Client-Side Built-in Browser AI for Digital Libraries and Archives

Case Studies of the Query Shaper and Local Chat Web Components

Sawood Alam
Internet Archive
San Francisco, CA, USA
sawood@archive.org

Abstract

Web browsers have begun shipping on-device language models that web pages can prompt directly, such as through Chrome's Prompt API. We demonstrate how this built-in AI can improve digital library and archive interfaces without a cloud-hosted AI service, using two open-source web components. *Query Shaper* translates natural language into structured queries in the backend's syntax, such as Lucene syntax for Elasticsearch, offering an alternative to complex Advanced Search forms. *Local Chat* is a chat widget grounded in page content that helps users navigate documentation and summarize or compare search results, as in the Ask Me Twice (AMT) Explorer. Because inference runs on the user's device, queries and conversations never leave the browser, and the institution pays nothing for inference. In browsers without built-in AI, both components fall back to the original interface.

1 Introduction

Digital libraries and archives expose rich, fielded search backends, yet most users type a few keywords. Advanced Search forms and query syntaxes (field prefixes, boolean operators, ranges) offer precision, but they take effort to learn and are rarely used [8]. Large language models (LLMs) can bridge this gap by interpreting what the user means. However, a cloud-hosted LLM sends every query to a third party and adds a per-request cost that many non-profit memory institutions cannot absorb at scale.

Browsers now offer another option: an on-device model that web pages can call from JavaScript. Chrome's Prompt API, exposed as `LanguageModel`, lets a page create sessions with the model, prime them with instructions, stream responses, and constrain output to a JSON Schema [6, 11]. This demonstration presents two case studies of this approach: Query Shaper [2] and Local Chat [1]. Both are framework-agnostic, open-source (AGPLv3) custom elements that we developed at the Internet Archive.

2 Query Shaper

A `<query-shaper for="id">` element attaches to an existing search input. As the user types, the on-device model suggests a mix of typo corrections, completions, query expansions, and fielded boolean queries (Figure 1). The host page describes its backend in three attributes: `fields` lists the searchable fields, `examples` gives sample inputs with good suggestions for each, and `notes` states domain rules in prose. Our Internet Archive demo declares `mediatype`, `year`, `creator`, `subject`, and `language`. For "audio recordings or texts about jazz, not from 2020", it may suggest something like:

```
(mediatype:audio OR mediatype:texts) AND subject:jazz AND -year:2020
```

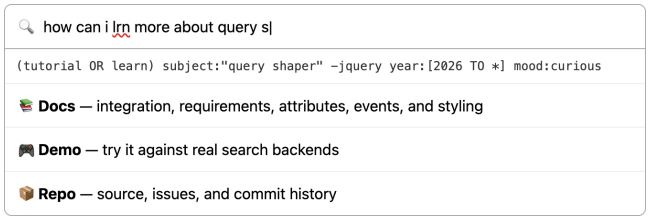


Figure 1: An illustration from the Query Shaper landing page, where a partially typed, misspelled question becomes a fielded boolean query.

The model always writes suggestions in Lucene-style syntax [4]. Our first design had it label each suggestion's kind and return field/value pairs as JSON, but it mislabeled kinds and invented operators; plain Lucene syntax, which the model knows well, proved far more reliable. Query Shaper drops suggestions that fail to parse, or that use fields when none are declared. It passes the rest through as-is for the `lucene` format (e.g., Elasticsearch's `query_string`), or re-renders them as `simple-query-string` with `+/-` operators [5], as URL parameters for facet-driven backends, or through a custom function. Accepting a suggestion can fill the input, submit the form, open an OpenSearch URL template [10], write to another element, or just fire an event. In `headless` mode, Query Shaper only emits events and the page shows suggestions in its own UI. Query Shaper's documentation page uses this for a site search whose only field is `section`, so questions lead to the relevant section. A short local History of past inputs and accepted suggestions is fed back to the model as examples of what this user wants, and it never leaves the device. A user who keeps turning "films" into `mediatype:movies` thus sets a precedent.

Query Shaper is a progressive enhancement. Without the Prompt API, the input works exactly as before. If the model has not been downloaded yet, the popup offers an "Enable" button to fetch it.

Suggestions are only useful if they arrive while the user is still typing. Creating a model session is slow, but cloning one is fast, so a page-wide session holds the general instructions, each instance clones it once and adds its fields, and each query runs in a throwaway clone. A flat `{suggestions: string[]}` output schema was much faster in our tests than a nested, per-field one. We also destroy the session of a superseded query instead of just aborting it, because an aborted call could keep the on-device model busy for tens of seconds.

3 Local Chat

`<local-chat>` is a floating, draggable, resizable chat widget that answers questions from page content (Figure 2). The host supplies

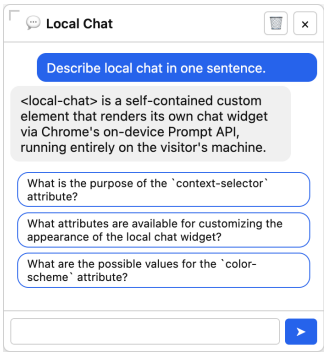


Figure 2: Local Chat answering from the documentation of its own page, with suggested follow-up questions.

this context with a CSS selector, whose matching elements' text is used, or directly as plain text or JSON, such as a list of search results. By default, the model is told to answer only from the context and to say so when the context does not cover a question. As in Query Shaper, one session is primed with the context once, and each conversation runs in a clone of it. Answers stream in as Markdown, followed by suggested follow-up questions. The host can also provide opening questions, or ask the model to propose some. Conversation history is kept locally until the user clears it. We have deployed Local Chat in two settings.

