Research Report
LLM-based Knowledge Agents

Banghao Chi[1]

[1]University of Illinois at Urbana-Champaign, NCSA, Urbana, US
banghao2@illinois.edu

May 22, 2025

Keywords: Natural Language Processing (NLP); Knowledge Representation; Large Language Models (LLMs)

1 Introduction

The exponential growth of information available on the internet has created both unprecedented opportunities and significant challenges in knowledge management and utilization. While the web contains vast amounts of valuable data, much of it exists in unstructured formats and are scattered everywhere, making it difficult to efficiently process, analyze, and leverage at scale. Traditional web scraping and data extraction techniques often struggle with the complexity and diversity of web content, leading to incomplete or inaccurate information retrieval.

Recent advancements in Large Language Models (LLMs) have opened new possibilities in natural language processing and understanding. These models have demonstrated remarkable capabilities in tasks such as text summarization, information extraction, and question answering(Zhu *et al.*, 2023). However, their full potential in systematically processing and structuring web-based information remains to be explored.

Structured outputs of LLMs is another critical consideration in developing effective knowledge extraction systems. By constraining LLMs to generate structured outputs, such as well-formed JSON or XML, the extraction process becomes more efficient and effective in terms of consistency, reliability, and downstream processability. As a result, guided generation techniques using Finite State Machine(FSM) can effectively constrain LLM outputs while maintaining their semantic understanding capabilities(Willard and Louf, 2023).

Problem setting: In our case, we want to extract the information of an entity(with desired properties) with a search query or specific URL(s) as an entry point and diverge from that with the power of LLMs. That is:

- **Input:** one single URL(or multiple, or just a single search query) as the starting point of the entity's information
- **Output:** a JSON object that contains the information of the entity with designated properties and corresponding values

2 Methodology

The system leverages Large Language Models (LLMs) and structured outputs to create a robust pipeline for entity information extraction. This approach combines web scraping, schema detection, recursive link analysis, and intelligent information gathering to build comprehensive entity profiles.

2.1 Algorithm Overview

The core algorithm can be described in the following pseudocode:

Algorithm 1: Enhanced Entity Information Extraction Pipeline

```
Input: Input query or URL(s), maximum recursion depth
Output: JSON files containing comprehensive entity information

// Phase 1: Process input and initialize logging
if IsQuery(input) then
    urls ← BraveSearch(input)
end
else
    urls ← ProcessInputURLs(input)
end
logger ← SetupLogging(model, maxDepth)

// Phase 2: Process each URL for entity information
foreach url ∈ urls do
    startTime ← CurrentTime()
    scrapeResult ← WebScraper.Scrape(url)
    // Schema Detection and Management
    schemaResult ← Agent.DetectSchema(scrapeResult)
    if schemaResult = "No match" then
        schemaType ← UserInput()
        newSchema ← Agent.GenerateNewSchema(scrapeResult, schemaType)
        SchemaManager.SaveNewSchema(schemaType, newSchema)
    end
    // Initial Entity Data Extraction
    entityData ← Agent.ExtractEntityData(scrapeResult, schemaType)
    noneKeys ← GetNoneValueKeys(entityData)
end

// Phase 3: Recursive Link Analysis
if noneKeys ≠ ∅ then
    (discoveredLinks, relevanceDict) ← GatherLinksRecursively(
            scrapeResult, entityData, noneKeys, schemaType, maxDepth)
end

// Phase 4: Information Enhancement
foreach (linkUrl, fields) ∈ relevanceDict do
    linkContent ← WebScraper.Scrape(linkUrl)
    entityData ← Agent.UpdateEntityData(entityData, linkContent, fields)
end

// Phase 5: Results Storage
outputPath ← GenerateOutputPath(entityData, schemaType)
SaveToJSON(entityData, outputPath)
processingTime ← CurrentTime() − startTime
CollectMetrics(url, processingTime, relevanceDict)
WriteProcessStats(urls, processingTimes, relevanceDict)

return entityData
```

