HyMark: Application of Hybrid AI for Markdown Syntax Generation

Khushnood Adil Rafique, Nabeel Khaliq, Christoph Grimm
University of Kaiserslautern-Landau (RPTU), Chair of Cyber-Physical Systems, Kaiserslautern, Germany
[khushnood.rafique|grimm]@cs.uni-kl.de, khaliq@rhrk.uni-kl.de

***Abstract*—Markdown, a markup language, faces accessibility barriers due to its moderate syntax complexity for users not so technically inclined. This paper presents a novel approach, merging rule-based and artificial intelligence (AI)-based strategies, leveraging natural language processing (NLP). By employing regular expressions, classifiers, and advanced language models, we automate the transformation of unstructured text into structured Markdown. This inclusive method enhances accessibility, making Markdown tools more user-friendly, and can rival the performance of large language models (LLMs) such as ChatGPT. Its novelty lies in the use of AI techniques combined with rule-based methods in a hybrid setting, to understand document semantics, and intelligently apply formatting, especially in optimal placements for headings and subheadings, extraction of code snippets, bullet points, and table generation. This technique minimizes the learning curve and manual effort in Markdown usage, aiding further adoption across various content creation domains, and thereby contributing to the documentation formatting practices. This manuscript will introduce and objectively compare the performance of our approach, *HyMark*, to that of ChatGPT.**

***Index Terms*—Markdown, Regular Expressions, ChatGPT, Natural Language Processing, Hybrid AI, Code Recognition, Natural language to Markdown, Information Extraction, Information Representation**

I. Introduction

In the contemporary landscape of digital technology, the generation of content has become a ubiquitous phenomenon. Daily, a large volume of articles, reports, blogs, and diverse content types is created to disseminate information globally. Markdown, distinguished by its plain-text-formatting grammar, has emerged as a preeminent choice among content creators and scientific experts. It serves as a lightweight markup language, enabling the creation of rich text through plain text editors, thereby enhancing the efficiency and effectiveness of content generation and presentation. Despite, its advantages, crafting formatted Markdown documents demands a functional understanding of the Markdown syntax. Untrained users often grapple with the challenge of investing time and effort in careful formatting, a task susceptible to errors. This involves paragraph organization, appropriate heading allocation, and the identification and formatting of links, code snippets, bullet points, and other elements crucial for achieving an optimal structure, and better readability.

Presently available solutions predominantly rely on manual processes, necessitating users to possess a good understanding of Markdown syntax or constant manual lookup. This poses a barrier for numerous potential users, especially those less versed in programming or scripting languages. Hence, the imperative lies in developing an intelligent system capable of automatically converting unstructured text into a well-formatted Markdown document, ensuring the preservation of logical flow and adherence to standard Markdown syntax. This approach would automate the arduous task of manual formatting, rendering Markdown more accessible to a broader audience. Our endeavors focus on the creation of a tool that transforms plain text into well-organized Markdown documents. While we now have access to tools like ChatGPT [1] that can perform similar tasks, we want to emphasize that HyMark performs the same task with better precision with consistently high accuracy. It achieves this by combining rule-based techniques and natural language processing (NLP) methods in a hybrid setting, and it often outperforms ChatGPT at Markdown-specific information extraction.

A. What is Markdown

Markdown, a lightweight markup language originating in 2004 and credited to John Gruber in collaboration with Aaron Swartz [2], was conceived to enhance the accessibility of writing and content sharing on the web, focusing particularly on plain text format. This fundamental markup language employs basic symbols to modulate the appearance and structure of the text, exemplified by utilizing a # sign before a heading (e.g., # Heading One) or employing numbers followed by a dot and a space for bullet points (e.g., 1. First item). Under its apparent simplicity lies a remarkable degree of adaptability. Markdown facilitates the seamless addition of links, images, list creation, and text organization. Ordered lists, employing numbers and periods (e.g., 1.), or unordered lists with dashes (-) or asterisks (*), are easily crafted. Furthermore, Markdown supports nested lists, block quotations, and code blocks, offering diverse expressive and structural choices to writers.

B. Markdown with SysMD

SysMD, outlined by Dalecke et al. (2022) [3], is a modeling tool rooted in SysML v2, aiming to democratize systems modeling for domain experts. This language uses near-natural-language statements, seamlessly integrating with Markdown files, unifying code, and documentation. SysMD documents enhance model accessibility with formatted text, tables, and external resource links. HyMark API integration

979-8-3503-5409-6/24/$31.00 ©2024 IEEE
DOI 10.1109/CAI59869.2024.00134

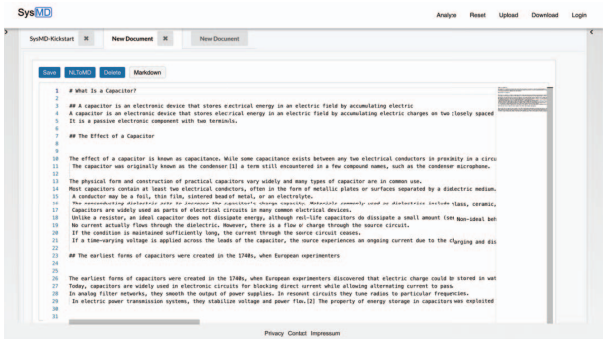

Figure 1. SysMD Notebook using HyMark API.

within SysMD via REST API enables Markdown text generation from unstructured text. Figure 1 shows the use of HyMark API within the SysMD system. Markdown makes SysMD accessible to users with limited modeling experience, streamlining well-documented and interconnected model construction by integrating code and documentation.

