Lighthouse++

A Search and Generative Engine Optimization Tool for VSCode

Bachelor's Thesis in Computer Science and Engineering

Adam Jöeäär
Daniel Alm-Eriksson
Kevin Pettersson
Mohammad Hamdan
Simon Johansson
Yousef Noufal

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2026
www.chalmers.se

Bachelor's Thesis 2026

Lighthouse++

A Search and Generative Engine Optimization Tool for VSCode

Adam Jöeäär
Daniel Alm-Eriksson
Kevin Pettersson
Mohammad Hamdan
Simon Johansson
Yousef Noufal

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2026

Lighthouse++
A Search and Generative Engine Optimization Tool for VSCode
Adam Jöeäär
Daniel Alm-Eriksson
Kevin Pettersson
Mohammad Hamdan
Simon Johansson
Yousef Noufal

(The authors are listed alphabetically by first name. All authors contributed equally to the project execution and the writing of this report.)

Supervisor: Haroon Elahi, Department of Computer Science and Engineering
Examiner: Patrik Jansson, Department of Computer Science and Engineering
Graded by teacher: Sandro Stucki, Department of Computer Science and Engineering

Bachelor's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The logo for the Lighthouse++ VS Code extension, featuring a stylized lighthouse.

Typeset in LaTeX
Gothenburg, Sweden 2026

Adam Jöeäär, Daniel Alm-Eriksson, Kevin Pettersson, Mohammad Hamdan, Simon Johansson, Yousef Noufal

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Search engine optimization (SEO) remains an important challenge for e-commerce businesses in the ever-expanding and competitive World Wide Web (WWW) landscape. The emerging AI-driven information retrieval trends have added new dimensions to the WWW utility and further complicated the SEO problem.

Traditional SEO is largely performed using external tooling, disrupting the flow as developers move back and forth between environments. In addition, current tools do not adequately consider the evolving AI-driven search landscape.

This thesis presents the design and implementation of Lighthouse++, a Visual Studio Code (VS Code) extension for automated analysis and improvement of webpage quality with a primary focus on on-page SEO and Generative Engine Optimization (GEO). The system is built around Google Lighthouse and extends it with AI-assisted suggestions, a set of custom metrics, and an exploratory GEO component intended to provide heuristic feedback rather than predictive ranking-based estimates.

In this regard, the thesis proposes a GEO-oriented evaluation framework based on semantic similarity and source visibility, intended to capture how webpage content may align with retrieval and answer generation in large language model systems. The developed extension audits webpages, identifies technical and structural issues, and generates suggested code changes intended to improve Lighthouse-related outcomes while also exposing GEO-related feedback as an exploratory complement to traditional SEO analysis. Finally, Lighthouse++ comes with improved usability by enabling developers to code and test their websites for S&GEO in their IDEs.

The project demonstrates the effectiveness of the proposed tool by evaluating and showing improved metrics across 80 pages from 18 different websites. The evaluation shows an increase in all metrics and that AI-assisted SEO and GEO are possible.

Sammandrag

Sökmotoroptimering (SEO) fortsätter att vara en ständig utmaning för e-handelsföretag i det expanderande och konkurrensutsatta landskapet för World Wide Web (WWW). De framväxande trenderna i AI-driven informationshämtning har infört nya dimensioner för användbarheten av WWW och fortsatt att komplicera SEO-problematiken.

Traditionell SEO utförs huvudsakligen med externa verktyg, vilket stör arbetsflödet när utvecklare behöver växla fram och tillbaka mellan olika miljöer. Dessutom tar nuvarande verktyg inte i tillräcklig grad hänsyn till det föränderliga AI-drivna söklandskapet.

Denna tes presenterar designen och implementeringen av Lighthouse++, ett Visual Studio Code (VS Code)-tillägg för automatisk analys och förbättring av webbplatsskvalitet, med primärt fokus på *on-page* SEO och generativ sökmotoroptimering (GEO). Systemet är byggt kring Google Lighthouse och expanderar det med AI-baserade förslag, egna mätvärden samt en explorativ GEO-komponent avsedd att ge heuristisk återkoppling snarare än prediktiva, rankingbaserade uppskattningar.

I detta avseende presenterar tesen en GEO-orienterat utvärderingsramverk baserat på semantisk likhet och källsynlighet, avsett att fånga hur hemsidors innehåll kan överensstämma med informationshämtning och svarsgenerering i system baserade på stora språkmodeller. Det utvecklade tillägget utvärderar hemsidor, identifierar tekniska och strukturella problem och genererar kodbaserade förbättringsförslag avsedda att förbättra Lighthouse-relaterade resultat, samtidigt som GEO-relaterad återkoppling presenteras som ett explorativt komplement till traditionell SEO-analys. Slutligen förbättrar Lighthouse++ användbarheten genom att låta utvecklare programmera och testa hemsidor för S&GEO direkt i utvecklingsmiljön.

Projektet demonstrerar effektiviteten i det föreslagna verktyget genom att utvärdera och påvisa förbättrade mätvärden för 80 sidor från 18 olika hemsidor. Utvärderingen visar en ökning av alla mätvärden och att AI-assisterad SEO och GEO är möjliga.

Keywords: SEO, GEO, Visual Studio Code, Web Optimization, Google Lighthouse, Large Language Models.

Acknowledgements

We would like to express our gratitude to our supervisor, Haroon Elahi, for his guidance, constructive feedback, and support throughout the project. His advice helped us refine both the technical direction of the work and the structure of this thesis.

Finally, we would like to thank our examiner, Sandro Stucki, and the Department of Computer Science and Engineering at Chalmers University of Technology and the University of Gothenburg, Sweden, for providing the academic framework in which this project was carried out.

Adam Jöeäär, Daniel Alm-Eriksson, Kevin Pettersson, Mohammad Hamdan, Simon Johansson, Yousef Noufal, Gothenburg, June 2026

Contents

1

Introduction

Search Engine Optimization (SEO) consists of the techniques for improving the quantity and quality of traffic to web pages via organic search results from search engines, such as Google Search or Bing. The term "organic" refers to search results that appear strictly due to their algorithmic relevance to the user's query, distinct from paid advertisements or sponsored placements [1].

The study of SEO is not merely a commercial pursuit. It sits at the intersection of computer science, information economy, and user behavior. From an academic standpoint, SEO presents a unique challenge in the field of information retrieval. It represents a continuous and dynamic relationship among search engine engineers, who aim to provide unbiased results, and webmasters, who aim to maximize visibility [2].

In a similar way, Generative Engine Optimization (GEO) concerns techniques for improving how web content is discovered and used by LLM-based systems. However, unlike SEO, which primarily targets ranking in conventional search results, GEO focuses on whether a webpage is retrieved, represented, and cited when a model generates an answer. As search behavior increasingly shifts from link-based retrieval toward direct answer generation, GEO becomes an important complement to traditional SEO.

This creates a complex engineering problem: ensuring website architecture is both performant for the user and accessible to the crawler. The rise of "Core Web Vitals" metrics has shifted the engineering focus toward a metric-driven user experience (UX), forcing developers to optimize Critical Rendering Paths and minimize layout shifts [3]. Furthermore, as search engines move toward semantic search, the field increasingly overlaps with computational linguistics and the study of structured data.

1.1 Challenges with SEO

Search engines serve as the primary tool for internet users seeking websites that meet their needs. Upon receiving a user query, search engines execute complex algorithms to match

the request with relevant websites based on various parameters [4].

Google dominates the search engine market. The specific mechanics of its crawling, indexing, and ranking systems remain largely treated as a black box. Although Google provides open-source tools like Lighthouse for performance analysis [5], the core logic behind the handling of searches remains undisclosed. This lack of transparency introduces significant complexity when trying to verify the trustworthiness of search data. To build a reliable and objective framework, we must remain skeptical of official claims and strive to decode the internal mechanisms/algorithms that drive these systems.

Modern applications are also built with diverse technologies, and developers have access to numerous development tools. The fundamental principle of web development relies on three core technologies: HTML to represent the main document structure, CSS for styling, and JavaScript to introduce dynamic behavior.

Today's web development primarily focuses on dynamic websites, where the user's web browser continuously receives updates about what content to display. These are often called JavaScript-heavy applications. This approach makes websites feel more like programs running directly in your browser rather than static pages [6].

Various frameworks are commonly used to implement these web applications, each offering different approaches to building interactive web applications. Although these frameworks share similarities, they have distinct characteristics and different implementation details [6]. This diversity adds a dimension of complexity to web scraping and content extraction. Instead of simply parsing what's rendered on the screen, we need to define methods for parsing and evaluating the content as it's received and processed by the browser.

1.2 Challenges with GEO

Another important aspect of the problem is the emergence of AI-powered search systems and Generative Engine Optimization (GEO), which are increasingly influencing how users discover web content [7]. Modern search engines integrate AI-generated summaries directly within the Search Engine Results Page (SERP), exemplified by features like Google's AI Overviews [8]. In addition, users are increasingly turning to AI-based assistants such as ChatGPT and Perplexity as first-hand sources of information retrieval, further shifting traffic away from traditional search result navigation [9].

Although these systems are based on different underlying ranking and retrieval algorithms, they represent a vital consideration for this project as they fundamentally change how

visibility is achieved.

As the field of GEO is new and emerging, there are no clear-cut metrics for judging whether a website is optimized for AI-based search. The models are highly probabilistic by nature, which introduces significant challenges when trying to devise a deterministic evaluation framework. There have, however, been various attempts at formalizing metrics, such as GC-GSEO-Bench [10]. They will serve as a basis for our subsequent research and work.

1.3 Existing tools

Multiple existing tools are attempting to tackle the aforementioned problems, coming in both commercial and open source forms.

Commercial variants include Semrush [11], Ahrefs [12], and SE Ranking [13]. However, as commercial platforms, their use involves recurring subscription costs, which may limit accessibility for individuals or small organizations. This motivates the exploration of open-source alternatives that reduce financial barriers while providing comparable functionality.

One such alternative is Lighthouse, a tool released by Google itself. It is an automated auditing tool, allowing web developers to discover problems with their websites. Lighthouse primarily audits four categories: Performance, Accessibility, Best Practices, and SEO [5]. Each of these categories is assigned a score from 0 to 100. Optimizing the metrics evaluated by Lighthouse gives website developers a means of improving page quality, which helps to increase ranking in search engines.

A common characteristic among these tools is that they operate independently of the software development process. Commercial variants require visiting an external website, while Lighthouse is tightly coupled to the Google Chrome browser [5]. Locating the problems identified by these tools within the source code is often difficult. Additionally, switching back and forth between the development environment and web browser induces a disruption of flow.

As the field of GEO is relatively new, most existing tools have limited or no support for it. Lighthouse, for instance, does not address AI-based search in any way.

1.4 Aims and Scope

To address the aforementioned challenges and the limitations of existing tools, this project aims to design and develop a Search and Generative Engine Optimization (S&GEO)

tool for automated analysis of modern websites. The project focuses on empowering developers in evaluating technical and structural aspects of individual web pages that influence search engine visibility and overall web quality, and improving the developer experience in performing such evaluations and relevant improvements.

