# **Web3 Daily Digest: Automated Newsletter Generator - Documentation**

## **1. Introduction: Navigating the Web3 Landscape with AI**

The Web3 space, encompassing blockchain, cryptocurrencies, decentralized finance (DeFi), NFTs, and more, is characterized by its rapid evolution and constant influx of information. Keeping up with the daily developments from numerous sources can be overwhelming, leading to information overload and the potential to miss critical insights. The **Web3 Daily Digest** project offers an innovative and automated solution to this challenge.

This project is designed to be your intelligent curator for Web3 news. It automates the process of gathering, summarizing, and delivering the most pertinent daily updates directly to your Telegram chat. By leveraging cutting-edge AI (specifically Large Language Models or LLMs) for summarization and employing smart deduplication techniques, the Web3 Daily Digest transforms a flood of information into concise, actionable insights, enabling users to stay informed without being drowned in data.

### **1.1 The Information Overload Problem in Web3**

The decentralized and permissionless nature of Web3 means that information is generated and disseminated across a vast array of platforms, including:

* **Specialized News Outlets:** CoinDesk, CoinTelegraph, The Defiant, etc.
* **Blockchain-Specific Blogs:** Project updates, technical deep dives.
* **Community Forums & Social Media:** Twitter (X), Reddit, Discord channels.
* **Developer Updates:** GitHub repositories, whitepapers.

Manually sifting through these sources daily to identify key developments, understand their implications, and filter out noise is a time-consuming and inefficient task. This often leads to:

* **Missed Opportunities:** Not reacting quickly to market shifts or project announcements.
* **Burnout:** The mental fatigue from constantly consuming and processing large volumes of data.
* **Redundancy:** Reading the same news from multiple sources.

### **1.2 The AI-Powered Solution**

The Web3 Daily Digest addresses these problems by automating the entire pipeline:

* **Intelligent Source Aggregation:** It systematically pulls news from a curated list of high-quality Web3 and crypto sources, intelligently choosing between efficient RSS feeds and robust HTML web scraping for comprehensive coverage.
* **Automated Summarization:** Instead of simply presenting article links, the project employs LLMs to generate concise summaries for each piece of content. This allows users to quickly grasp the essence of an article without needing to read the full text unless specifically interested.
* **Deduplication for Clarity:** News often breaks simultaneously across multiple platforms or is re-reported with minor variations. The project uses similarity checks to filter out redundant articles, ensuring the newsletter is as lean and informative as possible.
* **Scheduled Delivery:** The entire process runs automatically on a daily schedule, delivering a fresh digest directly to the user's preferred Telegram chat, fitting seamlessly into their routine.

This automated, AI-driven approach transforms information consumption from a chore into a seamless and insightful experience, keeping users ahead in the fast-paced Web3 landscape.

## **2. Core Architecture and Feature Breakdown**

The Web3 Daily Digest operates on a modular architecture, with distinct components handling different stages of the newsletter generation process. This design ensures maintainability, scalability, and allows for flexible customization.

### **2.1 Overall Workflow**

The daily newsletter generation process can be conceptualized in the following steps:

1. **Scheduled Trigger:** The scheduler.py script, running continuously in the background, triggers the newsletter generation at a predefined time (e.g., once a day).
2. **Article Collection:**
   * The system iterates through a predefined list of NEWS\_SOURCES.
   * For **RSS Feed** sources, it uses a dedicated library to parse the XML feeds and extract article metadata (title, URL, summary, date).
   * For **HTML Scraped** sources, it performs direct web scraping using configured CSS selectors to extract similar article information.
3. **Content Extraction (for Summarization):** For each collected article, the full content is extracted from its respective URL. This is often more involved for HTML scraped articles, requiring a robust content extraction mechanism (e.g., newspaper3k or custom parsing).
4. **AI-Powered Summarization:**
   * The extracted full content of each article is fed into a Large Language Model (LLM).
   * The LLM generates a concise summary of the article's key points.
5. **Deduplication:**
   * A comparison algorithm checks the similarity between the summaries (or full article texts) of all collected articles.
   * Articles exceeding a predefined DEDUPLICATION\_THRESHOLD are identified and removed, ensuring no redundant information is sent.
6. **Newsletter Formatting:**
   * The remaining unique and summarized articles are compiled into a structured newsletter.
   * The newsletter is formatted in both HTML and Markdown to support rich display across various platforms.
7. **Newsletter Delivery:**
   * The formatted newsletter is sent to the designated Telegram chat via the Telegram Bot API.
8. **Logging and Reporting:** Throughout the process, relevant information, warnings, and errors are logged to aid in monitoring and troubleshooting.

