A Cost-Effective, End-to-End Pipeline for Dissertation-Scale Knowledge Graph Construction from Web Corpora

Introduction

The proliferation of unstructured data on the World Wide Web presents both a significant challenge and a profound opportunity for knowledge discovery. For academic researchers, particularly those undertaking dissertation projects, the ability to systematically distill this vast repository of text into structured, machine-readable formats is paramount. Knowledge Graphs (KGs) have emerged as a powerful paradigm for representing entities and their interconnections, transforming raw text into a network of semantically rich information. However, the methodologies for constructing such graphs have historically been resource-intensive, often requiring substantial computational power, costly proprietary software, or large, manually annotated datasets, placing them beyond the reach of many student-led projects.

The recent advent of powerful, accessible Large Language Models (LLMs) has fundamentally altered this landscape. These models exhibit a remarkable capacity for zero-shot and few-shot information extraction, enabling the conversion of unstructured text into structured triples (subject, predicate, object) with minimal task-specific training. This capability opens a new frontier for building KGs in a cost-effective and scalable manner.

This report presents a comprehensive, cost-effective, and methodologically robust pipeline for constructing a Knowledge Graph from a curated corpus of 100-200 diverse web pages, designed specifically to meet the constraints of a dissertation project with a budget under ₹8000 INR (~$100 USD). The proposed end-to-end methodology prioritizes the use of free, open-source, and low-cost tools and services without compromising on academic rigor or technical sophistication.

The analysis is structured into five principal parts, each addressing a critical stage of the KG construction workflow:

1. **Web Content Acquisition and Preprocessing:** Detailing best practices for reliably scraping a heterogeneous collection of modern websites and filtering noise to isolate core textual content.
2. **LLM-Powered Information Extraction:** A deep dive into the core task of using LLMs for high-fidelity entity and relationship extraction, focusing on advanced prompt engineering and schema enforcement techniques.
3. **Cost-Effective LLM Deployment and Access:** A practical guide to accessing state-of-the-art LLMs for free or at near-zero cost, comparing local execution on consumer hardware with generous free API tiers.
4. **Knowledge Graph Storage and Management:** An analysis of storage solutions, comparing lightweight, in-memory libraries for prototyping against persistent graph databases for final analysis and visualization.
5. **End-to-End Pipeline Orchestration:** A blueprint for integrating all preceding components into a single, automated, and executable workflow using modern orchestration frameworks.

By providing detailed technical guidance, supported by code examples and a thorough analysis of trade-offs, this report serves as both a practical implementation manual and a methodologically sound foundation for a dissertation project in the field of automated knowledge engineering.

Part I: Web Content Acquisition and Preprocessing

The foundation of any data-driven project is the quality of its input data. In the context of building a Knowledge Graph from web pages, this initial phase of content acquisition and preprocessing is of paramount importance. The primary objective is to reliably and accurately extract the main textual content from a diverse set of 100-200 websites, which may feature a combination of static and dynamic content-loading mechanisms. This section outlines a strategic approach to scraping, content extraction, and noise filtration that prioritizes robustness and data integrity.

1.1 Strategic Scraper Selection for a Heterogeneous Corpus

The modern web is not a monolith; a corpus of 100-200 URLs will invariably include simple static pages, such as blog posts and news articles, alongside complex, dynamic pages that rely heavily on JavaScript to render their primary content. This heterogeneity necessitates a nuanced scraping strategy that balances speed with the ability to handle dynamic content.

Comparative Analysis of Scraping Technologies

Two primary technological stacks dominate the web scraping landscape: lightweight HTTP request libraries paired with HTML parsers, and full-featured browser automation frameworks.

- **requests + BeautifulSoup:** This combination represents the classic, lightweight approach to web scraping. The requests library sends an HTTP GET request to a URL and retrieves the raw HTML source code as it is initially served by the server. The BeautifulSoup library then parses this HTML, creating a navigable tree structure that allows for efficient extraction of specific elements.[1] The principal advantage of this method is its exceptional speed and low resource overhead. As it does not involve rendering a full web page in a browser, it consumes minimal CPU and memory, making it highly efficient for scraping static websites.[2] Benchmarks have shown that this approach can be significantly faster—in some cases, around 70% faster—than browser automation tools.[2] However, its critical limitation is its inability to handle content that is loaded dynamically via JavaScript after the initial page load.
- **Selenium / Playwright:** These tools are not mere HTTP clients; they are comprehensive browser automation frameworks. They programmatically control a real web browser (like Chrome or Firefox) to navigate to a page, execute JavaScript, and interact with page elements just as a human user would.[1] This makes them indispensable for scraping modern, dynamic websites where content is rendered client-side. They can handle tasks like clicking "load more" buttons, scrolling to trigger content loading, and waiting for AJAX requests to complete.[1] This power, however, comes with significant trade-offs. The process is inherently slower and more resource-intensive due to the overhead of running a full browser instance. Furthermore, the setup is more complex, requiring the installation and management of a corresponding WebDriver executable.[1]

A Hybrid, Fallback-Driven Decision Framework

For a dissertation project, methodological soundness and data completeness are more critical than marginal gains in processing speed. A scraping script that fails on a significant portion of the target URLs introduces bias and undermines the validity of the resulting Knowledge Graph. The absolute time difference between scraping 200 pages with requests (a few minutes) versus Selenium (perhaps 10-15 minutes) is trivial in the context of a multi-month research project. The true cost is not measured in CPU cycles but in failed extractions, where a URL yields no data.

Therefore, the most efficient and robust strategy is not to choose one tool over the other, but to implement a hybrid, fallback-driven pipeline. This approach optimizes for the common case (fast static scraping) while ensuring robustness for the difficult case (dynamic content). The recommended workflow is as follows:

1. **Primary Attempt:** For each URL, the pipeline first attempts to fetch and parse the content using the requests + BeautifulSoup combination.
2. **Content Validation:** After the initial scrape, a simple validation check is performed on the extracted text. For instance, the pipeline can check if the word count of the extracted content exceeds a minimum threshold (e.g., 50 words).
3. **Fallback to Browser Automation:** If the initial attempt fails (e.g., due to an HTTP error, a timeout) or if the content validation check does not pass (indicating a likely dynamic page where the main content was not in the initial HTML), the pipeline automatically triggers a fallback mechanism.
4. **Secondary Attempt:** In the fallback, the same URL is processed using Selenium or Playwright. The browser automation tool loads the page, waits for a reasonable period to allow JavaScript to execute, and then extracts the fully rendered HTML.

This hybrid strategy ensures that the pipeline benefits from the speed of requests for the majority of simpler web pages while guaranteeing that content from complex, JavaScript-heavy sites is also captured reliably. This maximizes data acquisition success, which is the most critical metric for this phase of the project.