2.2 Algorithm Breakdown

The enhanced algorithm operates in six distinct phases:

- **Phase 1: Input Processing and Initial Setup**
  - Processes input which can be either a search query or URL(s)
  - Performs Brave search if input is a query
  - Sets up comprehensive logging system for process tracking
- **Phase 2: Entity Processing**

  - Scrapes webpage content while excluding irrelevant elements
  - Detects appropriate schema for the entity
  - Generates new schema if no matching schema exists
  - Extracts initial entity information using LLM

- **Phase 3: Recursive Link Analysis**
  - Identifies fields with missing information
  - Recursively discovers and analyzes relevant links
  - Evaluates link relevance for specific missing fields
  - Maintains visited URL tracking to prevent cycles

- **Phase 4: Information Enhancement**
  - Processes each relevant link to extract additional information
  - Updates entity data with new information from each source

- **Phase 5: Results Storage**
  - Saves comprehensive entity information to JSON files
  - Generates detailed CSV reports of process statistics

The algorithm uses a modular approach that separates concerns between web scraping, schema management, information extraction, and data storage. Next, we will dive into how *GatherLinksRecursively* this function works.

2.3 *GatherLinksRecursively*

This algorithm implements intelligent recursive link discovery and analysis for missing entity information:

Algorithm 2: Intelligent Recursive Link Discovery and Analysis

Input: Initial webpage content, entity JSON data, empty field keys, schema type, maximum depth
Output: Set of relevant links, dictionary mapping URLs to relevant fields

```
// Base case - check recursion depth
if maxDepth ≤ 0 then
    return ∅, ∅
end
// Initialize tracking sets and dictionaries
visitedUrls ← ∅
relevantLinks ← ∅
relevanceDict ← {}
// Extract links from current page
pageLinks ← Agent.ExtractLinks(content, entityData)
validLinks ← FilterValidLinks(pageLinks)
// Process each discovered link
foreach link ∈ validLinks do
    if link.url ∈ visitedUrls then
    end
    if IsPDF(link.url) ∨ IsArxiv(link.url) then
    end
    visitedUrls ← visitedUrls ∪ {link.url}
    relevantFields ← ∅
    // Check relevance for each empty field
    foreach field ∈ emptyFields do
        relevance ← Agent.CheckLinkRelevance(link, field, entityData)
        if relevance.answer = "Yes" then
            relevantFields ← relevantFields ∪ {field}
        end
    end
    // Process relevant links recursively
    if relevantFields ≠ ∅ then
        relevantLinks ← relevantLinks ∪ {link}
        relevanceDict[link.url] ← relevantFields
        // Scrape and recurse on relevant link
        newContent ← WebScraper.Scrape(link.url)
        if newContent ≠ null then
            (nestedLinks, nestedRelevance) ← GatherLinksRecursively(
                    newContent, entityData, emptyFields, schemaType, maxDepth − 1)
            relevantLinks ← relevantLinks ∪ nestedLinks
            relevanceDict ← relevanceDict ∪ nestedRelevance
        end
    end
end
return (relevantLinks, relevanceDict)
```

2.4 Algorithm Breakdown

The *GatherLinksRecursively* algorithm operates in several key phases:

- **Initialization**
  - Sets up tracking for visited URLs to prevent cycles
  - Initializes collections for relevant links and their mappings
  - Validates depth parameter to enforce recursion limits
- **Link Discovery**

  - Extracts all links from current webpage content
  - Filters links for validity and accessibility
  - Removes PDFs and certain blocked domains (e.g., arXiv)

- **Relevance Analysis**
  - Uses Agent.CheckLinkRelevance to analyzes each link for relevance to missing fields

- **Intelligent Recursive Processing**
  - Processes relevant links by scraping their content
  - Recursively discovers nested links within depth limit and wisdom

- **Result Aggregation**
  - Combines relevant links from all depth levels
  - Merges relevance dictionaries maintaining field associations

Next, we will explore how all the agents algorithms work.

