A Research Blueprint for Autonomous Knowledge Graph Construction with Emergent Schema Discovery

Part I: Introduction and Conceptual Foundations

1.1 The Modern Information Deluge: A Crisis of Structure

The digital transformation has resulted in an unprecedented proliferation of information. Enterprises and the public web have become vast repositories of knowledge, but this knowledge is predominantly locked within unstructured and semi-structured formats.[1] Critical business intelligence, scientific data, and financial reporting are contained in heterogeneous sources such as dynamic web pages, technical blogs, and complex Portable Document Formats (PDFs) that embed both text and tables. This explosion of data presents a dual challenge and opportunity. While the potential for data-driven insight has never been greater, the ability to harness it is severely constrained by the very nature of the data itself.

Traditional methods of information retrieval and data analysis, which often rely on keyword matching or rigid, predefined database schemas, are ill-equipped to navigate this landscape.[1] These systems struggle to comprehend the nuanced semantic relationships and contextual connections that exist within a single document, let alone across a diverse corpus. They are unable to synthesize information scattered across different formats—for instance, connecting a procedural description in a blog post to a data table in a financial PDF. This fundamental limitation creates a significant bottleneck for knowledge discovery, strategic decision-making, and the development of truly intelligent applications.[2] The result is a crisis of structure, where vast quantities of valuable data remain underutilized, their potential unrealized due to the sheer difficulty of making them machine-readable and interoperable.

1.2 The Knowledge Graph Paradigm: Structuring the World's Data

In response to the challenge of unstructured data, the Knowledge Graph (KG) has emerged as a powerful and flexible paradigm for knowledge representation.[5] A KG is not merely a database; it is a knowledge base that employs a graph-structured data model to represent a network of real-world entities and the relationships between them, effectively creating a semantic network of interconnected facts.[5]

Fundamental Components

At its core, a knowledge graph is composed of three primary components that work in concert to model a domain [7]:

- **Nodes (Entities/Vertices):** Nodes represent the distinct objects, concepts, people, places, or events within a domain. These are the fundamental "things" about which information is stored. For example, in a financial KG, nodes could represent entities such as "Apple Inc.", "Tim Cook", "iPhone 15", or a specific "Q4 2023 Earnings Report".[6]
- **Edges (Relationships/Predicates):** Edges are the connections that define how nodes are related to one another. They represent the verbs or predicates that link the subjects and objects in the real world. For instance, an edge might connect the "Tim Cook" node to the "Apple Inc." node with the label is_CEO_of, while another connects "Apple Inc." to "iPhone 15" with the label manufactures.[7] These relationships can be directed (e.g., is_CEO_of has a clear direction) and can be weighted to represent the strength of a connection.[10]
- **Properties:** Properties are key-value pairs that provide additional attributes or metadata to describe nodes and, in some models, edges. For example, a Person node might have properties like dateOfBirth or title, while a Company node could have foundingDate or stockTicker.[10]

The Role of the Schema (Ontology)

A knowledge graph's power and consistency stem from its underlying schema, also known as an ontology.[11] An ontology is a formal specification that defines the "rules of the world" for a given domain. It provides a structured framework by specifying the allowed types of nodes (e.g.,

Person, Company, Product, FinancialMetric), the allowed types of relationships (has_CEO, reported_revenue, is_located_in), and the constraints on how they can be connected (e.g., a has_CEO relationship must connect a Person node to a Company node).[6]

This schema is what transforms a simple collection of triples into a coherent and machine-interpretable knowledge base. It enables several critical capabilities:

- **Data Integration:** The ontology provides a common structure that allows for the simplified integration of data from diverse and heterogeneous sources.[11]
- **Reasoning and Inference:** By defining hierarchical (subClassOf) and other logical relationships, the schema allows a system to infer new knowledge that is not explicitly stated in the source data.[12]
- **Data Quality and Consistency:** The schema acts as a validation layer, ensuring that new information added to the graph adheres to the domain's rules, thereby improving overall data quality.[11]
- **Explainability:** The explicit nature of the graph structure provides a transparent way to trace the reasoning paths used by an AI system to arrive at an answer, making the system more explainable.[12]

Historically, the creation of this domain ontology has been a significant bottleneck in KG construction. It is a complex, time-consuming, and often expensive process that requires deep domain expertise and manual effort to ensure accuracy and consistency.[3] This high barrier to entry has limited the widespread adoption of KG technology.

1.3 The LLM Revolution in Knowledge Engineering

The recent advent of powerful Large Language Models (LLMs) has fundamentally altered the landscape of knowledge engineering.[1] Trained on petabytes of text and code, these models have developed a remarkable capacity for understanding and

generating human language, enabling them to perform complex tasks with minimal task-specific training.[16] Of particular relevance to KG construction is their emergent ability to perform zero-shot and few-shot information extraction.[1] This means LLMs can be prompted to identify entities and extract relationships from unstructured text with high fidelity, dramatically accelerating the process of populating a knowledge graph.[1]

However, the most transformative potential of LLMs in this domain lies beyond simple fact extraction. The central thesis of this research plan is that LLMs can be leveraged to address the primary bottleneck in KG construction: the creation of the schema itself. Recent research indicates that LLMs possess higher-order reasoning capabilities that allow them to not only extract information but also to analyze, categorize, and structure it. They can be prompted to discover entity types, canonicalize relationship labels, and even induce entire taxonomic hierarchies from raw text.[15]

This capability opens a new frontier for a fully automated, end-to-end pipeline where the LLM performs two critical roles in tandem: it discovers the underlying schema from the data while simultaneously extracting the facts to populate it. This represents a paradigm shift from the traditional, schema-first approach to a dynamic, schema-as-a-service model, where the ontology is not a prerequisite for construction but an emergent property of the process itself. This research plan provides a comprehensive blueprint for developing such a system.

Part II: The Data Ingestion and Preparation Pipeline

The quality and comprehensiveness of any knowledge graph are fundamentally dependent on the quality of its input data. Before any intelligent extraction can occur, a robust and reliable pipeline must be established to acquire raw content from diverse sources and meticulously clean it to isolate the core textual signal from surrounding noise. This foundational engineering is critical for the success of the entire system.

2.1 A Hybrid Framework for Heterogeneous Source Acquisition

The modern web is not a monolithic entity; it is a heterogeneous collection of documents built with varying technologies. A corpus of websites, blogs, and financial portals will inevitably include simple, static HTML pages alongside complex, dynamic single-page applications that rely heavily on client-side JavaScript to render their primary content.[1] A single-method scraping strategy is therefore inherently brittle and destined to fail on a significant portion of sources. To ensure maximum data capture, a hybrid, fallback-driven architecture is required.

