ChatGPT

Scaling HVAC Manual Scraping and Processing

Scalable Data Collection Strategy for Multiple Manufacturers

To expand beyond Carrier and gather furnace manuals from other HVAC manufacturers, design a **modular web-scraping pipeline**. Leverage lessons from scraping Carrier's site and apply a similar process to each target brand's website:

- **Identify Accessible Manual Sources:** For each manufacturer, find how they publish product literature. Many have *public literature libraries* or product pages linking to PDF manuals. For example, Goodman provides a literature library organized by product category (with direct PDF links) [1] [2], which can be parsed easily. In contrast, Carrier's site requires searching by model number on a literature page [3] – you might gather model numbers by crawling product listings (e.g. all furnace model pages) and then query the literature search for each model. Other brands like Trane offer a public e-Library search with filters for installation manuals [4] [5]. Tailor your approach to each site's structure (e.g. by writing site-specific scraping functions or spiders).

- **Crawling and Filtering:** Use a crawler to traverse each site's relevant sections. Focus on pages likely to contain manual links (such as *product detail pages* or dedicated "Resources/Literature" sections). Parse HTML for anchor tags ending in ".pdf" and for keywords like "manual" or "installation" to filter relevant docs. This ensures you collect *only public PDFs* related to HVAC furnaces. If an official site lacks a convenient index, consider crawling the entire domain (with a depth limit and rate limit) and scraping all PDF links, then later filter out irrelevant files by name or content. This broad crawl approach should be a fallback – whenever possible, use the site's own indexing (product catalogs or search pages) for a more targeted and efficient scrape.

- **Modularity for New Brands:** Keep the scraping logic modular so you can add new manufacturers easily. For instance, define a configuration for each brand containing the base URL, any sitemap or known index page, and any unique parsing rules (HTML patterns or form parameters). Then your `scrape_pdfs.py` script can iterate through configured manufacturers. This way, adding "the others" (e.g. Trane, Lennox, Goodman, Rheem, York, etc.) is as simple as plugging in a new config and parser without rewriting the whole pipeline. Aim to scrape **50+ manuals** across brands for a robust knowledge base, as outlined in your project's goals [6] [7].

- **Concurrency and Rate Limiting:** To scale up crawling without overloading servers, introduce controlled concurrency. Since your stack is Python-based, you can use asynchronous requests (via `aiohttp` or `httpx`) or a crawling framework to fetch multiple pages in parallel. If using PySpark, you could distribute download tasks across executors, but a web-focused tool might be simpler. **Rate limiting** is essential – insert short delays or limits on requests per domain [8] to avoid being flagged as a bot. Many scraping frameworks allow setting a crawl rate; otherwise, manually ensure your code pauses appropriately (e.g. a few seconds between page requests). Concurrent crawling across different domains is fine (scraping Carrier and Goodman in parallel, for example), but serialize or throttle requests *per host* to stay polite. Logging and error handling (with retries for transient failures) will make the scraper more robust [8] [9].

- **Incremental Updates:** For long-term scalability, design the pipeline to handle updates. After initial scraping, periodically check for new or updated manuals without re-downloading everything. You can maintain a manifest of collected PDFs (file names or hashes) and have the scraper skip those on subsequent runs. Some sites provide publication dates in their literature listings (e.g. Trane's eLibrary shows dates [10]); you could use that to fetch only newer docs in future. This ensures your knowledge base stays up-to-date with minimal overhead.

Legal and Ethical Considerations

Scraping manufacturer manuals is generally feasible as long as you **access only publicly available documents** and abide by ethical guidelines:

- **Robots.txt and Site Policies:** Always check each site's `robots.txt` file to see what is permitted. The robots.txt is a file where websites communicate crawling rules to bots. For example, Carrier's `robots.txt` disallows certain old site sections but not the current residential docs section [11]. While the Robots Exclusion Protocol isn't a law, *ignoring it can lead to your scraper being detected and even legal complications* [12]. Use Python's built-in `urllib.robotparser` or a library to programmatically read robots.txt and confirm that your target paths (e.g. "/residential/en/us/products/...") aren't disallowed. If a site explicitly forbids scraping its manuals, respect that – limit your project to brands that make manuals publicly accessible.

