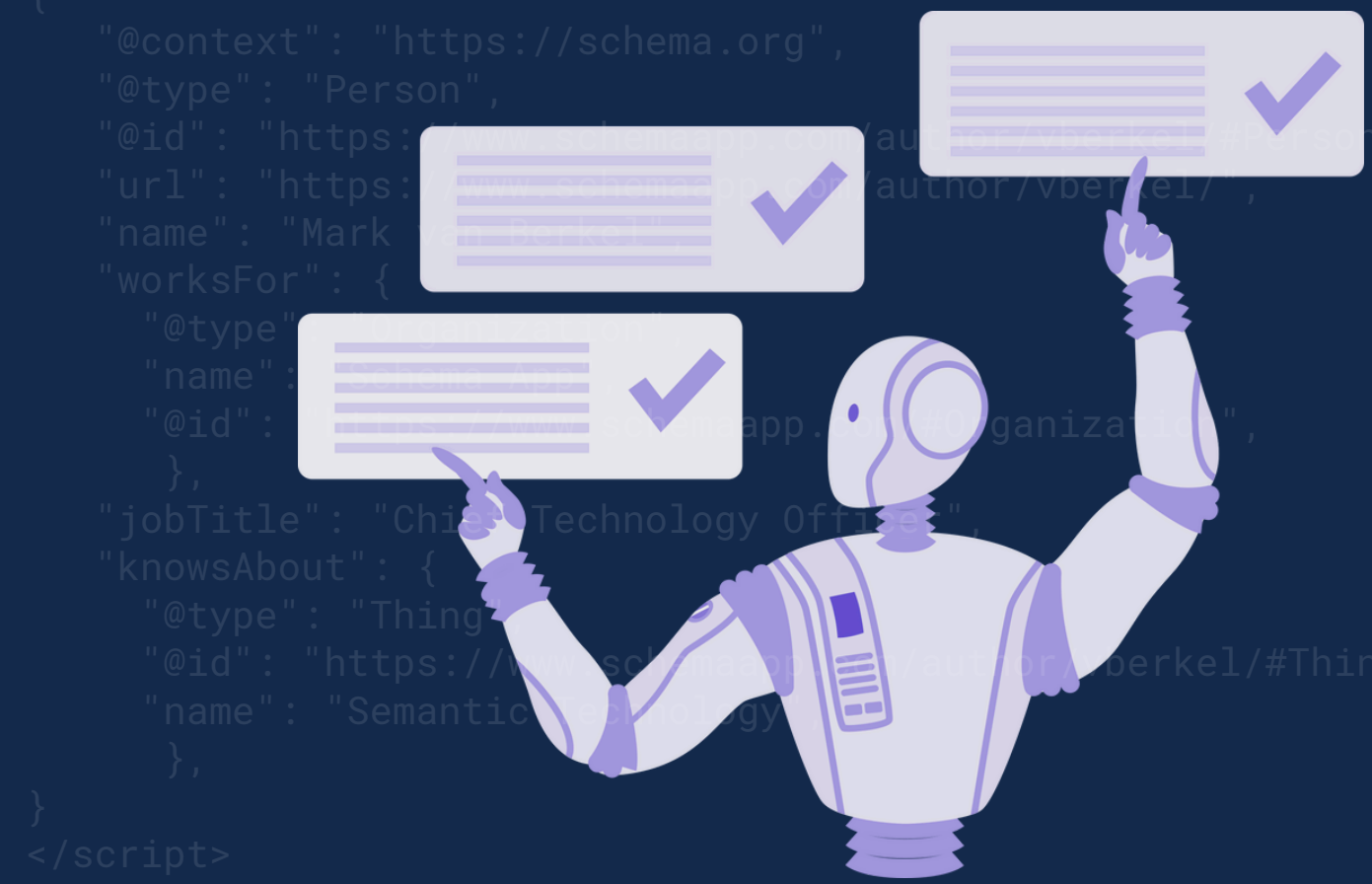
Schema App

NLWeb: A Technical Assessment for AI-Enhanced Website Functionality

Consuming Schema Markup for AI Functions: Implementation Findings and Lessons Learned



Executive Summary

One of the current challenges in using generative AI, as always in the data world, is making one's data consumable by AI tools. Whether it's an LLM-powered chatbot for a website or a new agent to automate IT tasks, these things all have one thing in common: they require quality, clean data — the more structured the better.