### **2.2 Key Features in Detail**

* **Diverse News Source Integration:**
  + **RSS Feeds:** The project leverages the efficiency of RSS feeds (Really Simple Syndication) which are designed for content syndication. This is the preferred method as it's less resource-intensive and less prone to breaking. Sources like CoinDesk, CoinTelegraph, The Defiant, and CryptoPanic provide reliable RSS feeds.
  + **HTML Web Scraping:** For sources that do not offer comprehensive or reliable RSS feeds (e.g., Bitcoinist), the system falls back to direct HTML scraping. This requires identifying specific CSS selectors (container, title, link, summary, date) to accurately extract article information from the web page's HTML structure. This method is more prone to breaking if the website's layout changes.
* **AI-Powered Summarization:**
  + At the heart of the "digest" functionality is the use of Large Language Models (LLMs). An LLM is a type of artificial intelligence program that can recognize, understand, and generate human language. Trained on vast amounts of text data, LLMs can distill complex articles into their core message.
  + The project passes the full text of each article to an LLM (e.g., via the OpenAI API or a compatible local model) with a prompt instructing it to summarize the content concisely. This allows users to quickly scan headlines and summaries to decide if they want to read the full article.
* **Intelligent Deduplication:**
  + To prevent users from receiving the same news repeatedly from different sources, the system employs a deduplication mechanism.
  + It calculates the similarity between article summaries (or potentially full texts) using a similarity threshold. If two articles are found to be highly similar (e.g., above 80% similarity), one of them is discarded. This ensures a clean and non-redundant newsletter.
* **Flexible Multi-Format Delivery:**
  + The generated newsletter is prepared in two common formats:
    - **HTML:** Allows for rich text formatting, hyperlinks, bolding, etc., which is ideal for display within Telegram's message formatting capabilities.
    - **Markdown:** Another lightweight markup language that also supports basic formatting and is widely used.
  + This dual-format approach ensures compatibility and optimal rendering across different platforms or future integrations.
* **Automated Daily Scheduling:**
  + The schedule library in Python is used to configure a daily job.
  + The scheduler.py script continuously runs, waiting for the designated time to trigger the generate\_newsletter function. This ensures hands-free, consistent delivery of the digest.
* **Customizable Content:**
  + The config.py file serves as a central hub for all configurable parameters.
  + Users can easily adjust the TOP\_ARTICLES\_DAILY limit, manage NEWS\_SOURCES (adding new RSS feeds or defining HTML scrape patterns), and fine-tune AI model parameters or DEDUPLICATION\_THRESHOLD to personalize their newsletter experience.

This comprehensive feature set and modular design make the Web3 Daily Digest a robust and user-friendly solution for staying informed in the dynamic Web3 landscape.

## **3. Setup, Configuration, and Usage**

Getting the Web3 Daily Digest up and running is designed to be a straightforward process, involving cloning the repository, installing dependencies, configuring environment variables, and starting the scheduler.

### **3.1 Project Structure**

The repository is structured to maintain clarity and separation of concerns:

Multi-Agent-Newsletter-Generator-using-LangChain/

├── .env.example # Example environment variables file

├── config.py # Central configuration for news sources, articles limit, etc.

├── main.py # Main logic for fetching, summarizing, deduplicating articles

├── newsletter\_formatter.py # Handles HTML and Markdown formatting of the newsletter

├── requirements.txt # List of Python dependencies

├── scheduler.py # Script to schedule and run the daily newsletter generation

├── telegram\_bot.py # Handles sending messages to Telegram

├── README.md # Project overview and quick start guide

└── LICENSE # Project's license

* **.env.example**: A template file illustrating the required environment variables.
* **config.py**: Centralized configuration file for all customizable parameters, including news sources, TOP\_ARTICLES\_DAILY limit, and deduplication thresholds.
* **main.py**: Contains the core logic for fetching articles from various sources, performing AI summarization, and deduplication.
* **newsletter\_formatter.py**: Dedicated module for taking processed article data and formatting it into visually appealing HTML and Markdown newsletters.
* **requirements.txt**: Lists all Python libraries required for the project.
* **scheduler.py**: The script responsible for setting up and running the daily automated task of generating and sending the newsletter. This script needs to be kept running.
* **telegram\_bot.py**: Manages interaction with the Telegram Bot API, including sending messages to the specified chat ID.