II. Related Work

This section reviews existing research in the domains of markup languages, information extraction, and syntax generation. The challenge of Markdown generation is pervasive across various tools such as GitHub, Bitbucket, Reddit, and SysMD Notebook. Users are often compelled to master Markdown syntax for their daily activities with these tools. Although some efforts like GitHub [5] and R Markdown [6] have attempted to address this with customizable editors, it remains a partial solution. Qian Xi and David Walker's work [7] introduces a context-free markup language tailored for semi-structured text. Their work explores design, features, and applications, making a significant contribution to information representation in semi-structured contexts. Topaz et al. [8] leverage regular expressions to extract fall-related data from clinical notes. NimbleMiner, a rule-based system, showcases the efficacy of regular expressions in discerning diverse lexical variations. In their work, Yin and Neubig [9] propose a syntactic neural model for general-purpose code generation. This model, trained on large datasets, captures syntactic structures, enhancing the quality of machine-generated code.

A. Role of HyMark

- **Innovative Markdown Syntax Generation:** Our work introduces an approach to automatically generate Markdown syntax from unstructured text, addressing a gap in existing literature, and compares its performance against state-of-the-art large language model (LLM), ChatGPT.
- **Hybrid Methodology:** Distinguishing itself, HyMark uses a hybrid methodology that seamlessly integrates rule-based and artificial intelligence (AI) techniques.
- **Semantic Understanding:** The core contribution lies in our system's ability to semantically understand unstructured text and present it in a structured fashion that can match the performance of ChatGPT.
- **Accuracy and Precision:** Leveraging rule-based techniques guarantees accuracy and precision in Markdown syntax, crucial for reliable information representation that on most occasions outperforms ChatGPT at identification tasks.
- **Filling a Crucial Gap:** The contribution of our work fills a critical gap in the scholarly landscape, specifically addressing the need for automatic markup language generation.
- **Enhancing Information Extraction:** Our approach enhances information extraction and representation in unstructured text, contributing to advancements in the field.
- **Versatility of HyMark Across Platforms:** HyMark demonstrates versatility across various scenarios and platforms. Its easy-to-use API seamlessly integrates with different systems, such as SysMD Notebook, facilitating diverse applications without significant implementation complexities.

III. Methodology

In this section, we delve into the developmental framework of HyMark. The process of transmuting raw unstructured text into refined markdown syntax is executed through a series of steps. The flow of HyMark's approach with a system like SysMD is outlined in figure 2, providing an overview of the process.

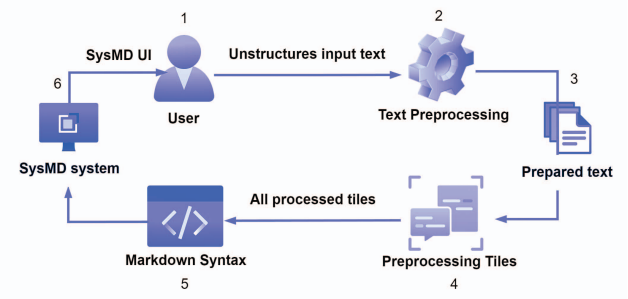

Figure 2. Markdown syntax generation pipeline.

A. Text Preprocessing

Before transforming raw text into organized Markdown documents, it undergoes text preprocessing. This section outlines our approach to achieving this task.

1) Structured Paragraph Transformation: Organizing large blocks of raw text requires a nuanced understanding of semantic structure beyond punctuation cues. We used sentence embeddings generated by employing the *all-mpnet-base-v2 model* [10] to convert sentences into meaningful vectors. Cosine similarity measurements between these vectors in a multi-dimensional space help group semantically related sentences, forming the basis for constructing coherent paragraphs. We then use the sentence embeddings to create a cosine similarity

matrix showing how similar each sentence is to the other ones. We create a new matrix by stacking each diagonal right to the main diagonal and then applying activation weights to each row to determine the closest sentences having the biggest activation weight. With this, we can create a similarity graph depicting maxima and minima between sentences with minima being used as splitting points for new paragraphs.

Text Block A

"Breadth First Search (BFS) is an algorithm used for traversing graphs or trees. Traversing means visiting each node of the graph. Breadth First Search is a recursive algorithm to search all the vertices of a graph or a tree. BFS in Python can be implemented by using data structures like a dictionary and lists. An integrated circuit (IC), also called a microelectronic circuit, microchip, or chip, is an assembly of electronic components, fabricated as a single unit."

