LLMManuGen © 2023 - Marko T. Manninen - All Rights Reserved

Use with ChatGPT + Advanced Data Analysis plug-in (ADA) activated.

Introduction

The LLMManuGen Manuscript class is designed to manage and traverse hierarchical data structures, such as a table of contents or a task list. While traversing throught the tree, one can let GPT to generate content for each section in a controlled manner.

Manuscript class functions as a persistent memory extension for ChatGPT, overcoming the token limitations inherent to vanilla ChatGPT. By doing so, it ensures that the context is not lost during the content generation process, enabling more precise and organized output.

Core functionalities include the ability to move through the sections in a step-by-step manner, add content to each section, and export the entire structure into JSON or Markdown formats. The class also offers methods for searching specific sections and updating them with new content, providing a flexible framework for text-based projects.

The manual you are currently navigating serves as a comprehensive guide to understanding and utilizing the Manuscript class. It covers everything from basic installation and setup to advanced features like content management and exporting options. The guide is structured into distinct sections, including Installation, Features, and Usage, each designed to offer detailed insights into specific functionalities. Additionally, you'll find a Troubleshooting section to help resolve common issues, and a FAQ for quick answers to general queries.

Installation

Utilizing LLMManuGen with ChatGPT and ADA (Advanced Data Analysis plug-in) doesn't require a traditional software installation. Instead, the process is initiated through a specific prompt mechanism designed to seamlessly integrate these technologies. The prompt serves as the entry point for all functionalities, encapsulating user-defined variables and directives that guide the behavior of the language models involved. It acts as a set of instructions that the language models adhere to, ensuring that the generated content meets user-defined criteria, be it length, tone, or subject matter.

Once the prompt is prepared, it can be fed into the ChatGPT and ADA models through their respective interfaces. The Manuscript class, which forms the core of LLMManuGen, takes over from this point. It maintains a structured, hierarchical data model that captures both the content and the metadata for the manuscript being generated. This hierarchical data model is an integral feature, allowing for a more organized and intuitive navigation through various sections and subsections of the content. It effectively serves as a dynamic table of contents that can be traversed linearly or jumped through as needed.

Moreover, the Manuscript class offers robust content and metadata management functionalities. Users have the flexibility to add or modify content within each section or subsection. Metadata such as instructions, guidelines, and constraints can also be defined for each section, offering a more refined control over the generated content. These directives guide the generation process, ensuring the output aligns with the user's expectations.

Additionally, the class provides options for exporting the manuscript into various formats, including JSON and Markdown, allowing for easy sharing and publication. It also includes a built-in search functionality, making it easier to locate specific content or sections based on different criteria. For those who wish to take a break or continue their work later, the Manuscript class supports data persistence. It allows users to save their progress in a JSON file, which can be loaded back into the system for a seamless continuation of work.

Below is the initialization prompt for copy & paste:

~~~
First, generate and execute Python code from the pseudo-class description. Use
compact/minimal naming conventions in method code blocks. Then, follow the
wizard.

```pseudocode

class Manuscript

    init(data dictionary, title field is "title", children field is "sections")
        set title and children fields as class attributes
        data = data dictionary
        data set children field to empty list by default if not exist
        validate_schema(data)
        reset_current_section_path()

    validate_schema(schema)
        _(schema)
            if schema is not dictionary -> False
            if title field is not in schema or is empty -> False
            if children field in schema
                if schema children field is not list -> False
                for child in schema children field
                    if not _(child) -> False
            True
        if not _(schema)
            raise ValueError
        True

    reset_current_section_path(path indices None by default)
        current path = path indices or first key of data sections as initial path

    get_section(path indices)
        section = data children field
        for index in path indices
            try
                if children field in section
                   section = section children field
                section = section index
            except
                None
        section

    get_current_section()
        get_section(current path)

    get_current_and_next_sections(without children is true)
~~~

```
        current section = get_current_section()
        temporary path = current path copy
        next section = None
        if move_to_next_section() is "continue"
            next section = get_current_section()
        current path = temporary path
        current section without children field if without children, next section
without children field if without children