1.2 Advanced Content Extraction and Noise Filtration

Once the raw HTML of a page is successfully acquired, the next challenge is to isolate the main article content and filter out the surrounding "boilerplate" or "noise," such as navigation bars, advertisements, headers, footers, and sidebars. Feeding this noisy text to an LLM can introduce irrelevant entities and relationships, degrading the quality of the final Knowledge Graph.

Dedicated libraries have been developed to solve this specific problem through heuristic and machine learning-based approaches. A comparative analysis of these tools is essential for selecting the most effective one.

- **trafilatura:** The research consistently points to trafilatura as the superior choice for general-purpose main content extraction.[3] It has been benchmarked as a leader in the field, achieving a high F1-score of 90.2%, indicating an excellent balance between precision and recall.[3] It is also noted for being lightweight and fast, processing HTML significantly quicker than many alternatives.[3] Crucially, its evaluation shows a strong balance between precision (the accuracy of the text it extracts) and recall (the completeness of the text it extracts).[4] For this project, high recall is vital to ensure that no part of the core content is missed.
- **goose3:** While goose3 demonstrates very high precision, meaning the content it extracts is very likely to be part of the main article, it suffers from significantly poor recall.[4] This means it has a higher tendency to miss relevant parts of the content. For a KG construction task where the goal is to build a comprehensive representation of the document's knowledge, this trade-off is unacceptable. Furthermore,
  goose3 is benchmarked as being considerably slower than trafilatura.[4]

The recommended implementation is to pass the raw HTML content obtained from the scraper (either requests or Selenium) directly to the trafilatura library. Using its standard or precision settings will yield the highest quality clean text, stripped of boilerplate and ready for the LLM extraction phase.

Python

```
import requests
from trafilatura import fetch_url, extract
```

```
# Example usage with a URL
url = "https://example-news-article.com/story"
downloaded = fetch_url(url) # trafilatura can fetch the URL directly
# Or, if using the hybrid scraper:
# response = requests.get(url)
# downloaded = response.text

# Extract the main content, comments, and metadata are excluded by default
main_content = extract(downloaded)

if main_content:
    print(main_content)
```

1.3 Intelligent Crawling with LLM-Aware Frameworks: Crawl4ai

For projects that may scale beyond the initial dissertation scope or require more integrated functionality, "intelligent crawling" frameworks like Crawl4ai warrant consideration. Crawl4ai is an open-source tool designed specifically for AI-centric web crawling, bundling scraping, cleaning, and even LLM-based extraction into a single, cohesive framework.[5]

Key features of Crawl4ai include asynchronous processing for high-speed crawling of multiple pages, robust handling of dynamic content through browser automation, and the ability to automatically convert messy HTML into clean Markdown, a format often preferred for LLM input.[5]

A critical question for this dissertation project is whether such an advanced tool can operate within the strict budgetary constraints. The documentation shows that Crawl4ai can be configured for "AI powered extraction," which by default may use paid services like the OpenAI API.[5] However, a crucial architectural detail is its use of the

litellm library as its backend for LLM connections.[7]

litellm is a versatile abstraction layer that supports over 100 LLM providers, including free and local options such as Ollama.[9]

This means Crawl4ai can be configured to use a free, locally hosted LLM, thereby

eliminating API costs. This is achieved by specifying the local provider in the LlmConfig object. For an Ollama model, the provider string would be "ollama/<model_name>".[8]

Python

```
# Conceptual example of configuring Crawl4ai with a local Ollama model
from crawl4ai import LlmConfig, LLMExtractionStrategy, CrawlerRunConfig

# Configure the LLM to use a local Ollama model via litellm
local_llm_config = LlmConfig(provider="ollama/llama3:8b-instruct")

# Define an extraction strategy using this local LLM
llm_strategy = LLMExtractionStrategy(
    llm_config=local_llm_config,
    instruction="Extract the key people and their affiliations from this text."
)

# Configure the crawler to use this strategy
run_config = CrawlerRunConfig(extraction_strategy=llm_strategy)

#... proceed with the crawl using the AsyncWebCrawler...
```

While Crawl4ai is a powerful and budget-compatible option, for the defined scope of 100-200 pages, the manual hybrid pipeline (requests/Selenium + trafilatura) offers greater transparency and simplicity. It allows for more granular control and easier debugging, which are valuable attributes in a research context. Crawl4ai should be noted as a viable path for future work or for scaling the project beyond its initial prototype phase.

Table 1: Scraper Technology Decision Matrix

To provide a concise summary for methodological justification, the following table compares the two primary scraping technologies based on the analysis.

Criterion	requests + BeautifulSoup	Selenium / Playwright
Primary Use Case	Fast parsing of static HTML and XML documents.	Interaction with dynamic web applications that render content using JavaScript.
Speed	High. Significantly faster due to the absence of browser rendering overhead.[2]	Low. Slower performance as it must launch and control a full browser instance.[1]
Dynamic Content (JS)	No native support. Can only access the initial HTML source.	Full support. Can execute JavaScript, simulate user actions (clicks, scrolls), and access the fully rendered page.[1]
Resource Intensity	Low. Minimal CPU and memory consumption.	High. Consumes significant CPU and RAM, comparable to running a graphical browser.[1]
Setup Complexity	Low. Requires simple pip install of libraries.	Higher. Requires installation of browser drivers (e.g., ChromeDriver) and more complex configuration.[1]
Dissertation Project Role	**Primary Scraper:** Use as the first-pass, high-speed tool for all URLs.	**Fallback Mechanism:** Use as the robust, secondary tool for URLs that fail the primary attempt or are identified as dynamic.

Part II: LLM-Powered Information Extraction

This section addresses the intellectual core of the dissertation: the transformation of cleaned, unstructured text into structured Knowledge Graph triples. The success of the entire project hinges on the ability to effectively prompt a Large Language Model to act as a high-fidelity information extractor. This involves a combination of sophisticated prompt engineering, strict schema enforcement, and the use of frameworks that streamline the process.

2.1 Prompt Engineering for High-Fidelity Triple Extraction

The prompt is the primary interface through which the researcher communicates intent to the LLM. A well-designed prompt is the difference between a sparse, inaccurate graph and a dense, correct one. An optimal prompt for KG extraction should be a composite of several effective techniques.