The proposed workflow is designed to optimize for the common case while ensuring robustness for the difficult case:

1. **Primary Attempt (Speed and Efficiency):** For each target URL, the pipeline will first attempt to fetch and parse the content using the classic, lightweight combination of the requests library and BeautifulSoup. The requests library executes a simple HTTP GET request to retrieve the initial HTML source code served by the server, while BeautifulSoup provides an efficient interface for parsing this static HTML.[1] This method is exceptionally fast and consumes minimal CPU and memory resources because it does not involve rendering a full web page in a browser.[25]
2. **Content Validation:** After the initial scrape, a simple automated validation check is performed. This check can be as straightforward as verifying that the word count of the extracted text exceeds a minimum threshold (e.g., 50 words). A very low word count is a strong indicator that the main content is loaded dynamically and was not present in the initial HTML source.[1]
3. **Fallback Mechanism (Robustness and Completeness):** If the primary attempt fails (e.g., due to an HTTP error or timeout) or if the content validation check does not pass, the pipeline automatically triggers a fallback to a full browser automation framework such as Selenium or Playwright.[1] These tools programmatically control a real web browser (e.g., Chrome, Firefox), allowing them to execute JavaScript, simulate user interactions like scrolling or clicking "load more" buttons, and wait for AJAX requests to complete. The framework then extracts the fully rendered HTML from the browser.[24]

The justification for this hybrid approach is rooted in the priorities of a research project. While the browser automation fallback is slower and more resource-intensive, the absolute time difference for a dissertation-scale corpus is trivial compared to the cost of data loss. A scraping script that fails introduces bias and undermines the validity of the final knowledge graph. Therefore, methodological soundness and data completeness must be prioritized over marginal gains in processing speed.[1] This hybrid strategy maximizes the data acquisition success rate, which is the most critical

metric for this phase.

The following table provides a concise summary of the technological decision-making process, justifying the selection of a hybrid architecture.

Table 1: Data Acquisition Technology Matrix

Criterion	requests + BeautifulSoup	Selenium / Playwright
Primary Use Case	Fast parsing of static HTML and XML documents.[24]	Interaction with dynamic web applications that render content using JavaScript.[24]
Speed	High. Significantly faster due to the absence of browser rendering overhead.[1]	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.[1]	Full support. Can execute JavaScript and access the fully rendered page.[1]
Resource Intensity	Low. Minimal CPU and memory consumption.[1]	High. Consumes significant CPU and RAM, comparable to running a graphical browser.[1]
Setup Complexity	Low. Requires simple pip install of libraries.[1]	Higher. Requires installation of browser drivers (e.g., ChromeDriver) and more complex configuration.[1]
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.

2.2 Isolating the Signal: Advanced Content Extraction

Once the raw HTML of a page is successfully acquired, the next critical 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.[1] This step is

crucial because feeding noisy, irrelevant text to the LLM can introduce spurious entities and relationships, degrading the quality and accuracy of the final knowledge graph.

While several libraries exist for this purpose, research and benchmarks consistently point to trafilatura as the superior choice for general-purpose main content extraction.[1] It has been benchmarked as a leader in the field, achieving a high F1-score that indicates an excellent balance between precision (the accuracy of the text it extracts) and recall (the completeness of the text it extracts).[1] For KG construction, where the goal is to build a comprehensive representation of a document's knowledge, high recall is vital to ensure that no part of the core content is inadvertently missed. In contrast, alternative libraries like

goose3 have been shown to have significantly poorer recall, making them less suitable for this task.[1]

The recommended implementation is to pass the raw HTML content obtained from the hybrid scraper directly to the trafilatura library's extract() function. This will yield the highest quality clean text, stripped of boilerplate and ready for the LLM extraction phase.[1]

2.3 Handling Complex Formats: The Case of Financial PDFs

Real-world knowledge is not confined to HTML. Financial reports, academic papers, and legal documents are often distributed as PDFs, a format that poses unique challenges for information extraction.[1] Standard text extraction tools often struggle with complex, multi-column layouts and, most importantly, fail to extract the structured data locked within tables, which are ubiquitous in financial and scientific documents.[4]

To address this, the research plan must incorporate specialized PDF parsing libraries designed for AI and LLM workflows. Tools such as PyMuPDF4LLM (or alternatives like LLMSherpa as mentioned in the project abstract [1]) are engineered specifically for this purpose. These libraries go beyond simple text extraction; they can intelligently parse the document structure, identify tables, and convert their contents into a machine-readable format like Markdown.[33] The Markdown representation preserves the tabular structure (rows and columns), making it an ideal input format for an LLM,

which can then be prompted to understand and extract the relationships contained within the table.

The process for handling a PDF would be as follows:

1. Ingest the PDF file.
2. Use a library like PyMuPDF4LLM to process the document.
3. The library converts the PDF's textual content and its embedded tables into a clean, structured Markdown string.
4. This Markdown string is then passed to the same downstream information extraction pipeline used for the cleaned HTML content, creating a unified workflow for all source types.

Part III: Core Information Extraction: From Text to Triples

With clean, high-quality text prepared from diverse sources, the next phase of the pipeline focuses on the first stage of knowledge creation: using LLMs to transform the unstructured text into structured data. The goal of this phase is to perform high-fidelity Open Information Extraction (OIE), generating a comprehensive set of raw (Subject, Predicate, Object) triples that will serve as the foundation for the subsequent schema discovery process.

3.1 Advanced Prompt Engineering for High-Fidelity Triple Extraction

The prompt is the primary instrument for controlling the LLM's behavior. A poorly designed prompt will yield noisy, inaccurate, and inconsistent results, while a well-crafted prompt can elicit high-quality, structured output. To achieve the best possible performance in triple extraction, a composite prompt structure that integrates several proven techniques is essential.

An optimal prompt for this task will include the following components:

- **Role-Playing:** The prompt should begin by assigning a role to the LLM, such as, "You are an expert knowledge graph engineer tasked with extracting entities and their relationships from a given text".[1] This primes the model to adopt the persona

and leverage the knowledge associated with that role.

- **Few-Shot Learning (In-Context Learning):** Instead of relying on zero-shot instructions alone, the prompt must include a small number (typically 2-5) of high-quality examples that demonstrate the desired input-to-output transformation.[1] Showing the model concrete examples of a text and the corresponding correct triple extractions significantly improves its accuracy, consistency, and adherence to the desired format.[1]
- **Chain-of-Thought (CoT) Reasoning:** To enhance the model's logical reasoning and reduce errors, the prompt should guide it to "think step by step".[1] For relation extraction, this involves instructing the model to follow a specific process: first, identify all candidate entities in the text; second, analyze the linguistic context connecting pairs of entities; and third, deduce the relationship and formulate the final triple. This externalized reasoning process leads to more logically sound extractions.[1]
- **Clear Output Specification:** The prompt must be unambiguous about the desired output format. It should explicitly request a list of triples, each formatted as `` or a similar machine-readable structure.[20]

By combining these elements into a single, comprehensive prompt, the system can maximize the fidelity of the initial, open-ended information extraction process.

3.2 Ensuring Structural Integrity: Schema Enforcement and Validation

A frequent and critical failure mode of LLMs in extraction tasks is producing output that is semantically correct but syntactically flawed.[1] The model might return a list of triples with missing quotation marks, trailing commas, or extraneous conversational text (e.g., "Here are the triples you requested:..."). Such inconsistencies will break any automated downstream parsing logic and halt the pipeline.[34] To build a robust and reliable system, the LLM's output must be programmatically constrained and validated.

The most effective modern solution for this challenge is to leverage the function-calling or JSON-mode capabilities of LLM APIs, in conjunction with a data validation library like Pydantic. This creates a powerful, self-correcting loop:

1. **Define Pydantic Models:** First, define the exact desired output structure using standard Python classes and type hints. For this task, one would create classes

such as Node(id: str, label: str), Relationship(source: str, target: str, label: str), and a container class KnowledgeGraph(nodes: List[Node], relationships: List).[1] These models serve as a formal "data contract."