    move_to_next_section()
        section = data children field
        path = current path copy
        for index in path except last item
            section = section index children field
        if children field in section path last index and section path last index
children field
            current path append 0
            "continue"
        if section length > path last index + 1
            increment current path last index
            "continue"
        else
            while current path length > 1
                current path pop
                sections = data children field
                for index in current path except last item
                    sections = sections index children field
                if sections length > current path last index + 1
                    increment current path last index
                    "continue"
            else
                "end"
        "continue"

    add_current_content(content dictionary)
        if current section is get_current_section()
            remove possible children field from content dictionary copy
            update current section with content dictionary

    static from_json(file path)
        Manuscript(json load file path)

    to_json(optional directory)
        validate_schema(data)
        if directory does not exists
            make directory
        write data to json string to safe filename in directory
        safe filename, data to json string

    get_safe_filename(extension)
        title = data title field or "Untitled" + "_"  + datetime now
        remove all except alphanumeric from title with spaces replaced with
underscore + extension

    get_table_of_contents(tree structure is true)
        current section = get_current_section()
        current title = current section title field
        _(sections, level, prefix is empty)
            for i, section in sections
                is_last = if i is sections length - 1
                if tree structure
                    new prefix = if is_last then "└── " else "├── "
                    spacer = if is_last then "    " else "|   "
                else
                    new prefix = empty
                    spacer = "  " times (level - 1)
                toc append prefix + new prefix + section title field with "*" if 
section title field is current title
                if section child field
                    add toc _(section child field, level + 1, prefix + spacer)
        _(data children field, level 1)
        toc string

    to_md(optional content field, optional directory)
        _(subsection, level)
            for section in subsection
                markdown add "#" times level + space + section title field
                if content field
                    markdown add (section content field or "Section content not 
present for the field: " + content field) wrapped with linebreaks
                markdown add _(section children field, level + 1)
        _(data children field, level 1)
        write markdown string to safe filename in directory
        safe filename, markdown string

    search(query as string or regex, optional field, optional path)
        search field = field or title field
        _(sections, new path is empty list)
            for i, section in sections
                local path = new path + [i]
                if (search field in section and regex query or string query match 
in section search field) and
                   (path is None or path is local path slice to path length)
                    results add {children field: section, path: local path}
                if children field in section
                    _(section children field, local path)
        _(data children field)
        results

    find_path_indices(field values)
        _(subsections, remaining fields, new path is empty list)
            for i, section in subsections
                if section title field is first from remaining fields
                    local path = new path + [i]
                    if remaining fields length is 1
                        return local path
                    if children field in section
                        return _(section children field, remaining fields slice 
from second, local path)
        _(data children field, field values)

\```
```

```
## WIZARD

### User guide

Purpose of the class: To extend vanilla ChatGPT functionality beyond the
limitations of the token window. Enhanced functionality is achieved by using
persistent memory in a Python environment for storing a hierarchical list of
tasks/table of contents/etc. and traversing through them in a step-by-step
manner. This enables the exact processing of content rather than eventually
losing the reference point and list items, as vanilla ChatGPT does without
persistent memory. The Manuscript class can upload and download JSON
representation of the structured data, which makes it possible to continue
traversal from any given point.

_Note: This class gives basic functionality for hierarchical traverse of the
tree-structured items and can be extended to give a more restricted form for the
schema and specific manuscript generation._

### Steps 1-7

Give a small intro about the functionality of Manuscript class and the following
steps. Then ask the user:

1a. To upload existing JSON file. It must meet the minimal structure requirements
-> output minimal example with mandatory title field and children field for
nested structures.

1b. To give a topic or table of contents. It is to generate a JSON data structure
for the Manuscript class initialization.

2a. Print the table of contents. Asterisk denotes the current section.

2b. Ask from which section the user wants to start. Then traverse to the starting
point.

3. Get the current and next section. The latter indicates what is coming next at
the iteration process, which enables building a more natural content bridge
between sections.

4. Ask what the user wants to do with the current section. For instance, store
either user-given or LLM/GPT generated data to the section (children field
excluded), might it be content, summary, prompt directives ({instructions,
guidelines, constraints}), created/updated/completed datetime, etc. fields.