- **Zero-Shot vs. Few-Shot Learning:** A zero-shot prompt provides the LLM with instructions but no concrete examples. While simple, its performance can be unreliable for a nuanced task like relation extraction. A few-shot prompt, conversely, includes a small number (typically 2-5) of high-quality examples demonstrating the desired input-to-output transformation.[12] This technique, known as in-context learning, significantly improves the model's accuracy and adherence to the desired format by showing, not just telling, what is required.[13] For a dissertation, a few-shot approach is strongly recommended for its superior performance and consistency.
- **Chain-of-Thought (CoT) Prompting:** This advanced technique dramatically enhances an LLM's reasoning capabilities. Instead of asking for the final output directly, the prompt instructs the model to "think step by step" or to externalize its reasoning process.[14] For relation extraction, this means guiding the model to first identify candidate entities, then analyze the linguistic context connecting them, and finally deduce the relationship type before constructing the final (Subject, Predicate, Object) triple. This structured reasoning process reduces errors and improves the logical consistency of the extractions.[15] Research into CoT for relation extraction (
  CoT-ER) suggests a multi-step process: first, identify the abstract types of the entities; second, extract the specific text span that serves as evidence for the relationship; and third, use this evidence to classify the relationship.[13]
- **Recommended Prompt Structure:** The most effective prompt for this project will be a carefully crafted hybrid that integrates these techniques.
  Example Hybrid Prompt Template:
  You are an expert knowledge graph engineer tasked with extracting entities and their relationships from a given text. Your goal is to create a structured list of (Subject, Predicate, Object) triples.
  Adhere strictly to the following schema for nodes and relationships:
  - Allowed Node Types: [Provide list, e.g., "Person", "Organization", "Location"]

- Allowed Relationship Types:

Follow these steps to perform the extraction:

1. **Entity Identification:** First, carefully read the text and identify all entities that match the allowed node types.
2. **Relationship Analysis:** For each pair of identified entities, analyze the sentence(s) connecting them to determine if a relationship of an allowed type exists between them.
3. **Triple Formulation:** Construct a triple in the format (Subject, Predicate, Object). Ensure the subject and object are the exact names of the identified entities and the predicate is one of the allowed relationship types.

Here are some examples of correct extraction:Example 1:Text: "Elon Musk, the CEO of SpaceX, also founded The Boring Company."Output:Example 2:Text: "Apple Inc. is headquartered in Cupertino, California."Output:Now, process the following text and generate the corresponding list of triples.Text:"""{input_text}"""Output:

This prompt structure provides role-playing, clear instructions, schema constraints, a Chain-of-Thought process, and few-shot examples, creating a robust framework for high-quality extraction.

2.2 Enforcing Structured Output via Schema Definition

A common failure mode of LLMs is producing output that is semantically correct but syntactically flawed. An LLM might return a list of triples with missing quotation marks, trailing commas, or conversational filler text, all of which will break any automated downstream parsing logic.[16] To build a reliable pipeline, the LLM's output must be strictly structured and validated.

The most effective solution for this is to define the desired output schema using a data validation library like Pydantic and leverage the LLM's function-calling or JSON-mode capabilities.[16]

Pydantic uses standard Python type hints to create data models that act as a "data contract".[16]

The process is as follows:

1. **Define Pydantic Models:** Create Python classes that precisely define the structure of the desired KG output. This includes defining types, required fields,

and even descriptions for each field, which can be passed to the LLM as additional guidance.[18]

```python
from pydantic import BaseModel, Field
from typing import List, Tuple

class Node(BaseModel):
    id: str = Field(..., description="The unique identifier for the entity.")
    label: str = Field(..., description="The type or category of the entity (e.g., Person, Organization).")

class Relationship(BaseModel):
    source: str = Field(..., description="The ID of the source node.")
    target: str = Field(..., description="The ID of the target node.")
    label: str = Field(..., description="The type of the relationship connecting the nodes.")

class KnowledgeGraph(BaseModel):
    nodes: List[Node]
    relationships: List
```

2. **Generate and Inject JSON Schema:** Most modern LLM frameworks and APIs can accept a JSON schema that forces the model's output to conform to the specified structure. Libraries like LangChain and instructor can automatically handle this by taking the Pydantic model as an argument and passing its JSON schema representation to the LLM API.[18] This ensures the LLM returns a syntactically valid JSON object that can be reliably parsed.
3. **Parse and Validate:** The JSON output from the LLM is then parsed directly into an instance of the Pydantic model. Pydantic automatically performs runtime validation, checking data types and ensuring all required fields are present. If validation fails, it raises a detailed error, which can even be used to re-prompt the model for a correction.[16] This creates a robust, self-correcting loop that guarantees data integrity.

2.3 Frameworks for Graph Transformation: A LangChain Deep Dive

While manual prompt engineering and output parsing are possible, frameworks like LangChain provide pre-built components that significantly streamline the process.

The key component for this project is the LLMGraphTransformer, located in the langchain_experimental package.[20]

The LLMGraphTransformer is designed to take a sequence of LangChain Document objects (containing the cleaned text from Part I) and use an LLM to automatically convert them into GraphDocument objects. Each GraphDocument is a container for the nodes and relationships extracted from the source text.[20]

A crucial feature of the transformer is its configurability. A user can provide lists of allowed_nodes and allowed_relationships during initialization.[21] This constrains the LLM's extraction to a predefined ontology, preventing it from inventing arbitrary node and relationship types and ensuring the resulting KG is clean and consistent.

While many online tutorials and documentation examples default to using a paid ChatOpenAI model, the LLMGraphTransformer is model-agnostic and works with any LangChain-compatible LLM wrapper.[25] This is the key to using it within the project's budget. By instantiating a local LLM via

Ollama or GPT4All and passing that LLM object to the transformer, the entire extraction process can be run locally and for free.

The following code provides a definitive example of using LLMGraphTransformer with a local Ollama model:

Python

```
from langchain_community.llms import Ollama
from langchain_experimental.graph_transformers import LLMGraphTransformer
from langchain_core.documents import Document

# 1. Instantiate the local LLM via the Ollama wrapper
# Ensure the Ollama service is running and the model has been pulled
llm = Ollama(model="llama3:8b-instruct", temperature=0)

# 2. Define the graph ontology (schema)
allowed_nodes = ["Person", "Organization", "Product"]
allowed_relationships =

# 3. Instantiate the transformer with the local LLM and schema
```

```
llm_transformer = LLMGraphTransformer(
    llm=llm,
    allowed_nodes=allowed_nodes,
    allowed_relationships=allowed_relationships
)

# 4. Prepare input documents
# In a real pipeline, these would come from the web scraper
docs_to_process =

# 5. Convert documents to graph documents
# This step invokes the local LLM to perform the extraction
graph_documents = llm_transformer.convert_to_graph_documents(docs_to_process)

# 6. Inspect the output
print("Extracted Graph Documents:")
for doc in graph_documents:
    print("Nodes:", [node.dict() for node in doc.nodes])
    print("Relationships:", [rel.dict() for rel in doc.relationships])
    print("-" * 20)
```

This approach combines the power of LangChain's abstraction with the cost-effectiveness of local models, providing a direct solution to the core research question. The main limitation is that the quality of the extraction is entirely dependent on the capability of the chosen local LLM, which may be less powerful than state-of-the-art proprietary models.[23]

2.4 Managing Context Windows for Long-Form Content

A practical challenge in processing web pages is that their content often exceeds the context window of the target LLM. For instance, many powerful 8B-parameter models have context windows of 8,192 tokens. A long article can easily surpass this limit, making it impossible to process the document in a single pass.