To achieve the stated aims, Lighthouse++ is designed and developed. Lighthouse++ analysis is primarily based on widely used industry-scale measurements used by tools, such as Google Lighthouse [14], which serve as an adopted foundation for evaluating performance, accessibility, and established best practices. In addition, Lighthouse++ integrates custom-defined measurements targeting on-page factors such as structural HTML usage, metadata completeness, and other relevant on-page factors. These additional measurements are used to better reflect SEO considerations that are not fully covered by existing tools.

In addition to traditional analysis, Lighthouse++ performs automated Generative Engine Optimization (GEO) analysis. This analysis generates heuristic and indicative recommendations intended to improve content visibility in AI-driven search and retrieval systems. Due to the proprietary and rapidly evolving nature of large language models' ranking mechanisms, this analysis does not attempt to predict or model AI ranking behavior.

Lighthouse++ collects and processes these measurements to produce an overall evaluation of a given webpage. Based on the results, Lighthouse++ generates readable feedback and improvement recommendations, aiming to support both developers and non-technical users in understanding how a webpage may be optimized. These recommendations are readily available in the user's development environment, reducing friction and maintaining a closer connection with the source code of the website.

To maintain a manageable project scope and enable a deeper analysis, several aspects of SEO are explicitly excluded from this work. The project does not address off-page factors, such as backlinks, domain authority, or external traffic data. These factors are excluded because they depend on external systems and large-scale data collection, which would significantly increase project complexity and reduce the focus on measurable, page-level analysis.

Furthermore, the tool is not intended to work as a generalized solution for multiple search engines. Instead, the analysis is focused on principles and metrics commonly used within Google's ecosystem. Supporting several search engines would require handling different ranking models and evaluation rules, which could reduce the accuracy and clarity of the results. By limiting the scope in this way, the project aims to deliver a more focused and higher-quality solution rather than a broad but less reliable one.

Finally, the system is intended as a decision support and evaluation prototype, not as a full commercial SEO platform. The goal of the tool is to demonstrate how automated measurements, together with custom-defined metrics and AI-assisted analysis, can be used to evaluate the SEO quality of individual web pages. The system is designed to provide meaningful feedback and improvement suggestions, not to provide complete ranking predictions or guarantees.

1.5 Contributions

In summary, these are the contributions that our tool and research intend to make:

- *Decoupling of Lighthouse and integration with common development environment*: Lighthouse++ enables developers to perform Search & Generative Engine Optimization without switching back and forth between code editor and browser.
- *Framework for evaluating Generative Engine Optimization*: A theoretical framework for generative engine optimization is designed to be integrated into Lighthouse++. This framework adds new features based on heuristic similarity and visibility metrics to assess whether website content is optimized for inclusion in generative engine outputs.
- *Automated GEO analysis*: Lighthouse++ implements and automates the aforementioned framework for GEO analysis, currently lacking or fully absent in existing tools available to developers.
- *AI-assisted improvement suggestions*: Lighthouse++ provides LLM-based insights and suggestions for code level improvements. It guides developers in how to interpret Lighthouse++ audit results and can automatically apply fixes. Lighthouse++ also includes both lightweight quick fixes and a full agentic approach that explores the codebase on a deeper level.

2

Background

To fully understand the following discussion and implementation, we first present the initial literature review, theory behind search engines, Lighthouse metrics and the technologies involved in Generative Engine Optimization.

2.1 Literature Review

The project began with a review of the literature on SEO and search engines in particular. Initially, all major search engines, such as Google, Bing, and Yandex, were part of the review. This included their own documentation [15][16][17], but also several external papers about the inner workings of search engines and optimization of search results. Realizing that search engines mostly follow a similar ranking process, Google was chosen as the primary research subject. It accounts for a strong majority of the market [18], and we felt that a narrower focus would lead to a better result.

It was also important to study existing tools and previous contributions in this field to avoid reinventing the wheel. Multiple industry-standard commercial products, such as Semrush [11] and Ahrefs [12], were briefly studied, mainly to determine common features in such products. The primary focus, however, was directed towards open source tools, including the tool proposed in the paper "Search Engine Optimization (SEO) Auditing Tool" [19], SeoNaut [20], and Google Lighthouse [5]. These were used to guide us in the design and implementation of our own tool.

Since GEO is a fresh and evolving field, there is still no widely established methodology or standard tool support comparable to what already exists for SEO. This made the pre-study especially important, since we first needed to understand how GEO is currently discussed in industry and research, what kinds of optimization signals are considered relevant, and how such signals could be translated into a practical evaluation workflow.

The review, therefore, also included an investigation of how large language models retrieve, rank, and use web content in generated answers. One paper that proved especially useful

was CC-GSEO-Bench [10]. It acted as a conceptual basis for the two main aspects that later shaped our implementation: semantic similarity and source visibility in LLM-generated responses.

2.2 Search Engines Mechanisms

During the search process, search engines operate as large-scale information retrieval (IR) systems, designed to satisfy user intent. They use a ranking process to prioritize the inclusion and positioning of a webpage in the search results presented to the user. This ranking process is dynamic and relies on proprietary algorithms. To understand how optimization techniques influence visibility, one must first understand the underlying mechanisms of the search engine itself.

The processing of a search query generally involves three primary stages: crawling, indexing, and ranking [4].

2.3 Crawling and Indexing

Crawling is the discovery process in which search engine bots (often referred to as "spiders" or "crawlers") scour the internet for content. This process begins with a known list of Uniform Resource Locators (URLs) and follows hyperlinks on those pages to discover new URLs.

From an engineering perspective, crawling is a resource-intensive operation due to the massive number of websites. Search engines operate under a "crawl budget," allocating specific resources to a domain based on its server performance and update frequency [21]. Once a page is discovered, it is rendered—including all JavaScript—and stored in a database known as the index, from which it can be retrieved.

2.4 Ranking Algorithms

Once a user inputs a query, the search engine acts as an IR system to select the most relevant resources from its index. The ranking process is governed by proprietary algorithms to determine the order of the results [4]. While the exact algorithms are undisclosed, they generally fall into two categories:

- **On-page:** These refer to the semantic relevance of the content to the user's query, the structure of the HTML (headings, meta tags), and the technical performance of the page (loading speed, mobile-friendliness)[22].

- **Off-page:** These primarily involve the authority of the domain, historically calculated using algorithms like PageRank [23]. This view hyperlinks from other reputable sites as "votes of confidence," suggesting that the target content is trustworthy.

Modern ranking has evolved from simple keyword matching to semantic understanding. With the integration of Natural Language Processing (NLP) models, search engines can now interpret the intent behind a query rather than simply matching strings of text [24].

2.5 Lighthouse Metrics

Since part of the evaluation of the proposed tool relies on Lighthouse scores, it is important to understand the metrics that contribute to these scores. Lighthouse evaluates webpages across four categories: Performance, Accessibility, Best Practices, and SEO [25].

Each category score is computed from a set of underlying audits. These audits are aggregated using a weighted scoring model, where each audit contributes differently depending on its relative importance. The resulting score ranges from 0 to 100, where higher values indicate better compliance with Lighthouse [26].

The Performance category measures the page loading speed and responsiveness from the user's perspective. The five metrics are:

- First Contentful Paint (FCP) measures the time from navigation until the first visual element is rendered on the screen.
- Speed Index (SI) measures the time it takes for content to visually populate the screen.
- Largest Contentful Paint (LCP) measures the time until the largest visible content element in the viewport is rendered.
- Total Blocking Time (TBT) measures the total time during which the main thread is blocked and unable to respond to user input after FCP.
- Cumulative Layout Shift (CLS) measures the extent of unexpected layout shifts occurring during page load.

The Accessibility category evaluates whether a webpage can be effectively used by a wide range of users, including individuals with disabilities. Lighthouse performs a wide set of audits to aggregate a score. Examples of evaluated accessibility audits include color contrast ratios, accessible names for buttons and links, form labels, document language specification, image alternative text, heading structure, and keyboard navigation support

[27].

The Best Practices category evaluates whether a webpage adheres to established web development practices. The category includes audits related to security, compatibility, and maintainability. Examples of evaluated aspects include the use of HTTPS, specification of the viewport meta tag, presence of a valid HTML doctype, browser console errors, and charset declaration [28].

The SEO category evaluates whether a webpage follows fundamental technical SEO practices that help search engines discover, crawl, and understand its content. The category includes audits related to metadata, crawlability, mobile usability, and content discoverability. Examples of evaluated aspects include the presence of a meta description, descriptive link text, canonical URLs, robots.txt configuration, and proper indexing directives [29].

2.6 Text Embeddings and Similarity Measures

Traditional information retrieval systems often relied on the occurrence of identical words in both the query and the document [30]. However, text can express the same intent using different vocabulary, thus making exact matching insufficient for capturing semantic relationships.

To address this limitation, text can be represented as numerical vectors known as embeddings. Embeddings are generated from machine learning models trained on large bodies of text to capture the meaning and intent behind words and text segments. These models map words and text into a high-dimensional vector space. In this representation, text with similar meanings are positioned closer to each other, while semantically unrelated text are further apart [31].

Once text has been converted to embeddings, similarity measures can be used to quantify the semantic relationship between them. One popular approach is to use cosine similarity, which is calculated as follows:

$$\text{sim}(q_i, c_j) = \frac{\vec{q_i} \cdot \vec{c_j}}{\|\vec{q_i}\| \, \|\vec{c_j}\|}$$

Here, similarity is computed as the dot product of the two vectors divided by the product of their magnitudes. By dividing by their magnitudes, cosine similarity is normalized and depends only on the angle between the vectors rather than their absolute lengths [32].

Cosine similarity lies in the interval $[-1, 1]$ where the values can be interpreted as follows:

- 1 means the vectors point in the same direction (maximum similarity)

- 0 means no directional similarity (orthogonal vectors)
- -1 means opposite direction.

Embeddings are widely used in modern semantic search and retrieval systems. By converting both queries and documents into vector representations, similarity measures such as cosine similarity can be used to identify content that is semantically relevant even when the same keywords are not present. This principle forms the basis of the retrieval component used in the GEO evaluation presented in this thesis.

2.7 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) is an approach in which a large language model generates answers using not only its internal learned parameters, but also external information retrieved at inference time. In a typical RAG setting, a user query is first used to retrieve relevant documents or chunks from an external collection. The retrieved material is then combined with the original query and provided to the language model as context for answer generation [33].

This approach is especially relevant in knowledge-intensive settings, where relying only on the model's internal parameters may lead to outdated, incomplete, or hallucinated responses. By incorporating retrieved external information, RAG can support more grounded answer generation and make it possible to use domain-specific or up-to-date content during inference [33].

RAG is relevant to the present project because it reflects the type of environment in which Generative Engine Optimization (GEO) becomes important. If large language models retrieve webpage content and use it as part of the answer-generation process, then optimization is no longer only about achieving visibility in traditional ranked search results. It also becomes important that webpage content can be retrieved as relevant context and that it can contribute to the final generated answer. This is directly reflected in the present work, where retrievability and visibility are used as two central aspects for evaluating GEO.