### **3.2 Setup & Quick Start Guide**

Follow these steps to deploy and run your automated Web3 Daily Digest:

**Clone the Repository:** Open your terminal or command prompt and execute the following commands to clone the project to your local machine:  
Bash  
git clone https://github.com/karthikeyapranav/Multi-Agent-Newsletter-Generator-using-LangChain.git

cd Multi-Agent-Newsletter-Generator-using-LangChain

**Install Python Dependencies:** It's highly recommended to use a Python virtual environment to manage dependencies.  
Bash  
# Create a virtual environment

python -m venv venv

# Activate the virtual environment

# On Windows:

# .\venv\Scripts\activate

# On macOS/Linux:

# source venv/bin/activate

# Install the required libraries

pip install -r requirements.txt

1. **Configure Environment Variables:** The project relies on environment variables for sensitive information like API keys.
   * Create a new file named .env in the root directory of the project (the same directory as README.md).
   * Populate this file with your Telegram Bot Token and Chat ID.
   * **TELEGRAM\_BOT\_TOKEN**:
     + Open Telegram and search for @BotFather.
     + Start a chat with @BotFather and send the /newbot command.
     + Follow the prompts to choose a name and a username for your bot (username must end with bot).
     + @BotFather will provide you with an HTTP API token. Copy this token. It will look something like 1234567890:ABCDEF1234567890abcdef1234567890.
   * **TELEGRAM\_CHAT\_ID**:
     + Start a chat with your newly created bot (or the group/channel where you want the newsletter to be sent).
     + Send any message to the bot (or in the group/channel where the bot is an admin).
     + Forward that message to @getidsbot (or any similar bot like @RawDataBot).
     + The bot will reply with various IDs, including your chat\_id. For a private chat with the bot, it will be a positive number. For a group or channel, it will typically be a negative number.
   * **OPENAI\_API\_KEY (Optional):** If you plan to use OpenAI's models for summarization (which often offer better quality), you will need an API key. Add this to your .env file if applicable.
     + Sign up for an OpenAI account and generate an API key from their platform.

Your .env file should look like this:  
Code snippet  
TELEGRAM\_BOT\_TOKEN=YOUR\_TELEGRAM\_BOT\_TOKEN\_HERE

TELEGRAM\_CHAT\_ID=YOUR\_TELEGRAM\_CHAT\_ID\_HERE

# OPENAI\_API\_KEY=sk-your\_openai\_api\_key\_if\_used

1. *Important*: Never share your .env file or commit it to version control.

**Run the Scheduler:** This is the main entry point to start the daily newsletter process. The scheduler.py script will run continuously, triggering the newsletter generation at the configured daily interval.  
Bash  
python scheduler.py

1. You will see logs in your console indicating that the scheduler has started and is waiting for the next run. The newsletter will be delivered to your specified Telegram chat automatically once a day.

### **3.3 Advanced Configuration (config.py)**

The config.py file provides a centralized location for fine-tuning the behavior of your newsletter generator.

* **TOP\_ARTICLES\_DAILY**:
  + **Purpose:** Controls the maximum number of articles that will be included in each daily newsletter.
  + **Default:** (Check config.py for default value, typically around 5-10)
  + **Adjustment:** Modify this integer value to get more or fewer articles.
* **NEWS\_SOURCES**:
  + **Purpose:** A list of dictionaries, each defining a news source to scrape. This is the most crucial part for customizing content.
  + **Adding/Removing Sources:**

**For RSS Feeds ("type": "rss"):**Python  
{

"name": "NewRSSSource",

"url": "https://newsource.com/rss\_feed.xml",

"type": "rss"

},

* + - Simply provide a unique name and the url to the RSS feed.