2) Enhancing Paragraph Structure with Text Tiling: We further improve precision by defining a similarity threshold, a numerical line that sentences must cross to be categorized into a single topic category. Figure 3 shows the similar sentence similarity threshold for text block A, in which the first few sentences of the text are about breadth-first search (BFS) and later, from the fourth sentence the topic is shifted to the subject of circuits.

This procedure guarantees that the paragraph has a single context or a group of ideas that are tightly related to one another. After the initial division of raw text into paragraphs, we employ the NLTK [11] technique of text tiling to further enhance coherence. Text tiling identifies topic shifts and divides the text into tiles, each focusing on a different topic.

B. Tile Processing

This section provides an exploration of the *Tile Processing* component within HyMark. Following the segmentation of text into smaller tiles, the focus is then on the refinement of each tile. Parsing of textual components, including headings, subheadings, bullet points, hyperlinks, and code snippets, is conducted, and their formatting is adjusted.

Text Block B

"Breadth First Search (BFS) is an algorithm used for traversing graphs or trees. Traversing means visiting each node of the graph. Breadth First Search is a recursive algorithm to search all the vertices of a graph or a tree. BFS in Python can be implemented by using data structures like a dictionary and lists."

1) Automated Heading Generation: The automation of heading generation from context is particularly crucial in unstructured text to Markdown conversion, ensuring that every segment, from small tiles to entire pages, is accompanied by appropriate headings and subheadings. For the task of generating contextually relevant headings, we leveraged the T5 (Text-to-Text Transfer Transformer) model [12]. Developed by Google, T5 is a flexible machine-learning model proficient in tasks such as summarization and translation. The model integration was tested to generate meaningful headings within their context, for example, it generated the heading, "Breadth-First Search in Python" for text block B.

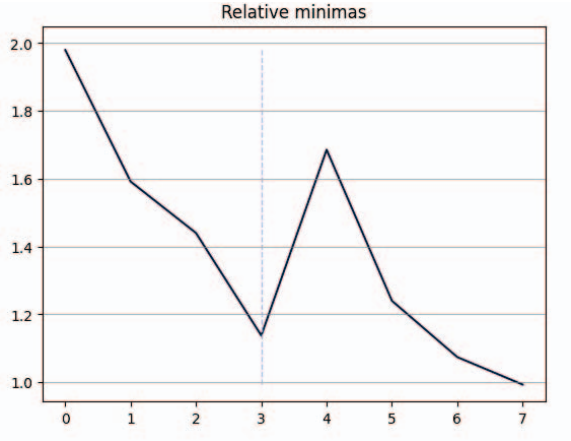

Figure 3. Sentence similarity threshold curve.

The ROUGE scores [14] provide a quantitative measure of the similarity between the heading generated by HyMark using T5 and a reference heading, generated by ChatGPT for text block B: "Understanding Breadth First Search (BFS) Algorithm in Graphs and Trees with Python Implementation". The rouge-1 and rouge-L metrics indicate a moderate overlap in unigrams and longest common subsequences, with recall, precision, and F1-scores (see figure 4). However, the lack of overlap in bigrams (rouge-2) suggests that consecutive word pairs in the two headings do not match well. Text block A posed a challenge for HyMark due to a thematic shift after sentence 4. ChatGPT generated the heading, "Exploring Breadth First Search (BFS) Algorithm and Introduction to Integrated Circuits" which we used as a reference heading to compute the ROUGE scores again for text block A. HyMark did not take into account the thematic change in the text and opted to follow the theme that is more prevalent (in terms of the sentence count) and generated the same heading as it did for text block B, "Breadth-First Search in Python". The ROUGE scores are represented in figure 4.

2) Rule-Based Identification of Textual Elements: HyMark uses a rule-based methodology utilizing Regular Expressions (Regex) [15] to achieve the extraction of textual elements like bullet points, hyperlinks, and image URLs. Regular expressions, or Regex Patterns, represent specific character sequences designed for string matching to create search patterns. Within HyMark, Regex plays a crucial role in systematically filtering through each text tile, searching for patterns that match different text elements. This precision is important for preserving the original context and meaning of the extracted information. An overview of its application is as follows:

- **Bullet Points:** HyMark identifies lists in the text by de-

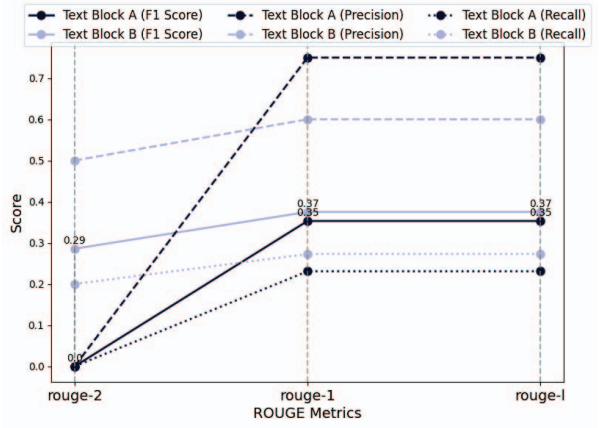

Figure 4. Comparison of ROUGE Scores for HyMark's performance across Key Metrics (Rouge-2, Rouge-1, and Rouge-L) – showcasing the recall, precision, and F1 scores, demonstrating HyMark's promising potential at heading generation.