3

Methodology

This chapter presents the methodology applied throughout the project. It covers the selection of development platforms and tools, the established workflow, the formalization of the metrics defined for Generative Engine Optimization, and the evaluation used to assess the developed tool.

3.1 SEO Open Source Alternatives

Due to the large availability of existing open source SEO tools and limited time, we chose to extend the existing Google Lighthouse tool rather than implementing one from scratch. Being widely adopted in the developer community and published under an Apache 2.0 license [14], it formed a good foundation for our tool.

One benefit of Lighthouse is that it doesn't just inspect static HTML; it runs a headless Chromium browser instance to evaluate a web page under controlled conditions. This overcomes some of the challenges with modern JavaScript-based websites, where content is often dynamically rendered.

Lighthouse operates through a structured pipeline that transforms a live website into a series of scores via three main phases: Gathering, Auditing, and Reporting. It begins with the Gatherers; it controls the browser instance to load the page under specific conditions, such as simulated mobile hardware or throttled network speeds, to collect raw outputs, which are called Artifacts. These artifacts include the network logs, the DOM snapshot, and many more. Once gathered, this raw data is passed to the Audits, which serve as the judges. During this stage, Lighthouse analyzes the artifacts to calculate specific metrics compared to best practices. Finally, these individual audit results are compiled into a report, where they are grouped into categories: Performance, Accessibility, Best Practices, and SEO.

3.2 Selection of Development Platform

Visual Studio Code (VS Code) is a code editor developed by Microsoft [34]. It is commonly employed by web developers to build websites. According to StackOverflow Developer Survey 2025 [35], over 75% of developers use it in their primary tool set.

VS Code also provides a rich interface for developing extensions that can extend the editor with additional functionality [36]. In addition, VS Code is free and available on multiple operating systems, which makes it available to a wide range of developers. These features can be used to spend more time on the development of the engine. Saving time on developing a solution outside an existing environment.

These qualities justified the choice of VS Code as the primary target ecosystem.

3.3 Software Development Approach

The project was carried out by a team of six students during the spring semester of 2026. Due to the interdisciplinary nature of the team, a structured development process was necessary to ensure effective coordination and task distribution.

An Agile development methodology was adopted, combining Scrum principles with a Kanban-style workflow managed through GitHub Projects. The project was initiated by decomposing high-level requirements into smaller tasks and milestones, which were then tracked as individual tickets.

Progress was monitored during weekly meetings, where the team reviewed completed work and planned upcoming tasks. In addition, the role of team lead and Scrum facilitator was rotated bi-weekly to distribute coordination responsibilities.

Task management followed a Kanban workflow where issues were assigned to team members, moved through development stages, and marked as completed upon review. This provided transparency regarding task ownership and project progress.

Version control was managed using Git and GitHub using a structured commit convention with prefixes such as ADD, FIX, and DELETE to improve traceability of changes. Feature development was performed on separate branches, and all changes were integrated into the main branch through code reviews to ensure code quality and consistency.

3.4 Choice of programming language and technologies

Since VS Code extensions are natively powered by TypeScript, the project is written in TypeScript. This alignment ensures better compatibility and performance. Additionally, using TypeScript allows for seamless integration with Lighthouse (which is written in JavaScript) as an NPM module.

The project utilizes Node.js as the industry-standard JavaScript runtime. To provide flexibility, our project uses the `openai` NPM package, which can be paired with any OpenAI-compatible API, such as ChatGPT, Groq, OpenRouter, or Ollama.

Most of the development utilized OpenRouter due to its unified interface, which simplifies switching between different LLMs. The platform also offers clear visibility into the pricing of various models, enabling task-specific optimization based on budget. The main limitation is the requirement for pre-purchased credits.

Providing support for Ollama also enables users to run local models, which greatly favors privacy and control over resource consumption [37].

3.5 Generative Engine Optimization Framework

This section presents the proposed Generative Engine Optimization (GEO) framework. The framework consists of two evaluation metrics, similarity and visibility, together with an LLM-based content optimization stage intended to improve estimated GEO performance.

3.5.1 Semantic Similarity Metric

Let $Q = \{q_1, \ldots, q_n\}$ be a set of generated candidate queries related to the webpage under analysis, intended to approximate the space of likely user information needs. Since no ground-truth dataset exists that maps arbitrary web pages to real user queries, this approach is used as a proxy to approximate a plausible query distribution over the document. It is important to note that this does not represent observed user queries, but rather a synthetically generated query set derived from the document itself using a large language model.

Next, let T denote the actual text content of the target webpage, and let $C = \{c_1 \ldots c_m\}$ be a partitioning of T into coherent chunks. Chunking is necessary as a pre-processing step before embedding, as it breaks a large document into smaller semantically focused pieces. This enables the model to identify relevant parts of the document rather than treating the entire page as equally relevant. To quantify how well a webpage aligns semantically with

potential user queries, we first embed both content chunks and queries into a shared vector space.

Formally, let $\phi(x)$ denote an embedding function that maps a piece of text to a vector representation in $\mathbb{R}^d$.

$$\vec{q}_i = \phi(q_i), \quad \vec{c}_j = \phi(c_j)$$

We define the semantic similarity between a query q_i and a chunk c_j using cosine similarity, which measures the similarity between two vectors in the high-dimensional embedding space $\mathbb{R}^d$. It is based on the cosine of the angle between them. We chose this approach due to its effectiveness in measuring semantic alignment in embedding spaces, independent of vector magnitude.

Next, for each query q_i, we find the chunk $c \in C$ that maximizes semantic similarity. This is based on the assumption that satisfying a query requires at least one strongly aligned content segment. We define the overall similarity score of the webpage as the mean of the maximum similarity score per query:

$$\text{Similarity}(T, Q) = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \max_{c \in C} \text{sim}(q_i, c)$$

This metric captures the semantic coverage of the webpage with respect to a generated query space, by evaluating how well different potential queries can be matched to at least one relevant content segment.

It is important to note that since the query set Q is conditioned on T, this might create a bias toward the resulting similarity score. However, this design is intentional, as the objective is not to estimate absolute user behaviour in real-world search settings, but to ensure a consistent and comparable evaluation framework across different webpages under the same query generation process.

3.5.2 Content Visibility Metric

While the similarity metric captures how well a webpage semantically aligns with potential user queries, it does not capture whether retrieved content is utilized in generated responses. To address this, we introduce a visibility metric that measures how webpage content is selected, utilized, and cited by a generative model when answering queries in the presence of competing sources.

Given the same set of queries Q (see Section 4.5.1), we first select a subset $Q' \subset Q$ that best represents and covers the content of the target webpage. This reduces redundancy as

many generated queries correspond to similar semantic intents and can be answered by the same content chunk. Including all such queries would bias the visibility score towards overrepresented topics and increase computational cost without improving coverage.

For each query $q \in Q'$, we construct a retrieval environment consisting of both the target webpage and a set of competing webpages. All documents are parsed into chunks and then embedded and stored in a joint retrieval index. Next, for each query, we retrieve the top-k most relevant chunks from the retrieval index based on semantic similarity with the query. These chunks may originate from either the target page or the competing webpages. The retrieved documents are then provided as context to a generative model, which is instructed to produce an answer grounded in the provided documents and to include explicit source citations.

To quantify how the generated answer utilizes different sources, we analyze the citation structure of each response at the span level. Let $S_q = \{s_{q,0}, \ldots, s_{q,n-1}\}$ denote the ordered set of cited spans in the generated response for query q, and let i indicate the position (index) of the span in the response. Further, consider $S_q^{\text{target}} \subseteq S_q$ as the subset of spans supported by the target webpage.

The target span share measures how much of a generated answer is supported by the target page.

$$\text{SpanShare}(q) = \frac{|S_q^{\text{target}}|}{|S_q|}$$

The coverage measures how often the target page contributes to the generated response:

$$\text{Coverage} = \frac{|\{q \in Q' \quad | \quad |S_q^{\text{target}}| > 0\}|}{|Q'|}$$

The metric includes a positional weight depending on where in the response the target is cited. Studies in web search and information retrieval show a strong positional bias. That is, items presented earlier receive the highest attention and click-through rates [38]. To account for this, we assign a higher importance to citations appearing earlier in the response. We model the positional influence as a linearly decaying function over the position of a cited span as:

$$w(s_{q,i}) = 1 - \frac{i}{|S_q| - 1}$$

We adopt a linear decaying function to avoid introducing bias beyond the assumption that earlier cited spans in the response receive the most attention. The positional score is

defined as the average over all spans supported by the target page:

$$\text{PositionalScore} = \frac{1}{|S_q^{\text{target}}|} \sum_{s_{q,i} \in S_q^{\text{target}}} w(s_{q,i})$$

We aggregate span share by averaging over the set of selected queries. However, the positional score is averaged over the subset of queries where the target page was cited, as it is undefined when no target spans are present. Coverage is computed directly as the fraction of queries where the target was present. The total visibility score is then the weighted sum of SpanShare, Coverage, and PositionalScore:

$$\text{Visibility}(Q') = \alpha \cdot \text{SpanShare} + \beta \cdot \text{Coverage} + \gamma \cdot \text{PositionalScore}$$

Coverage is weighted highest as it reflects whether the page is used at all, which is a prerequisite for any visibility.

3.5.3 Content Optimization

Although the metrics provide quantitative estimates of GEO performance, they do not directly explain how webpage content can be improved. To address this, we introduce an LLM-based optimization stage that transforms audit signals into actionable recommendations intended to improve estimated GEO performance. In particular, the system identifies:

- Low-similarity queries, representing cases where the best-matching content chunk lacked strong semantic alignment with the query.
- Retrieved-but-not-cited queries, representing cases where the generated answer included citations to competing sources while excluding the target webpage from the cited evidence.

The audit signals and webpage content are then provided as input to a large language model. The model is instructed to generate structured recommendations aimed at improving semantic retrievability, factual clarity, and citation likelihood in AI-generated responses. Rather than rewriting the entire webpage, the system produces concise and structured suggestions. This reduces computational cost and context length limitations while maintaining interpretability.

The output is guided by the E-E-A-T framework (Experience, Expertise, Authoritativeness, Trustworthiness), a widely used set of principles for evaluating and improving informational content quality [39]. The generated output consists of identified key issues,

structural improvements, and example content additions.

3.6 Evaluation

This section details the empirical procedure used to evaluate the effectiveness of the developed tool. The objective of this evaluation is to assess the tool's capacity to iteratively improve Lighthouse outcomes when operating in *Agent Mode*. It also includes the evaluation of the GEO pipeline, which follows a separate methodology designed to assess content visibility in generative engine outputs.

3.6.1 Benchmark and Execution

The evaluation was conducted using a benchmark of 20 repositories selected from GitHub. To ensure a representative sample of page structures, five distinct pages were selected from each repository, resulting in a total of 100 unique targets. Each target was processed in *Agent Mode* with a three-pass optimization loop using `openai/gpt-5.4-mini` via OpenRouter:

1. Baseline Audit: An initial Google Lighthouse audit was performed to establish the baseline scores for each target page.
2. Autonomous Intervention: Lighthouse++ analyzed the audit artifacts and applied code-level optimizations to the source code automatically.
3. Repeated Measurement: The current page state was audited again before subsequent passes, and a final Lighthouse rerun after the third pass recorded the paired post-intervention scores.