NLWeb, short for Natural Language Web, presents a compelling solution for those interested in fast-tracking the behind-the-scenes data work and rolling out AI functions for their websites.

This paper documents our evaluation of **Microsoft's NLWeb toolkit** as both a semantic search replacement for WordPress and a broader AI enablement platform.

Through A/B testing, implementation analysis, and hands-on prototyping, we aim to provide technical insights for organizations considering NLWeb adoption.

Introduction to NLWeb

NLWeb is a toolkit from Microsoft that makes AI integration remarkably accessible. At a higher level view of its functionality, its standout feature is an 'ETL' pipeline built to load Schema.org structured data from one's website into a vector database, following the new Model Context Protocol (MCP) standard. The NLWeb toolkit, available at https://github.com/nlweb-ai/NLWeb, comes with several components:

Data Ingestion

- Web crawler that navigates your site and retrieves schema-marked-up elements
- Data loader that loads website data into a vector database of your choice with all the proper embeddings for the LLM model you wish to use
- Connectors for popular LLM endpoints

Data Storage

- Built-in MCP (Model Context Protocol) servers for standardized LLM interactions
- Conversation storage

Ease of Use

- Natural language handling through the prompt control flow
- Boilerplate front end accelerating prototyping

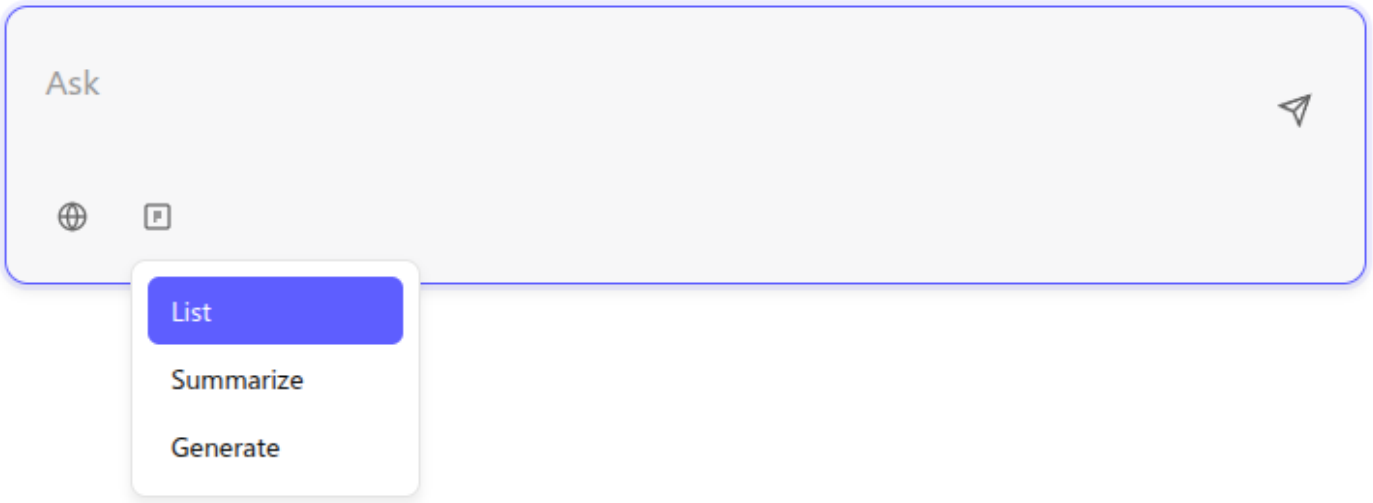
As people who have built pipelines for retrieval-augmented generation (RAG) will tell you, there are many hidden complexities around data extraction, chunking strategies, embedding consistency, and vector store optimization that can consume much development time.

What using NLWeb means for website teams is that now your site is navigable, findable, and usable by AI with just a couple of hours of setup, avoiding data-loading complexities. The toolkit also comes with a collection of system prompts meant to make querying your website data more reliable, allowing those not ready to step into the dark arts of prompt engineering to make their first steps into LLM utilization with NLWeb's preconfigured prompting.