The solution is **text chunking**: splitting the long document into smaller, semantically coherent pieces that each fit within the context window.[26] The LLM then processes each chunk individually, and the resulting graph fragments are aggregated to

reconstruct the graph for the entire document.

Several chunking strategies exist, with varying levels of sophistication and effectiveness for KG extraction [28]:

- **Fixed-Length Chunking:** The simplest method, splitting text every N characters or tokens. This is strongly discouraged as it frequently cuts sentences and ideas in half, destroying the semantic context needed for accurate relation extraction.
- **Sentence-Based Chunking:** A significant improvement, this method uses natural language processing libraries (like nltk) to split the text along sentence boundaries. This ensures that each chunk contains whole sentences, preserving local context.
- **Recursive Character Text Splitting:** This is a popular and effective strategy implemented in LangChain. It attempts to split text hierarchically, first by paragraph separators (\n\n), then by sentence separators (.), and finally by words, to keep semantically related text grouped together as much as possible.
- **Semantic Chunking:** This is the most advanced strategy. It involves generating vector embeddings for each sentence and then grouping adjacent sentences that are semantically similar. Chunks are created where the semantic distance between consecutive sentences exceeds a certain threshold. This produces the most contextually coherent chunks but is more computationally intensive.

For this project, **Recursive Character Text Splitting** offers the best balance of simplicity and effectiveness. When chunking, it is crucial to use an **overlap**, where a portion of the end of one chunk is repeated at the beginning of the next. This overlap provides the LLM with surrounding context, which is vital for resolving coreferences (e.g., understanding that "he" in chunk 2 refers to a person named in chunk 1) and maintaining continuity in relationships that span chunk boundaries. The final aggregation step must then include an entity resolution process to merge duplicate nodes created from different chunks that refer to the same real-world entity.

Part III: Cost-Effective LLM Deployment and Access

A central constraint of this dissertation project is the strict budget of under ₹8000 (~$100 USD). This necessitates a strategy for accessing powerful LLMs at little to no cost. This section details the two primary avenues for achieving this: running open-source models locally on consumer hardware and leveraging the generous free

tiers of commercial LLM APIs.

3.1 Local LLM Execution on Consumer Hardware

The most direct and reliable method for eliminating recurring costs is to run LLMs locally. This approach not only guarantees zero inference fees but also offers maximum data privacy, as the source documents and extracted data never leave the user's machine.[29] The open-source community has produced several user-friendly tools that make setting up and running local LLMs accessible even to non-experts.

- **Ollama:** This is a powerful command-line tool that dramatically simplifies the process of running local LLMs. It bundles model weights, configurations, and a web server into a single, easy-to-manage application.[30] With a simple command like
  ollama run llama3, the tool downloads the specified model and immediately makes it available for chat in the terminal or via an OpenAI-compatible API endpoint at http://localhost:11434.[31] This seamless API compatibility makes integration with frameworks like LangChain trivial.[31]
- **LM Studio:** For users who prefer a graphical user interface (GUI), LM Studio provides an excellent alternative.[33] It features a "Discover" tab for browsing and downloading models from the Hugging Face Hub, a "Chat" tab for interacting with them, and a "Server" tab to start a local, OpenAI-compatible server with a single click.[34] This tool is particularly well-suited for beginners or for quickly experimenting with different models and configurations without writing any code.[35]

For a dissertation project, starting with small yet capable models is recommended. Open-source models like **Meta Llama 3 (8B)**, **Mistral (7B)**, and **Microsoft Phi-3** represent the state-of-the-art in their size class and are excellent candidates for local execution.

3.2 The Role of Quantization: The Key to Feasibility

Running multi-billion parameter models on consumer hardware (e.g., a laptop with 16 GB of RAM and a GPU with 6-8 GB of VRAM) is made possible by a single, critical

technology: **quantization**. A standard 7-billion parameter model stored at 16-bit precision requires approximately 14 GB of memory, which exceeds the VRAM of most consumer GPUs and consumes a large portion of system RAM. Quantization is the process of reducing the numerical precision of the model's weights, for example, from 16-bit floating-point numbers to 4-bit integers.[36] This can reduce the model's memory footprint by a factor of four or more, bringing a 14 GB model down to a much more manageable 3.5-4 GB.

This reduction is not merely a performance optimization; it is the **enabling technology** that makes the local-first, budget-friendly methodology of this dissertation feasible. Without quantization, running these models on typical student hardware would be impossible.

Two primary quantization formats are prevalent, and the choice between them is critical:

- **GGUF (GPT-Generated Unified Format):** This is the most versatile and recommended format for this project. GGUF is designed for efficient inference on CPUs but also supports **offloading a portion of the model's layers to a GPU**.[38] This hybrid approach is ideal for consumer hardware with limited VRAM. A user can load as many layers as will fit into their GPU's VRAM for accelerated processing, while the rest of the model runs on the CPU, utilizing system RAM.[36] This flexibility allows for running larger models than would otherwise be possible. Tools like
  Ollama and LM Studio almost exclusively use GGUF-quantized models.
- **GPTQ (GPT-based Quantization):** This is a GPU-only quantization format. It can offer very high performance, but only if the *entire* model fits within the GPU's VRAM.[39] It cannot fall back to using system RAM, making it far less flexible and generally unsuitable for users with limited VRAM.

Recommendation: The researcher should exclusively download models in the **GGUF** format from the Hugging Face Hub. When selecting a model file, look for specific quantization levels that offer a good balance between size and performance, such as **Q4_K_M** or **Q5_K_M**, which are widely regarded as providing excellent quality with significant size reduction.[37]

3.3 Leveraging Free and Generous API Tiers

As an alternative or a supplement to local models, several commercial providers offer free API tiers that provide access to more powerful, state-of-the-art models without any cost, provided usage remains within specified limits.

- **Google Gemini API:** The free tier for the **Gemini 1.5 Flash** model is particularly compelling for this project. It offers **15 requests per minute (RPM), 500 requests per day (RPD), and 250,000 tokens per minute (TPM)**.[40] The official pricing documentation confirms that usage within this tier is free of charge.[41] With a corpus of 200 web pages, the 500 RPD limit is more than sufficient to process the entire dataset over a single day, making this a highly viable, zero-cost option.
- **Groq API:** Groq is renowned for its exceptional inference speed, powered by its custom LPU hardware. It provides a free "Dev Tier" designed for developers and small-scale projects.[42] While the specific rate limits may fluctuate, the existence of this tier makes Groq an excellent choice for rapid experimentation and tasks where low latency is beneficial.[44]
- **Hugging Face Inference API:** Hugging Face offers a free tier for its serverless Inference API, which provides access to a vast number of models on the Hub. The free tier for registered users includes a small amount of monthly credits and rate limits on the order of 300 requests per hour, which can be useful for testing and smaller tasks.[46]

For the primary task of processing the 200-page corpus, the **Google Gemini 1.5 Flash API free tier** stands out as the most promising API-based solution due to its high model capability, generous limits, and clear zero-cost structure.