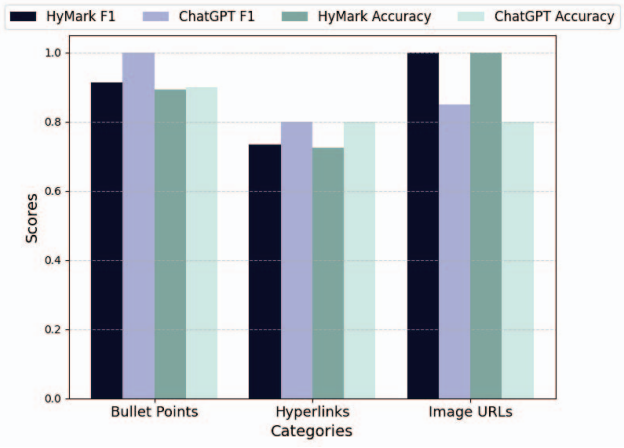

Figure 5. Performance Metrics (F1-Score and Accuracy) comparison between HyMark and ChatGPT at bullet point, hyperlink, and image URL extraction.

tecting patterns indicative of bullet points. It subsequently reformats them into markdown bullet points.

- **Hyperlinks:** Recognizable patterns reflecting common URL formats are used to extract hyperlinks. These are then converted into clickable Markdown links.
- **Image URLs:** Similar to hyperlinks, image URLs adhere to recognizable patterns with a path to an image resource. Our tool identifies these URLs and converts them to display the actual images in the markdown.

While AI models demand extensive datasets and substantial computing power, Regex provides simplicity, efficiency, and control. In scenarios requiring high accuracy and rule adherence, the rule-based approach with Regex ensures robust reliability. Additionally, the rule-based approach proves practical for obtaining quick results without extensive computational requirements.

We validated our point by comparing the performance of ChatGPT against that of HyMark. We used 30 unseen pieces of text when evaluating the performance metrics. The text segments contained about 100-150 words each and included bullet point hints (running numbered text elements, etc.), hyperlinks, and image URLs. Some text segments also did not contain one or more of these elements to conduct a varied test. We compared the information extraction performances across the 3 categories. Figure 5 shows how HyMark fared against ChatGPT. It performed fairly well but occasionally fell short on bullet point extraction since that is much more difficult for a rule-based implementation to achieve.

Assuming, TP_i, FP_i, and FN_i represent the true positives, false positives, and false negatives for the i-th piece of text, respectively. The precision (P_i), recall (R_i), and F1-score ($F1_i$) for each textual element were calculated as follows:

$$P_i = \frac{TP_i}{TP_i + FP_i}, \quad R_i = \frac{TP_i}{TP_i + FN_i}, \quad F1_i = \frac{2 \cdot P_i \cdot R_i}{P_i + R_i}$$

Since the F1-score does not paint the full picture, we decided to also compare the accuracies of the two systems. The accuracy of the extraction is computed as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Finally, the performance is gauged with the macro average of the F1-scores and the average accuracy:

$$F1score_{avg.} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{F1}_i, Accuracy_{avg.} = \frac{1}{n}\sum_{i=1}^{n} Accuracy_i$$

C. Code Snippet Extraction

The delineation between regular text and code is essential for the proper formatting of code snippets in the final Markdown document, enhancing the overall clarity. To achieve the accurate extraction of code snippets while preserving their syntax and functional utility in the document, HyMark utilizes a Random Forest Classifier (RFC) [16].

1) Dataset Curation: The first segment of the dataset comprised 131,455 samples of everyday language, mimicking the linguistic patterns humans use in their daily communication and the second segment, comprised 131,455 code snippets, with 34 of the most popular high-level programming languages. Figure 6, visually represents the combination of text and code snippets constituting the dataset.