3.6.2 Analysis and Inclusion Criteria

Only targets with complete before-and-after measurements were included in the final paired dataset. This exclusion rule ensured that the score comparisons were based on matched observations rather than partial runs.

Data analysis was conducted at both the page level and the repository level. Aggregating results by repository allows for a broader view of the tool's effectiveness across shared codebases and reduces the weight of potentially redundant page structures. The analysis focuses specifically on the four core categories: Performance, Accessibility, Best Practices, and SEO. Statistical significance of the paired score improvements was assessed using the signed-rank test p-values for all comparisons.

3.6.3 GEO Evaluation

The GEO component was evaluated separately from the Lighthouse-based SEO pipeline. Therefore, the GEO pipeline was decoupled from the Lighthouse++ optimization loop to enable an isolated and controlled evaluation environment.

The evaluation dataset consists of 123 pages spanning five content domains: news articles, wiki-style pages, blog posts, documentation pages, and how-to guides. These categories were selected to represent a range of informational content types where GEO-related effects are expected to be observable. Pages such as landing pages and purely navigational homepages were excluded, as they do not contain sufficient structured informational content to meaningfully evaluate generative engine optimization.

The GEO benchmark evaluates each page by extracting the signals defined in Sections 3.5.1 and 3.5.2. These signals are aggregated into a single baseline score using their arithmetic mean, providing a unified representation of the initial page-level GEO performance. Next, each page is modified by applying the content optimization suggestions as described in Section 3.5.3. Specifically, example content additions are appended to the original page text. The same evaluation procedure is then applied to the modified page, producing a post-optimization score that is directly comparable to the baseline.

4

Implementation

This chapter provides a technical overview of the system implementation. The solution was developed as a Visual Studio Code (VS Code) extension, leveraging the editor's extension framework to manage the user interface, data persistence, and security-related functionality. The chapter focuses particularly on the integration with Google Lighthouse, including the implementation of the proposed GEO metrics and the use of large language models for evaluation and optimization.

4.1 Visual Studio Code extension

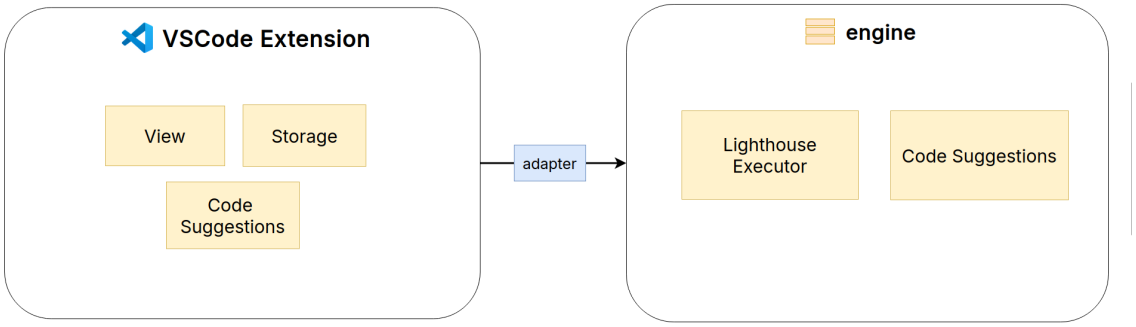

Figure 4.1: High-level architecture of extension and engine

Lighthouse++, a VS Code extension, features a modular architecture centered around an encapsulated core known as the `engine`. By decoupling this engine from VS Code, as seen in Figure 4.1, the extension ensures that its primary functionality remains portable, allowing for integration of a standalone command-line interface (CLI) or other user interfaces.

To facilitate integration with Visual Studio Code, an adapter layer was implemented to act as an interface between the editor and the engine. This adapter handles all interactions with the Visual Studio Code API, including accessing the workspace, displaying messages

to the user, and managing user-triggered actions.

This separation of concerns ensures that the core functionality can be reused across different platforms in the future, where only a new adapter is needed.

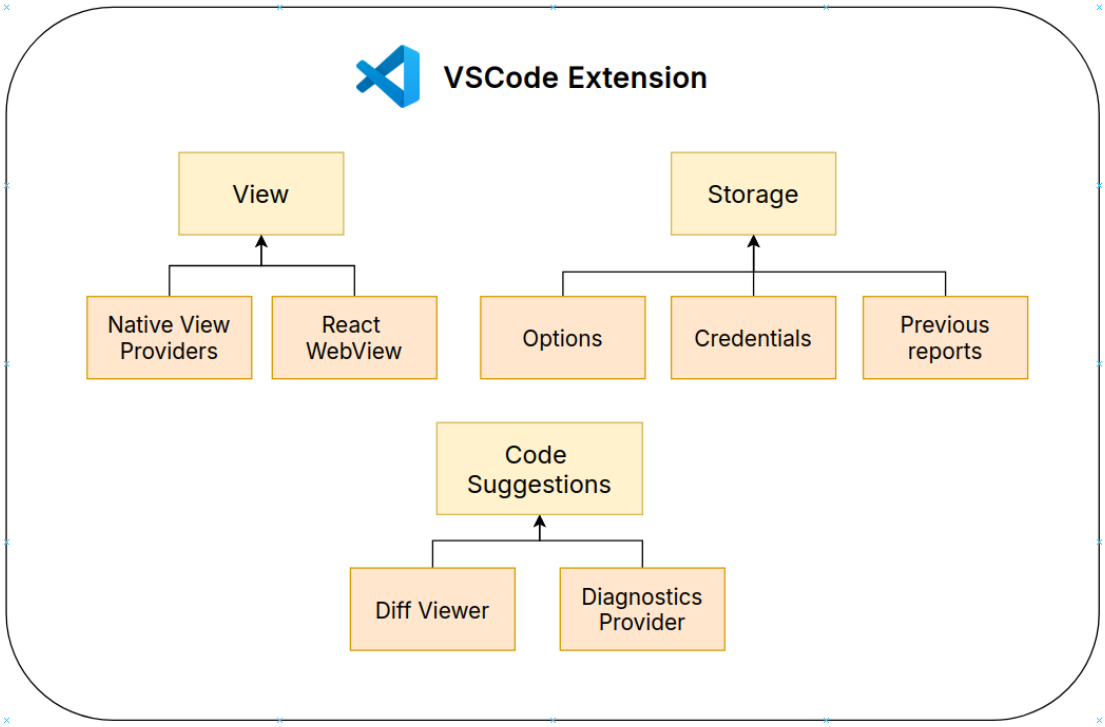

Figure 4.2: High-level architecture of extension

Much of VS Code's robust Extensibility API [36] was leveraged to manage UI components, webviews, and persistent storage to form the submodules seen in Figure 4.2.

4.1.1 User interface

The tool utilizes Native API components for the sidebar to ensure a consistent look and feel for method selection, while opting for HTML-based Web views to handle more complex data visualization. Specifically, both the welcome (see Figure 4.5) and settings (see Figure 4.6) pages and the Lighthouse result reports are built using ReactJS and rendered within these Web views. This approach allows us to utilize a modern framework to build reusable components and manage complex layouts that go beyond standard editor elements.

With these views in place, the extension provides easy-to-use interfaces for the user. Figure 4.3 showcases the integrated panel view inside VS Code. Here, the user selects the options they are interested in auditing.

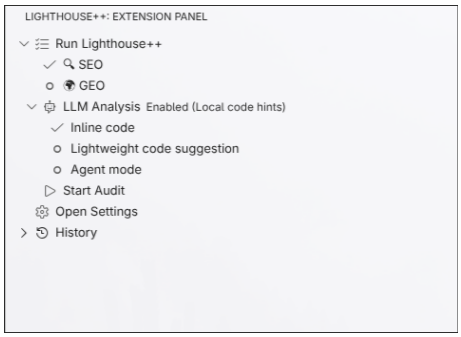

Figure 4.3: Option panel

When the Lighthouse++ pipeline has finished, the user is presented with an HTML report in a new tab within the VS Code workspace (see Figure 4.4). The report presents the complete audit results without requiring the user to leave the development environment, enabling a seamless and efficient workflow.

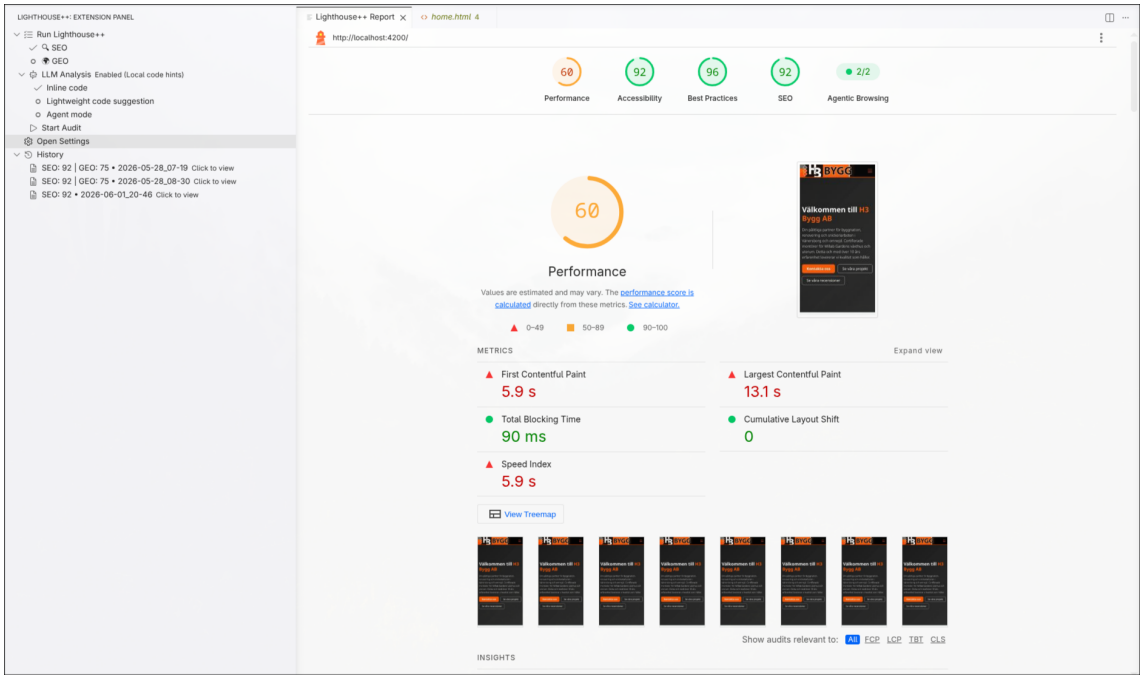

Figure 4.4: Lighthouse++ report directly seen in VS Code

For new users, a welcome screen is displayed when the extension is first launched (see Figure 4.5). From this screen, the user can go through the setup process and connect their APIs and models to the extension. A quick link to the documentation is also provided.