**For HTML Scraped Sources ("type": "html"):**Python  
{

"name": "NewHTMLSource",

"url": "https://newsource.com/news",

"type": "html",

"scrape\_pattern": {

"container": "article.news-item",

"title": "h2.article-title a",

"link": "h2.article-title a",

"summary": "p.article-summary",

"date": "span.article-date"

}

},

* + - This requires more effort:
      * **name**: A unique name for the source.
      * **url**: The main URL of the news listing page.
      * **type**: Must be "html".
      * **scrape\_pattern**: A dictionary of CSS selectors:
        + "container": The CSS selector for the main block that contains each individual article on the listing page. This helps the scraper identify distinct articles.
        + "title": CSS selector to extract the article's title *within* its container.
        + "link": CSS selector to extract the article's URL *within* its container. This will usually be an <a> tag's href attribute.
        + "summary": (Optional) CSS selector for a brief summary or excerpt on the listing page. If not available, the AI will summarize the full article.
        + "date": (Optional) CSS selector for the publication date. The scraper will attempt to parse this.
      * **Identifying CSS Selectors:** Use your web browser's "Developer Tools" (usually F12 or right-click -> "Inspect") to inspect the elements on the target website and identify the correct CSS selectors. This is a crucial step for adding new HTML sources and requires some familiarity with HTML and CSS.
* **DEDUPLICATION\_THRESHOLD**:
  + **Purpose:** A float value (e.g., 0.7 to 0.9) that determines how similar two article summaries (or texts) must be to be considered duplicates. A higher value means articles need to be more identical to be removed.
  + **Adjustment:** Experiment with this value to fine-tune the deduplication aggressive-ness. Start with a value like 0.8 and adjust based on the results.
* **AI Model Selection and Parameters:**
  + If using an LLM, config.py might contain parameters for the model name, temperature (creativity), and maximum tokens. Refer to the specific LLM integration details in main.py or associated modules for exact parameters.
  + Ensure that any required API keys for these models are correctly set in your .env file.

By carefully configuring these options in config.py, you can tailor the Web3 Daily Digest to your exact preferences and news consumption habits.

## **4. Troubleshooting and Best Practices**

While designed for robustness, automated systems can encounter issues, especially when interacting with external services like websites and APIs. This section provides guidance on common problems and best practices for running the Web3 Daily Digest efficiently and ethically.

### **4.1 Troubleshooting Common Issues**

* **403 Forbidden Errors when Scraping HTML:**
  + **Cause:** The website has detected and blocked your scraping requests. This often happens if the website uses bot detection mechanisms, such as checking User-Agent headers, Referer headers, or analyzing request frequency.
  + **Solution:**
    - **User-Agent Rotation:** While not explicitly implemented in this basic setup, for more advanced scraping, you might need to rotate User-Agent headers to mimic different browsers.
    - **Rate Limiting:** Introduce longer time.sleep() delays between requests to the problematic source in your scraping logic (main.py). This reduces the frequency of requests, making them appear more "human-like."
    - **Proxy Services:** For sustained, large-scale scraping, consider using a proxy service to rotate IP addresses, making it harder for the target website to block your requests.
    - **robots.txt and ToS:** Always check the website's robots.txt file (e.g., https://example.com/robots.txt) and their Terms of Service. Some websites explicitly forbid scraping. Respect these rules.
* **Timeouts During Article Fetching:**
  + **Cause:** The server hosting the news source (either RSS or HTML) is slow to respond, or your network connection is unstable. This can cause the script to hang or fail before receiving data.
  + **Solution:** Increase the timeout parameter in the network requests. For requests calls (used for HTML scraping) or libraries like feedparser (for RSS), look for a timeout argument and set it to a higher value (e.g., 30 or 60 seconds).
* **Partial Articles or Missing Content:**
  + **Cause:**
    - **RSS:** The RSS feed itself might only provide a summary or truncated content, not the full article text.
    - **HTML:** The scrape\_pattern CSS selectors in config.py are incorrect or outdated, causing the scraper to fail to locate or extract the full content of an article. Websites frequently update their layouts, breaking old selectors.
  + **Solution:**
    - **For RSS:** Verify the content provided by the RSS feed directly in a browser or an RSS reader. If the feed itself is truncated, consider switching to HTML scraping for that source if full content is essential.
    - **For HTML:**
      1. **Re-inspect:** Open the problematic article page in your web browser.
      2. **Developer Tools:** Use the browser's developer tools (F12) to meticulously inspect the HTML structure of the article title, content, author, etc.
      3. **Update Selectors:** Adjust the scrape\_pattern in config.py with the correct, current CSS selectors. Pay attention to unique IDs, class names, and element hierarchies.
* **Newsletter Not Sending to Telegram:**
  + **Cause:**
    - Incorrect TELEGRAM\_BOT\_TOKEN or TELEGRAM\_CHAT\_ID in your .env file.
    - The bot has not been added to the chat/group/channel, or doesn't have necessary permissions (e.g., if it's a channel, it needs to be an administrator).
    - Network issues preventing the script from connecting to the Telegram Bot API.
    - Telegram API rate limits (less common for a single user bot).
  + **Solution:**
    - **Verify Environment Variables:** Double-check TELEGRAM\_BOT\_TOKEN and TELEGRAM\_CHAT\_ID in your .env file against the values obtained from @BotFather and @getidsbot.
    - **Bot Permissions:** If sending to a group or channel, ensure your bot has been added as a member and granted the necessary administrative permissions (e.g., "Post Messages").