2. **Generate and Inject JSON Schema:** Frameworks like LangChain or the specialized instructor library can automatically convert these Pydantic models into a corresponding JSON Schema. This schema is then passed to the LLM API along with the prompt.[1] This forces the LLM to generate a response that is a syntactically valid JSON object conforming precisely to the specified structure.
3. **Parse and Validate:** The JSON output from the LLM is then parsed directly into an instance of the corresponding Pydantic model. Pydantic automatically performs runtime validation, checking that all fields are present and that their data types are correct.[1] If validation fails, it raises a detailed error, which can even be used to re-prompt the model with a correction request, thus guaranteeing the structural integrity of the data entering the pipeline.

3.3 Orchestration with LangChain's LLMGraphTransformer

While the components above can be implemented manually, frameworks like LangChain provide pre-built abstractions that significantly streamline the development process. For this project, a key component is the LLMGraphTransformer, which is designed specifically for converting text into graph structures.[1]

The LLMGraphTransformer takes a list of LangChain Document objects (which would contain the cleaned text from the ingestion pipeline in Part II) and uses a specified LLM to automatically convert them into GraphDocument objects. Each GraphDocument is a container for the nodes and relationships that the LLM has extracted from the source text.[1]

A crucial feature of this transformer is its configurability. It allows the user to provide lists of allowed_nodes and allowed_relationships to constrain the extraction to a predefined ontology.[1] However, for the initial phase of this research plan—which precedes schema discovery—these constraints will be intentionally left open. This allows the transformer to function as a powerful tool for Open Information Extraction (OIE), capturing all potential entities and relationships as the LLM identifies them.[38] The resulting raw, un-normalized graph data will then become the input for the schema discovery module detailed in Part IV. This approach decouples the problem neatly: first, capture as much information as possible in an open-ended manner;

second, impose structure and normalization on that captured information.

3.4 Managing Long-Form Content with Context-Aware Chunking

A significant practical challenge is that many source documents, such as long-form articles or detailed financial reports, will exceed the context window of the target LLM.[1] An 8,000-token context window, for example, is insufficient for a 20-page PDF. This makes it impossible to process the document in a single pass and necessitates a text chunking strategy.

The choice of chunking strategy has a direct impact on the quality of relation extraction. Several methods exist, with varying levels of sophistication:

- **Fixed-Length Chunking:** The simplest method, which splits text every N characters or tokens. This is strongly discouraged as it has no semantic awareness and frequently cuts sentences, phrases, and ideas in half, destroying the local context required for the LLM to accurately identify relationships.[1]
- **Recursive Character Text Splitting:** A much-improved strategy, popular in LangChain, which attempts to split text hierarchically along semantic boundaries. It tries to split first by paragraph separators (\n\n), then by sentence separators (.), and only then by words. This approach does a much better job of keeping semantically related text grouped together.[1]
- **Semantic Chunking:** The most advanced strategy. This method involves generating vector embeddings for each sentence and then grouping adjacent sentences that are semantically similar. A new chunk is created when the semantic distance between consecutive sentences exceeds a certain threshold. This produces the most contextually coherent chunks but is more computationally intensive.[1]

For this project, **Recursive Character Text Splitting** offers the best balance of implementation simplicity and effectiveness. Regardless of the method chosen, it is crucial to implement an **overlap** between chunks, where a portion of the end of one chunk is repeated at the beginning of the next. This overlap ensures that the LLM has sufficient surrounding context to resolve coreferences (e.g., understanding that "he" in chunk 2 refers to a person named in chunk 1) and to capture relationships that span across chunk boundaries.[1] The final aggregation of graph fragments from each chunk must then include an entity resolution step to merge duplicate nodes that refer to the

same real-world entity.

Part IV: The Frontier of Automation: Dynamic Schema Discovery and Refinement

This section addresses the intellectual core of the research plan: moving beyond simple triple extraction to the automated discovery and refinement of the knowledge graph's schema. This task, known as Ontology Learning (OL), has traditionally been the most significant bottleneck in KG construction, requiring extensive manual effort from domain experts.[15] The methodology proposed here leverages a hybrid approach, using statistical techniques to find latent structure in the data and LLMs to provide the semantic interpretation of that structure.

4.1 From Extraction to Induction: The Ontology Learning Problem

Ontology Learning is the process of automatically or semi-automatically constructing a formal ontology—including concepts, relationships, and axioms—from various data sources, most commonly unstructured text.[15] This stands in stark contrast to traditional KG pipelines, which operate on the assumption that a comprehensive, manually-crafted ontology already exists.[43] The goal of this research is to integrate OL as a dynamic, data-driven step within the KG construction workflow.

The output from Part III is a large, noisy set of raw triples generated via Open Information Extraction. This set contains a multitude of entity names (e.g., "Apple," "Tim Cook," "California") and relationship phrases (e.g., "is CEO of," "leads," "is based in"). However, these elements lack formal types and canonicalization. The task of this phase is to induce a coherent schema from this raw material. The proposed approach achieves this by first using unsupervised machine learning to discover statistical patterns (clusters) in the extracted data, and then employing LLMs to perform the higher-level cognitive task of labeling and refining these patterns into a meaningful ontology.

4.2 Inducing Entity Types via Unsupervised Clustering

The first step in schema induction is to discover the types of entities present in the corpus. The raw extraction process identifies entity names but not their categories (e.g., it finds "Tim Cook" but does not know it is a Person). The following methodology can be used to discover these types automatically:

1. **Generate Contextual Entity Embeddings:** For each unique entity string extracted in Part III, a rich contextual vector embedding must be generated. This is not simply an embedding of the entity name itself, but an embedding that captures its usage. A robust method is to take the source sentence(s) in which the entity appeared and pass them through a sentence-transformer model (e.g., from the Sentence-BERT family) to generate a vector that represents the entity in its original context.
2. **Cluster Entity Embeddings:** The resulting set of entity embeddings is then clustered using an unsupervised algorithm like K-Means or, more robustly, HDBSCAN. The core hypothesis is that entities of the same semantic type will be used in similar contexts, and therefore their embeddings will form distinct clusters in the vector space.[44] For example, the embeddings for "Tim Cook," "Satya Nadella," and "Elon Musk" should cluster together, separately from the cluster containing "Apple," "Microsoft," and "Tesla."
3. **LLM-Powered Cluster Labeling:** The output of the clustering step is a set of unlabeled groups of entities. The final, crucial step is to use an LLM to assign a meaningful semantic type label to each cluster. This is achieved by prompting the LLM with a representative sample of entities from a single cluster. For example, the prompt could be: "The following is a list of related items:. What is the common type or category for these items? Provide a single, concise label in PascalCase (e.g., 'Person', 'Organization', 'City')." This leverages the LLM's vast world knowledge to interpret the statistical groupings found by the clustering algorithm and assign them a human-understandable type.

4.3 Defining the Lexicon of Relationships

A similar process is applied to the relation phrases extracted during OIE. The raw output will contain many redundant and stylistically varied phrases that refer to the same underlying semantic relationship (e.g., "is CEO of," "leads the company," "is the

chief executive of").[43] These must be canonicalized into a clean, consistent set of relationship types (e.g.,

hasCEO).