Documentation help. The Local Chat and Query Shaper documentation pages each embed an instance that answers from the page itself. The Local Chat page simply points it at its `main` element. The Query Shaper page puts a list of its section links in Markdown ahead of the page text, so answers can end with "See also" links to the relevant sections. When the links came after the page text, the model often missed them in its small context window and made up links that did not exist.

Summarizing and comparing results. The Ask Me Twice (AMT) Explorer lets users browse a longitudinal archive of AI chatbot responses to the same questions, collected daily from many models [3, 9]. When a user opens one response, or several side by side, and the on-device model is available, the Explorer offers a Local Chat grounded in those response records. Each record carries the question, model, collection date, and answer, so the model can compare answers across models or days. Its opening questions depend on the selection: "Summarize this response" or "Any inaccuracies or gaps in this answer?" for one response, "How do these answers differ?" for several models' answers to one question, and "How has this model's answer changed here?" for one model over time. Researchers can thus examine archived records with an LLM at no inference cost to the archive, and their questions never leave the browser.

4 Demonstration

Attendees will use both components in Chrome with the on-device model already downloaded. They will type natural-language queries into the Internet Archive, Open Library, and Wayback Machine collection search demos and watch the suggested queries change as they type. They will switch output formats and edit `fields` and `examples` to see how the suggestions respond. They will then ask the documentation assistants how to integrate the components, and use the AMT Explorer to summarize and compare AI responses. Finally, we will show the same pages in a browser without built-in AI. Both components are live at https://internetarchive.github.io/query-shaper/ and https://internetarchive.github.io/local-chat/. Attendees can also try both components on their own laptops in Chrome.

5 Discussion and Limitations

On-device inference changes who pays for AI features and who sees the data. The institution serves static JavaScript and pays for no GPUs, API calls, or extra servers. Queries, conversations, and History stay in the user's browser, in keeping with libraries' long-standing commitment to patron privacy. The browser downloads one model and shares it across sites, so a small library gets the same capability as a large one. Nothing changes on the backend: a script tag and one element enhance even a legacy interface, and the backend keeps receiving queries in the syntax it already supports.

The approach has limits. The Prompt API is currently available only in Chrome. The model needs a one-time, multi-gigabyte download and fairly capable hardware, so many users will see only the fallback. On-device models are small and have limited context windows. Their output can look plausible but be wrong, which is why Query Shaper only suggests queries and leaves the choice to the user. Context is also untrusted input: archived AI responses may contain text that tries to steer the model, so Local Chat's answers are reading aids, not authoritative summaries. Quality and latency vary by device, and our tuning reflects one evolving API implementation. We have not yet formally evaluated either component, and we hope to gather feedback at the demonstration. Looking ahead, WebMCP, a proposed standard now in an origin trial, lets pages expose functions as JavaScript tools that AI agents can call [7]. Once it is generally available, an agent could, for example, apply facets or page through results on the user's behalf.

References

[1] Sawood Alam. 2026. Local Chat: An On-Device AI Chat Widget Grounded in Page Context. https://internetarchive.github.io/local-chat/. Accessed September 25, 2026.
[2] Sawood Alam. 2026. Query Shaper: On-Device AI Query Reformulation for Search Inputs. https://internetarchive.github.io/query-shaper/. Accessed September 25, 2026.
[3] Sawood Alam, Matt Blumberg, Rachita Bisht, Amrita Nambiar, and Mark Graham. 2026. Ask Me Twice: A Longitudinal Dataset for Studying Change in AI Chatbot Responses. In *Proceedings of the 2026 ACM/IEEE Joint Conference on Digital Libraries (JCDL '26)*. ACM, Frisco, TX, USA. doi:10.1145/3805696.3848492
[4] Apache Software Foundation. 2026. Apache Lucene Classic Query Parser Syntax. https://lucene.apache.org/core/. Accessed September 25, 2026.
[5] Elastic. 2026. Full Text Queries: `query_string` and `simple_query_string`. https://www.elastic.co/docs/reference/query-languages/query-dsl/full-text-queries. Accessed September 25, 2026.
[6] Google Chrome. 2026. The Prompt API. https://developer.chrome.com/docs/ai/prompt-api. Accessed September 25, 2026.
[7] Google Chrome. 2026. WebMCP. https://developer.chrome.com/docs/ai/webmcp. Accessed September 25, 2026.
[8] Marti A. Hearst. 2009. *Search User Interfaces*. Cambridge University Press, Cambridge, UK. doi:10.1017/CBO9781139644082
[9] Internet Archive. 2026. Ask Me Twice Explorer. https://wayback-api.archive.org/services/amt/explore/. Accessed September 25, 2026.
[10] OpenSearch.org. 2005. OpenSearch 1.1 Specification. https://github.com/dewitt/opensearch. Accessed September 25, 2026.
[11] W3C Web Machine Learning Community Group. 2026. Explainer for the Prompt API. https://github.com/webmachinelearning/prompt-api. Accessed September 25, 2026.