There are three default query templates:

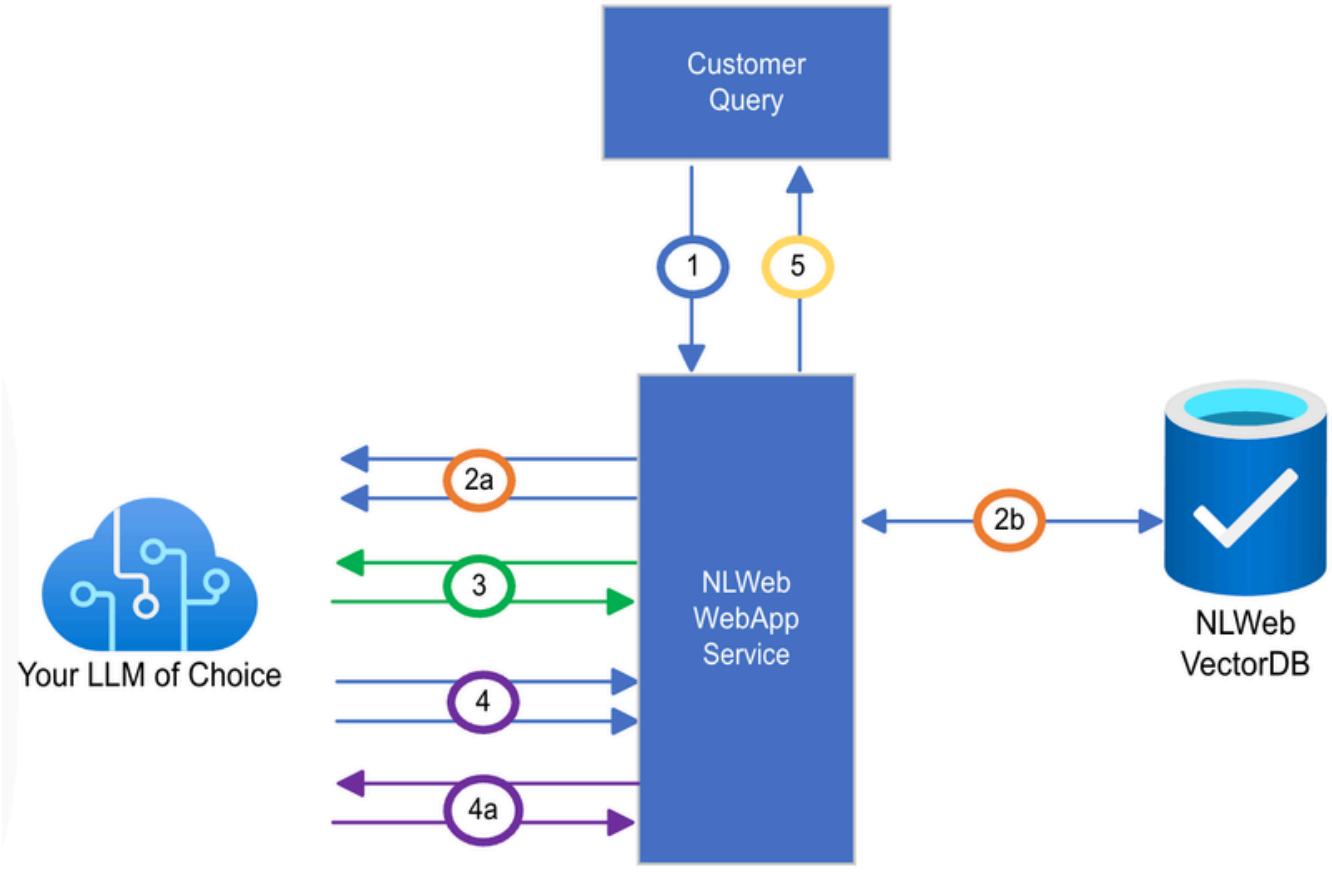
- List
- Summarize
- Generate

A company wanting to replace its basic website search functionality with something AI-powered and semantic could use the NLWeb list functionality to take a search keyword as input and provide a list of relevant results.



Pictured below is a top-level diagram of how the NLWebApp service works once deployed. The app takes a query **[1]** and contextualizes that query **[2]** using multiple parallel calls to the LLM and vector store, utilizing initially requested query types (list, summarize, generate), conversation history, and more.

The service may fast-track the query past contextualization **[2b]** if it is simple and does not need contextualization. The LLM will then select a tool to use **[3]** based on query contextualization, for example, search, item details, or ensemble queries. The service then takes the results and, again depending on the type of tool selected, compares results **[4]** and prepares them for display back to the user with optional post-processing **[4a]**.