2) Implementation: The methodology starts with tokenization, involving the division of text, specifically code snippets, into smaller units or tokens. The challenge with code tokenization lies in using specific patterns that consider the distinctive syntax of programming languages, a departure from the simplistic approach of splitting by spaces in ordinary text. The subsequent phase in the process is Vectorization, which involves the conversion of the cleaned tokens into a numerical format or vector interpretable by the model. HyMark also employs the TF-IDF (Term Frequency-Inverse Document Frequency) [17] statistical measure to evaluate the

Unnamed: 0	content	language
0	/*\n *\tPCI handling of I2O controller\n *\n *...	Code
1	/**********...**********...**********...**********...	Code
2	/* { dg-do run } */\n\n#include <stdlib.h>\n#i...	Code
3	/*\n * Copyright (C) 2012, Samsung Electronic...	Code
4	/*\n *\n * Copyright (C) 2013 Google, Inc.\n *...	Code
...	...	...
252895	That is the same wiki airlied referred you to	Text
252896	thanks for the help guys.	Text
252897	do you know much about editing the grub file? ...	Text
252898	i love wobly windows it soo funny	Text
252899	ok heres one for all u linux gurus as im sure ...	Text

Figure 6. Data frame showing text and code snippet.

significance of a term within a corpus of texts. By transforming the text into a numerical representation, the model gains the ability to comprehend and learn from the data, recognizing patterns and distinctions that distinguish regular text from code. The higher the TF-IDF value, the more important the term is to that specific document in the collection.

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

$$\text{where} \quad \text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

$$\text{IDF}(t, D) = \log\left(\frac{\text{Total number of documents in the corpus } |D|}{1 + \text{Number of documents containing term } t}\right)$$

3) Model Selection: The process of selecting a suitable model was crucial. After careful consideration and experimentation, RFC emerged as the right choice. This model, by design and operational logic, constructs multiple decision trees during the training phase and derives a classification from each tree for prediction. The inherent simplicity of this approach, coupled with the collaborative decision-making process inherent in a random forest, substantially mitigates the risk of errors associated with a single, potentially overfitted decision tree. This strategic choice ensures a robust and reliable classification framework.

4) Findings: RFC demonstrated exceptional efficacy, achieving remarkable accuracy in distinguishing between standard text and code. The evaluation encompassed a diverse array of programming languages, ranging from the more conventional to the less common, such as SysMD, and languages closer to natural languages, like SQL. The model consistently displayed accurate detection of code snippets across varied linguistic contexts, demonstrating its adaptability and high utility for real-world applications. When compared to ChatGPT across 30 unseen text segments containing popular high-level languages like Python, Java, SQL, SysML, etc, HyMark performed exceptionally well with equal precision, recall, and F1-scores. When it came to a language such as SysMD (also see section I-B) that both systems had not seen during training, we observed a slight advantage for the HyMark over ChatGPT as demonstrated in figure 7.

D. Text to Table

The integration of tables within markdown documents serves as an organizational element and enhances readers' ability to compare data effectively. Recognizing the significance

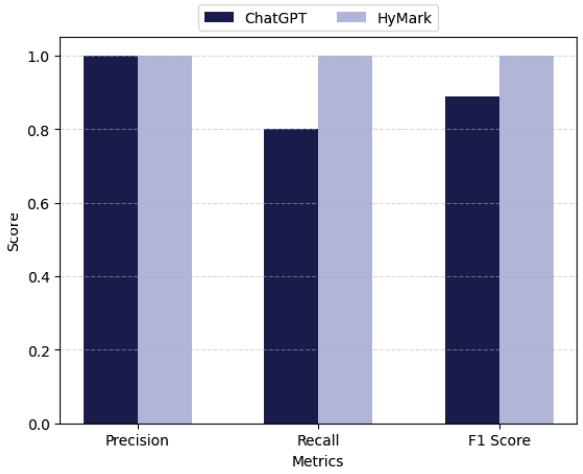

Figure 7. The comparison graph illustrates the performance metrics (Precision, Recall, and F1 Score) for SysMD Code Recognition between ChatGPT and HyMark.

of tables in Markdown, HyMark incorporates a text-to-table conversion feature. However, it is imperative to acknowledge that the current capability of HyMark may not meet the highest expectations. The inherent challenges of the extensive text-to-table conversion problem, coupled with the constraints posed by a relatively modest one-dimensional dataset, warrant the need for refinement and future work in this aspect.

1) Dataset Selection: The WikiTableText dataset [18], sourced from Wikipedia, provides text samples paired with single-row tables, simplifying complexity. However, its limitation lies in the absence of multi-row, multi-column tables commonly encountered in real-world scenarios, which reduces its applicability for tasks requiring more intricate table structures.

2) Model Selection & Training: Our tool uses a sequence-to-sequence (seq2seq) model, fine-tuned from the BART-based model [19]. Inspired by the methodologies discussed in the work of Wu et al. [20], we selected the BART model for its adeptness in text generation. Enhancing the fundamental seq2seq architecture, our model generated tokens sequentially, with each token dependent on the preceding ones. Training data consisted of input text and corresponding output table pairs, denoted as $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. To refine model accuracy, we used two techniques: Table Constraint (TC) and Table Relation Embeddings (TRE). TRE incorporated table relations, while TC imposed constraints on table structure. Evaluation across datasets showcased significant performance improvements with the combined use of TC and TRE.

3) Findings: The evaluation of the text-to-table information extraction model on the WikiTableText dataset yielded good results. In direct comparison, the proposed approach demonstrated superior performance when benchmarked against both the vanilla seq2seq model and a baseline employing *named*

entity recognition (NER). Figure 8 shows the comparison between the proposed technique and the NER & vanilla seq2seq models. Although table generation is possible with HyMark, it still needs a context cue to determine what part of the text needs conversion.