2.5 Agent Methods

The Agent class implements several key methods for entity processing and information extraction:

2.5.1 Agent.DetectSchema

This method analyzes webpage content to determine the appropriate schema type:

Algorithm 3: Schema Detection

Input: Webpage content as text
Output: Schema type and detection reason

`// Initialize schema detection`
$availableSchemas \leftarrow$ SchemaManager.GetSchemaNames()
$response \leftarrow \emptyset$
`// Prepare LLM prompt for schema detection`
$systemPrompt \leftarrow$ CreateSystemPrompt($availableSchemas$)
$userPrompt \leftarrow$ CreateUserPrompt($webpageContent$)
`// Query LLM for schema detection`
$response \leftarrow$ QueryLLM($systemPrompt, userPrompt$)
`// Validate and return result`
if $response.schema \notin availableSchemas \wedge response.schema \neq$ *"No match"* **then**
    return *"No match", "Invalid schema detected"*
end
return *response.schema, response.reason*

2.5.2 Agent.GenerateNewSchema

This method creates a new Pydantic schema when no existing schema matches:

Algorithm 4: Dynamic Schema Generation

```
Input: Webpage content, desired schema type
Output: New Pydantic schema code
// Load example schema template
exampleSchema ← LoadExampleSchema()
// Prepare LLM prompt for schema generation
systemPrompt ← CreateSchemaGenerationPrompt(exampleSchema)
userPrompt ← CreateUserPrompt(content, schemaType)
// Generate new schema
schemaCode ← QueryLLM(systemPrompt, userPrompt)
// Validate generated schema
if !IsValidPydanticSchema(schemaCode) then
    return Error("Invalid schema generated")
end
return schemaCode
```

2.5.3 Agent.ExtractEntityData

This method extracts entity information using the appropriate schema:

Algorithm 5: Entity Data Extraction

```
Input: Webpage content, schema type
Output: Structured entity data
// Get schema definition
entitySchema ← SchemaManager.GetSchema(schemaType)
// Prepare LLM prompt for data extraction
systemPrompt ← CreateExtractionPrompt(schemaType, entitySchema)
userPrompt ← CreateUserPrompt(webpageContent)
// Extract entity data
response ← QueryLLM(systemPrompt, userPrompt)
// Validate extracted data against schema
if !ValidateAgainstSchema(response, entitySchema) then
    return Error("Invalid data structure")
end
return response
```

2.5.4 Agent.CheckLinkRelevance

This method evaluates if a link might contain information about a specific entity field:

Algorithm 6: Link Relevance Evaluation

Input: URL, display text, target field, entity data, schema type
Output: Relevance assessment with reason

```
// Initialize relevance assessment
entityName ← GetEntityName(entityData)
systemPrompt ← CreateRelevancePrompt(schemaType)
// Prepare link context
linkContext ← {
    url : url,
    displayText : displayText,
    targetField : targetField,
    entityName : entityName
}
// Query LLM for relevance assessment
response ← QueryLLM(systemPrompt, linkContext)
// Validate response structure
if !IsValidResponse(response) then
    return Error("Invalid response structure")
end
// Return structured assessment
assessment ← {
    answer : response.answer,
    reason : response.reason
}
return assessment
```

2.5.5 Agent.UpdateEntityData

This method updates entity information with data from additional sources:

Algorithm 7: Entity Data Update

Input: Original entity data, new content, target fields
Output: Updated entity data

```
// Prepare update prompt
systemPrompt ← CreateUpdatePrompt(schema, targetFields)
userPrompt ← CreateUserPrompt(originalData, newContent)
// Update entity data
updatedData ← QueryLLM(systemPrompt, userPrompt)
// Merge and validate updates
foreach field ∈ targetFields do
    if HasNewInformation(updatedData, field) then
        originalData[field] ← MergeInformation(originalData[field], updatedData[field])
    end
end
// Validate final structure
if !ValidateAgainstSchema(originalData, schema) then
    return Error("Invalid update structure")
end
return originalData
```