Source https://github.com/nlweb-ai/NLWeb/blob/main/docs/life-of-a-chat-query.md

The service includes a debugging mode that lets you inspect intermediary LLM calls. Below is an example output from the tool selection prompt, along with the prompt template that NLWeb passes to the LLM to generate this result.

The orchestration process ensures prompts are handled at a finer granularity, allowing the system to adjust tool selection based on context and stored history without overloading the LLM's context window.

tool_selection 06:57:56


```
{
  "message_type": "tool_selection",
  "selected_tool": "search",
  "score": 80,
  "parameters": {
    "score": 80,
    "search_query": "blog posts by author Martha"
  },
  "query": "Blog posts written by Martha",
  "time_elapsed": "3.448s",
  "query_id": ""
}
```


Tool Selection Prompt Template

The user has the following query: **{request.query}**

The search tool finds items that match specific criteria, such as:

- Recipes, movies, books, or products that meet certain requirements
- Discovering items based on attributes like genre, cuisine type, price range, or features
- Broad exploration when users want to see what's available

The search tool is *not* the best choice if the user is asking for details about a specific named item (like ingredients, instructions, cast, or price).

Assign a score from 0 to 100 for whether the search tool is appropriate, and provide the search query that should be passed to the tool.

Implementation Architecture and Setup

Infrastructure Configuration

Our implementation was deployed on AWS EC2, providing a controlled environment for testing and evaluation. The setup process, while straightforward, revealed several important considerations:

Documentation Challenges: The NLWeb documentation proved somewhat unorganized with several undocumented features, such as the user agent setting for the crawler. This required additional exploration and experimentation during setup. For instance, we needed to modify the provided NLWeb code to use our crawler's identity.

Vector Database Selection: We selected Qdrant as our vector database server based on its recommendation for quick setup and the ability to bundle it as a Docker container. This approach offers significant advantages:

- Simplified deployment alongside NLWeb on the same server
- Potential for data isolation in future customer deployments
- Embedding-agnostic flexibility for different model configurations

However, our implementation revealed a critical limitation: the containerized version of Qdrant supports only one connection at a time, preventing simultaneous reading and writing operations.

This means NLWeb must be stopped to add new data to the datastore, which slowed initial experimentation but shouldn't cause significant friction for production deployments with established content loading schedules.

LLM and Embedding Configuration

Our testing revealed important insights about model selection and configuration:

Initial Model Experimentation: We initially tested various LLM options including Gemini 2.0 Flash and GPT 4.1. Gemini 2.0 Flash performed poorly, while Gemini 2.5 Pro wouldn't function at all within the NLWeb framework.

Optimal Configuration: We achieved best results using Azure OpenAI for the LLM endpoint, likely because this received the most internal testing by the Microsoft NLWeb team. For embeddings, we switched to OpenAI's embedding models, creating a stable and reliable configuration.

Lesson Learned:
For endpoint and embedding selection, we found it best to keep these at Microsoft Azure defaults — other LLM endpoints behaved more inconsistently.

Data Ingestion Process

The data ingestion process proved remarkably efficient. Using existing Schema App markup data, we exported content in CSV format with URL and JSON-LD as the two columns. The entire setup took approximately one hour to have a functioning system providing AI-powered answers — a significant improvement over developing similar functionality from scratch.

Website Search Prototype and Testing Methodology

Current State Analysis

Our website, like many, uses our content management system's (WordPress) search functionality. This provides adequate *keyword-based* search but lacks *semantic understanding* — it cannot interpret the meaning of search requests or the content it retrieves.

Since we're in the business of making semantic data useful, replacing default search functionality presented an ideal NLWeb prototyping opportunity.

Semantic Search Implementation

NLWeb transforms search from keyword matching to semantic understanding through the orchestration of LLM calls with generation augmented by the retrieval of structured data.

When a user enters a search term, NLWeb's behind-the-scenes process:

- Makes fine-grained prompted calls to contextualize the query
- Categorizes entities of interest within the search
- Uses additional LLM calls to rank relevance between categorized queries and extracted Schema Markup in the vector database

This enables search to interpret queries as being about certain types of things, then find pages related to those conceptual categories rather than just keyword matches.