Text Block C

"Breadth First Search (BFS) is an algorithm used for traversing graphs or trees. Traversing means visiting each node of the graph. Breadth First Search is a recursive algorithm to search all the vertices of a graph or a tree. BFS in Python can be implemented by using data structures like a dictionary and lists. (Potential table) C. Y. Lee implemented Breadth First Search (BFS) as a wire routing algorithm in 1961. Breadth First Search (BFS) is an algorithm used to traverse graphs, and its application can be understood through the analogy of solving a Rubik's Cube. In this analogy, the cube's different states represent nodes in a graph, and the possible cube actions correspond to the graph edges. The BFS algorithm aims to visit all vertices of the graph while avoiding cycles. It operates by starting from a node, checking nodes at increasing distances in layers, and it employs a queue to keep track of nodes to be visited. The steps of the algorithm work as follows: 1. Start by putting any one of the graph's vertices at the back of the queue. 2. Now take the front item of the queue and add it to the visited list. 3. Create a list of that vertex's adjacent nodes. Add those that are not within the visited list to the rear of the queue. 4. Keep continuing steps two and three till the queue is empty. Now, we will see how the source code of the program for implementing breadth-first search in Python. Consider the following graph which is implemented in the code below: https://favtutor.com/resources/images/uploads/mceu_1015606421160784810956 3.jpg def bfs(visited, graph, node): visited.append(node) queue.append(node) while queue: m = queue.pop(0) print (m, end = " ") for neighbour in graph[m]: if neighbor not in visited: visited.append(neighbour) queue.append(neighbour)"

IV. Results

In this section, we explore HyMark's real-world application and performance. Text block C introduces Breadth First Search (BFS), illustrating its implementation in Python. A code snippet showcases the Python BFS implementation, alongside a graph image. The HyMark output corresponding to the text block C is organized in Listing 1.

```
# Breadth First Search in Python

## Breadth-First Search
Breadth First Search (BFS) is an algorithm used for
     traversing graphs or trees. Traversing involves
     visiting each node of the graph. It is a recursive
     algorithm designed to search all the vertices of a
     graph or a tree. BFS in Python can be implemented using
      data structures like a dictionary and lists.

```

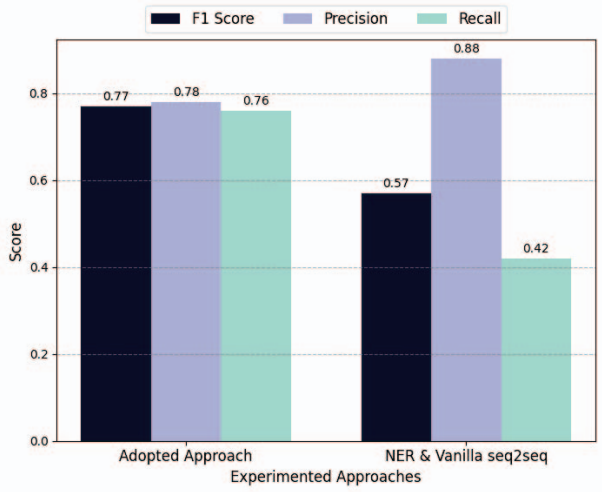

Figure 8. Comparison between the proposed approach and NER & vanilla seq2seq.

```
| title | subtitle | year | algorithm |
| ----- | -------- | ---- | --------- |
| C. Y. Lee | wire routing algorithm | 1961 | Breadth-first
    -search (BFS) |

## Breadth-First Search - How to search Graphs
Breadth First Search (BFS) is an algorithm used to traverse
     graphs, and its application can be understood through
    the analogy of solving a Rubik's Cube.  In this analogy
    , the cube's different states represent nodes in a
    graph, and the possible cube actions correspond to the
    graph edges. The BFS algorithm aims to visit all
    vertices of the graph while avoiding cycles. It
    operates by starting from a node, checking nodes at
    increasing distances in layers, and it employs a queue
    to keep track of nodes to be visited.

The steps of the algorithm work as follows:
1. Start by putting any one of the graphs vertices at
    the back of the queue.
2. Now take the front item of the queue and add it to the
    visited list.
3. Create a list of that vertex's adjacent nodes. Add those
     which are not within the visited list to the rear of
    the queue.
4. Keep continuing steps two and three until the queue is
    empty.

## Breadth First Search Algorithm
Now, we will see how the source code of the program for
    implementing breadth-first search in Python.

Consider the following graph implemented in the code below:
![image](https://favtutor.com/resources/images/uploads/
    mceu_1015606421160784810956 3.jpg)
```
def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
    while queue:
        m = queue.pop(0)
        print(m, end=" ")
        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

```
```

Listing 1. HyMark's output for text block C.

In the output, Python code is encapsulated within a markdown code block. Moreover, HyMark effectively extracts the historical context of BFS by transforming it into a structured table, using the cue provided in the text.