3.4 Comparative Analysis: Local Llama3:8b vs. API Gemini 1.5 Flash

A crucial part of the dissertation's methodology will be to justify the choice of LLM. This requires a comparison between the leading local option and the leading free API option.

- **Extraction Quality:** Direct, peer-reviewed benchmarks for Knowledge Graph extraction are still emerging.[48] However, general-purpose LLM benchmarks provide strong indicators. Across reasoning and knowledge-based tasks like MMLU-Pro, Gemini models consistently show a performance advantage over Llama 3 models of a similar size.[50] One study on Chinese relation extraction found that
  gemini-1.5-flash achieved significantly higher recall than Llama models,

suggesting it is better at identifying a comprehensive set of relationships, though its precision was lower than top-tier models like GPT-4.[52] This suggests that for KG construction,
Gemini 1.5 Flash is likely to produce a more *complete* graph, while a local Llama 3 8B might produce a *cleaner* but less comprehensive one.

- **Latency and Reliability:** Local models offer predictable latency determined by the user's hardware. API calls introduce variability from network latency and server load. However, highly optimized services like Groq and Gemini are often faster than running a model on a consumer-grade CPU.
- **Recommendation for Dissertation Methodology:** The most rigorous approach is to conduct a small-scale pilot study. The researcher should process a representative sample of 10-20 web pages from their corpus using both a quantized Llama-3:8b-instruct.gguf model running locally via Ollama and the Gemini 1.5 Flash API. The resulting sets of extracted triples should then be manually evaluated for precision (correctness) and recall (completeness). This project-specific, empirical comparison will provide a much stronger justification for the final choice of LLM than relying solely on generic external benchmarks.

Table 2: LLM Access Modality Trade-Offs

The decision between local execution and API access is central to the project's design. The following table summarizes the trade-offs.

Factor	Local Execution (via Ollama/LM Studio)	Free API Tiers (e.g., Gemini 1.5 Flash)
Cost	Effectively zero after initial hardware purchase. No per-request fees.[29]	Free within specified rate and usage limits. Risk of incurring fees if limits are exceeded.[41]
Data Privacy	Maximum. Data and prompts never leave the local machine.[29]	Lower. Data is sent to a third-party provider and is subject to their privacy policies and terms of service.
Model Capability	Limited to smaller, open-source models (typically 3B to 13B parameters on consumer hardware).	Access to large, state-of-the-art proprietary models (e.g., Gemini 1.5) that may not be publicly available.

Inference Speed	Dependent on local hardware (CPU, RAM, GPU). Can be slow, especially on CPU-only systems.	Generally very fast and low-latency due to optimized, data-center-grade hardware.[48] Subject to network conditions.
Setup & Maintenance	Requires one-time setup of tools (Ollama), model downloads (several GB), and local resource management.[30]	Simpler. Requires only signing up for an API key and managing credentials.[40]
Offline Use	Fully functional without an internet connection once models are downloaded.	Requires a constant and stable internet connection to make API calls.

Part IV: Knowledge Graph Storage and Orchestration

The final stage of the pipeline involves storing the extracted graph data in a persistent and queryable format and orchestrating the entire workflow from URL to graph. This section addresses the selection of an appropriate graph data store and provides a blueprint for integrating all components into a cohesive, automated script.

4.1 Selecting a Graph Data Store: NetworkX vs. Neo4j

Once the (Subject, Predicate, Object) triples have been extracted, they must be loaded into a graph structure for analysis. For a Python-based project, two primary options present themselves: a lightweight, in-memory library or a full-featured, persistent graph database.

- **NetworkX:** This is a ubiquitous Python library for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.[53] It represents graphs in memory, making it incredibly fast and easy to integrate into a Python script. It requires no external database setup—a simple pip install is all that is needed.[54]
  NetworkX is ideal for rapid prototyping, programmatic analysis, and running a wide array of classic graph algorithms (e.g., centrality, pathfinding).[53] Its main

limitation is that it is not a persistent database; the graph exists only in RAM during script execution and must be explicitly saved to and loaded from a file format like Pickle or GraphML if persistence is required.[54]

- **Neo4j:** In contrast, Neo4j is a mature, enterprise-grade, and transactional graph database designed for persistent storage and operational workloads.[55] It provides a robust server environment, managed easily for local development via Neo4j Desktop, and uses the powerful and declarative Cypher query language for complex graph traversals.[57] It also offers rich visualization tools like Neo4j Bloom, which allows for interactive, no-code exploration of the graph.[58] The Neo4j Community Edition is free and more than capable of handling the scale of this dissertation project.[55]

A Two-Stage Storage Strategy for the Research Lifecycle

A dissertation project has distinct phases, each with different requirements. The initial phase involves heavy development, experimentation, and rapid iteration of the extraction pipeline. The final phase focuses on analysis, visualization, and demonstration of the results. This suggests that a single storage solution may not be optimal for the entire project.

The most effective methodology is a two-stage approach that leverages the strengths of both tools:

1. **Stage 1: Prototyping with NetworkX:** During the development of the scraping and LLM extraction pipeline, NetworkX should be the primary target. Its in-memory nature and simple Python API allow for extremely fast iteration. The entire pipeline can be re-run in seconds or minutes, and the resulting graph can be immediately analyzed within the same script without the overhead of connecting to and clearing an external database. This agility is invaluable during the experimental phase.
2. **Stage 2: Finalization with Neo4j:** Once the extraction pipeline is stable and has produced the final set of triples for the full corpus, the data should be ingested into a local Neo4j instance. This provides a persistent, queryable, and visualizable artifact. The researcher can then use Cypher for powerful ad-hoc queries, leverage Neo4j Bloom for interactive exploration to uncover interesting patterns, and easily connect the database to a web application or other demonstration tools.

This two-stage strategy aligns the choice of technology with the evolving needs of the research process, using the right tool for the right job.

4.2 Local Graph Database Implementation with Neo4j

Setting up a local Neo4j instance for the final storage phase is straightforward.

- **Setup:** The first step is to download and install Neo4j Desktop, which provides a simple GUI for creating and managing local database projects.[59] Within Desktop, the user can create a new project and a new database instance with a few clicks.[59] The instance can then be started, making it accessible for connections from a Python script.
- **Data Ingestion:** The most critical task is to efficiently load the LLM-generated triples into the database. Performing individual CREATE or MERGE queries for each of the thousands of triples would be extremely inefficient. The correct and performant approach is to perform a batch insertion within a single transaction. This is achieved by passing the entire list of triples as a parameter to a Cypher query that uses the UNWIND clause to iterate over the list and create the graph elements.[61]

The following Python function demonstrates how to use the official neo4j-driver to perform this batch ingestion. It takes a list of triple tuples, connects to the database, and executes a single, parameterized query.