A/B Testing Results

We conducted comprehensive A/B testing using the top 10 search queries from Google Analytics, comparing NLWeb results against WordPress search as the control.

Criteria	WordPress Search	NLWeb Search
Search Result Relevance	Returns results based on keyword matching, consistently surfacing exact-match content, especially for titles and headings	Leverages generative and semantic search, capable of providing summaries and context-rich answers. Occasionally surfaces highly relevant but less literal matches, which may be less predictable for users expecting keyword-based results
Performance Characteristics	Fast and predictable, with results optimized for existing content structure	Slightly slower due to model inference, but acceptable for most queries. Performance expected to improve as underlying models are optimized
Text Search Accuracy	Excels at literal string matching	Excels at finding semantically similar content but may miss exact-match scenarios unless explicitly tuned

Real-World Example

In the default WordPress search, entering 'entity' returns only pages with that word in the title or subtitle HTML tags. Since we launched a product named '**Entity Hub**', WordPress search results were dominated by those pages. Using NLWeb, results provided a much more balanced representation of blog posts, how-tos, demos, and user stories, because NLWeb uses LLM calls and Schema Markup to understand the meaning of website content.

Testing Challenges and Observations

Search Term Limitations: We encountered an issue where NLWeb produced no results for the search term "saving in schema." This suggests certain domain-specific terminology may require additional prompt tuning or training data consideration.

Localization Issues: During testing, unusual Japanese text appeared in debug information, indicating potential localization configuration needs.

Result Quality: While positional matches between WordPress and NLWeb were rare, NLWeb seemed to prioritize relevance more effectively, leading to improved user experience. Manual review categorized non-matching results as either improvements or equally relevant to WordPress results.

Configuration Best Practices and Optimization

Prompt Engineering for Domain Specificity

For semantic search effectiveness, we recommend reviewing and customizing NLWeb's default prompts, particularly the DetectItemTypePrompt (https://github.com/nlweb-ai/NLWeb/blob/main/config/prompts.xml). By default, the system recognizes limited categories. Tailoring these categories to reflect your website's content provides significant improvement.

Recommendation:
This presents an especially interesting opportunity to incorporate structured data (RDF, OWL) to enhance search result quality and semantic understanding.

Content Type Differentiation

Challenge: NLWeb requires additional configuration to distinguish between content types (blogs vs. landing pages vs. documentation).

Solution Directions:

- Develop structured prompts or metadata signals to help NLWeb distinguish content types
- Consider hybrid approaches combining keyword and semantic search for more predictable results
- Implement user feedback mechanisms for iterative search quality improvement

Production Readiness Assessment

Minimum Criteria for WordPress Replacement

To serve as a production-ready WordPress search replacement, NLWeb must:

- Surface all relevant content types with high accuracy
- Allow filtering or prioritization by content type
- Provide predictable results for common queries
- Have few regressions from WordPress search performance for exact-match scenarios