A. HyMark compared to ChatGPT

When ChatGPT was presented with text block C along with the prompt shown in figure 9, it produced the output demonstrated in listing 2.

You

You are an automatic Markdown language syntax generator. In the following piece of unstructured text, use your expertise to structure it into the following Markdown elements:

1. Heading for the text
2. Subheadings according to context
3. Identify bullet points and structure it in markdown syntax
4. Identify hyperlinks and structure it in markdown syntax
5. Identify topic shifts and structure it in markdown syntax
6. Identify image URLs and structure it in markdown syntax
7. Identify code snippets and structure it in markdown syntax
8. Identify table creation opportunity and structure it in markdown syntax:

Figure 9. ChatGPT prompt for Markdown syntax generation for text block C.

```
# Breadth First Search (BFS) Algorithm

## Introduction
Breadth First Search (BFS) is an algorithm used for
    traversing graphs or trees. It involves visiting each
    node of the graph, making it useful for various
    applications.

### BFS Overview
BFS is a recursive algorithm designed to search all the
    vertices of a graph or tree. The algorithm can be
    implemented in Python using data structures such as
    dictionaries and lists.

### Historical Context
C. Y. Lee pioneered the implementation of Breadth First
    Search (BFS) as a wire routing algorithm in 1961.

## Algorithm Explanation
BFS is applied to traverse graphs, with an analogy to
    solving a Rubik's Cube. The algorithm's goal is to
    visit all vertices of the graph while avoiding cycles.
    It operates by starting from a node, checking nodes in
    layers, and using a queue to manage nodes.

#### Algorithm Steps
1. Start by placing any graph vertex at the back of the
    queue.
2. Extract the front item from the queue and add it to the
    visited list.
3. Create a list of adjacent nodes for the current vertex.
    Add unvisited nodes to the rear of the queue.
4. Repeat steps two and three until the queue is empty.

## Implementation in Python
Here is a sample Python code for implementing BFS:

'''python
def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
    while queue:
        m = queue.pop(0)
        print(m, end=" ")
        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)
```

Listing 2. ChatGPT's output for text block C.

B. Evaluation & Comparison

Text block C was chosen as the text to evaluate the performance of both systems for this final real-world application. A short survey was conducted with a group of 20 computer science students (all currently pursuing MSc degrees) to decide what would qualify as headings and subheadings, bullet points, and table creation opportunities. The chosen metrics were precision and recall for each category. The categories were "headings", "bullet points", "hyperlinks," "code snippets", "image URLs", and "table generation"

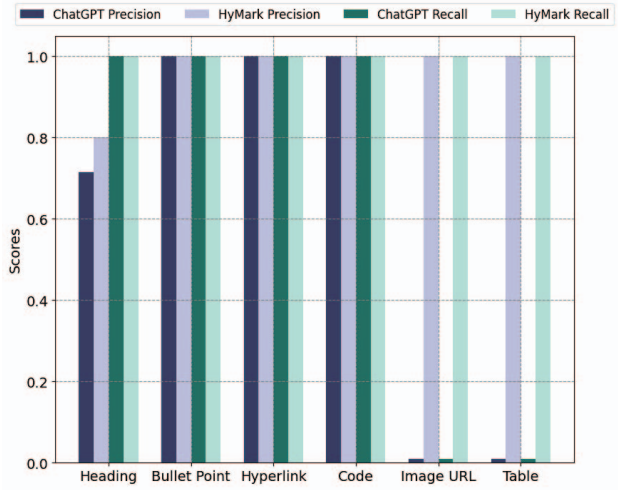

Figure 10. ChatGPT compared to HyMark at identifying different elements of the unstructured text block C.

Each metric was applied separately to the output generated by ChatGPT and HyMark. This evaluation process provided a glimpse of the technical reliability of HyMark in handling specific markdown elements. Figure 10 showcases the performance of HyMark and ChatGPT. It can be seen that HyMark performed at par with ChatGPT and also managed to create a table where ChatGPT could not identify the opportunity to generate a table relative to the context even though a hint in the text was provided. ChatGPT also missed the image URL extraction, although it generated slightly better headings and subheadings that summarized the text well.

C. Case Study

The 20 computer science students also evaluated HyMark by testing it on 20 segments of previously unseen text with approximately 200-300 words each. The evaluation process involved converting structured text into unstructured format, with a copy of the original structured text provided for comparison purposes. Each segment of unstructured text was then individually inputted into HyMark for analysis. The evaluation results were generally satisfactory, with HyMark consistently accurately identifying bullet points, hyperlinks, code snippets, and image URLs. However, there were precisely 4 instances where HyMark missed opportunities to generate tables and

overlooked hints while generating satisfactory headings and subheadings in all cases.