2.6 Implementation Details

The Agent methods share several key characteristics:

- Uses a consistent interface for LLM queries
- Temperature control (0.0) for deterministic outputs

- Guided JSON schemas for structured responses

3 Results

Given the novel nature of our LLM-based entity information extraction topic, there are no direct state-of-the-art frameworks available for comparative evaluation. Traditional information extraction systems typically focus on specific domains or predefined schemas, while our approach offers flexible, schema-driven extraction across diverse entity types. Therefore, we present comprehensive results across multiple dimensions to demonstrate the effectiveness of our system.

3.1 FSM and LLM Integration Performance

To evaluate the fundamental effectiveness of integrating Finite State Machines (FSM) with Large Language Models, we conducted a comparative analysis between our FSM-guided extraction approach and GPT-4o mini's baseline performance. This comparison focuses on one-time information extraction accuracy, testing both semantic understanding and structured information extraction capabilities.

Table I: FSM-LLM Integration Performance Results on Car

Models	JSON Validity	Key Similarity	Value Exactness	Numeric Similarity	String Similarity
	1.000	1.000	0.969	0.980	0.910
Qwen2.5-72B	Standard Deviation				
	0.000	0.000	0.000	0.044	0.064
	1.000	1.000	0.972	0.989	0.930
GPT-4o-mini	Standard Deviation				
	0.000	0.000	0.005	0.035	0.052

The FSM-LLM integration performance was evaluated across five key metrics using 30 distinct samples of entity schema Car, Professor, and Movie. For each model (Qwen2.5-72B and GPT-4o-mini), we report both the mean performance and standard deviation to capture result consistency and reliability. Here, we report the results on Car entity as an example. As seen in Table I, both models achieved perfect scores in JSON validity and key similarity (1.000), demonstrating robust structured outputs capability. For value exactness, GPT-4o-mini showed a slightly higher average (0.972) compared to Qwen2.5-72B (0.969), though the difference is minimal. In numeric similarity, GPT-4o-mini outperformed with 0.989 versus 0.980, showing better handling of numerical data. String similarity results favored GPT-4o-mini (0.930) over Qwen2.5-72B (0.910), indicating slight superior text matching capabilities. The standard deviations reveal that both models maintain consistent performance across samples, with numeric and string similarities showing the most variation (standard deviations ranging from 0.035 to 0.064), while maintaining perfect stability in JSON validity and key similarity metrics.

As a result, the difference is model-specific, with our focus being on JSON validity and value exactness. For more results on other schemas such as Professor and Movie, checkout the Github repository.

3.2 Web Scraping Performance

The effectiveness of our information extraction pipeline also slightly depends on the quality of the initial web scraping. Our custom web scraper was evaluated across various website structures and content types as compared to Firecrawl , an open-source project which turns websites into LLM-ready data. The results can be found in the Github repository. By measuring and validating the content of the results, we have successfully achieved 100% accuracy as for the scraping performance.

3.3 Final Results

We conducted evaluation of our FSM-guided LLM extraction system across multiple entity schemas and models. The evaluation encompassed 15 samples across five distinct entity types: research papers, courses, students, language models, and professors. Two state-of-the-art quantized models were compared: Qwen2.5-72B-Instruct-AWQ and Meta-Llama-3.3-70B-Instruct-AWQ-INT4. Below is the table demonstrating averaged results. For more information, please checkout this file.

