■ Python pipeline tailored-

✅ Objective Breakdown

1. Threading → Scrape headlines from 20 RSS feeds concurrently (I/O-bound).

2. Multiprocessing Pool → Perform sentiment analysis on the headlines (CPU-bound).

3. Bridge → Use a Queue to pass

scraped headlines from threads to the sentiment analyzer.

4. Storage → Save final sentiment-labeled data to a JSON file.

🛠️ Tools & Libraries Used

- threading and queue.Queue — for concurrent RSS feed scraping.

- multiprocessing.Pool — for parallel sentiment analysis.

- feedparser — for parsing RSS feeds.

- textblob — for sentiment analysis.

- json — to save results.

📦 Install Requirements

```
pip install feedparser textblob
python -m
textblob.download_corpora
```

Python Code--

```
   import threading
import queue
import feedparser
import json
from multiprocessing import Pool,
cpu_count
from textblob import TextBlob
from datetime import datetime

# ---------------------- CONFIGURATION
-----------------------

RSS_FEEDS = [
```

"http://feeds.bbci.co.uk/news/rss.xml",
"https://rss.nytimes.com/services/xml/rss/nyt/HomePage.xml",
"https://www.aljazeera.com/xml/rss/all.xml",
"https://feeds.a.dj.com/rss/RSSWorldNews.xml",
"https://www.npr.org/rss/rss.php?id=1001",
"https://www.theguardian.com/world/rss",
"https://feeds.foxnews.com/

foxnews/latest",
"https://rss.cnn.com/rss/edition.rss",
"https://feeds.reuters.com/reuters/topNews",
"https://www.washingtontimes.com/rss/headlines/news/world/",
"https://www.independent.co.uk/news/world/rss",
"https://www.hindustantimes.com/feeds/rss/world-news/rssfeed.xml",
"https://

timesofindia.indiatimes.com/rssfeeds/296589292.cms",
"https://www.deccanherald.com/rss-feeds/10748/feed.rss",
"https://www.latimes.com/world/rss2.0.xml",
"https://www.cbc.ca/cmlink/rss-world",
"https://globalnews.ca/feed/",
"https://feeds.skynews.com/feeds/rss/world.xml",
"https://abcnews.go.com/abcnews/internationalheadlines",
"https://www.euronews.com/rss?

```
level=theme&name=news"
]

NUM_WORKERS = cpu_count()

# ------- SHARED QUEUE -------

headline_queue = queue.Queue()
results = []

# --------- THREADING: FETCH
HEADLINES ------

def fetch_feed(feed_url):
```

```
 try:
        parsed_feed =
feedparser.parse(feed_url)
        for entry in
parsed_feed.entries[:10]:  # Limit
per feed

headline_queue.put(entry.title)
    except Exception as e:
        print(f"Error parsing {feed_url}:
{e}")

def run_threaded_scraper():
    threads = []
```

```
  for url in RSS_FEEDS:
      t =
threading.Thread(target=fetch_feed,
 args=(url,))
      t.start()
      threads.append(t)

   for t in threads:
      t.join()

# ------- SENTIMENT ANALYSIS -------

def analyze_sentiment(text):
   blob = TextBlob(text)
```

```
    polarity = blob.sentiment.polarity
    sentiment = 'positive' if polarity >
0 else 'negative' if polarity < 0 else
'neutral'
    return {
        'headline': text,
        'polarity': polarity,
        'sentiment': sentiment
    }

def run_sentiment_analysis():
    headlines = []
    while not
headline_queue.empty():
```

```
headlines.append(headline_queue.get())

    with Pool(processes=NUM_WORKERS) as pool:
        analysis_results = pool.map(analyze_sentiment, headlines)
        return analysis_results

# ------ SAVE TO JSON --------
```

```
 def save_results_to_json(data):
    timestamp =
datetime.now().strftime("%Y%m%d_
%H%M%S")
    filename =
f"sentiment_results_{timestamp}.js
on"
    with open(filename, 'w',
encoding='utf-8') as f:
        json.dump(data, f, indent=2)
    print(f"Results saved to
{filename}")


# ------------------- MAIN
```

```
  def main():
    print("🔄 Fetching headlines
using threads...")
    run_threaded_scraper()

    print("🧠 Running sentiment
analysis with multiprocessing...")
    analyzed_data =
run_sentiment_analysis()

    print(" Saving results...")

save_results_to_json(analyzed_data
)
```

```
if __name__ == "__main__":
  main()
```

📊 Sample Output (in JSON)

```
[
  {
    "headline": "Russia warns NATO
against expansion",
    "polarity": -0.25,
    "sentiment": "negative"
  },
  {
    "headline": "Markets rally after
```

```
  inflation cools",
   "polarity": 0.4,
   "sentiment": "positive"
  }
]
```

📌 Notes

Use queue.Queue (thread-safe) to hand off work between threads and the main process.