Python

```
from neo4j import GraphDatabase
from typing import List, Tuple

def batch_ingest_triples_to_neo4j(uri, user, password, triples: List]):
    """
    Connects to a Neo4j database and ingests a list of (subject, predicate, object)
    triples in a single, efficient batch transaction.

    Args:
        uri (str): The Bolt URI for the Neo4j instance.
```

```
    user (str): The username for the database.
    password (str): The password for the database.
    triples (List]): A list of triples to ingest.
  """
  driver = GraphDatabase.driver(uri, auth=(user, password))

  # Format triples into a list of dictionaries for the query parameter
  triples_data = [
      {"subject": s, "predicate": p, "object": o} for s, p, o in triples
  ]

  # This Cypher query uses UNWIND to iterate over the list of triples.
  # It uses MERGE to create nodes, preventing duplicates if they already exist.
  # It then uses MERGE to create the relationship between the nodes.
  # This is highly efficient as it's a single query and a single transaction.
  query = """
  UNWIND $triples AS t
  MERGE (source:Entity {id: t.subject})
  MERGE (target:Entity {id: t.object})
  MERGE (source)-->(target)
  """

  with driver.session() as session:
      session.run(query, triples=triples_data)
      print(f"Successfully ingested {len(triples)} triples into Neo4j.")

  driver.close()

# Example Usage:
# triples_from_llm =
# batch_ingest_triples_to_neo4j("bolt://localhost:7687", "neo4j", "password", triples_from_llm)
```

This approach, synthesized from best practices for batch operations, provides a robust and scalable method for populating the final graph database.[61]

4.3 End-to-End Pipeline Orchestration

The final step is to integrate all the components—scraping, cleaning, chunking,

extraction, and storage—into a single, automated pipeline. Frameworks like LangChain and its more explicit state-machine-based counterpart, LangGraph, are designed precisely for this purpose, allowing developers to chain components together where the output of one step serves as the input to the next.[65]

The logical flow of the final, orchestrated script would be:

1. **Input:** The script takes a list of URLs as its initial input.
2. **Iteration:** It iterates through each URL.
3. **Scraping and Cleaning:** For each URL, it calls a function that implements the hybrid scraper (from 1.1) and uses trafilatura (from 1.2) to return clean, unstructured text.
4. **Chunking:** The clean text is passed to a LangChain text splitter (e.g., RecursiveCharacterTextSplitter) to create a list of Document objects suitable for the LLM (from 2.4).
5. **Graph Extraction:** This list of Document objects is fed into the LLMGraphTransformer, which is configured with a local Ollama LLM and the predefined schema (from 2.3). The transformer outputs a GraphDocument.
6. **Aggregation and Storage:** The nodes and relationships from the GraphDocument are collected. After processing all URLs, the aggregated list of triples is passed to the batch ingestion function (from 4.2) to populate the Neo4j database.

This complete workflow, encapsulated in a single Python script, directly answers the core research question by providing a demonstrable, end-to-end pipeline from web page to Knowledge Graph.[66]

Table 3: Graph Storage Solution Comparison (Prototyping Focus)

To justify the recommended two-stage storage strategy, this table highlights the key differences between NetworkX and Neo4j in the context of a dissertation prototype.

Feature	NetworkX	Neo4j Community Edition
Architecture	In-memory Python library. The graph exists only within the running script's memory.[54]	Standalone, persistent graph database server with on-disk storage.[56]

Setup & Dependencies	Simple pip install networkx. No external services required.	Requires download and installation of Neo4j Desktop application; management of a database instance.[59]
Data Persistence	Not persistent by default. Graph must be explicitly saved to and loaded from a file (e.g., Pickle, GraphML).	Fully persistent and transactional (ACID-compliant). Data is safely stored on disk between sessions.[70]
Querying Mechanism	Programmatic. Queries are performed by writing Python code to iterate over nodes and edges.[53]	Declarative. Uses the powerful Cypher query language to express complex graph patterns.[57]
Performance	Extremely fast for graph analysis and algorithms as long as the graph fits in RAM.[54]	Optimized for querying large, on-disk graphs. May have driver and network overhead for Python interaction.[70]
Visualization	Basic static plotting capabilities via integration with libraries like matplotlib.[53]	Rich, interactive visualization and exploration via built-in tools like Neo4j Browser and Neo4j Bloom.[58]
Project Role	**Ideal for Prototyping:** Best for rapid development, iteration, and programmatic analysis of the KG during the pipeline-building phase.[54]	**Ideal for Finalization:** Best for persistent storage, ad-hoc querying, and interactive visualization of the final KG artifact.[55]

Conclusion and Recommendations

This report has detailed a comprehensive, cost-effective, and methodologically sound end-to-end pipeline for constructing a Knowledge Graph from a small, diverse corpus of web pages, tailored specifically for the constraints of a dissertation project. The analysis demonstrates that such a project is not only feasible but can be executed with a high degree of technical sophistication while adhering to a budget of under

₹8000 INR (~$100 USD).

The key recommendations synthesized from the analysis are as follows:

1. **Adopt a Hybrid Web Scraping Strategy:** Prioritize data integrity over marginal speed gains. Implement a pipeline that first attempts to scrape content using the fast requests and BeautifulSoup combination, then automatically falls back to a full browser automation tool like Selenium or Playwright for dynamic pages that fail the initial attempt. This ensures maximum data capture from a heterogeneous corpus. For content cleaning, trafilatura is the recommended tool due to its superior balance of precision and recall.
2. **Embrace a Local-First LLM Approach:** To eliminate recurring costs and ensure data privacy, the primary strategy for LLM access should be local execution. Tools like Ollama and LM Studio make it straightforward to download and run powerful open-source models. This is made feasible on consumer hardware through the use of **GGUF-quantized models**, which significantly reduce memory requirements. The Llama 3 8B Instruct model in a Q4_K_M GGUF format is an excellent starting point.
3. **Leverage Free API Tiers as a High-Capability Alternative:** For access to even more powerful models, the free tier of the **Google Gemini 1.5 Flash API** is a highly recommended alternative. Its generous rate limits are sufficient for processing the entire corpus at no cost. A pilot study comparing the extraction quality of the local Llama 3 model against the Gemini 1.5 Flash API should be conducted to provide empirical justification for the final model choice in the dissertation.
4. **Enforce Rigorous Schema and Prompting Standards:** The quality of the extracted graph is directly proportional to the quality of the prompt. A hybrid prompt should be employed, incorporating role-playing, clear instructions, a Chain-of-Thought reasoning structure, and few-shot examples. To guarantee reliable, machine-readable output, the desired graph structure should be defined using Pydantic models, and the resulting JSON schema should be used to enforce structured output from the LLM.
5. **Utilize a Two-Stage Graph Storage Strategy:** The research lifecycle calls for different tools at different stages. Use the lightweight, in-memory **NetworkX** library during the rapid prototyping and development phase for its speed and simplicity. Once the extraction pipeline is finalized, ingest the complete graph into a local **Neo4j Community Edition** instance for persistent storage, advanced querying with Cypher, and rich, interactive visualization.
6. **Orchestrate with LangChain:** Use the LangChain framework, specifically the LLMGraphTransformer, to tie all components together. This transformer, when

configured with a local Ollama LLM and a predefined schema, provides a streamlined and powerful method for converting cleaned text into structured graph documents.