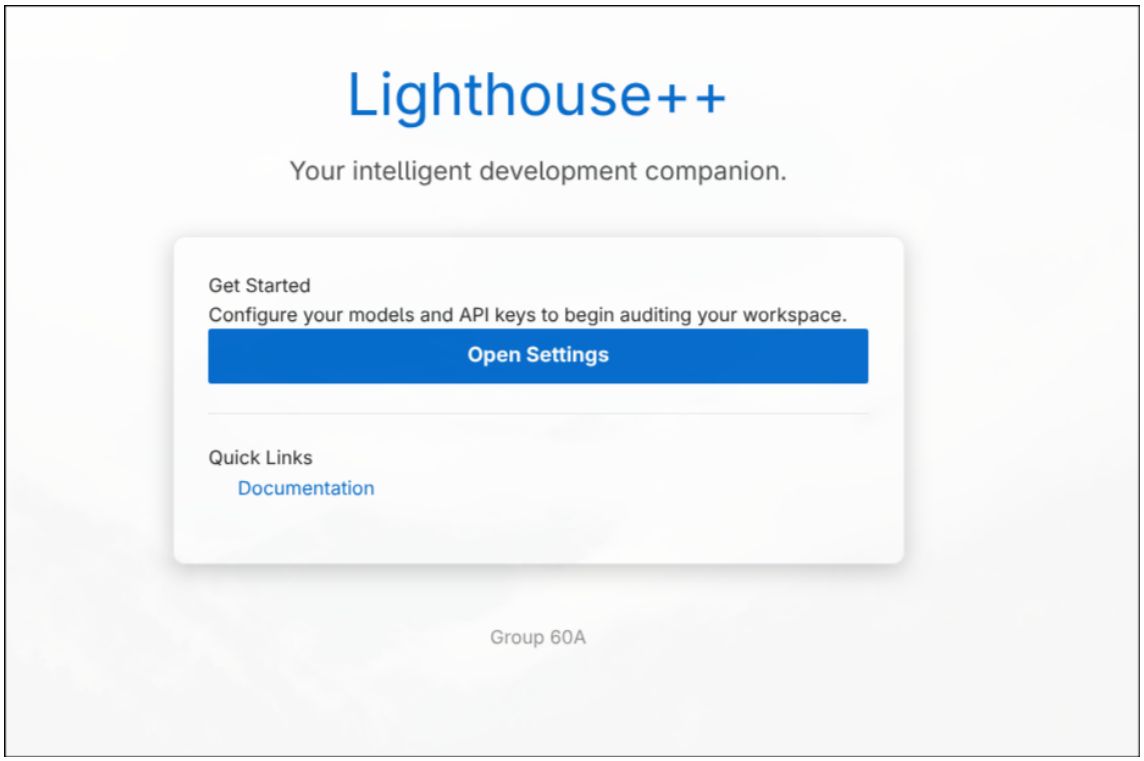

Figure 4.5: Welcome screen

Figure 4.6 shows the interface used for configuring the target URL that the extension should navigate to, the selected text and embedding models, and the API key.

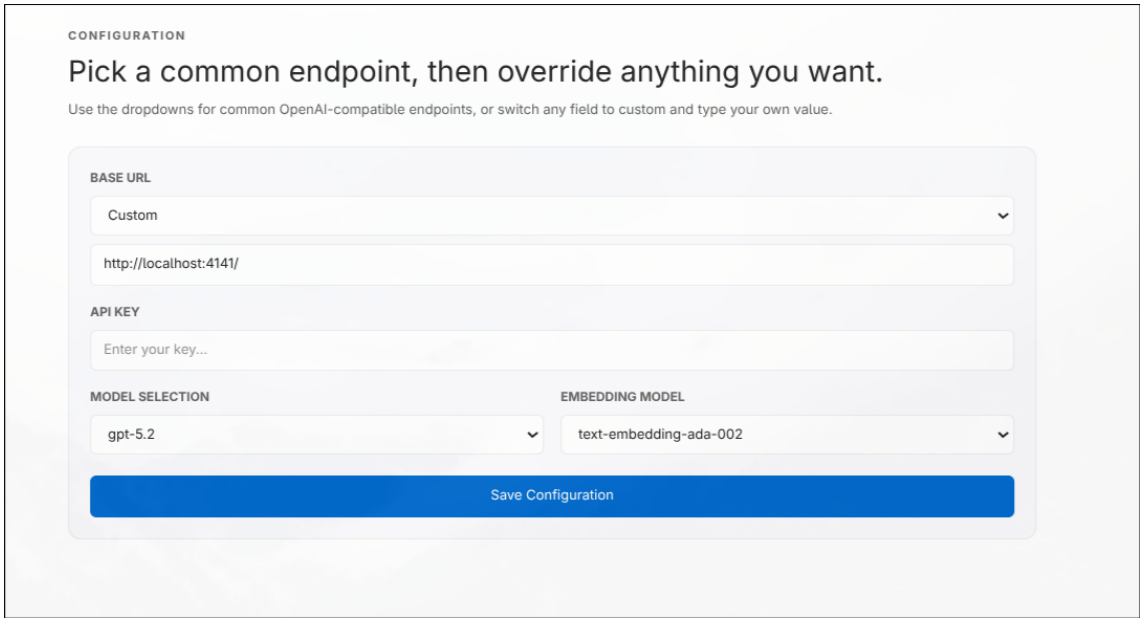

Figure 4.6: Settings page

4.1.2 User storage

Data persistence is handled through VS Code's dedicated storage solutions to balance accessibility with security. We utilize Global Storage to maintain user settings and a history of the last three generated reports (see Figure 4.7). However, to prioritize security, all sensitive information, such as API keys, is stored exclusively in VS Code's Secret Storage, ensuring that credentials remain encrypted and isolated from general configuration data.

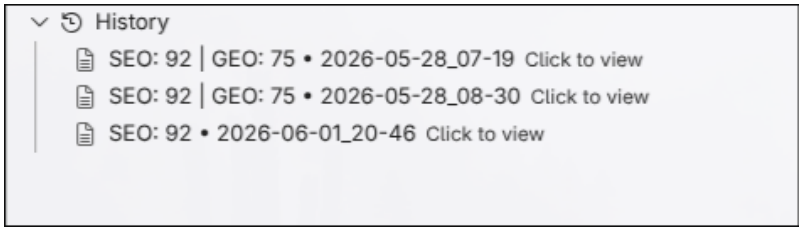

Figure 4.7: Storage panel

4.2 Lighthouse integration

An important part of this project was to decouple Lighthouse from the usual flow inside Google Chrome and instead provide reports directly in the coding environment. The solution for this uses the `lighthouse` NPM package to programmatically generate reports. The output HTML is then rendered in a VS Code WebView.

This still requires the use of the `chrome-launcher` library that will spawn a headless Chrome instance in the background. It is, however, not in the way of the user and provides a seamless integration with VSCode.

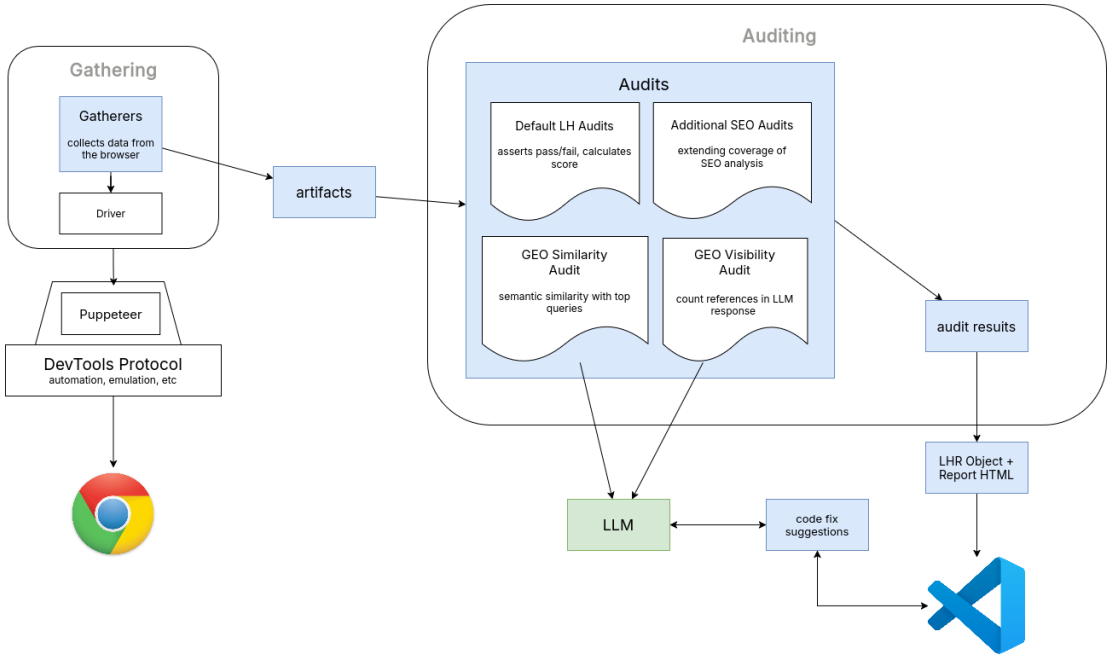

Figure 4.8: SEO architecture in our Lighthouse++

In order to extend Lighthouse with additional functionality, we relied on their extensive gather and audit pipeline. To define our own metrics, we developed gatherer and auditing classes similar to Lighthouse's own, and registered them in a custom configuration.

```
{
    audits: [
        ...defaultConfig.audits,
        { implementation: SimilarityAudit },
        { implementation: VisibilityAudit },
    ],
    artifacts: [
        ...defaultConfig.artifacts,
        {
            id: "GeoProfile",
            gatherer: { instance: new GeoProfile(llmConfig) },
        },
        {
            id: "LLMConfig",
            gatherer: { instance: new LLMConfigGatherer(llmConfig) },
        }
    ],
}
```

Listed below are the custom Lighthouse audits that the tool implements.

4.2.1 SEO audits

Table 4.1: Custom-defined SEO metrics

Audit	Description
Title	Checks whether the page title is optimal length for inclusion on search result pages.
JSON-LD	Checks for JSON-LD markup, used by search engines to provide rich widgets from the content of a site.

4.2.2 GEO audits

Table 4.2: Custom-defined GEO metrics

Audit	Description
Similarity	Evaluates how well webpage content matches generated queries in the embedding space. The audit compares query embeddings with page chunk embeddings using cosine similarity and reports a mean similarity score, with query coverage as a supporting indicator.
Visibility	Evaluates whether a webpage is used and cited in LLM-generated answers in the presence of competing sources. The audit retrieves target and competitor content, generates citation-based answers, and computes a visibility score based on target citation coverage, target span share, and citation position.

4.3 LLM-based feedback

Lighthouse audit results can be difficult to interpret, and it's generally hard to pinpoint the relevant lines on a code level.