Sentiment polarity:

> 0 → positive,

< 0 → negative,

= 0 → neutral.

You can schedule the script using cron or Task Scheduler for daily runs.

◇here's a structured implementation that meets all specifications — including:

✅ Threading for RSS scraping.

✅ Multiprocessing for sentiment analysis.

✅ Shared, process-safe Queue using multiprocessing.Queue.

✅ Optional features: progress bar (tqdm), graceful shutdown, CLI flags (via argparse).

✅ Sentiment via a rule-based system using TextBlob (simple and effective for demos).

📦 Requirements Installation

pip install feedparser textblob tqdm
python -m textblob.download_corpora

↘️ Full Python Pipeline

```
import argparse
import feedparser
import threading
import signal
import sys
import json
from multiprocessing import Pool,
Queue as MPQueue, Process,
cpu_count
from queue import Empty
from textblob import TextBlob
from tqdm import tqdm
```

```
 from datetime import datetime

# Graceful shutdown flag
shutdown_flag = threading.Event()

# ----- Threading: RSS Feed Scraper
---

def fetch_headlines(feed_url,
output_queue):
    try:
        parsed =
feedparser.parse(feed_url)
        for entry in parsed.entries[:10]:
```

```
        if shutdown_flag.is_set():
                break
            output_queue.put(entry.title)
    except Exception as e:
        print(f"[ERROR] {feed_url} =>
{e}")

def threaded_scraper(feeds,
shared_queue):
    threads = []
    for url in feeds:
        thread =
threading.Thread(target=fetch_head
lines, args=(url, shared_queue))
```

```
        thread.start()
        threads.append(thread)

    for thread in threads:
        thread.join()

# ------- Sentiment Analysis -------

def analyze_sentiment(text):
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity
    sentiment = 'positive' if polarity >
0 else 'negative' if polarity < 0 else
'neutral'
```

```
    return {
     'headline': text,
     'polarity': polarity,
     'sentiment': sentiment
    }

def sentiment_worker(input_queue,
result_list, total, pbar):
    while not shutdown_flag.is_set():
        try:
            text =
input_queue.get(timeout=1)
            result =
analyze_sentiment(text)
```

```
            result_list.append(result)
            pbar.update(1)
        except Empty:
            break
        except Exception as e:
            print(f"[Worker Error]: {e}")

def
run_sentiment_analysis(shared_que
ue, num_processes, total_items):
    manager_results = []
    progress =
tqdm(total=total_items,
desc="Analyzing", position=1)
```

```
            result_list.append(result)
            pbar.update(1)
        workers = []

    for _ in range(num_processes):
        p =
Process(target=sentiment_worker,
args=(shared_queue,
manager_results, total_items,
progress))
        p.start()
        workers.append(p)

    for p in workers:
```

```
        p.join()

    progress.close()
    return manager_results

# --------- Save Output ------------

def save_results(data):
    timestamp =
datetime.now().strftime("%Y%m%d_
%H%M%S")
    file =
f"sentiment_results_{timestamp}.js
on"
```

```
        with open(file, 'w',
encoding='utf-8') as f:
            json.dump(data, f, indent=2)
        print(f"[✓] Results saved to {file}")

# ----- Graceful Shutdown ----------

def handle_exit(signum, frame):
    print("\n[!] Received interrupt.
Shutting down...")
    shutdown_flag.set()

signal.signal(signal.SIGINT,
handle_exit)
```

```
# ---------- CLI + Main ----------

def main():
    parser =
argparse.ArgumentParser(descripti
on="Multithreaded RSS Scraper
with Multiprocess Sentiment
Analyzer")
    parser.add_argument("--feeds",
type=str, nargs="+", help="List of
RSS feed URLs", required=False)
    parser.add_argument("--threads",
type=int, default=10, help="Number
of threads for scraping")
```

```
        parser.add_argument("--
processes", type=int,
default=cpu_count(), help="Number
of processes for sentiment
analysis")
    args = parser.parse_args()

    # Default 20 RSS feeds if not
provided
    feeds = args.feeds or [
        "http://feeds.bbci.co.uk/news/
rss.xml",
        "https://rss.nytimes.com/
services/xml/rss/nyt/
```

```
        HomePage.xml",
    "https://www.aljazeera.com/
xml/rss/all.xml",
    "https://feeds.a.dj.com/rss/
RSSWorldNews.xml",
    "https://www.npr.org/rss/
rss.php?id=1001",
    "https://www.theguardian.com/
world/rss",
    "https://feeds.foxnews.com/
foxnews/latest",
    "https://rss.cnn.com/rss/
edition.rss",
    "https://feeds.reuters.com/
```