By following this end-to-end pipeline, a researcher can successfully build a high-quality, dissertation-scale Knowledge Graph from web data in a manner that is transparent, replicable, and exceptionally budget-friendly. This methodology not only provides a solution to the immediate research question but also equips the researcher with a flexible and scalable framework for future explorations in the domain of automated knowledge engineering.

Works cited

1. BeautifulSoup vs. Selenium: A Detailed Comparison for Web ..., accessed June 20, 2025, https://www.browserstack.com/guide/beautifulsoup-vs-selenium
2. Selenium vs. BeautifulSoup in 2025: Which Is Better? - ZenRows, accessed June 20, 2025, https://www.zenrows.com/blog/selenium-vs-beautifulsoup
3. Comparative Analysis of Open-Source News Crawlers - htdocs.dev, accessed June 20, 2025, https://htdocs.dev/posts/comparative-analysis-of-open-source-news-crawlers/
4. Evaluation — Trafilatura 2.0.0 documentation, accessed June 20, 2025, https://trafilatura.readthedocs.io/en/latest/evaluation.html
5. Crawl4AI: AI-Ready Web Crawling - DEV Community, accessed June 20, 2025, https://dev.to/ali_dz/crawl4ai-the-ultimate-guide-to-ai-ready-web-crawling-2620
6. unclecode/crawl4ai: Crawl4AI: Open-source LLM Friendly Web Crawler & Scraper. Don't be shy, join here: https://discord.gg/jP8KfhDhyN - GitHub, accessed June 20, 2025, https://github.com/unclecode/crawl4ai
7. Crawl4AI Tutorial: Build a Powerful Web Crawler for AI Applications ..., accessed June 20, 2025, https://www.pondhouse-data.com/blog/webcrawling-with-crawl4ai
8. LLM Strategies - Crawl4AI Documentation (v0.6.x), accessed June 20, 2025, https://docs.crawl4ai.com/extraction/llm-strategies/
9. LiteLLM - Getting Started | liteLLM, accessed June 20, 2025, https://docs.litellm.ai/
10. Providers - LiteLLM, accessed June 20, 2025, https://docs.litellm.ai/docs/providers
11. Ollama - LiteLLM, accessed June 20, 2025, https://docs.litellm.ai/docs/providers/ollama
12. Chain Of Thought Prompting: Everything You Need To Know - Annotation Box, accessed June 20, 2025, https://annotationbox.com/chain-of-thought-prompting/
13. arxiv.org, accessed June 20, 2025, https://arxiv.org/html/2311.05922v3
14. Chain-of-Thought Prompting | Prompt Engineering Guide, accessed June 20, 2025, https://www.promptingguide.ai/techniques/cot
15. Understanding Chain-of-Thought in LLMs through Information Theory - arXiv, accessed June 20, 2025, https://arxiv.org/html/2411.11984v1

16. A Practical Guide on Structuring LLM Outputs with Pydantic - DEV ..., accessed June 20, 2025, https://dev.to/devasservice/a-practical-guide-on-structuring-llm-outputs-with-pydantic-50b4
17. Using Pydantic Models for Structured Output | CodeSignal Learn, accessed June 20, 2025, https://codesignal.com/learn/courses/expanding-crewai-capabilities-and-integration/lessons/using-pydantic-models-for-structured-output
18. Using Pydantic for Dynamic LLM Response Models - Instructor, accessed June 20, 2025, https://python.useinstructor.com/concepts/models/
19. Build an Extraction Chain - LangChain, accessed June 20, 2025, https://python.langchain.com/docs/tutorials/extraction/
20. How to construct knowledge graphs | 🦜 LangChain, accessed June 20, 2025, https://python.langchain.com/docs/how_to/graph_constructing/
21. How to Build a Knowledge Graph for RAG Applications? - ProjectPro, accessed June 20, 2025, https://www.projectpro.io/article/knowledge-graph-rags/1089
22. How to Build a Knowledge Graph in Minutes (And Make It Enterprise-Ready), accessed June 20, 2025, https://towardsdatascience.com/enterprise-ready-knowledge-graphs-96028d863e8c/
23. How to construct knowledge graphs - LangChain.js, accessed June 20, 2025, https://js.langchain.com/docs/how_to/graph_constructing/
24. LLMGraphTransformer — LangChain documentation, accessed June 20, 2025, https://python.langchain.com/api_reference/experimental/graph_transformers/langchain_experimental.graph_transformers.llm.LLMGraphTransformer.html
25. Enhancing RAG-based application accuracy by constructing and leveraging knowledge graphs - LangChain Blog, accessed June 20, 2025, https://blog.langchain.dev/enhancing-rag-based-applications-accuracy-by-constructing-and-leveraging-knowledge-graphs/
26. Text Chunking - GraphRAG, accessed June 20, 2025, https://graphrag.com/guides/chunking/
27. Knowledge Graph Extraction and Challenges - Graph Database & Analytics - Neo4j, accessed June 20, 2025, https://neo4j.com/blog/developer/knowledge-graph-extraction-challenges/
28. 5 RAG Chunking Strategies for Better Retrieval-Augmented ... - Lettria, accessed June 20, 2025, https://www.lettria.com/blogpost/5-rag-chunking-strategies-for-better-retrieval-augmented-generation
29. Run models locally | 🦜 LangChain, accessed June 20, 2025, https://python.langchain.com/docs/how_to/local_llms/
30. How to Set Up and Run DeepSeek-R1 Locally With Ollama - DataCamp, accessed June 20, 2025, https://www.datacamp.com/tutorial/deepseek-r1-ollama
31. How to Set Up and Run Llama 3 Locally With Ollama and GPT4ALL ..., accessed June 20, 2025, https://www.datacamp.com/tutorial/run-llama-3-locally
32. OllamaLLM - LangChain, accessed June 20, 2025,

