Towards Leveraging Semantic Web Technologies for Automated UI Element Annotation

Trisanth Srinivasan
Cyrion Labs
trisanth@cyrionlabs.org

***Abstract*—This paper presents a novel Chrome extension for automated web UI element annotation using Semantic Web technologies. The primary objective is to enable Visual Language Models (VLMs) and Large Language Models (LLMs) to quickly comprehend and interact with the web by transforming unstructured visual data into structured semantic information. Our approach integrates natural language processing, vector embeddings, and FAISS for real-time similarity search. In addition to detailing the system architecture and implementation, we propose a custom algorithm for semantic annotation and provide a comparison with traditional methods. Evaluations via BLEU scores and search time measurements demonstrate the extension's efficiency.**

***Index Terms*—Semantic Web, UI Annotation, NLP, FAISS, Chrome Extension, Structured Annotation, Real-time Systems**

I. Introduction

The rapid growth of unstructured web data demands tools that can automatically convert visual elements into structured semantic representations. The Semantic Web, as envisioned by Berners-Lee et al. [1], offers a framework for machines to understand web content contextually. Our work builds on this vision by introducing a fully automated Chrome extension that annotates UI elements in real time. Unlike traditional approaches that rely on manual or semi-automated methods, our solution integrates advanced NLP models and FAISS-based vector search to achieve both speed and accuracy. The novelty of our research lies in the seamless combination of these technologies to empower VLMs and LLMs with detailed, context-rich web annotations.

II. Related Work

Semantic annotation and vector search have long been used to convert unstructured data into actionable insights. Berners-Lee et al. [1] laid the groundwork for the Semantic Web, while later studies [4]–[8] extended these ideas to UI annotation and semantic grouping. However, most existing methods either depend on manual intervention or are limited by high computational overhead when processing dynamic web content. Our work addresses these limitations by providing a fully automated, scalable, and computationally efficient solution that fills this research gap.

III. System Architecture

Our system comprises two primary modules: UI Element Detection and Semantic Annotation with Embedding Generation. The UI detection module employs a Chrome extension to monitor DOM changes using `MutationObserver` and track URL modifications, ensuring dynamic and real-time capture of UI elements. The annotation module uses OpenAI's `gpt-4-turbo` API for semantic JSON annotation and the `text-embedding-ada-002` API to generate high-dimensional vector embeddings. These embeddings are later indexed using FAISS for efficient similarity searches.

Figure 1 provides an overview of the system architecture. All figures and tables are explicitly cited in the text.

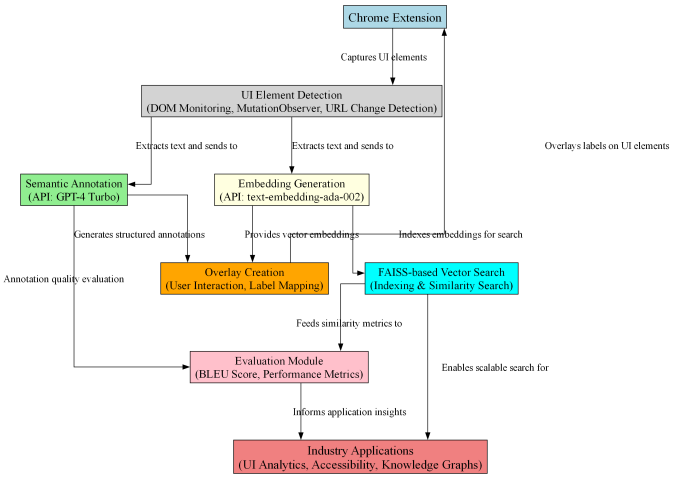

Fig. 1. Overall System Architecture for Automated UI Element Annotation. The diagram illustrates the integration of DOM monitoring, semantic annotation, and FAISS-based vector search.

IV. Implementation Details

A. Chrome Extension Logic

The Chrome extension is responsible for detecting UI elements and initiating their annotation. It uses a `MutationObserver` to monitor DOM changes and detects URL updates by overriding `pushState` and `replaceState`. This ensures robust and dynamic annotation in both traditional multi-page and single-page applications.