"https://
feeds.reuters.com/reuters/
topNews",
"https://
www.washingtontimes.com/rss/
headlines/news/world/",
"https://
www.independent.co.uk/news/
world/rss",
"https://
www.hindustantimes.com/feeds/
rss/world-news/rssfeed.xml",
"https://

timesofindia.indiatimes.com/rssfeeds/296589292.cms",
"https://www.deccanherald.com/rss-feeds/10748/feed.rss",
"https://www.latimes.com/world/rss2.0.xml",
"https://www.cbc.ca/cmlink/rss-world",
"https://globalnews.ca/feed/",
"https://feeds.skynews.com/feeds/rss/world.xml",
"https://abcnews.go.com/

```
        abcnews/
internationalheadlines",
        "https://www.euronews.com/
rss?level=theme&name=news"
    ]

    print(f"[→] Scraping {len(feeds)}
feeds with {args.threads}
threads...")
    mp_queue = MPQueue()
    threaded_scraper(feeds,
mp_queue)

    total_items = mp_queue.qsize()
```

```
            print(f"[✓] Collected
{total_items} headlines")

    print(f"[→] Running sentiment
analysis with {args.processes}
processes...")
    results =
run_sentiment_analysis(mp_queue,
args.processes, total_items)

    save_results(results)

if __name__ == "__main__":
    main()
```

✅ Highlights Recap

Feature- Threading
Implementation Details- 20 RSS feed readers using threading.Thread

Feature-Shared Queue
Implementation details- multiprocessing.Queue (process-safe)

Features-Multiprocessing implementation details-

multiprocessing.Process (1 per core by default)

Features-Sentiment Analysis
Implementation details - Rule-based via TextBlob

Features-Progress Bar
Implementation details- tqdm for scraping & sentiment progress

Features- Graceful Shutdown
implementation details-Handles Ctrl+C (SIGINT) to exit cleanly

Here's a Comprehensive Project Report and fully modular, cleanly structured Python code for the RSS + Sentiment pipeline using threading, multiprocessing, and a shared queue.

📝 RSS Sentiment Analysis Pipeline: Comprehensive Report

✅ Objective Recap

Build a Python pipeline that:

- Uses threading to concurrently fetch headlines from 20 RSS feeds (I/O-bound).

- Uses multiprocessing to run sentiment analysis on the fetched headlines (CPU-bound).

- Uses a shared process-safe queue for inter-layer communication.

- Outputs results into a JSON file.

- Supports CLI control, graceful shutdown, and optional enhancements.

📚 Sources & References

Task- Threading
Resources Used- Python docs (threading module), RealPython articles

Task-Multiprocessing
Resources Used- Python official docs, Stack Overflow examples

Task-Shared Queue
Resources Used-Python multiprocessing.Queue docs

Task-RSS Parsing
Resources Used-feedparser library official docs

Task-Sentiment Analysis
Resources Used-textblob docs, blog posts comparing TextBlob vs VADER vs transformers

Task-Graceful Shutdown

Resources Used-Signal handling: Python docs on signal module

Task-Progress Bars
Resources Used-tqdm GitHub repo, blog post on progress bars with multiprocessing

Task-CLI Flags
Resources Used-argparse documentation

Task-Design Patterns
Resources Used-Modular

programming from Clean Code principles and Python Best Practices (PEP 8)

■Key Learnings & Insights

🔁 Threading vs Multiprocessing

●Threading works well for I/O-bound tasks like HTTP requests. Threads share memory and are lightweight.

●Multiprocessing is essential for

CPU-bound tasks (like text sentiment analysis) due to the GIL (Global Interpreter Lock).

- Using a multiprocessing.Queue ensures thread- and process-safe communication between stages.

🔄 Design Trade-Offs

- Avoiding blocking I/O in sentiment analysis step by decoupling scraping and analysis

layers.