https://python.langchain.com/docs/integrations/llms/ollama/
33. Tool Use | LM Studio Docs, accessed June 20, 2025, https://lmstudio.ai/docs/advanced/tool-use
34. LM Studio - For New Coders - Cline, accessed June 20, 2025, https://docs.cline.bot/running-models-locally/lm-studio
35. mistral-7b-instruct-v0.3 - LM Studio, accessed June 20, 2025, https://lmstudio.ai/model/mistral-7b-instruct-v0.3
36. LLMs on CPU: The Power of Quantization with GGUF, AWQ, & GPTQ, accessed June 20, 2025, https://www.ionio.ai/blog/llms-on-cpu-the-power-of-quantization-with-gguf-awq-gptq
37. Quantize Llama models with GGUF and llama.cpp - Origins AI, accessed June 20, 2025, https://originshq.com/blog/quantize-llama-models-with-gguf-and-llama-cpp/
38. newsletter.maartengrootendorst.com, accessed June 20, 2025, https://newsletter.maartengrootendorst.com/p/which-quantization-method-is-right#:~:text=GGUF%3A%20GPT%2DGenerated%20Unified%20Format,GPU%20for%20a%20speed%20up.
39. LLM-FineTuning-Large-Language-Models/LLM_Techniques_and_utils/GGUF_GGML_GPTQ-basics.md at main - GitHub, accessed June 20, 2025, https://github.com/rohan-paul/LLM-FineTuning-Large-Language-Models/blob/main/LLM_Techniques_and_utils/GGUF_GGML_GPTQ-basics.md
40. Rate limits | Gemini API | Google AI for Developers, accessed June 20, 2025, https://ai.google.dev/gemini-api/docs/rate-limits
41. Gemini Developer API Pricing | Gemini API | Google AI for Developers, accessed June 20, 2025, https://ai.google.dev/gemini-api/docs/pricing
42. Enterprise Access - Groq is Fast AI Inference, accessed June 20, 2025, https://groq.com/enterprise-access/
43. What's Groq AI and Everything About LPU [2025] - Voiceflow, accessed June 20, 2025, https://www.voiceflow.com/blog/groq
44. Is Groq Free? Access the Groq API Without Cost - BytePlus, accessed June 20, 2025, https://www.byteplus.com/en/topic/404714
45. What are the rate limits for the Groq API, for the Free and Dev tier plans? | Community, accessed June 20, 2025, https://community.groq.com/help-center-14/what-are-the-rate-limits-for-the-groq-api-for-the-free-and-dev-tier-plans-127
46. Pricing and Billing - Hugging Face, accessed June 20, 2025, https://huggingface.co/docs/inference-providers/pricing
47. Free Hugging Face Inference api now clearly lists limits + models : r/LocalLLaMA - Reddit, accessed June 20, 2025, https://www.reddit.com/r/LocalLLaMA/comments/1fi90kw/free_hugging_face_inference_api_now_clearly_lists/
48. Zero-Shot End-to-End Relation Extraction in Chinese: A Comparative Study of Gemini, LLaMA, and ChatGPT - arXiv, accessed June 20, 2025, https://www.arxiv.org/pdf/2502.05694

49. Gemini 1.5 Pro (Sep '24) vs Llama 3.1 Instruct 70B: Model Comparison, accessed June 20, 2025, https://artificialanalysis.ai/models/comparisons/gemini-1-5-pro-vs-llama-3-1-instruct-70b
50. Gemini 1.5 Flash 8B vs Llama 3.1 8B Instruct - LLM Stats, accessed June 20, 2025, https://llm-stats.com/models/compare/gemini-1.5-flash-8b-vs-llama-3.1-8b-instruct
51. Llama 3 8B Instruct vs Gemini 1.5 Flash-8B - Detailed Performance & Feature Comparison, accessed June 20, 2025, https://docsbot.ai/models/compare/llama-3-8b-instruct/gemini-1-5-flash-8b
52. Zero-Shot End-to-End Relation Extraction in Chinese: A Comparative Study of Gemini, LLaMA, and ChatGPT - arXiv, accessed June 20, 2025, https://arxiv.org/html/2502.05694v1
53. 354 - Knowledge Graphs in Python Using NetworkX library - YouTube, accessed June 20, 2025, https://www.youtube.com/watch?v=n7BTWc2C1Eg
54. Simple In-Memory Knowledge Graphs for Quick Graph Querying, accessed June 20, 2025, https://safjan.com/simple-inmemory-knowledge-graphs-for-quick-graph-querying/
55. Is Neo4j a good database for small apps where one may never have big data? - Quora, accessed June 20, 2025, https://www.quora.com/Is-Neo4j-a-good-database-for-small-apps-where-one-may-never-have-big-data
56. Graph Databases Explained: Better Way to Represent Connections - Cognee AI, accessed June 20, 2025, https://www.cognee.ai/blog/fundamentals/graph-databases-explained
57. Tutorials - Getting Started - Neo4j, accessed June 20, 2025, https://neo4j.com/docs/getting-started/appendix/tutorials/tutorials-overview/
58. 15 Best Graph Visualization Tools for Your Neo4j Graph Database, accessed June 20, 2025, https://neo4j.com/blog/graph-visualization/neo4j-graph-visualization-tools/
59. Installation - Neo4j Desktop, accessed June 20, 2025, https://neo4j.com/docs/desktop-manual/current/installation/
60. Installation - Neo4j Desktop, accessed June 20, 2025, https://neo4j.com/docs/desktop/current/installation/
61. [Query] Efficient way to batch write · Issue #213 · neo4j/neo4j-python-driver - GitHub, accessed June 20, 2025, https://github.com/neo4j/neo4j-python-driver/issues/213
62. UNWIND - Cypher Manual - Neo4j, accessed June 20, 2025, https://neo4j.com/docs/cypher-manual/current/clauses/unwind/
63. (Neo4j-driver) - How to do batch insert relationship with Python - Stack Overflow, accessed June 20, 2025, https://stackoverflow.com/questions/57407926/neo4j-driver-how-to-do-batch-insert-relationship-with-python
64. Create relationship between nodes created from UNWIND list - Stack Overflow,

accessed June 20, 2025, https://stackoverflow.com/questions/70876421/create-relationship-between-nodes-created-from-unwind-list
65. A Step-by-Step Guide to Build an Automated Knowledge Graph ..., accessed June 20, 2025, https://www.marktechpost.com/2025/05/15/a-step-by-step-guide-to-build-an-automated-knowledge-graph-pipeline-using-langgraph-and-networkx/
66. Extracting Knowledge Graphs from User Stories Using Langchain - arXiv, accessed June 20, 2025, https://www.arxiv.org/pdf/2506.11020
67. Constructing a Knowledge Graph from unstructured data with LLMs by Diana O | Contra, accessed June 20, 2025, https://contra.com/p/EA2P2MRs-constructing-a-knowledge-graph-from-unstructured-data-with-ll-ms
68. Building Knowledge Graphs for RAG: Exploring GraphRAG with Neo4j and LangChain, accessed June 20, 2025, https://hackernoon.com/building-knowledge-graphs-for-rag-exploring-graphrag-with-neo4j-and-langchain
69. What are the differences between Neo4j and other graph databases? - Quora, accessed June 20, 2025, https://www.quora.com/What-are-the-differences-between-Neo4j-and-other-graph-databases
70. [D] Seeking Advice - For graph ML, Neo4j or nah? : r/MachineLearning - Reddit, accessed June 20, 2025, https://www.reddit.com/r/MachineLearning/comments/wav15e/d_seeking_advice_for_graph_ml_neo4j_or_nah/