1) DOM and URL Change Detection: The extension employs a comprehensive DOM monitoring strategy that detects node additions, removals, and modifications. URL change detection is implemented by comparing the current URL with

a stored value, ensuring annotations are refreshed as users navigate through the web.

Before detailing further components, we now illustrate a key function that is central to our Chrome extension's annotation process.

B. Semantic Annotation and Embedding Generation

Upon detection, UI element text (prioritizing `innerText`, then `placeholder`, and finally `aria-label`) is sent to the `gpt-4-turbo` API for semantic annotation. Simultaneously, the `text-embedding-ada-002` API generates a 1536-dimensional embedding vector. These embeddings are later converted to NumPy arrays and indexed using FAISS for efficient similarity search.

Listing 1 shows the function used for semantic annotation. Notice how we now insert descriptive commentary before and after the listing to improve readability and separation from adjacent content.

```
function fetchSemanticAnnotation(text) {
  return fetch(API_URL, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${API_KEY}`,
    },
    body: JSON.stringify({
      model: "gpt-4-turbo",
      messages: [
        {
          role: "system",
          content: "You are an AI that provides
              structured semantic annotations for
              web elements."
        },
        {
          role: "user",
          content: `Provide a structured JSON
              annotation for this web element: "${
              text}". Include a short description.`
        }
      ],
    }),
  })
  .then((response) => response.json())
  .then((data) => {
    let annotation = data.choices[0]?.message?.
        content || "No annotation available";
    collectedAnnotations.push({ text, annotation });
    fetchEmbedding(text).then((embedding) => {
      collectedEmbeddings[text] = embedding;
    });
    return annotation;
  })
  .catch((err) => {
    console.error("Annotation fetch failed:", err);
    return "No annotation available";
  });
}
```

Listing 1. Function to Fetch Semantic Annotations

Following this function, we describe our overall approach with a custom algorithm that integrates both semantic annotation and embedding generation.

C. Proposed Algorithm for Semantic Annotation

The following algorithm (Algorithm 1) details the step-by-step process for annotating UI elements and generating the corresponding vector embeddings.

Algorithm 1 Automated UI Element Semantic Annotation

Input: Web page DOM
Output: JSON annotations and vector embeddings for each UI element
for all detected UI element in DOM **do**
    Extract text content (use `innerText`, then `placeholder`, then `aria-label`)
    annotation ← `fetchSemanticAnnotation(text)`
    embedding ← `fetchEmbedding(text)`
    Store {text, annotation, embedding} in data repository
end for
Convert embeddings to NumPy arrays
Build FAISS index with the embedding array
return FAISS index and annotations

After presenting the algorithm, we now discuss how the system creates interactive overlays for user interaction.

D. Overlay Creation and User Interaction

Once semantic annotations are obtained, the next step involves generating overlay labels that are dynamically created and strategically positioned adjacent to the corresponding UI elements. These elements are identified and localized within the viewport using their bounding rectangles, which are retrieved through the `getBoundingClientRect()` method. This method provides precise positional and dimensional information about each element relative to the viewport, allowing the overlays to be accurately aligned with their targets. The overlay labels serve as visual and interactive affordances that bridge the semantic annotations with the actual interface components. Each overlay is designed to be context-aware and is typically rendered as a lightweight HTML element layered on top of the interface. These overlays not only display informative labels but also offer interactive capabilities such as highlighting, scrolling the associated UI component into view, and simulating user actions like clicks. These interactions are facilitated through event listeners that are bound to the overlays and trigger DOM manipulations or API calls that mimic user behavior.

E. Evaluation Pipeline

Evaluation is performed using a Python script that leverages NumPy, FAISS, and NLTK. The performance evaluation involves:

- Building a FAISS index from the generated embeddings.
- Measuring average search time (approximately 0.446 seconds).
- Computing annotation quality using BLEU scores.

To better understand the evaluation process, we now provide an excerpt from the Python script used in our experiments.