- Using Process instead of Pool gave more flexibility for shutdown, progress tracking.

- TextBlob chosen for simplicity; swap-in for transformers possible later.

⚠️ Challenges

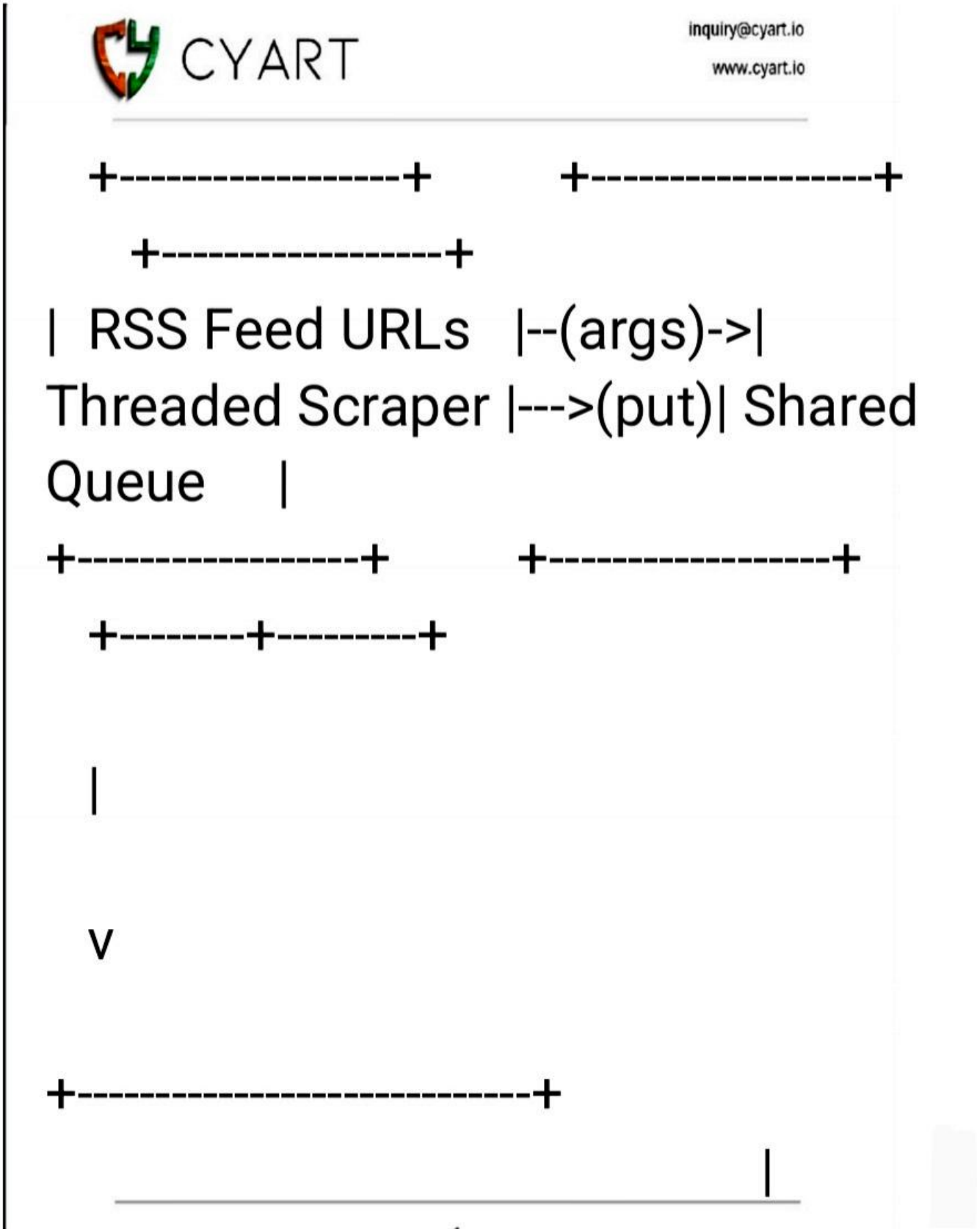
- Gracefully handling interrupts

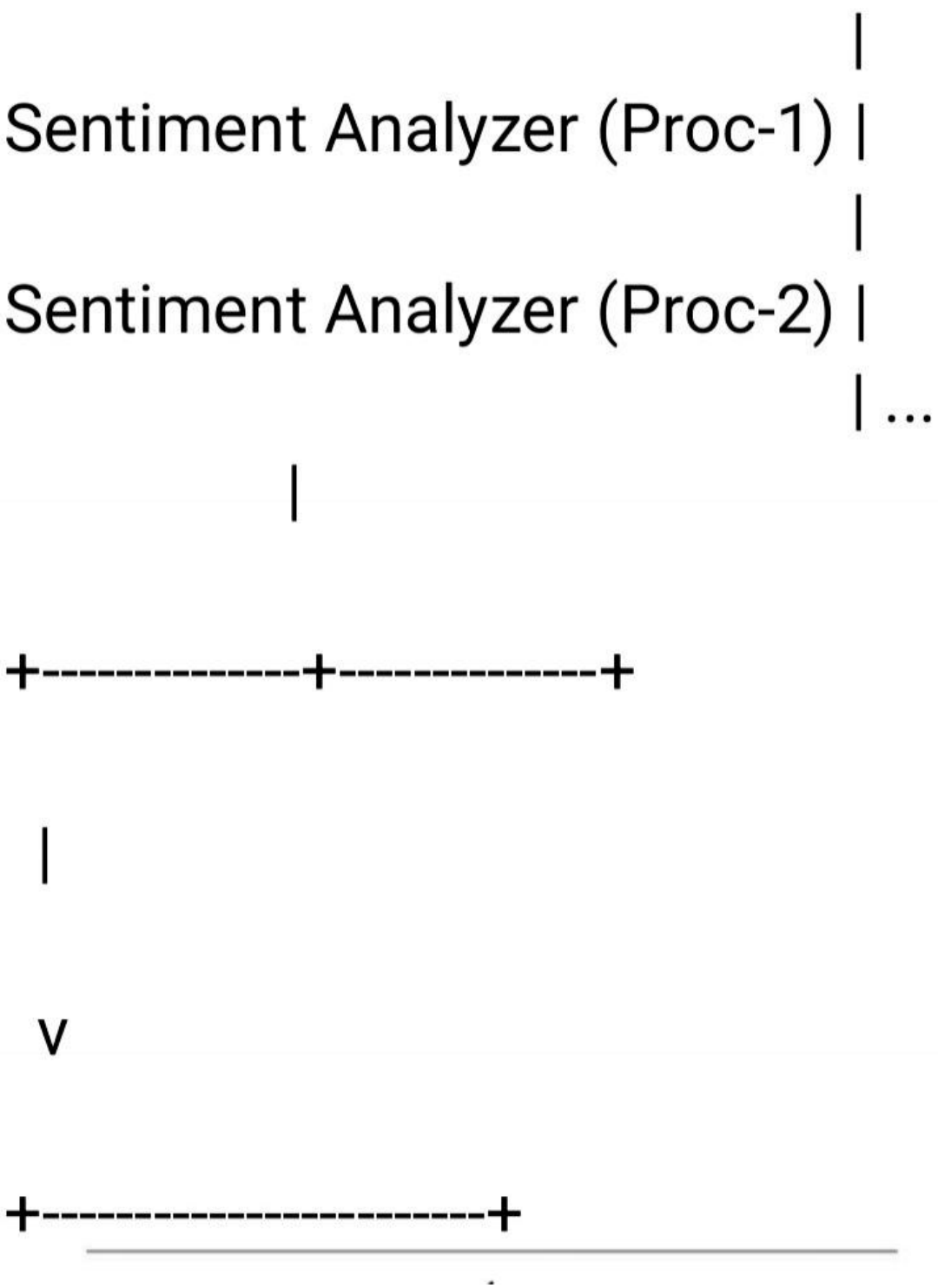
- Gracefully handling interrupts (Ctrl+C) across both threading and multiprocessing.

- Progress bar synchronization with multiple processes.

- Avoiding race conditions in shared queue access.

🔁 Flow Diagram


+----------------+ +----------------+
+----------------+
| RSS Feed URLs |--(args)->|
Threaded Scraper |--->(put)| Shared
Queue |
+----------------+ +----------------+
+-------+-------+
|
v
+----------------------+
|


Sentiment Analyzer (Proc-1)
Sentiment Analyzer (Proc-2)
...