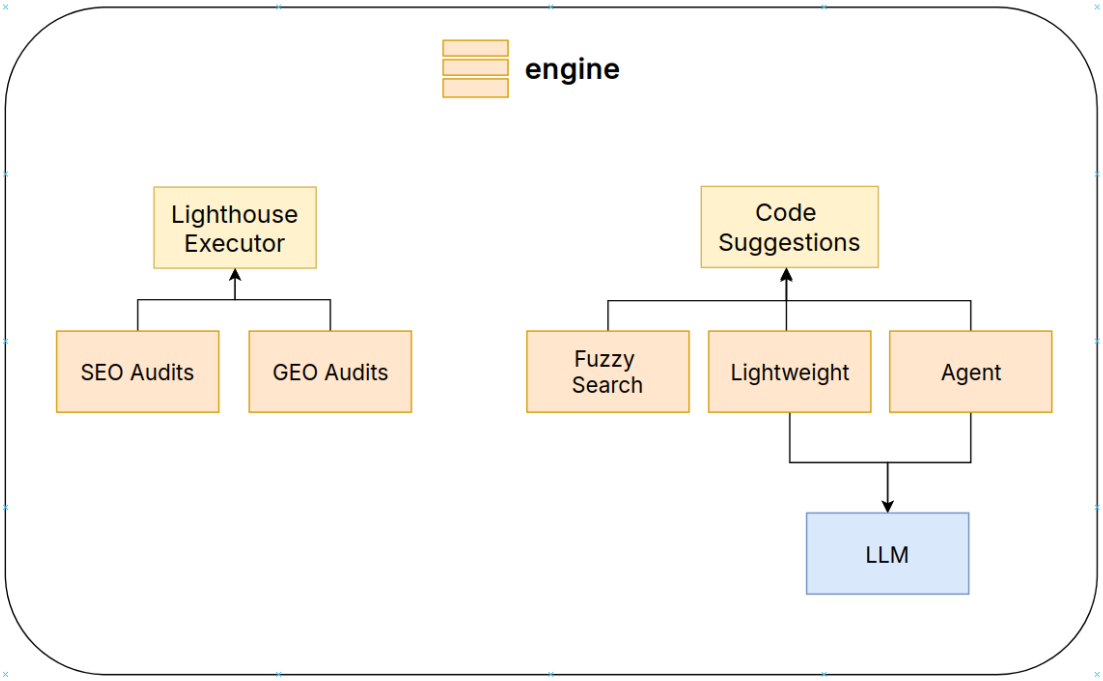

Figure 4.9: High-level architecture of Lighthouse++ engine

To solve this, the tool is divided into three different modes depicted under "Code suggestions" in Figure 4.9:

- Inline hints through fuzzy search on specific code snippets in the audit
- Lightweight (Inline errors with LLM-based code suggestions)
- Full agent, LLM can see all relevant files in the code base

These depend on the user's resources and preferences.

4.3.1 Inline hints

Lighthouse can mention specific code bits and then have a description of what's causing the problem, for example:

```
{
    "snippet": "<src="assets/image/10.jpg" alt="example text" width="500" height="700">"
    "description": "Reducing the download time of images
        can improve the perceived load time of the page and LCP."
    "displayValue": "Est savings of 921 KiB",
}
```

The implementation of this mode was to use a library called `ripgrep` to attempt to map this output to the corresponding source code. This has the advantage of being both efficient

and deterministic. With this, we get the code position of the mentioned problem and then package it into a node. This node is used when applied to form an information transfer called CodeSuggestion (appendix A).

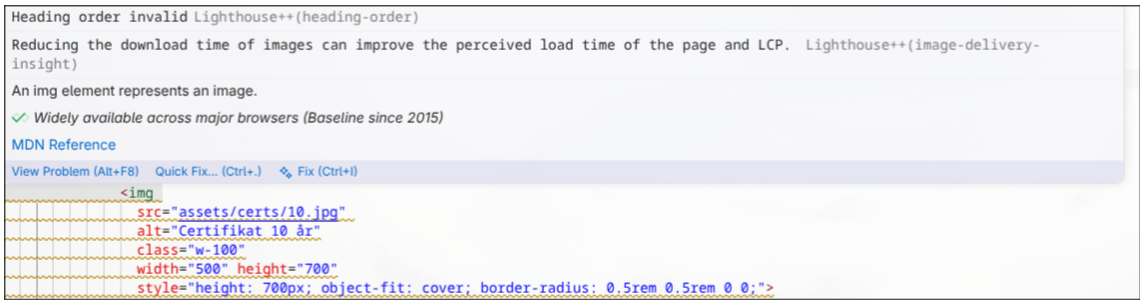

Figure 4.10: Yellow hint underlying the code, showing the problem from the audit

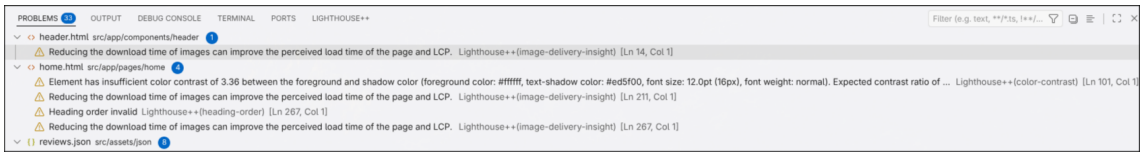

Figure 4.11: Problems Tab, showing the errors or hints

This CodeSuggestion interface gets populated with every bit of information the user needs to quickly handle the problem. With this, we create an inline hint in the user's code editor. This hint makes it easily findable in the "Problems" (See 4.11) tab, but also has a yellow underline on the code seen in figure 4.10. This addresses the problem of pinpointing audit results in large code bases.

This mode does not use LLMs and is completely based on algorithms and logic, allowing users without LLM access to still work efficiently.

4.3.2 Lightweight LLM-based suggestions

This mode builds on top of the inline hints and is capable of providing code fix suggestions. The service is provided by an LLM that takes the information provided by the CodeSuggestion interface and simply replies with the code the LLM thinks is suitable. Then it's stored in the same suggestion interface. That code that the LLM recommends is then easily applied by the user simply by pressing the quick fix seen in Figure 4.10, and an option called apply fix will pop up. This will then replace the user's existing code with the suggested fix.

This implementation allows the user to decide what LLM to use, seen in 4.6. The tool also supports locally hosted LLM for this mode, because we relay only the relevant information

along with the current code that's causing the problem, and then only minimal computation is needed for the LLM to complete. The system prompt used for the lightweight mode is visible in appendix C.

4.3.3 Agent mode

The shortcomings of the previous modes are that the LLM doesn't see the full picture and cannot work iteratively. That's where agent mode comes into play. This mode is made for LLMs with tool-calling capabilities, giving the LLM the ability to access certain files or bits of information in them. This also lets the LLM solve problems in the audit that do not contain code-level mappings and instead point to problems at a greater level.

The mode uses an agentic approach similar to the one discussed in the MKRL paper [40]. This approach is what powers multiple AI-assisted coding tools such as Cursor, GitHub Copilot, and Claude Code.

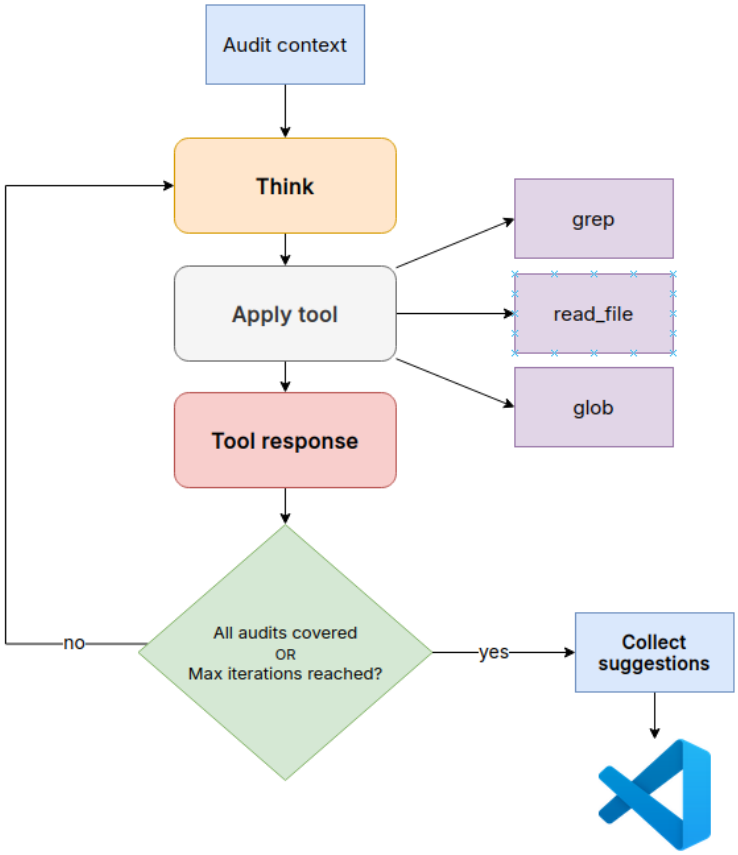

Figure 4.12: Agentic loop in Lighhouse++

With this approach, the LLM has access to a specific set of tools that help it analyze the entire codebase on the specific parts mentioned in the audit. A system prompt [D] is used to instruct the LLM on the workflow and establish the rules it must follow.

It will run multiple iterations (see figure 4.12) until it has gained enough context to be able to provide meaningful code suggestions.

The current set of tools includes:

- *glob* - Search for files and directory names matching a pattern
- *grep* - Search for source code lines matching a pattern
- *read_file* - Read a source code file
- *edit* - Propose code changes through text replacement

Generally, the agent will first use the `glob` and `grep` tools to find the files that the Lighthouse audit mentions. After this, it uses `read_file` on relevant files to gain a better understanding of the codebase.

A large benefit of using LLMs with reasoning capabilities is that it enables support for modern frameworks such as React, where the output HTML often differs substantially from the source code [41]. Traditional algorithms would typically require one implementation for each framework.

After the initial exploration of the codebase, the agent iteratively applies the `edit` tool to propose code changes. It is capped at a maximum of 50 iterations to maintain reasonable token usage. Edits are stored in memory and never applied directly to the files, allowing the user to review them before approval.