This listing is separated by explanatory text from the surrounding content.

```
import json, time, numpy as np, faiss
from nltk.translate.bleu_score import sentence_bleu,
     SmoothingFunction

with open("extension_data.json", "r") as f:
    data = json.load(f)
embeddings_dict = data["embeddings"]
element_texts = list(embeddings_dict.keys())
element_embeddings = np.array(list(embeddings_dict.
    values()), dtype='float32')

dimension = element_embeddings.shape[1]
index = faiss.IndexFlatL2(dimension)
index.add(element_embeddings)

query = "Sign in" # Example query
results, distances = index.search(query_embedding,
    1)

smooth_fn = SmoothingFunction().method1
for record in annotations:
    candidate = record["annotation"]
    reference = ground_truth[record["text"]]
    score = sentence_bleu([reference.lower().split()
        ], candidate.lower().split(),
        smoothing_function=smooth_fn)
    print(f"{record['text']}: {score:.3f}")
```

Listing 2. FAISS Search and BLEU Score Computation

F. Comparison Summary

Table I provides a summary comparison between the proposed model and traditional models for UI annotation.

TABLE I
COMPARISON BETWEEN PROPOSED AND TRADITIONAL UI ANNOTATION MODELS

Criteria	**Traditional Model**	**Proposed Model**
Automation Level	Semi-automated/manual	Fully automated
Annotation Method	Rule-based/Manual	NLP-based with AI
Vector Search	Not integrated	FAISS-enabled
Real-time Performance	Limited	Near real-time (0.446s)
Scalability	Low	High

V. DISCUSSION

Our technical evaluation demonstrates the system's capability for near-real-time performance and semantic annotation. The computational cost mainly arises from API calls to OpenAI models and FAISS indexing; however, by processing tasks in parallel, we maintain efficiency even as the number of UI elements increases. Effectiveness is measured via BLEU scores and search latency, and while BLEU scores indicate areas for improvement, they provide a consistent measure of annotation quality.

VI. CONCLUSION AND FUTURE WORK

This paper introduced a Chrome extension for automated semantic annotation of web UI elements that leverages Semantic Web technologies, advanced NLP models, and FAISS for scalable similarity search. The novelty of our approach lies in its complete automation, real-time performance, and integration of state-of-the-art embedding methods. Practical limitations include potential API latency, domain-specific annotation challenges, and the need for further fine-tuning of the NLP models. Future work will focus on addressing these limitations, expanding support for diverse web applications, and incorporating additional datasets for evaluation [2], [3].

DATASET REFERENCE

In our current implementation, evaluation data is generated from live web interactions and OpenAI API responses.

REFERENCES

[1] Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5), 34-43.

[2] Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3), 535-547.

[3] OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., et al. (2024). GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*. Retrieved from https://arxiv.org/abs/2303.08774.

[4] Xiao, S., Chen, Y., Song, Y., Chen, L., Sun, L., Zhen, Y., & Chang, Y. (2024). UI Semantic Group Detection: Grouping UI Elements with Similar Semantics in Mobile Graphical User Interfaces. In *Displays 2024*.

[5] Kim, H., Mitra, K., Li Chen, R., Rahman, S., & Zhang, D. (2024). MEGAnno+: A Human-LLM Collaborative Annotation System. In *Proceedings of the EACL 2024 Demo Track*.

[6] Park, S., Song, Y., Lee, S., Kim, J., & Seo, J. (2025). Leveraging Multimodal LLM for Inspirational User Interface Search. In *Proceedings of the 2025 SIGCHI Conference on Human Factors in Computing Systems*.

[7] Sunkara, S., Wang, M., Liu, L., Baechler, G., Hsiao, Y., Chen, J., Sharma, A., & Stout, J. (2022). Towards Better Semantic Understanding of Mobile Interfaces. In *Proceedings of the 29th International Conference on Computational Linguistics (COLING 2022)*.

[8] Baechler, G., Sunkara, S., Wang, M., Zubach, F., Mansoor, H., Etter, V., Cărbune, V., Lin, J., Chen, J., & Sharma, A. (2024). ScreenAI: A Vision-Language Model for UI and Infographics Understanding. In *Proceedings of the 2024 International Joint Conference on Artificial Intelligence (IJCAI-24)*.