removals often occurred at the edges of the newspaper where the image may be faded and thus difficult for either OCR method to read.

We decided not to incorporate this step in our pipeline because of the runtime of Tesseract in processing the TextLine images. Tesseract required approximately 2.5 seconds per TextLine image to perform OCR. Running this over many TextLines thus made this step computationally expensive. Due to this time cost and the lack of significant improvement by our model from removing these artifact TextLines in preprocessing, we decided not to use Tesseract in our model.

LLaVA

LLaVA (Large Language and Vision Assistant) is a multimodal model that integrates large language models with vision encoders, allowing for reasoning on mixtures of text and image prompts. This model is built upon a transformer architecture and combines a visual encoder (e.g., CLIP) with a language model (e.g., Llama), using learned adapters to align vision and text features for tasks like visual question answering and image-based reasoning.

There are a large variety of LLaVA[4] and LLaVA adjacent models that have open-source code and open-weights allowing for the vision model to be run locally. The ability to run the model locally (and potentially fine-tune it) was a key consideration in this model selection. By running LLaVA locally, model inference would likely be less expensive than if we sent the prompts to GPT-4o or another API-based closed source model.

We attempted to apply LLaVA to many different tasks within our model pipeline, including title identification, post-processing correcting title text, artifact identification, headline identification and correction, byline identification and correction, and extracting full article content. While we did not include any of these in the final model pipeline, these prompting strategies formed the basis of our prompting strategies for GPT-4o which we use extensively in our model pipeline.

One error that we saw frequently in using LLaVA models for content extraction is shown in **Figure 34** below. Instead of printing all of the titles as prompted, the model repeats the same title many times until the output reaches the output content limit.

[4] https://llava-vl.github.io/

```
Loading checkpoint shards: 100% 4/4 [01:03<00:00, 15.82s/it]
USER: Here is a list of article titles that are in this newspaper: [].  Your ta
The attention mask and the pad token id were not set. As a consequence, you may
Setting `pad_token_id` to `eos_token_id`:2 for open-end generation.
ASSISTANT: - Dancing Houri Cave-in Kills 4; Two Injured
- Death was caused by hypo-static compression
- Drugs Complicated Case
- Latest Photo of Wallace Reid and Family
- SINUOUS SHOULDERS AS JUDGE WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
- WAGGLES WORK IN 11 HOURS REQUIRED
- SINUOUS WIGGLER WAGGLES
-
USER:
```

Figure 34. Common LLaVA content extraction error

Beyond this error, the LLaVA series of models frequently hallucinated text, especially when the text within the input images was small. This may be a result of the text within input images being too small for the vision encoder to translate into readable input embeddings for the model which results in the model "guessing" the content that it is unable to "see" clearly. When switching to GPT-4o, while we did not encounter the repeated content error, we did witness hallucinated text in the model output, albeit to a lesser extent than in LLaVA outputs.

Despite success in some parts of our task, we chose not to use a LLaVA model in favor of GPT-4o. This decision was made for the following reasons:

First, GPT-4o had better base performance. We found that prompting LLaVA and GPT-4o with the same prompts usually resulted in better performance from the GPT-4o model (even when using prompts we optimized for LLaVA). Further, the errors that GPT-4o's output contained were far less harmful to the model pipeline than the errors that LLaVA's output contained. LLaVA hallucinated text and repeated content much less often than GPT-4o (see above paragraphs in this section). Since hallucinated text and repeated content were difficult errors to process within our model pipeline, GPT-4o was a better choice in this regard.

Second, we did not possess fine-tuning data to attempt to fine-tune LLaVA with. One of the reasons we initially selected LLaVA for testing was the prospect of fine-tuning the model for our tasks. While there are fine-tuning notebooks available, we found that we lacked sufficient data to tune the model for any of the tasks we considered given that all truthing data was done manually by the team. This eliminated one of the benefits that we foresaw in using LLaVA over a closed

source model like GPT-4o. Additionally, OpenAI released a fine-tuning API which allows for easier fine-tuning of GPT-4o, meaning that LLaVA does not possess an advantage over GPT-4o in fine-tuning ability anymore.

Third, we believed that technology improvements in VLMs were coming soon and that the increased implementation complexity of LLaVA compared to GPT-4o meant that replacing LLaVA inference in the pipeline with a newer model would be much more challenging than replacing an API call to GPT-4o. Choosing GPT-4o over LLaVA therefore meant that our code would be more maintainable and resilient to future improvements in vision models.