V. Conclusion & Inferences

The driving force behind our work was to assist individuals unfamiliar with Markdown syntax, like the many users of the SysMD notebook. HyMark was created to alleviate the manual efforts associated with document organization and styling. The goal was to develop a system capable of contextually understanding text, emulating human-like comprehension, and autonomously selecting optimal structures and formats. This automation significantly streamlined the time-intensive manual formatting process. Furthermore, this solution can support other tools like GitHub and R Markdown, where Markdown is an integral part of the projects.

Our approach integrated rule-based and AI-based methodologies such as NLP. Rule-based aspects, employing regular expressions, efficiently handled straightforward pattern detection tasks, while NLP methods were used for semantic understanding and information extraction. We systematically compared each feature of HyMark with ChatGPT to demonstrate that a hybrid approach to tasks like Markdown syntax generation can be performed at a consistently high level similar to and even better than state-of-the-art LLM like ChatGPT.

Yet the question remains, *"Why use HyMark when ChatGPT exists?"*. The answer is simple - precision and reliability. While testing, ChatGPT struggled to consistently deliver high-quality extraction results. It often required multiple attempts with the same prompts to achieve the desired outcomes, and prompt engineering posed challenges, consuming additional time in crafting them accurately. In the results section (see section IV) of this manuscript, ChatGPT exhibited limitations by missing a labeled table generation opportunity and image URL extraction, consistent with trends observed in earlier iterations of testing. In contrast, HyMark uses specialized AI tools tailored for specific tasks and a rule-based approach to simpler tasks for consistent precision. Whether applied in academic writing or content creation, HyMark streamlines and accelerates the writing process, offering a reliable solution that saves valuable time.

Acknowledgment

This work was funded by the BMBF Projects, KI4Boardnet and GENIAL. The authors would also like to acknowledge that ChatGPT was used to enhance the linguistic clarity of this manuscript, specifically to correct grammar and sentence structure.

References

[1] OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://openai.com.

[2] J. Gruber, *Markdown Syntax*. [Online]. Available: http://daringfireball.net/projects/markdown/syntax. [Accessed: June 2012].

[3] Š. Dalecke, K. A. Rafique, A. Ratzke, C. Grimm, and J. Koch, *SysMD: Towards "Inclusive" Systems Engineering*, in *2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS)*, Coventry, United Kingdom, 2022, pp. 1-6, doi: 10.1109/ICPS51978.2022.9816856.

[4] OMG SysML v2 Revision Task Force, *Systems Modeling Language (SysML) - Version 2*, Object Management Group, 202X. [Online]. Available: https://www.omg.org/spec/SysML/2.0/. [Accessed: Date Accessed].

[5] N. Bleiel, "Collaborating in GitHub," 2016 IEEE International Professional Communication Conference (IPCC), Austin, TX, USA, 2016, pp. 1-3, doi: 10.1109/IPCC.2016.7740497.

[6] D. Udwin and B. S. Baumer, "R Markdown," WIREs Computational Statistics, vol. 7, no. 3, pp. 167-177, 2015. https://doi.org/10.1002/wics.1348

[7] Q. Xi and D. Walker, *A context-free markup language for semi-structured text*, *SIGPLAN Not.*, vol. 45, no. 6, pp. 221-232, Jun. 2010.

[8] M. Topaz, L. Murga, K. M. Gaddis, M. V. McDonald, O. Bar-Bachar, Y. Goldberg, and K. H. Bowles, *Mining fall-related information in clinical notes: Comparison of rule-based and novel word embedding-based machine learning approaches*, *Journal of Biomedical Informatics*, vol. 90, pp. 103103, 2019.

[9] P. Yin and G. Neubig, *A syntactic neural model for general-purpose code generation*, *arXiv preprint arXiv:1704.01696*, 2017.

[10] Hugging Face, *all-mpnet-base-v2*, 2023. Online. Available: https://huggingface.co/sentence-transformers/all-mpnet-base-v2/blame/86eb780758622d085bac2d7f6aea99c157a5ee28/README.md.

[11] S. Bird, E. Klein, and E. Loper, *NLTK: The Natural Language Toolkit*, 2009. [Online]. Available: http://www.nltk.org/. [Accessed: August 12, 2023].

[12] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*, arXiv preprint arXiv:1910.10683, 2019.

[13] Medium, *[Online]. Available: https://medium.com/. [Accessed: August 13, 2023].*

[14] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in *Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004)*, Barcelona, Spain, 2004.

[15] M. Spencer, "Regular Expressions: A Comprehensive Guide," *Regex101*, 2023. [Online]. Available: https://regex101.com/. [Accessed: August 3, 2023].

[16] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

[17] R. A. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[18] "WikiTableText Dataset." GitHub, https://github.com/sean0042/Open_WikiTable, Accessed: August 20, 2023.

[19] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," arXiv preprint arXiv:1910.13461, 2019.

[20] X. Wu, J. Zhang, and H. Li, "Text-to-table: A new way of information extraction," arXiv preprint arXiv:2109.02707, 2021.

[21] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[22] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[23] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[24] K. Elissa, "Title of paper if known," unpublished.

[25] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[26] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[27] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.