1. **Generate Relation Embeddings:** For each unique relation phrase, generate a contextual embedding. This can be derived from the embedding of the full sentence in which the relation appeared.
2. **Cluster Relation Embeddings:** Apply a clustering algorithm to the set of relation embeddings. The goal is to group semantically equivalent phrases together.[48] The phrases "is based in," "is headquartered in," and "has its main office in" should ideally fall into the same cluster.
3. **LLM-Powered Canonicalization and Labeling:** For each resulting cluster of relation phrases, use an LLM to generate a single, canonical predicate name. The prompt would provide the LLM with the list of phrases from the cluster and ask it to synthesize a standardized label. For instance: "The following phrases all describe a similar relationship: ['is CEO of', 'leads the company', 'is the chief executive of']. Propose a single, concise, canonical predicate name for this relationship in a standardized format like camelCase or snake_case (e.g., 'hasCEO' or 'has_ceo')." This step cleans and formalizes the relational lexicon of the new ontology.

The following table compares the proposed hybrid approach for schema discovery with the alternative of using an LLM to directly generate an ontology from text.

Table 2: Schema Discovery Technique Comparison

Approach	Description	Strengths	Weaknesses
Unsupervised Clustering + LLM Labeling	Use statistical clustering on embeddings of raw extractions to find latent structure (groups of similar entities/relations). Then, use an LLM's semantic knowledge to assign meaningful labels to these clusters.[44]	Highly data-driven; discovers patterns and categories directly from the corpus content. More transparent, as statistical groupings can be inspected. Less prone to large-scale hallucination than direct generation.	More complex, two-stage process. The quality of the final schema is highly dependent on the quality of the initial embeddings and the effectiveness of the clustering algorithm.

Direct LLM Ontology Generation	Prompt a powerful LLM to read a document or a set of documents and directly generate a complete ontology (classes, properties, hierarchies) in a single step.[15]	Conceptually simple and can be implemented with a single, complex prompt. Can capture high-level conceptual structures that might be missed by bottom-up clustering.	Highly susceptible to LLM hallucination, where the model may invent concepts or relationships not supported by the text. The process is a "black box," making it difficult to trace why a particular schema element was created. May miss fine-grained details present in the text.

4.4 LLM-driven Schema Refinement and Normalization

The clustering and labeling steps produce a flat list of entity and relation types. The final stage of schema discovery involves refining this list into a coherent ontology and using it to normalize the extracted data.

- **Taxonomy Induction:** A key feature of a robust ontology is a hierarchical structure of its concepts (a taxonomy). An LLM can be used to arrange the discovered entity types into this structure. For example, given the flat list of types [Person, CEO, Company, Employee, Founder], a prompt can instruct the LLM to organize them using subClassOf relationships: "Given the following list of entity types: [...], arrange them into a hierarchical 'is-a' or 'subClassOf' structure. For example, a CEO is a type of Employee." This can produce a taxonomy like CEO subClassOf Employee, Founder subClassOf Person, Employee subClassOf Person.[15]
- **Ontology Alignment:** When processing data from many different sources, the system may induce multiple, slightly different schemas. For instance, one set of documents might lead to a Corporation entity type, while another leads to a Company type. Ontology alignment is the task of identifying and merging these semantically equivalent concepts. LLMs are well-suited for this task, as they can be prompted to assess the semantic similarity between two concepts or relations from different schemas and propose a mapping.[22]
- **Entity Normalization and Resolution:** This is the critical process of mapping all textual mentions of a single real-world entity to a single, canonical node in the

graph. For example, the strings "IBM," "International Business Machines," and "I.B.M." should all resolve to the same unique Company node. This is a complex disambiguation task. LLMs, provided with the context of each mention, can be prompted to determine if different strings refer to the same underlying entity, leveraging their vast knowledge base to assist in this resolution.[52] This process is essential for creating a clean, interconnected graph rather than a fragmented collection of disconnected mentions.

Part V: An Advanced Architectural Model: Multi-Agent Systems for KG Construction

While a linear pipeline (Ingest -> Extract -> Discover Schema) provides a logical starting point, it fails to capture the inherently cyclical and iterative nature of knowledge graph construction and schema refinement. A more sophisticated and powerful architectural paradigm is that of a multi-agent system, where a team of collaborative, LLM-powered agents work together to build and refine the graph.

5.1 Limitations of a Linear Pipeline

A strictly linear, one-shot pipeline suffers from a critical flaw: error propagation and a lack of feedback. The quality of the discovered schema in Part IV is entirely dependent on the quality of the raw extractions from Part III. If the initial Open Information Extraction is poor, the resulting schema will be flawed. Conversely, a well-defined schema can be used to guide and constrain the extraction process, leading to much higher accuracy than unconstrained OIE. For example, once the system knows that the foundedBy relationship connects a Company to a Person, it can filter out erroneous extractions that violate this rule. A linear pipeline struggles to accommodate this feedback loop, where an improved schema should inform a new, improved round of extraction.

5.2 The Multi-Agent Paradigm

A multi-agent system addresses this limitation by decomposing a complex problem into a set of specialized tasks, each handled by an autonomous agent.[54] In this context, an "agent" is an LLM given a specific role, a persona, a set of tools (e.g., functions for web search, database queries), and a goal. These agents can communicate and collaborate, passing information back and forth to iteratively solve the problem.

This architecture is exceptionally well-suited for KG construction because the overall task can be naturally broken down into distinct yet interdependent sub-tasks like discovery, extraction, alignment, and verification.[57] Frameworks like LangGraph are explicitly designed to facilitate the creation of such cyclical, stateful workflows, making the implementation of multi-agent systems more accessible.[59]

5.3 A Proposed Agent-Based Architecture (Inspired by KARMA)

Drawing inspiration from frameworks like KARMA [54] and other multi-agent research [55], this plan proposes a system composed of several specialized, collaborative agents:

- **Central Controller Agent:** This agent acts as the orchestrator of the entire system. It receives new documents, maintains a queue of tasks, prioritizes them based on utility or uncertainty, and routes them to the appropriate specialist agent. It is the "project manager" of the agent team.
- **Discovery Agent:** This agent's primary function is to perform the initial schema induction. It takes a representative sample of the input corpus, performs the full process outlined in Part IV (raw extraction, clustering, labeling, taxonomy induction), and produces the first version of the domain ontology (Schema v1.0).
- **Extraction Agent:** This agent is a specialist in information extraction. It receives the current version of the schema (initially v1.0) from the Controller. It then processes documents from the main corpus, performing *schema-guided* extraction. Its prompts are constrained by the known entity and relation types, which makes its output significantly more accurate and consistent than the initial open-ended extraction.
- **Schema Alignment Agent:** As the Extraction Agent processes a wider range of documents, it will inevitably encounter entities or relationships that do not fit the current schema. When this happens, the item is passed to the Schema Alignment

Agent. This agent's job is to determine if the new item is simply a synonym or variation of an existing concept (in which case it updates the normalization rules) or if it represents a genuinely novel concept. If the concept is novel, this agent proposes an update to the schema (e.g., adding a new entity type or relationship), which, upon approval by the Controller, leads to Schema v2.0.[54]

- **Conflict Resolution & Verification Agent:** This agent acts as the fact-checker and quality control specialist. Its role is twofold. First, it is tasked with resolving conflicts. If two different source documents produce contradictory information (e.g., one states a company was founded in 1998, another says 1999), this agent can be triggered to investigate the sources (e.g., checking their known reliability) and attempt to resolve the discrepancy. Second, it serves as a crucial defense against LLM hallucination. By cross-referencing extracted facts against other parts of the graph or even external sources, it can verify the plausibility of new information before it is permanently integrated.[14]

This multi-agent architecture transforms the KG construction process from a static, linear pipeline into a dynamic, self-improving ecosystem. It creates a feedback loop where the schema and the data are refined in tandem, addressing the "chicken-and-egg" problem and leading to a more robust, accurate, and scalable solution for automated knowledge discovery.