Current Gaps and Mitigation Strategies

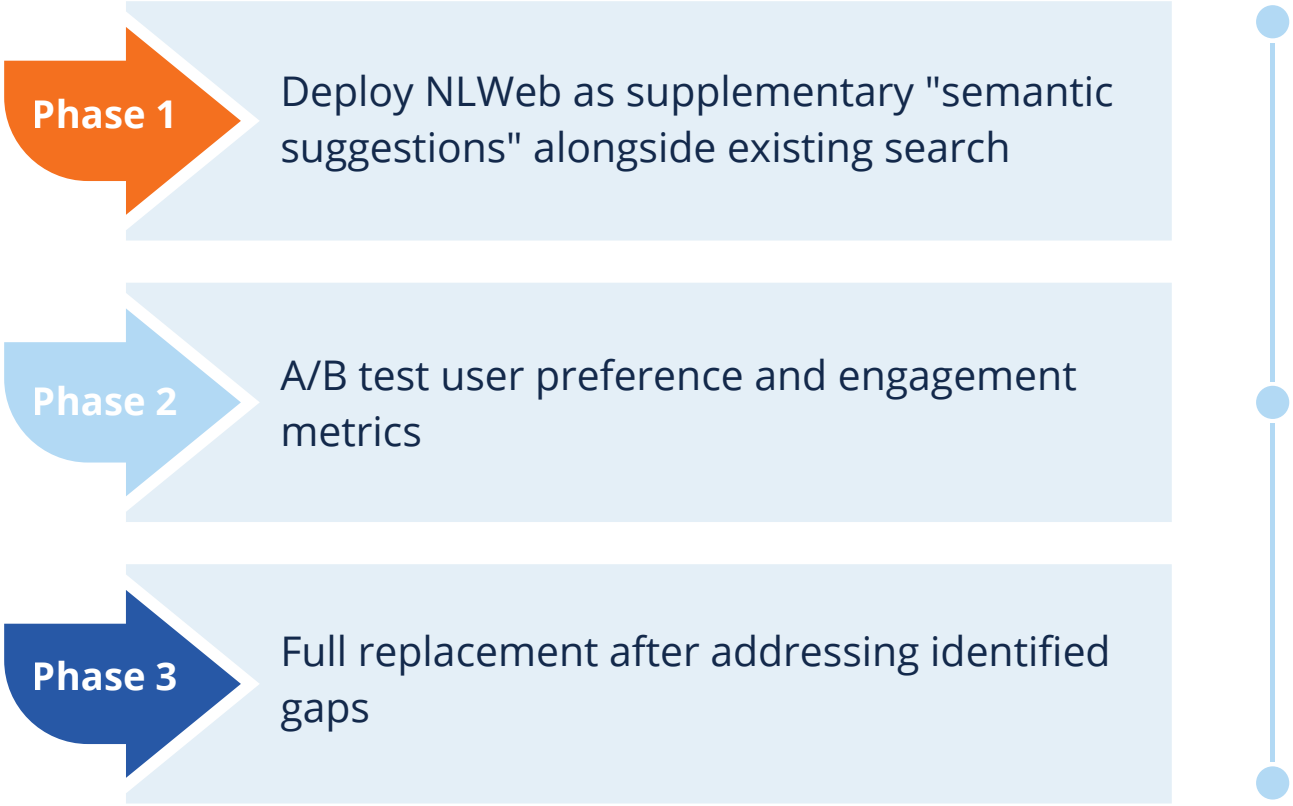
Predictability: NLWeb's generative approach, while powerful for summarization and contextual answers, needs refinement to consistently meet user expectations for direct content retrieval.

Performance Tuning: Optimize model serving for lower latency to match WordPress response times.

Hybrid Integration: Consider embedding NLWeb results as "smart suggestions" within existing WordPress search UI rather than complete replacement.

Strategic Implementation Recommendations

Phased Adoption Approach



Technical Architecture Considerations

Containerization: Leverage Docker containers for simplified deployment and customer data isolation

Monitoring: Implement comprehensive logging and performance monitoring for model inference and search quality

Feedback Loops: Build user feedback mechanisms to continuously improve search relevance

Conclusions and Future Directions

NLWeb presents a compelling solution for organizations seeking to leverage existing Schema Markup for AI functionality.

The remarkably quick setup time — approximately one hour from start to functioning AI-powered search — represents a significant advantage over custom development.

Key Strengths

- Rapid deployment and integration
- Effective utilization of existing Schema Markup
- Semantic search capabilities that understand content meaning
- Pre-built components that solve complex data loading challenges

Areas for Enhancement

- Prompt tuning for domain-specific terminology
- Content type differentiation capabilities
- Performance optimization for production workloads
- Predictability improvements for exact-match scenarios

Strategic Value: For organizations with schema.org markup, NLWeb offers a fast track to AI-enhanced website functionality. The toolkit's ability to make websites "navigable, findable and usable by AI" with minimal setup makes it particularly attractive for rapid prototyping and proof-of-concept development.

Next Steps

1. **Technical Documentation:** Develop comprehensive implementation guides with reproducible test cases
2. **Integration Strategy:** Explore hybrid approaches that leverage both traditional and semantic search strengths
3. **Performance Benchmarking:** Conduct detailed performance analysis for production scalability planning

NLWeb represents a significant step forward in making AI functionality accessible to organizations with structured data, providing a practical path from Schema Markup to intelligent, context-aware website interactions.

This assessment is based on hands-on implementation and testing conducted by the Schema App Data Team. For detailed technical implementation guides and reproducible test cases, additional documentation assets are available upon request.

Interested in having Schema App set up NLWeb on your website?
Contact our sales team today.

If you are a current Schema App customer, reach out to your Customer Success Manager for further inquiry.

Hello@SchemaApp.com

www.SchemaApp.com

Schema App