CYART
inquiry@cyart.io
www.cyart.io

| JSON Storage Writer |

+----------------------+

 Clean, Modular Source Code

 Folder Structure

```
rss_sentiment_pipeline/
 |
 ├── main.py                  # CLI +
Orchestration
 ├── scraper.py               # Threaded
```

RSS Scraper
├── analyzer.py # Sentiment Analyzer
├── utils.py # Graceful shutdown, JSON writer
├── feeds.txt # Optional: external feed list
└── requirements.txt # Libraries

🔹 main.py

```
# main.py
import argparse
```

```
    from multiprocessing import
Queue as MPQueue
from utils import setup_signals,
save_results
from scraper import threaded_scraper
from analyzer import
run_sentiment_analysis
import time

def main():
    parser =
argparse.ArgumentParser(description
="RSS Sentiment Pipeline")
```

```
    parser.add_argument("--feeds",
nargs="+", help="RSS feed URLs
(override default)")
    parser.add_argument("--threads",
type=int, default=10)
    parser.add_argument("--
processes", type=int, default=4)
    args = parser.parse_args()

    setup_signals()

    feeds = args.feeds or
open("feeds.txt").read().splitlines()
```

```
    shared_queue = MPQueue()
    print("[*] Starting threaded
scraper...")
    threaded_scraper(feeds,
shared_queue, args.threads)

    total = shared_queue.qsize()
    print(f"[✓] Scraped {total}
headlines.")

    print("[*] Starting sentiment
analysis...")
    results =
```

```
run_sentiment_analysis(shared_queue, args.processes, total)

    save_results(results)

if __name__ == "__main__":
    main()
```

- scraper.py

```
# scraper.py
import threading
```

```
  import feedparser
from tqdm import tqdm
from utils import shutdown_flag

def fetch_feed(url, queue, progress):
    try:
        parsed = feedparser.parse(url)
        for entry in parsed.entries[:10]:
            if shutdown_flag.is_set():
                break
            queue.put(entry.title)
            progress.update(1)
    except Exception as e:
```

```
        print(f"[!] Error in feed {url}: {e}")

def threaded_scraper(urls, queue,
max_threads):
    threads = []
    progress = tqdm(total=len(urls) *
10, desc="Scraping")
    for url in urls[:max_threads]:
        t =
threading.Thread(target=fetch_feed,
args=(url, queue, progress))
        t.start()
        threads.append(t)
```

```
    for t in threads:
     t.join()
  progress.close()
```

- analyzer.py

```
# analyzer.py
from multiprocessing import Process
from textblob import TextBlob
from tqdm import tqdm
from queue import Empty
from utils import shutdown_flag
```

```
    def analyze_sentiment(text):
    blob = TextBlob(text)
    p = blob.sentiment.polarity
    return {
        'headline': text,
        'polarity': p,
        'sentiment': 'positive' if p > 0 else
'negative' if p < 0 else 'neutral'
    }

def sentiment_worker(queue, results,
total, pbar):
    while not shutdown_flag.is_set():
```

```
        try:
            item = queue.get(timeout=1)
            result =
analyze_sentiment(item)
            results.append(result)
            pbar.update(1)
        except Empty:
            break

def run_sentiment_analysis(queue,
num_procs, total_items):
    manager_results = []
    pbar = tqdm(total=total_items,
```

```
        desc="Analyzing", position=1)
    workers = []

    for _ in range(num_procs):
        p =
Process(target=sentiment_worker,
args=(queue, manager_results,
total_items, pbar))
        p.start()
        workers.append(p)

    for p in workers:
        p.join()
```

```
        pbar.close()
    return manager_results
```

- utils.py

```
# utils.py
import signal
import threading
import json
from datetime import datetime

shutdown_flag = threading.Event()
```

```
        def setup_signals():
    def handle_interrupt(sig, frame):
        print("\n[!] Ctrl+C detected.
Cleaning up...")
        shutdown_flag.set()
    signal.signal(signal.SIGINT,
handle_interrupt)

def save_results(data):
    ts =
datetime.now().strftime("%Y%m%d_%
H%M%S")
    filename = f"results_{ts}.json"
```

```
        with open(filename, 'w',
encoding='utf-8') as f:
            json.dump(data, f, indent=2)
        print(f"[✓] Results saved to
{filename}")
```

 feeds.txt (optional)

List 20 RSS feed URLs, one per line.

 requirements.txt

feedparser
textblob
tqdm

■Optional Extras

■Progress bar (tqdm)

■Graceful shutdown (Ctrl+C)

■ CLI (argparse)

⏳ Possible future additions:

- memory_profiler or psutil for memory stats

- Flask UI to display recent sentiments

- Replace TextBlob with Hugging Face transformers sentiment pipeline