Table II: Comprehensive Evaluation Results Across All Schemas

Models	JSON Validity	Key Similarity	Value Exactness	Numeric Similarity	String Similarity
	1.000	1.000	0.911	0.934	0.966
Qwen2.5-72B			Standard Deviation		
	0.000	0.000	0.057	0.064	0.032
	1.000	1.000	0.872	0.903	0.782
Meta-Llama-3.3-70B			Standard Deviation		
	0.000	0.000	0.061	0.089	0.087

3.3.1 Overall Performance

Both models demonstrated excellent performance in structural accuracy, achieving perfect scores (1.000) in key similarity across all schemas, indicating robust adherence to the prescribed JSON structure. The Qwen2.5-72B model consistently outperformed Meta-Llama-3.3-70B across all major metrics.

3.3.2 Key Findings

- **Numerical Similarity:** The difference of the numerical value is mainly due to year(the model made it up), unit transformation(e.g., training tokens in billion, but the model reports raw number such as 72700000000 instead of 72).
- **Number of Depths:** The results of depths of one are generally better than that of two, indicating the potential best number of depths.
- **Model Comparison:** Qwen2.5-72B-Instruct-AWQ demonstrated superior performance across all metrics, with particularly notable advantages in string similarity (18.4% higher) and value exactness (4.5% higher) compared to Meta-Llama-3.3-70B.

These results demonstrate the robustness of our FSM-guided extraction system across diverse entity types and its ability to maintain high accuracy while processing various information types. The consistent superior performance of Qwen2.5-72B-Instruct-AWQ suggests it as the preferred model for deployment scenarios requiring high accuracy in information extraction tasks.

4 Assessment

Our semester goals focused on developing a robust, LLM-based system for automated entity information extraction with adaptive schema management. Assessing our progress against these objectives reveals several key achievements and areas for future development:

4.1 Core Objectives Achievement

- **Automated Information Extraction**
  - Successfully implemented a fully automated pipeline integrating FSM with LLMs
  - Achieved high accuracy in information extraction (96.9-97.2% value exactness)
  - Demonstrated good final results across all tests
- **Intelligent Recursive Information Discovery**
  - Implemented depth-controlled and intelligent link discovery and relevance assessment
  - Successfully merged information from multiple sources maintaining data integrity
- **Schema Flexibility**
  - Developed dynamic schema detection and generation capabilities

4.2 Technical Milestones

- **Model Integration**
  - Successfully integrated and compared multiple LLM models, featuring open-source models such as Qwen2.5-72B, Llama-3.3-70B and Mistral-Large(all AWQ-quantized)
  - Demonstrated competitive performance between Qwen2.5-72B and GPT-4o-mini

5 Reflection

5.1 Research Incentives

Learnt the core of concept of research, which is to research things driven by peronal interests and rational thinking

5.2 Technical Skills

- Gained deep understanding of LLM decoding operations and evaluation
- Enhanced database management and Docker containerization skills

5.3 Feedback for Collaboration

5.3.1 Strengths

- Strong advisory support from Dr. Kevin Chang
- Access to excellent research infrastructure through NCSA
- Clear project milestones and progression

5.3.2 Areas for Enhancement

- More incentives and commmitment to the project during semester

6 Future Work

Several key areas have been identified for future development.

6.1 Framework Extensions

- [ ] Explore more intelligent ways of:
  - [ ] Scraping relevant links
    - [ ] Utilize more of the search engine?
    - [ ] Utilize browser-use-webui for browser control?
  - [ ] Updating the JSON object
    - [ ] Update field by field instead of the entire JSON object?
- [x] Dynamic schema creation
- [x] Bypass anti-scraping by rendering pages in the local browser
- [ ] Information effectiveness evaluation
- [ ] Database operation
- [ ] Modularize the codebase

References

Willard, Brandon T and Louf, Rémi (2023). “Efficient guided generation for large language models”, *arXiv preprint arXiv:2307.09702*,

Zhu, Yutao *et al.*, (2023). “Large language models for information retrieval: A survey”, *arXiv preprint arXiv:2308.07107*,