5. Traverse to the next item in the list if not "end".

6. Repeat steps 3-5

_Note: The list can be traversed multiple times, each time adding more data to
sections. At the end, the user should have a complete schema ready for MD output
or exporting manuscript schema to the other application for the publication
process._

7. Generate JSON/MD files for download. JSON file can be used in an iterative way
as re-input to the manuscript class. MD file is read-only.
```

```
---

Start by creating the Manuscript class and then follow the wizard.
```

The rest is basically following the instruction and step-by-step guide wizarded by ChatGPT.

However, it is good to know the internal working of the class to have a better control over the flow of generating the hierarchical tree and traversing throught it.

Features

The Manuscript class is rich in features, each designed to address specific aspects of manuscript creation and management. Below are some of the key features, listed in order of their significance and utility:

Hierarchical Structure: The class offers a tree-like hierarchical representation for organizing sections and subsections. This feature enables a more organized approach to content management, making it easier to navigate through different layers of the manuscript.

Dynamic Traversal: The Manuscript class allows for both linear and non-linear navigation through the manuscript's sections. Whether you want to jump to a specific section or move sequentially, this feature offers the flexibility needed for effective content generation.

Content and Metadata Management: Not only can you add and modify content within each section, but you can also include prompt directives like instructions, guidelines, and constraints. This feature automates the addition of timestamps, providing a detailed account of when each section was created or updated.

Search Functionality: The class offers robust search capabilities, enabling users to find specific content or sections based on various criteria. This functionality is crucial for large manuscripts where pinpointing a specific section can be time-consuming.

Data Persistence: The Manuscript class allows for the saving and loading of the manuscript's state in JSON format. This feature facilitates a seamless continuation of work, especially in cases where the content generation spans multiple sessions.

Export Options: With various export options, the Manuscript class supports the transformation of the manuscript into Markdown format. This feature is particularly useful for sharing or publishing the content in platforms that support Markdown.

Table of Contents: The class automatically generates a dynamic table of contents, providing an overview of the manuscript's structure. This is particularly beneficial for large documents that require quick navigation.

ADA Compatibility: Finally, the class is designed to work seamlessly with ChatGPT and ADA (Advanced Data Analysis). The compatibility ensures that users can leverage the advanced features of these language models while using the Manuscript class.

Each of these features contributes to making the Manuscript class a versatile and powerful tool for manuscript creation and management, addressing a wide range of needs and requirements.

Usage

This section outlines the internal functionality of the Manuscript class. While users typically interact with ChatGPT and ADA through human language, or a combination of human and programming languages, the following guide provides an in-depth look into the class's programmable features.

1: Initialize the Manuscript Class

Begin by initializing the Manuscript class with a data dictionary. The dictionary should have at least a 'title' and optional 'sections' for nested content.

```
data = {'title': 'My Manuscript', 'sections': []}
manuscript = Manuscript(data)
```

2: Navigating Through Sections

Use `move_to_next_section()` to navigate through sections. The method returns 'continue' if the navigation is successful and 'end' if you reach the end of the manuscript.

```
status = manuscript.move_to_next_section()
```

3: Adding Content

To add content to a section, use `add_current_content()`. This method updates the current section with a content dictionary.

```
content = {'summary': 'This is the introduction.'}
manuscript.add_current_content(content)
```

4: Search Functionality

The `search()` method enables you to find specific sections based on different criteria. You can pass in a query string or regex pattern.

```
results = manuscript.search('introduction')
```

5: Data Persistence

Save the current state of the manuscript using `to_json()`. This method validates the schema and saves the manuscript in JSON format.

```
manuscript.to_json('path/to/save')
```

6: Reloading Manuscript

You can reload a saved manuscript using `from_json()`.

```
manuscript = Manuscript.from_json('path/to/json')
```

7: Exporting to Markdown

The `to_md()` method allows you to export the manuscript to Markdown format, making it easier to publish or share.

```
manuscript.to_md('path/to/save')
```

8: Updating Metadata

You can include specific directives for each section, guiding the content generation process. Each section allows for metadata like guidelines, constraints, and updated timestamps. This ensures a comprehensive manuscript management system.

9: Final Checks

Before exporting or sharing, make sure to validate the manuscript using `validate_schema()`. This ensures that your manuscript adheres to the predefined structure.