Part VI: Implementation Strategy and Technical Architecture

This section provides a concrete technical blueprint for building the proposed system, detailing the specific tools, services, and strategies required to translate the conceptual framework into a functional implementation. The recommendations prioritize cost-effectiveness, flexibility, and methodological rigor, making them suitable for a dissertation-level research project.

6.1 LLM Selection and Deployment: Local vs. API

Access to a powerful LLM is the cornerstone of the entire system. For a researcher operating under typical budget constraints, two primary avenues provide access at

little to no cost.[1] The choice between them involves a critical trade-off between privacy, control, and raw capability.

- **Local Execution with Open-Source Models:** The most direct path to zero recurring costs and maximum data privacy is to run open-source LLMs locally on consumer-grade hardware.[1] This is made feasible by two key technologies:
  - **Orchestration Tools:** Tools like Ollama and LM Studio dramatically simplify the process of downloading, managing, and serving local LLMs. They provide an OpenAI-compatible API endpoint, allowing for seamless integration with frameworks like LangChain.[1]
  - **Quantization:** This is the enabling technology that makes local execution practical. Quantization reduces the numerical precision of a model's weights (e.g., from 16-bit floats to 4-bit integers), drastically shrinking its memory footprint.[65] A model in the **GGUF** format is particularly versatile, as it is optimized for CPU inference but can offload layers to a GPU if available. This allows powerful models (e.g., 7-8 billion parameters) to run effectively on a machine with 16 GB of RAM and a GPU with 6-8 GB of VRAM.[1] Recommended starting models include Meta's Llama-3-8B, Mistral-7B, and Microsoft's Phi-3.
- **Leveraging Free-Tier Commercial APIs:** As an alternative or for benchmarking purposes, several commercial providers offer generous free API tiers that provide access to state-of-the-art proprietary models. The **Google Gemini 1.5 Flash API** is a particularly compelling option. Its free tier offers high daily request limits that are more than sufficient to process a dissertation-scale corpus of several hundred documents at zero cost, provided usage stays within the specified limits.[1] This provides access to a potentially more capable model without financial outlay.

The following table summarizes the critical trade-offs between these two access modalities.

Table 3: LLM Access Modality Trade-Offs

Factor	Local Execution (via Ollama)	Free API Tiers (e.g., Gemini 1.5 Flash)
Cost	Effectively zero. No per-request fees.[18]	Free within specified rate and usage limits. Risk of fees if limits are exceeded.[45]

Data Privacy	Maximum. Data and prompts never leave the local machine.[18]	Lower. Data is sent to a third-party provider and is subject to their privacy policies.[1]
Model Capability	Limited to smaller, open-source models (typically 3B to 13B parameters on consumer hardware).[1]	Access to large, state-of-the-art proprietary models that may not be publicly available.[1]
Inference Speed	Dependent on local hardware. Can be slow on CPU-only systems.[1]	Generally very fast due to optimized, data-center-grade hardware. Subject to network latency.[15]
Setup & Maintenance	Requires one-time setup of tools (Ollama), model downloads, and local resource management.[19]	Simpler. Requires signing up for an API key and managing credentials.[44]
Offline Use	Fully functional without an internet connection once models are downloaded.	Requires a constant and stable internet connection to make API calls.

To ensure methodological rigor, the recommended approach is to conduct a small-scale pilot study. The researcher should process a representative sample of 10-20 documents using both a locally-run quantized model (e.g., llama3:8b-instruct.gguf) and the Gemini 1.5 Flash API. The resulting sets of extracted triples should be manually evaluated for precision and recall. This project-specific empirical comparison will provide a much stronger justification for the final choice of LLM than relying solely on generic external benchmarks.[1]

6.2 A Dual-Mode Storage Strategy: NetworkX and Neo4j

The knowledge graph, once extracted, must be stored. The choice of storage technology significantly impacts the development workflow and the final analytical capabilities. A single solution is not optimal for the entire research lifecycle, which consists of distinct phases of development and analysis. Therefore, a two-stage storage strategy is proposed.[1]

- **Stage 1: Prototyping and Development with NetworkX:** During the initial phase, which involves heavy development, experimentation, and rapid iteration of the extraction and schema discovery pipelines, the in-memory Python library NetworkX is the ideal choice. It is lightweight, requires no external database setup (pip install is sufficient), and its simple Python API allows for extremely fast iteration cycles.[1] The entire pipeline can be re-run and the resulting graph can be immediately analyzed programmatically within the same script, avoiding the overhead of connecting to and managing an external database. This agility is invaluable during the experimental phase.[67]
- **Stage 2: Finalization and Analysis 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 persistent, transactional graph database. Neo4j (using the free Community Edition) is the industry standard and an excellent choice.[33] It provides a robust server environment, a powerful and declarative query language (Cypher) for complex graph traversals, and rich, interactive visualization tools like Neo4j Bloom that allow for no-code exploration of the final graph.[1] This creates a persistent, queryable, and demonstrable artifact for the final analysis and presentation of the research.

For efficient data loading into Neo4j, it is critical to perform a batch insertion within a single transaction rather than creating nodes and relationships one by one. 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 the MERGE clause to create the graph elements idempotently.[1]

The following table compares these two storage solutions in the context of a research prototype.

Table 4: Graph Storage Solution Comparison

Feature	NetworkX	Neo4j Community Edition
Architecture	In-memory Python library. Graph exists only in the script's RAM.[67]	Standalone, persistent graph database server with on-disk storage.[70]
Setup & Dependencies	Simple pip install networkx. No external services required.[1]	Requires installation of Neo4j Desktop application; management of a database

		instance.[71]
Data Persistence	Not persistent by default. Must be explicitly saved to/loaded from a file.[67]	Fully persistent and transactional (ACID-compliant). Data is safely stored on disk.[72]
Querying Mechanism	Programmatic. Queries are written as Python code iterating over nodes/edges.[32]	Declarative. Uses the powerful Cypher query language to express complex graph patterns.[24]
Performance	Extremely fast for analysis as long as the graph fits in RAM.[67]	Optimized for querying large, on-disk graphs. Incurs driver/network overhead for Python interaction.[72]
Visualization	Basic static plotting via libraries like matplotlib.[32]	Rich, interactive visualization via built-in tools like Neo4j Browser and Neo4j Bloom.[26]
Project Role	**Ideal for Prototyping:** Best for rapid development and iteration during the pipeline-building phase.[67]	**Ideal for Finalization:** Best for persistent storage, ad-hoc querying, and visualization of the final KG artifact.[33]

6.3 The End-to-End Orchestrated Workflow

The final step in implementation is to integrate all the preceding components into a single, automated pipeline. Modern orchestration frameworks like LangChain and its more explicit, state-machine-based counterpart, LangGraph, are designed precisely for this purpose.[59]

The logical flow of the final, orchestrated script, particularly if adopting the advanced multi-agent architecture, would be managed by LangGraph. LangGraph excels at creating cyclical graphs where agents can pass state back and forth, making it a perfect fit for the iterative refinement process. The high-level workflow would be:

1. **Input:** The system takes a list of URLs and/or file paths as its initial input.
2. **Ingestion:** A data ingestion function, implementing the hybrid scraper and trafilatura cleaning, processes each source to produce clean text.