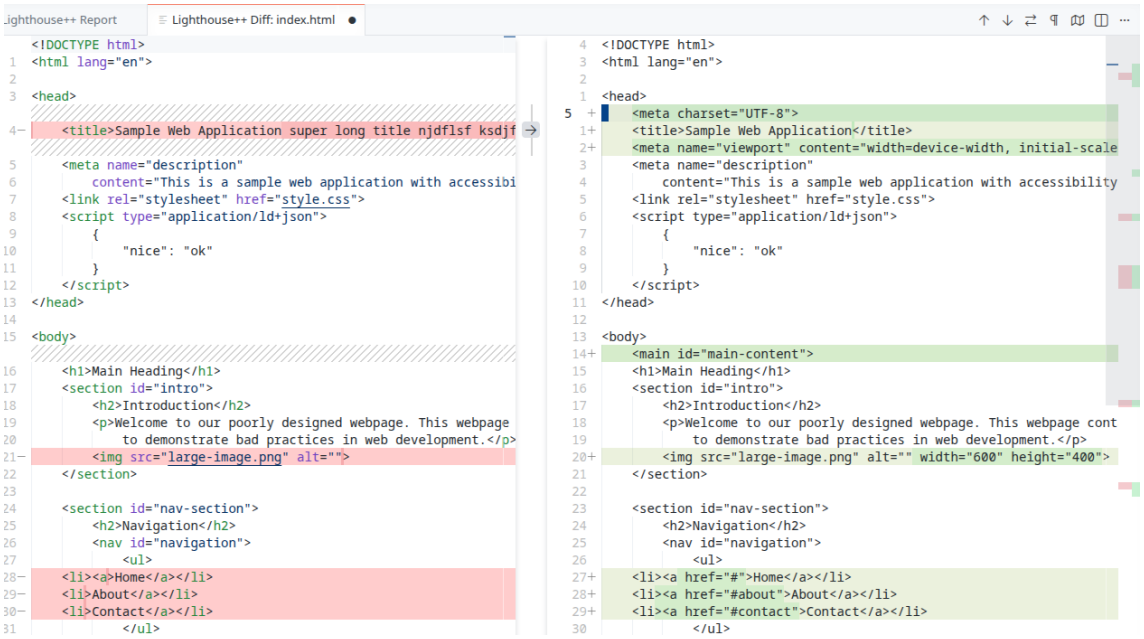

Figure 4.13: Recommended changes displayed in the Diff Viewer

The tool relies on VSCode's built-in diff viewer [42], which allows the developer to see exactly what the file will look like after applying the suggestions. It is also possible to revert individual changes as needed.

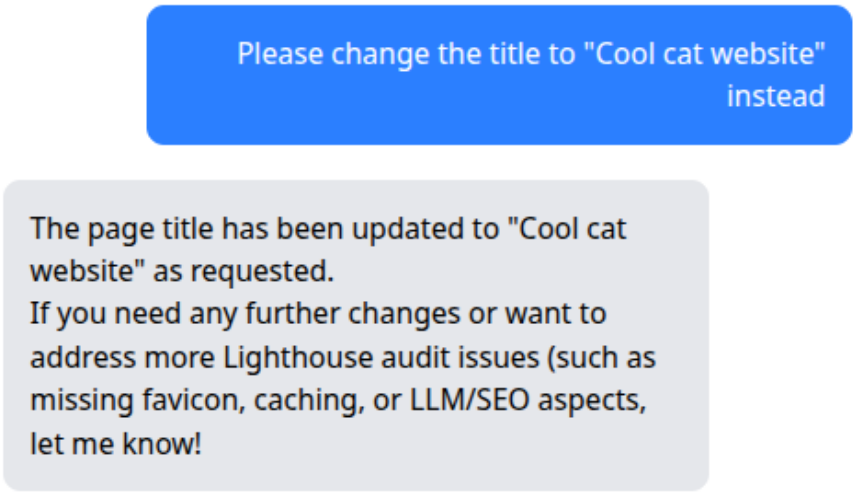

Figure 4.14: Interaction between agent and user in the "Chat" view

Another important feature of AI-assisted tools is the ability to ask follow-up questions or request modifications to the agent's original proposal. This was implemented with the help of a dedicated "Chat" view. To ensure that the model does not forget previous discussions, the entire message history is passed in for each new message.

An improvement suggestion to our current solution would be to only retain the most relevant context, for example, by summarization [43]. This would help mitigate the extensive token usage that agentic mode incurs.

4.4 Security

Security is a significant issue for AI-driven systems [44]. Given the tool's direct access to a user's source code, data privacy and security are fundamental to its architecture. To protect sensitive information, the engine is strictly prohibited from accessing high-risk files. This restriction includes files like .env to keep API keys and credentials hidden from the LLM, while also filtering out project irrelevant file structures. Furthermore, file reading is confined to the current workspace directory, preventing the agent from accessing global paths.

Modern AI agents also commonly allow access to arbitrary command execution [45], further empowering the LLM. This functionality was intentionally left out of the project due to the significant security considerations that such features impose.

For users requiring maximum data sovereignty, the extension supports integration with locally hosted LLMs, keeping all source code within the user's private environment.

Finally, as the tool operates within an NPM environment and relies on external libraries, we maintain a proactive stance toward supply-chain security to defend against potential malicious library distributions [46].

4.5 Generative Engine Optimization

This Section describes the implementation of the proposed GEO metrics in Lighhouse++. The system was designed to evaluate how well a webpage is adapted for retrieval and use by large language models. The implementation consists of three main parts: content collection, similarity analysis, and visibility analysis. In addition, supporting modules were implemented for query generation, embeddings, retrieval, citation analysis, and optimization suggestion generation.

The implementation is written in TypeScript and follows Lighthouse's audit and gatherer architecture. Gatherers are used to extract and prepare data from the target webpage, while the audit processes this data and computes the final metric values.

4.5.1 System Overview

The evaluation begins by collecting the visible and semantically important content of the webpage. This content is then divided into smaller text chunks and transformed into a representation that can be processed by embedding modules. From the page content, the system also generates a target query and a set of candidate queries intended to reflect how a user might ask an LLM about the page topic.

The generated queries and page chunks are then used in two separate but connected analyses:

- The similarity analysis, which measures how well the content of the page matches the generated queries in embedding form.
- The visibility analysis, which estimates whether the page would be represented and cited in an LLM-generated answer compared to competing pages.

This design makes it possible to evaluate both retrievability and LLM visibility, which correspond to two different aspects of GEO. Figure 4.15 illustrates the overall implementation pipeline of the proposed GEO evaluation system.

4.5.2 Data Collection and Preprocessing

The webpage content is collected in the GeoGatherer. This component extracts the page title, meta description, headings, and main body text from the DOM. This implementation focuses on meaningful content rather than raw HTML structure by isolating the main article-like content of the page.

The extracted text is normalized and divided into sentence-based chunks. Chunking is necessary because embedding and retrieval are performed at the chunk level rather than on the full page. This allows the system to compare generated queries against smaller semantic units and identify which parts of the page are most relevant.

At the same time, the system generates one primary query and several candidate queries from the extracted page text. These queries are produced by an LLM using a structured prompt (see appendix B). The purpose of the prompt is to simulate realistic information-seeking prompts that a user might submit when asking about the page content.

The gathered information is stored in a shared memory structure so that later audits can reuse the same page representation without recomputing it.

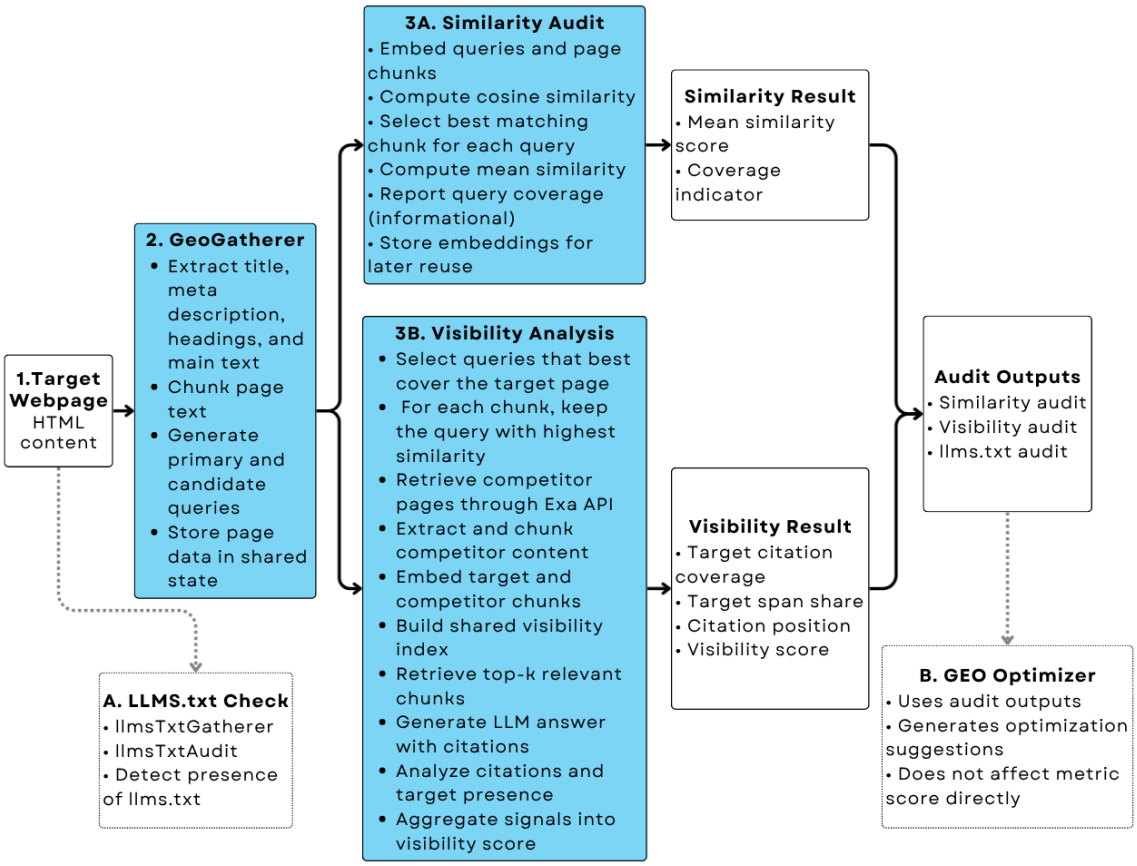

Figure 4.15: Implementation pipeline of the proposed GEO evaluation system. The figure shows the main flow from webpage extraction and query generation to similarity and visibility analysis, together with the additional `llms.txt` check and the post-analysis GEO optimizer.

4.5.3 Semantic Similarity

The similarity component estimates how well the target page content aligns with likely user queries. Its purpose is to measure whether the content can be semantically retrieved when an embedding-based system searches for relevant information.

After query generation, the system computes vector embeddings for both the generated queries and the page chunks. These embeddings are produced through an embedding API and represent the semantic meaning of the text in a high-dimensional vector form.

Each query embedding is compared against all chunk embeddings using cosine similarity. For each query, the chunk with the highest similarity score is selected as the best match. This produces one best match result per query.

The main output of the similarity audit is based on the cosine similarity values between queries and their best matching chunks. The implementation computes:

- The best matching chunk for each query.
- The cosine similarity for that match.
- The mean similarity across all query-chunk pairs.
- The fraction of queries whose best match exceeds a defined threshold.

The final similarity score is calculated as the mean of the highest chunk similarity score for each generated query:

$$\text{Similarity}(T, Q) = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \max_{c \in C} \text{sim}(q_i, c)$$

Here, Q is the set of generated queries, C is the set of page chunks, and $\text{sim}(q_i, c)$ is the cosine similarity between query q_i and chunk c.

In addition to the final similarity score, the implementation also computes query coverage as a complementary indicator. Query coverage shows how many queries achieve a similarity above a defined threshold. A page whose content contains chunks that are semantically close to the generated queries will obtain a higher mean similarity value, while query coverage is reported separately to indicate how consistently this holds across the full set of queries.

In addition to producing its own result, the similarity audit stores the generated queries and chunk embeddings for use in the visibility stage. This means that the component functions both as a standalone metric and as an input to the next part of the evaluation.