```
manuscript.validate_schema()
```

With these steps, you'll be well-equipped to utilize all features of LLMManuGen, making manuscript generation a structured and efficient process.

Troubleshooting

Common Content Generation Issues

1. Token Limitation

Problem: Running into token limits during content generation, causing incomplete sections.
Solution: Break down large sections into smaller subsections or adjust your content to fit within the token limitations of the language model you're using.

2. Context Loss

Problem: The model loses track of the context in deeply nested or complex manuscripts.
Solution: Simplify the structure or reduce the depth of nesting in your manuscript. Ensure that each section or subsection is concise and to the point.

3. Prompt Ambiguity

Problem: Ineffectual or ambiguous prompts leading to off-target content generation.
Solution: Use clear and specific prompts. Test the prompts beforehand to ensure they yield the desired results.

4. Instruction Overload

Problem: Too many or too complex directives make it difficult for the model to generate appropriate content.
Solution: Limit the number of directives to essential ones and make them as clear as possible. Try to avoid over-complicating your instructions.

5. Consistency Issues

Problem: Discrepancies in tone, style, or facts across different sections generated by the LLM.
Solution: Proofread the entire manuscript for consistency. If you are using a particular set of guidelines, ensure they are applied uniformly across all sections.

Specific Technical Issues

6. Initial Prompt Malfunction

Problem: The pseudo-code does not yield the correct Python code.
Solution: If pseudo-class generation fails, you can remove the search methods to shorten the class or replace the pseudocode with native Python code from the GitHub project: GitHub Repo.

7. Navigation Difficulties

Problem: Trouble navigating through the hierarchical structure, especially for complex manuscripts.
Solution: Familiarize yourself with the navigation methods provided by the Manuscript class. Practice using the table of contents to better understand the manuscript's structure. Additionally, using search, find, and resetting index methods can aid navigation.

8. Export Errors

Problem: Issues with exporting the manuscript to formats like JSON or Markdown.
Solution: Double-check the export options and ensure that the manuscript validates against the schema before attempting to export. The class contains only a simplified version of MD export. Users may want to extend the functionality via a custom class and redefined to_md method.

9. Version Compatibility

Problem: Issues arising from using LLMManuGen with different versions of ChatGPT or ADA.
Solution: There is no quick solution to this. If ChatGPT or ADA changes over time, the code generation and prompt execution may fail. One just needs to figure out relevant changes to the initial prompt or see the GitHub repo for possible updates.

FAQ

1. What is LLMManuGen?

LLMManuGen is a tool that extends the capabilities of ChatGPT and the Advanced Data Analysis plug-in, allowing for more structured and persistent content generation and management.

2. How do I get started with LLMManuGen?

To get started, follow the instructions in the manual's Installation section or find the initial prompt here.

3. What are the system requirements?

LLMManuGen - initial prompt and Manuscript class generation from the pseudo-class - requires GPT version 4 (which is commercial as of now) and the Advanced Data Analysis plug-in activated. This allows ChatGPT to access the Python environment.

4. How do I navigate through the manuscript?

To navigate, you can use phrases like 'move to next section,' 'show current section,' or 'go to a specific section.' To see what's coming next, you can ask 'what's the next section?' To reset your position to the beginning or a specific section, say 'reset to start' or 'reset to [section name].' If these phrases are misinterpreted, you can use direct method names like `move_to_next_section()`, `get_current_section()`, `reset_current_section_path()`, and `get_current_and_next_sections()`.

5. Can I export my manuscript?

Yes, you can say, 'Export to JSON' or 'Export to Markdown' to export your manuscript. For more control, you can use the methods `to_json()` and `to_md()`.

6. Is LLMManuGen compatible with different versions of ChatGPT and ADA?

Unfortunately, compatibility across different versions cannot be guaranteed.

7. How do I troubleshoot common issues?

For common issues and solutions, refer to the 'Troubleshooting' section of this manual.

8. Is my data safe and secure?

LLMManuGen does not store data externally, so your manuscript remains in the Python environment until exported, or until the environment is reseted by the system. Unfortunately the reset happens quite often if the Enterprise version is not used. For OpenAI's data policy, refer to their site and documentation.

10. Can I customize LLMManuGen?

Absolutely, LLMManuGen is designed to be extensible. You can redefine any method, extend current functionality, or even write support for previous versions of the software. The pseudo-code provides a baseline, but you may need to paste classes and functions in separate chunks if you run into input and output token limitations. Moreover, you may want to extend the whole functionality to the ChatGPT+Noteable or autopilot it by using OpenAI API.

Acknowledgments

Special thanks to Ville-Veli Einari Saari for being a valuable pilot user and for providing insightful feedback.