3. **Agentic Loop (Orchestrated by LangGraph):**
    - The Central Controller agent receives the clean text.
    - It routes a sample of the text to the Discovery Agent to generate an initial schema (v1.0).
    - The schema is stored in a shared state. The Extraction Agent is invoked, using the schema to perform guided extraction on new documents.
    - If the Extraction Agent encounters concepts that violate the schema, it passes them to the Schema Alignment Agent.
    - The Schema Alignment Agent proposes updates, which are reviewed and integrated by the Controller, potentially creating a new schema version (v2.0).
    - Conflicting facts are routed to the Conflict Resolution Agent for verification.
4. **Storage:** As verified and normalized triples are produced, they are collected. Periodically, or at the end of a run, they are passed to the batch ingestion function to populate the persistent Neo4j database.

This architecture, orchestrated by LangGraph, provides a demonstrable, end-to-end pipeline that fulfills the core research objective: transforming a collection of unstructured documents into a high-quality knowledge graph with an emergent, dynamically refined schema.

Part VII: Evaluation, Challenges, and Future Directions

A comprehensive research plan must not only detail the construction of a system but also provide a rigorous framework for its evaluation, anticipate inherent challenges, and outline avenues for future work.

7.1 A Multi-faceted Evaluation Framework

Evaluating a system that generates its own schema is inherently more complex than evaluating a standard information extraction system against a fixed, predefined ontology. A simple measure of triple correctness is insufficient, as it does not capture the quality of the discovered schema itself. Therefore, a two-level evaluation framework is proposed.

- **Level 1: Triple-level Correctness (Extraction Quality):** The fundamental accuracy of the information extraction component must be measured. This requires the manual creation of a "gold standard" dataset by annotating a small, representative subset of the source corpus with all correct entities and relationships. Against this golden set, the system's extracted triples can be evaluated using standard information retrieval metrics [2]:
    - **Precision:** Of the triples the system extracted, what fraction are correct?
    - **Recall:** Of all the correct triples present in the text, what fraction did the system extract?
    - **F1-Score:** The harmonic mean of precision and recall, providing a single, balanced measure of extraction accuracy.
- **Level 2: Schema-level Quality (Ontology Quality):** This level assesses the quality of the emergent ontology, which is a more qualitative and task-oriented endeavor. Several methods can be employed:
    - **Schema Coherence and Utility:** A human domain expert (or the researcher acting as one) evaluates the discovered ontology for its logical consistency, clarity, and usefulness. Are the entity types meaningful? Are the relationship labels unambiguous? Does the taxonomy make logical sense?
    - **Task-Based Evaluation:** The ultimate test of a knowledge graph is its utility. A set of complex questions that require multi-hop reasoning or information synthesis should be designed. The generated KG is then used to answer these questions (e.g., via a GraphRAG system). The accuracy and completeness of the answers serve as an excellent end-to-end, task-based proxy for the overall quality of the graph and its underlying schema.
    - **Comparison to Existing Ontologies:** For well-defined domains like finance or biomedicine, public or proprietary ontologies may already exist. The automatically discovered schema can be compared to these reference ontologies using ontology alignment metrics to measure its overlap and consistency with established expert knowledge.

7.2 Anticipating and Mitigating Inherent Challenges

Building such a system is fraught with challenges. A proactive research plan must anticipate these issues and incorporate mitigation strategies.

- **LLM Hallucination:** LLMs are prone to generating plausible but false information.[6] This is the most significant risk to the integrity of the KG. Mitigation

strategies include:
    - Grounding all extractions in the source text, a core principle of Retrieval-Augmented Generation (RAG).
    - Employing a dedicated Conflict Resolution & Verification Agent (as described in Part V) to cross-reference facts.[54]
    - Enforcing strictly structured, non-conversational output using Pydantic models to reduce the likelihood of conversational filler and invented facts.[1]
- **Entity Ambiguity and Resolution:** Natural language is inherently ambiguous. An entity name like "Apple" can refer to a technology company or a fruit. The process of entity resolution—correctly disambiguating and linking all mentions to the correct real-world entity—is a non-trivial challenge and a critical area of focus for the Schema Alignment Agent.[21]
- **Data Sparsity and Noise:** The principle of "garbage in, garbage out" applies. The quality of the final KG is directly dependent on the quality and information density of the source documents. Some documents may be too sparse or noisy to yield meaningful extractions, and the system must be robust to this.[3]
- **Temporal Drift and Information Obsolescence:** Knowledge is not static; it changes over time. A CEO may leave a company, financial data becomes outdated, and technical procedures evolve. A KG built from a static corpus is a snapshot in time. The proposed system does not inherently address real-time updates. This limitation must be acknowledged, and mechanisms for dynamic KG updating and managing the timeliness of information should be flagged as a key area for future research.[47]

7.3 Avenues for Future Research

The successful implementation of this research plan would serve as a powerful foundation for numerous future explorations.

- **Human-in-the-Loop Schema Refinement:** The fully automated system could be augmented with a user interface that allows a human domain expert to interact with the schema discovery process. The expert could review the agent-proposed schema, correct errors, merge concepts, or provide guidance, creating a collaborative human-AI system that combines the scale of automation with the precision of expert knowledge.[1]
- **Multi-Modal Knowledge Graphs:** The current plan focuses exclusively on textual and tabular data. A significant extension would be to incorporate multi-modal

information extraction. This could involve using vision-language models to extract information from images, charts, and diagrams within documents, or leveraging layout-aware models to understand the visual structure of a page, adding another layer of context to the KG.[32]

- **Dynamic and Streaming Knowledge Graph Updates:** The current architecture is primarily batch-oriented. A major area for future work is the development of mechanisms for the KG to be updated in near real-time as new documents are ingested or streamed into the system, rather than requiring a full, periodic rebuild. This would transform the KG from a static repository into a living, evolving knowledge base.