- **Terms of Service:** In addition to robots.txt, review the website's Terms of Use if available [13]. Some companies allow personal or research use of documentation but prohibit bulk downloading or commercial use. Staying within **public access only** is wise – don't attempt to scrape content behind logins or paywalls (e.g. certain pro portals for contractors). If a manufacturer only provides manuals through authorized dealer logins, that's effectively not public, so skip those to avoid legal risk. There are often alternative public sources for such manuals (for instance, third-party distributor sites or aggregators), but use official public sources whenever possible to ensure data accuracy and legality.

- **Politeness and Load:** Scraping should be gentle. Besides rate limiting, identify yourself with a sensible User-Agent string and potentially contact the site owner if you plan heavy use. Given the limited scope (technical manuals), the traffic from your project should be small, but it's good practice to avoid any action that could be seen as affecting the website's performance. Also, store the downloaded PDFs locally (as your pipeline does) and serve queries from your own database – this way, after the one-time scrape, your assistant isn't repeatedly hitting manufacturer servers.

- **Attribution and Non-Commercial Use:** Since you are building a local assistant, you're not republishing the manuals, just enabling QA on them. This falls under fair use in many cases, but to be safe, keep the content usage internal. If you eventually share this tool, clarify that it uses manufacturer documentation and ensure it's not used in a way that violates the documents' usage rights. As long as you're just helping end-users find info (which they could manually get from the PDFs), and not selling the content, the risk is minimal. Still, it's good to have a note in your README about the source of the manuals and perhaps a disclaimer that those PDFs are © by their respective companies.

Open-Source Tools and Library Recommendations

You've chosen a **100% open-source stack** for this project, which avoids any paid API dependencies [14] [15]. Here are some high-level tool recommendations to enhance your pipeline while keeping it cost-free:

- **Web Scraping Frameworks:** For a scalable crawler, consider using **Scrapy**. Scrapy is a powerful open-source Python framework that has become the *de facto* standard for large-scale web scraping [16]. It provides an efficient engine for sending asynchronous requests, parsing HTML, and handling pipelines (e.g. saving files). Scrapy will automatically respect robots.txt by default and has built-in throttling and retry mechanisms, which could save you time coding those features. If your scraping needs grow (hundreds of pages or more), Scrapy's robust architecture and scheduling can handle it [16] [17]. On the other hand, for small-scale or one-off scraping tasks, your current approach using **Requests** + **BeautifulSoup4** is perfectly fine [18]. BeautifulSoup is easy to use for parsing HTML DOM and extracting links or text. In fact, for each manufacturer's site, you might start with requests/BS4 to prototype the extraction logic. If you find yourself writing a lot of repetitive code or needing more speed, that's when switching to Scrapy could help (you can integrate your parsing code into a Scrapy spider with minimal changes).

- **Handling JavaScript-heavy Sites:** Most HVAC documentation pages are static HTML or have predictable URLs for PDFs, so you likely *don't need a browser automation tool*. But if you encounter a site where manuals are only accessible via a complex JavaScript application (for example, a search that renders results client-side), you might use **Selenium** or **Playwright** in headless mode to simulate a user. Both are open source (Selenium WebDriver and Microsoft's Playwright Python library) and allow you to fill out forms or click buttons and then scrape the resulting page. Keep in mind, these tools are heavier – they control a browser – so use them only if absolutely necessary for a particular site. Often, there's a way to find the underlying request (via network calls) and use Requests instead, avoiding the need for a headless browser. But it's good to know these are available if needed for tricky websites.

- **Downloading and Parsing PDFs:** Continue using **python-requests** to download PDF files. It's efficient and straightforward – just be sure to stream the response and write to file, to handle large PDFs without using too much memory. For parsing the text, you already use **pdfplumber**, which is a solid choice for PDF text extraction in Python. Another library to keep in mind is **PyMuPDF** (also known as `fitz`). PyMuPDF is known for its high performance in parsing PDFs. In benchmark comparisons, PyMuPDF tends to be faster at extracting text than pdfplumber or other pure-Python libraries [19]. If you end up processing hundreds of PDFs or very large documents, switching to PyMuPDF could speed up text extraction significantly. Both pdfplumber and PyMuPDF are open source – you won't need any paid API for PDF processing. For now, if pdfplumber is working and performance is acceptable, it's fine to stick with it (it has nice features for table extraction and such). Just note PyMuPDF as an option if you encounter performance bottlenecks, since it "processes PDFs swiftly, making it the fastest option for data extraction" [19] according to recent analyses.