**Test Connectivity:** Try a simple test using curl or a Python script to send a basic message via the Telegram API to confirm connectivity and token validity. Example curl command (replace with your token and chat ID):  
Bash  
curl -X POST \

https://api.telegram.org/botYOUR\_TELEGRAM\_BOT\_TOKEN\_HERE/sendMessage \

-d chat\_id=YOUR\_TELEGRAM\_CHAT\_ID\_HERE \

-d text="Test message from newsletter bot."

* **AI Summarization Issues (e.g., low quality, errors):**
  + **Cause:**
    - Insufficient or poorly formatted input text to the LLM.
    - Rate limits or API errors from the LLM provider.
    - Suboptimal prompt engineering for summarization.
    - The chosen LLM model is not suitable for the task.
  + **Solution:**
    - **Check Input:** Ensure the full article text extracted is clean and complete before sending it to the LLM.
    - **API Key/Billing:** Verify your OPENAI\_API\_KEY is correct and that your account has sufficient credits or is correctly billed.
    - **Prompt Engineering:** Experiment with different prompts for the LLM. A more detailed prompt can guide the LLM to produce better summaries (e.g., "Summarize this article in 3 bullet points, focusing on key Web3 developments and their implications.").
    - **Error Logging:** Implement more detailed logging around LLM API calls to catch specific error messages from the provider.

### **4.2 Optimization & Best Practices**

To ensure the efficient, reliable, and ethical operation of your newsletter generator:

* **Prioritize RSS Over HTML Scraping:**
  + **Reason:** RSS feeds are explicitly designed for content syndication, making them resource-efficient for both the scraper and the target server. They are also less likely to be blocked. HTML scraping is more fragile (due to website layout changes) and can place a higher load on the target server.
  + **Action:** Always prefer an RSS feed if available and comprehensive. Only resort to HTML scraping when no suitable RSS feed exists.
* **Set Realistic Timeouts for Network Requests:**
  + **Reason:** Prevents your script from hanging indefinitely if a server is unresponsive or slow.
  + **Action:** Implement timeouts (e.g., 15-30 seconds) for all HTTP requests made by your scraper.
* **Implement Caching (Future Enhancement):**
  + **Reason:** Reduces redundant work, saves API costs (for LLMs), and speeds up execution.
  + **Action:** For AI summaries or article content that doesn't change frequently, consider storing them in a local cache (e.g., a simple file-based cache or SQLite database). Before processing an article, check if its summary or content is already cached.
* **Monitor Source Rate Limits and Introduce Delays:**
  + **Reason:** Aggressive scraping can lead to temporary or permanent IP bans from news sources.
  + **Action:** Introduce small, random delays (time.sleep(random.uniform(min\_delay, max\_delay))) between requests to the same domain. Be mindful of the number of requests you make within a given timeframe.
* **Robust Error Handling and Logging:**
  + **Reason:** Crucial for diagnosing issues quickly and ensuring the scraper continues to run even if one source fails.
  + **Action:** Use try-except blocks around all network requests, parsing logic, and external API calls. Implement comprehensive logging to record successful operations, warnings, and detailed error messages (including tracebacks). This helps in understanding *why* something failed.
* **Run in a Persistent Environment:**
  + **Reason:** For daily automation, the scheduler.py script needs to be running continuously.
  + **Action:** Deploy the script on a cloud server (e.g., AWS EC2, Google Cloud Run, DigitalOcean Droplet) or a Raspberry Pi, and use process managers like systemd, Supervisor, or pm2 (for Node.js-based systems, but concept applies) to ensure it stays running and restarts on crashes.
* **Version Control for config.py:**
  + **Reason:** While .env should be ignored, config.py contains the logic for sources and settings. Keep it under version control.
  + **Action:** Commit changes to config.py to track modifications to your news sources and parameters.

By adhering to these best practices, you can create a more resilient, efficient, and ethical automated newsletter generator.