Works cited

1. Knowledge Graph Construction Research Plan
2. Open Information Extraction for Knowledge Graph Construction | Request PDF, accessed July 20, 2025, https://www.researchgate.net/publication/344329177_Open_Information_Extraction_for_Knowledge_Graph_Construction
3. Challenges and Advances in Information Extraction From Scientific Literature: A Review - OSTI, accessed July 20, 2025, https://www.osti.gov/servlets/purl/1869908
4. (PDF) Challenges and Advances in Information Extraction from ..., accessed July 20, 2025, https://www.researchgate.net/publication/355098954_Challenges_and_Advances_in_Information_Extraction_from_Scientific_Literature_a_Review
5. en.wikipedia.org, accessed July 20, 2025, https://en.wikipedia.org/wiki/Knowledge_graph
6. What Is a Knowledge Graph? | IBM, accessed July 20, 2025, https://www.ibm.com/think/topics/knowledge-graph
7. www.ibm.com, accessed July 20, 2025, https://www.ibm.com/think/topics/knowledge-graph#:~:text=This%20information%20is%20usually%20stored,the%20relationship%20between%20the%20nodes.
8. An Introduction to Knowledge Graphs - TextMine, accessed July 20, 2025, https://textmine.com/post/an-introduction-to-knowledge-graphs
9. www.schemaapp.com, accessed July 20, 2025, https://www.schemaapp.com/schema-markup/the-anatomy-of-a-content-knowledge-graph/#:~:text=At%20its%20simplest%20form%2C%20a,delineating%20the%20relationships%20between%20them.
10. Knowledge Graphs 101: How Nodes and Edges Connect All the World's Real Estate Data, accessed July 20, 2025, https://blog.cherre.com/2022/04/08/knowledge-graphs-101-how-nodes-and-edges-connect-all-the-worlds-real-estate-data/
11. What is a Knowledge Graph? - Onlim, accessed July 20, 2025, https://onlim.com/en/what-is-a-knowledge-graph/

12. Knowledge graphs | The Alan Turing Institute, accessed July 20, 2025, https://www.turing.ac.uk/research/interest-groups/knowledge-graphs
13. What are the main components of a knowledge graph? - Milvus, accessed July 20, 2025, https://milvus.io/ai-quick-reference/what-are-the-main-components-of-a-knowledge-graph
14. Knowledge Graphs and Their Reciprocal Relationship with Large Language Models - MDPI, accessed July 20, 2025, https://www.mdpi.com/2504-4990/7/2/38
15. Ontology Learning from Text: an Analysis on LLM Performance - CEUR-WS.org, accessed July 20, 2025, https://ceur-ws.org/Vol-3874/paper5.pdf
16. The Role of Ontologies with LLMs - Enterprise Knowledge, accessed July 20, 2025, https://enterprise-knowledge.com/the-role-of-ontologies-with-llms/
17. Information extraction with LLMs using Amazon SageMaker JumpStart | Artificial Intelligence, accessed July 20, 2025, https://aws.amazon.com/blogs/machine-learning/information-extraction-with-llms-using-amazon-sagemaker-jumpstart/
18. Zero-Shot Prompting - Prompt Engineering Guide, accessed July 20, 2025, https://www.promptingguide.ai/techniques/zeroshot
19. [2402.11142] Grasping the Essentials: Tailoring Large Language Models for Zero-Shot Relation Extraction - arXiv, accessed July 20, 2025, https://arxiv.org/abs/2402.11142
20. Small-size LLMs: Open Information Extraction | by Yitao Li | Medium, accessed July 20, 2025, https://medium.com/@yitaoli416/small-size-llms-open-information-extraction-b172155f8e92
21. Traditional NER vs LLMs: Dual Approaches to Building Knowledge Graphs, accessed July 20, 2025, https://www.bluetickconsultants.com/dual-approaches-to-building-knowledge-graphs-traditional-techniques-or-llms/
22. RELRaE: LLM-Based Relationship Extraction, Labelling ... - arXiv, accessed July 20, 2025, https://arxiv.org/pdf/2507.03829
23. NeurIPS Poster End-to-End Ontology Learning with Large Language Models, accessed July 20, 2025, https://neurips.cc/virtual/2024/poster/94942
24. Selenium vs. BeautifulSoup in 2025: Which to Choose? - Oxylabs, accessed July 20, 2025, https://oxylabs.io/blog/selenium-vs-beautifulsoup
25. Selenium versus BeautifulSoup for web scraping [closed] - Stack Overflow, accessed July 20, 2025, https://stackoverflow.com/questions/17436014/selenium-versus-beautifulsoup-for-web-scraping
26. Python Web Scraping Using Selenium and Beautiful Soup: A Step-by-Step Tutorial, accessed July 20, 2025, https://www.codecademy.com/article/web-scrape-with-selenium-and-beautiful-soup
27. Web Scraping Dynamic Sites With Both Selenium and Beautiful Soup(DIM) - Medium, accessed July 20, 2025,

https://medium.com/@georgemichaeldagogomaynard/web-scraping-dynamic-sites-with-both-selenium-and-beautiful-soup-dim-c679743018de
28. Web Scraping Tutorial Using Selenium & Python (+ examples) - ScrapingBee, accessed July 20, 2025, https://www.scrapingbee.com/blog/selenium-python/
29. Evaluation — Trafilatura 2.0.0 documentation, accessed July 20, 2025, https://trafilatura.readthedocs.io/en/latest/evaluation.html
30. Main Content Extraction from Web Pages - YTG Central, accessed July 20, 2025, https://central.yourtext.guru/main-content-extraction-from-web-pages/
31. Quickstart — Trafilatura 2.0.0 documentation - Read the Docs, accessed July 20, 2025, https://trafilatura.readthedocs.io/en/latest/quickstart.html
32. Information Extraction from Visually Rich Documents using LLM-based Organization of Documents into Independent Textual Segments - arXiv, accessed July 20, 2025, https://arxiv.org/html/2505.13535v1
33. Using PyMuPDF4LLM: A Practical Guide for PDF Extraction in LLM & RAG Environments, accessed July 20, 2025, https://medium.com/@danushidk507/using-pymupdf4llm-a-practical-guide-for-pdf-extraction-in-llm-rag-environments-63649915abbf
34. How to Use LLMs to Extract Document Information - 57Blocks, accessed July 20, 2025, https://57blocks.io/blog/how-to-use-llms-to-extract-document-information
35. Automate building knowledge graphs with LLMs and Neo4J — 2 ways - GoPenAI, accessed July 20, 2025, https://blog.gopenai.com/automate-building-knowledge-graphs-from-scientific-abstracts-with-llms-and-neo4j-2-ways-b9d5152a12da
36. Automating Knowledge Discovery from Scientific Literature via LLMs: A Dual-Agent Approach with Progressive Ontology Prompting - arXiv, accessed July 20, 2025, https://arxiv.org/html/2409.00054v1
37. Building Knowledge Graphs with LLM Graph Transformer | by Tomaz Bratanic - Medium, accessed July 20, 2025, https://medium.com/data-science/building-knowledge-graphs-with-llm-graph-transformer-a91045c49b59
38. Knowledge-Graph-Tutorials-and-Papers/topics/Open Information Extraction.md at master - GitHub, accessed July 20, 2025, https://github.com/heathersherry/Knowledge-Graph-Tutorials-and-Papers/blob/master/topics/Open%20Information%20Extraction.md
39. A Survey on Open Information Extraction from Rule-based Model to Large Language Model, accessed July 20, 2025, https://arxiv.org/html/2208.08690v6
40. SyRACT: zero-shot biomedical document-level relation extraction with synergistic RAG and CoT | Bioinformatics | Oxford Academic, accessed July 20, 2025, https://academic.oup.com/bioinformatics/advance-article/doi/10.1093/bioinformatics/btaf356/8169328
41. A Short Review for Ontology Learning: Stride to Large Language Models Trend - arXiv, accessed July 20, 2025, https://arxiv.org/html/2404.14991v2
42. Exploring large language models for ontology learning - International Association for Computer Information Systems, accessed July 20, 2025,