- **Data Storage and Querying:** Your plan to use local embeddings via **SentenceTransformers** and a vector DB like **FAISS or ChromaDB** is great for an open-source solution [20] [21]. These libraries have no usage cost. To keep things fully local, you might also consider using an open-source LLM for the question-answering step (so you don't have to call an API like OpenAI). Models like Llama 2 or smaller instruct models can be run with libraries like HuggingFace Transformers or LangChain's local pipeline. That said, the heavy lifting of getting the right info is done by your RAG setup, so even a smaller local model should suffice to generate answers from the retrieved text. This ensures your "NLP assistant for HVAC" truly runs without proprietary services.

- **Workflow and Containerization:** Since you've containerized with Docker, ensure all these tools and their dependencies are included in your image/compose. For example, if you use Scrapy, you'll need to add it to `requirements.txt`. Likewise, PyMuPDF if you add it, and possibly system packages (MuPDF C library). Keep using **Docker Compose** to manage services (though most of your pipeline is offline, so one container running the app is fine). The Docker approach will help with scalability too – you could deploy the container on a stronger machine or cloud VM with no code changes if you need to speed up processing or handle more data.

In summary, a **high-level plan** is: identify the public documentation sources for each manufacturer, use open-source scraping tools (starting with requests/BS4, and scaling up to Scrapy if needed) to collect PDFs, then parse and index them with open libraries (pdfplumber/PyMuPDF for text, SentenceTransformers for embeddings, FAISS/Chroma for vectors). By respecting websites' rules and staying within public data, you avoid legal issues while building a powerful, local HVAC Q&A system. All of this can be accomplished with free, open-source components – no API fees required – so you can continue developing your HVAC assistant without hitting a paywall. Good luck applying the Carrier experience to the rest of the industry's data!

Sources: Your project README and development notes, plus relevant references on web scraping and PDF parsing best practices:

- Project README – *HVAC Assistant* (JohnYanez95): Technology stack and features [15] [18].
- Project TODO – Development roadmap highlights goals like robust PDF scraping, rate limiting, and 50+ manuals in knowledge base [8] [6].
- ZenRows Blog – *How to Read robots.txt for Web Scraping*: Importance of respecting robots.txt (avoid detection and legal issues) [12].
- Firecrawl Dev Blog – *Scrapy for Large-Scale Crawling*: Scrapy is a proven open-source framework for efficient, scalable scraping in Python [16] [22].
- Medium Article – *PDF Extraction Libraries Comparison*: PyMuPDF offers top performance for PDF text extraction, outperforming other libraries in speed [19].

---

[1] [2] Literature Library | Product Specification | Goodman
https://www.goodmanmfg.com/support/literature-library

[3] Product Literature | Carrier Residential
https://www.carrier.com/residential/en/us/homeowner-resources/product-literature/

[4] [5] [10] Search Results: - Public
https://elibrary.tranetechnologies.com/public/search?term=

[6] [7] [8] [9] raw.githubusercontent.com
https://raw.githubusercontent.com/JohnYanez95/hvac-assistant/main/TODO.md

[11] www.carrier.com
https://www.carrier.com/robots.txt

[12] How to Read robots.txt for Web Scraping - ZenRows
https://www.zenrows.com/blog/robots-txt-web-scraping

[13] Need Help with Finding an Owner’s Manual | Carrier
https://www.carrier.com/residential/en/us/contact-us/help-with-finding-owners-manual/

[14] [15] [18] [20] [21] raw.githubusercontent.com
https://raw.githubusercontent.com/JohnYanez95/hvac-assistant/main/README.md

[16] [17] [22] Best Open-source Web Scraping Libraries in 2025
https://www.firecrawl.dev/blog/best-open-source-web-scraping-libraries

[19] A Comparative Analysis of PDF Extraction Libraries: Choosing the Fastest Solution | by abhiyan timilsina | Medium
https://abhiyantimilsina.medium.com/a-comparative-analysis-of-pdf-extraction-libraries-choosing-the-fastest-solution-3b6bd8588498