https://iacis.org/iis/2024/4_iis_2024_299-310.pdf
43. information extraction - Ontology-based knowledge graph construction from free-text, accessed July 20, 2025, https://datascience.stackexchange.com/questions/118727/ontology-based-knowledge-graph-construction-from-free-text
44. A Clustering-Oriented Method for Open-Domain Named Entity Recognition | OpenReview, accessed July 20, 2025, https://openreview.net/forum?id=ZsbHbm5mXk&referrer=%5Bthe%20profile%20of%20Hong%20Yao%5D(%2Fprofile%3Fid%3D~Hong_Yao1)
45. WebSets: Extracting Sets of Entities from the Web Using Unsupervised Information Extraction - CiteSeerX, accessed July 20, 2025, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=bec8ac8fd6cd4b4ee8724262840a899f88f426e2
46. Open Information Extraction with Global Structure Constraints - Jiawei Han, accessed July 20, 2025, http://hanj.cs.illinois.edu/pdf/www18_qzhu.pdf
47. Challenges from Information Extraction to Information Fusion, accessed July 20, 2025, https://blender.cs.illinois.edu/paper/challenge.pdf
48. A Relation-Oriented Clustering Method for Open Relation Extraction - ACL Anthology, accessed July 20, 2025, https://aclanthology.org/2021.emnlp-main.765.pdf
49. A Comprehensive Guide to Ontologies and Large Language Models - Docdigitizer, accessed July 20, 2025, https://www.docdigitizer.com/blog/ontologies-large-language-models-guide/
50. OLaLa: Ontology Matching with Large Language Models | Request PDF - ResearchGate, accessed July 20, 2025, https://www.researchgate.net/publication/376230467_OLaLa_Ontology_Matching_with_Large_Language_Models
51. Ontology Matching with Large Language Models and Prioritized Depth-First Search - arXiv, accessed July 20, 2025, https://arxiv.org/html/2501.11441v1
52. LLMs for Knowledge Graph 3: Challenges and Opportunities for GPT in KGs - GraphAware, accessed July 20, 2025, https://graphaware.com/blog/llms-for-knowledge-graph-3/
53. Entity Normalization As Part of Google's Knowledge Graph - Go Fish Digital, accessed July 20, 2025, https://gofishdigital.com/blog/entity-normalization/
54. KARMA: Leveraging Multi-Agent LLMs for Automated Knowledge Graph Enrichment - arXiv, accessed July 20, 2025, https://arxiv.org/html/2502.06472v1
55. From Unstructured Communication to Intelligent RAG: Multi-Agent Automation for Supply Chain Knowledge Bases - arXiv, accessed July 20, 2025, https://arxiv.org/pdf/2506.17484
56. From Unstructured Communication to Intelligent RAG: Multi-Agent Automation for Supply Chain Knowledge Bases - Amazon Science, accessed July 20, 2025, https://assets.amazon.science/ed/f6/3ac345ca40c1ba28ea0d6affcfea/from-unstructured-communication-to-intelligent-rag-multi-agent-automation-for-supply-chain-knowledge-bases.pdf
57. [Literature Review] KARMA: Leveraging Multi-Agent LLMs for Automated

Knowledge Graph Enrichment - Moonlight, accessed July 20, 2025, https://www.themoonlight.io/en/review/karma-leveraging-multi-agent-llms-for-automated-knowledge-graph-enrichment
58. KARMA: Leveraging Multi-Agent LLMs for Automated Knowledge Graph Enrichment, accessed July 20, 2025, https://powerdrill.ai/discover/summary-karma-leveraging-multi-agent-llms-for-automated-cm70yvllybobq07s3xrbumuie
59. LangGraph - LangChain, accessed July 20, 2025, https://www.langchain.com/langgraph
60. LLM-Powered Knowledge Graphs for Enterprise Intelligence ... - arXiv, accessed July 20, 2025, https://arxiv.org/pdf/2503.07993
61. Guide to Local LLMs - Scrapfly, accessed July 20, 2025, https://scrapfly.io/blog/posts/guide-to-local-llm
62. Run LLMs Locally with Ollama: Privacy-First AI for Developers in 2025 - Cohorte Projects, accessed July 20, 2025, https://www.cohorte.co/blog/run-llms-locally-with-ollama-privacy-first-ai-for-developers-in-2025
63. Build a RAG-Powered LLM Service with Ollama & Open WebUI : A Step-by-Step Guide | by S. Hassan Tabatabaei | Medium, accessed July 20, 2025, https://medium.com/@hassan.tbt1989/build-a-rag-powered-llm-service-with-ollama-open-webui-a-step-by-step-guide-a688ec58ac97
64. Step-by-Step Guide: How to Use LLMs on Your Local Machine Using Ollama - Medium, accessed July 20, 2025, https://medium.com/@munsifrazaofficial/step-by-step-guide-how-to-use-llms-on-your-local-machine-using-ollama-0f2b2a9dbb2a
65. Using Quantized Models with Ollama for Application Development - MachineLearningMastery.com, accessed July 20, 2025, https://machinelearningmastery.com/using-quantized-models-with-ollama-for-application-development/
66. Graph Cache-Augmented Generation: Enhancing Contextual LLMs with Neo4j and NetworkX [GraphCAG] | by Dr. Volkan OBAN | Medium, accessed July 20, 2025, https://medium.com/@drfolkan/graph-cache-augmented-generation-enhancing-contextual-llms-with-neo4j-and-networkx-graphcag-89c16f2da120
67. Demystifying Information Extraction using LLM - Towards AI, accessed July 20, 2025, https://towardsai.net/p/l/demystifying-information-extraction-using-llm
68. Fire up your Centrality Metric engines: Neo4j vs NetworkX — a drag race, of sorts... | by Kristof Neys | TDS Archive | Medium, accessed July 20, 2025, https://medium.com/data-science/fire-up-your-centrality-metric-engines-neo4j-vs-networkx-a-drag-race-of-sorts-18857f25be35
69. 15 Best Graph Visualization Tools for Your Neo4j Graph Database, accessed July 20, 2025, https://neo4j.com/blog/graph-visualization/neo4j-graph-visualization-tools/
70. [D] Entity Extraction with LLMs : r/MachineLearning - Reddit, accessed July 20, 2025,

https://www.reddit.com/r/MachineLearning/comments/1dwbcp5/d_entity_extraction_with_llms/
71. Build a Smart Web Scraper in Python Using Selenium & BeautifulSoup | by Arjun tewari, accessed July 20, 2025, https://medium.com/@arjuntewari0505/build-a-smart-web-scraper-in-python-using-selenium-beautifulsoup-c9d4ef656b2e
72. This Week in Neo4j - Neo4j vs NetworkX, Accessing Neo4j with Spring Boot 2.4, Image Annotation on GCP - Graph Database & Analytics, accessed July 20, 2025, https://neo4j.com/blog/twin4j/this-week-in-neo4j-neo4j-vs-networkx-accessing-neo4j-with-spring-boot-2-4-image-annotation-on-gcp/
73. Core functions — Trafilatura 2.0.0 documentation - Read the Docs, accessed July 20, 2025, https://trafilatura.readthedocs.io/en/latest/corefunctions.html
74. Knowledge Graph Construction: Extraction, Learning, and Evaluation - MDPI, accessed July 20, 2025, https://www.mdpi.com/2076-3417/15/7/3727