4.5.4 Content Visibility

While similarity measures whether the page can be retrieved, visibility measures whether it is actually used and cited in an LLM-generated answer when competing sources are present. This part of the implementation simulates a retrieval-augmented answering scenario.

The visibility analysis begins by selecting a subset of generated queries that best cover the content of the target page. Rather than using all candidate queries or selecting queries solely based on the highest similarity values, the implementation compares query embeddings with the embeddings of the target page chunks and, for each chunk, identifies the query with the highest similarity. The selected queries are therefore those that best match different parts of the page. This reduces redundancy, since multiple high-similarity queries may correspond to the same chunk rather than extending the coverage of the page content.

These selected queries are then used to retrieve competitor pages through the Exa API. For each retrieved result, the main text content is extracted and chunked in the same way as for the target page. All target chunks and competitor chunks are then embedded in the same vector form and stored in a common visibility index.

This creates a retrieval environment where the target page competes with alternative sources for relevance.

For each selected query, the system retrieves the highest-ranked chunks from the visibility index. These retrieved chunks may come from either the target page or competitor pages. The top chunks are then formatted into a document context and passed to an LLM together with the query.

The LLM is prompted (appendix E) to answer using only the provided documents and to include citations in the form of source URLs. This makes it possible to examine not only the generated answer, but also which sources the model relied on.

Once the answer has been generated, the system performs citation analysis. This step parses the output text, identifies cited spans, normalizes the cited URLs, and checks whether the target page URL appears among them.

The implementation derives several signals from this analysis:

- Whether the target page was cited at all.
- How many cited spans included the target page?
- The proportion of cited spans supported by the target page.

- The average position of target-supported citations in the response.

The final visibility score is calculated as a weighted combination of target span share, target citation coverage, and positional score:

$$\text{Visibility}(Q') = \alpha \cdot \text{SpanShare} + \beta \cdot \text{Coverage} + \gamma \cdot \text{PositionalScore}$$

Here, Q' is the selected subset of queries used in the visibility analysis. SpanShare represents the average proportion of cited spans supported by the target page, Coverage represents the fraction of selected queries where the target page was cited at least once, and PositionalScore represents how early target-supported citations appear in the generated responses. The weights α, β, and γ control the relative importance of these three components in the final score.

4.5.5 Supporting Components

In addition to the main similarity and visibility modules, several supporting components were implemented.

The llmsTxtGatherer and llmsTxtAudit are used to check whether the website provides an llm.txt file. This serves as an additional signal indicating whether the site offers machine-readable guidance for language models.

The citationAnalysis utility is responsible for parsing and evaluating URL citations in generated answers. This module provides the low-level support needed by the visibility metric.

The textUtils module handles normalization, document formatting, and chunking, while ragRetrieval performs top-k retrieval over embedded chunks. The configManager and related type definitions are used to manage model and API settings.

Finally, the geoOptimizer module uses the outputs of the similarity and visibility audits to generate GEO optimization suggestions. It does not contribute directly to the metric score, but instead interprets the audit results and translates them into practical recommendations for improving the page.

4.6 Testing

The project implements a JUnit testing suite, divided into two distinct directories: the Engine and the Extension. Testing within the Extension focuses on its integration with

the VS Code API, specifically validating the user interface, storage, and user experience, as well as the adapter that facilitates communication with the engine. The Engine tests prioritize the Lighthouse instance, external LLM connectivity, and the accuracy of the core code-suggestion logic.

5

Lighthouse++ Evaluation

This Chapter presents the results from the three-pass evaluation of Lighthouse++. The benchmark and analysis procedure are described in Section 3.6. The evaluation is divided into two main parts. The first part covers the core Lighthouse category metrics, analyzed at both the page and repository levels, as well as pass-by-pass. The second part presents an evaluation of GEO metrics.

For the primary SEO evaluation, the final paired comparison after the third optimization pass contains 80 pages from 18 repositories, since only targets with complete before-and-after measurements were included. Across these targets, Lighthouse++ produced 754 concrete suggestions.

5.1 Page-Level Comparison

The final page-level comparison contains 80 pages with usable baseline and post-pass-3 measurements. Table 5.1 summarizes the category-wise changes.

Table 5.1: Page-level before-and-after comparison after three optimization passes.

Category	n	Before mean	After mean	Mean delta
Performance	80	75.28	77.89	+2.61
Accessibility	80	91.40	95.09	+3.69
Best Practices	80	96.88	97.66	+0.79
SEO	80	93.16	95.84	+2.68

At the page level, metric scores in all four categories: performance, accessibility, best practices, and SEO improved on average after three optimization passes. The largest mean gain was observed for Accessibility, which increased from 91.40 to 95.09 (+3.69). SEO increased from 93.16 to 95.84 (+2.68), Performance increased from 75.28 to 77.89 (+2.61), and Best Practices increased from 96.88 to 97.66 (+0.79).

Most pages remained unchanged after intervention, but positive changes were more common than negative changes in every category. For Performance, 26 pages increased, 38 remained unchanged, and 16 decreased. For Accessibility, 23 increased, 53 were unchanged, and 4 decreased. For Best Practices, 15 increased, 62 were unchanged, and 3 decreased. For SEO, 18 pages increased, 62 were unchanged, and none decreased.

The sign-test p-values for the page-level comparison were 0.1641 for Performance, < 0.001 for Accessibility, 0.0075 for Best Practices, and < 0.001 for SEO. Under a conventional threshold of 0.05, Accessibility, Best Practices, and SEO showed statistically significant positive changes in the final paired page-level comparison, while Performance did not.

5.2 Repository-Level Comparison

To reduce the influence of repositories containing many similar pages, the same data was also aggregated at the repository level. The repository-level comparison contains 18 repositories with complete baseline and post-pass-3 coverage. Table 5.2 summarizes these results.

Table 5.2: Repository-level before-and-after comparison after three optimization passes.

Category	n	Before mean	After mean	Mean delta
Performance	18	74.02	76.34	+2.32
Accessibility	18	91.36	94.63	+3.28
Best Practices	18	97.02	97.72	+0.70
SEO	18	93.23	95.61	+2.38

The repository-level view preserves the same overall direction as the page-level comparison. Accessibility showed the largest mean increase, from 91.36 to 94.63 (+3.28). SEO increased from 93.23 to 95.61 (+2.38), Performance increased from 74.02 to 76.34 (+2.32), and Best Practices increased from 97.02 to 97.72 (+0.70).

At the repository level, 8 repositories improved in Performance, 5 improved in Accessibility, 5 improved in Best Practices, and 5 improved in SEO. For Performance, 4 repositories were unchanged, and 6 decreased. For Accessibility, 11 were unchanged, and 2 decreased. For Best Practices, 12 were unchanged and 1 decreased. For SEO, 13 were unchanged, and none decreased.

The sign-test p-values were 0.7905 for Performance, 0.4531 for Accessibility, 0.2188 for Best Practices, and 0.0625 for SEO. None of the final repository-level category changes reached statistical significance under a 0.05 threshold, although SEO was close to that boundary.

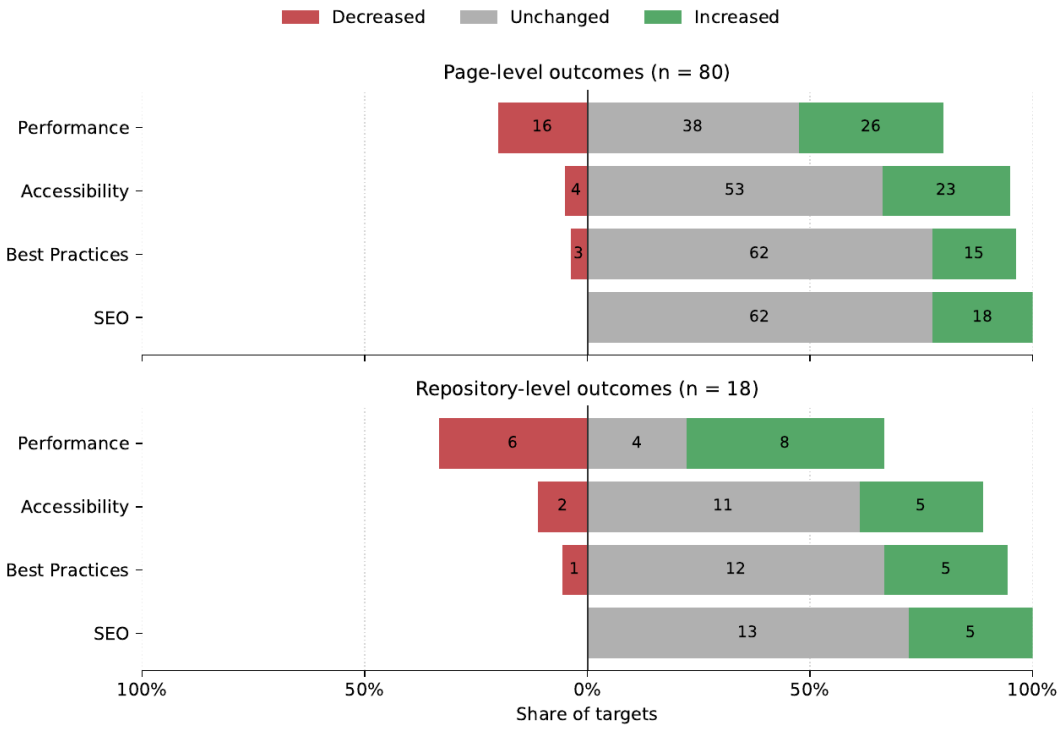

Figure 5.1: Direction of score changes after three optimization passes at the page and repository level. Bars show the share of targets that decreased, remained unchanged, or increased in each category, while the numbers inside the bars show raw counts.

Figure 5.1 summarizes the direction of change at both page and repository levels. This visualization is more informative than a box plot for the present data, because most targets remained unchanged, and several category distributions collapsed around zero. The figure makes clear that positive changes were more common than negative changes in all four categories, while unchanged targets still formed the largest group in most cases.

5.3 Pass-by-Pass Progression

To show how the benchmark evolved across the run, Table 5.3 reports the stage-specific page-level means at baseline and after each optimization pass.

Table 5.3: Page-level progression across the three-pass evaluation.

State	n	Performance	Accessibility	Best Practices	SEO
Baseline	80	75.28	91.40	96.88	93.16
After pass 1	80	78.13	93.88	96.78	95.18
After pass 2	80	78.43	94.60	97.23	95.61
After pass 3	80	77.89	95.09	97.66	95.84

The first optimization pass produced the largest immediate gains in Performance, Accessibility, and SEO. The second pass increased all four category means further. The third pass continued to increase Accessibility, Best Practices, and SEO to their highest observed means, while Performance decreased slightly from its after-pass-2 peak but remained clearly above baseline.

At the repository level, the same pattern was observed. The stage-specific repository means moved from 74.02, 91.36, 97.02, and 93.23 at baseline to 76.58, 93.56, 96.93, and 95.02 after pass 1, 76.86, 94.20, 97.33, and 95.41 after pass 2, and 76.34, 94.63, 97.72, and 95.61 after pass 3.

5.4 GEO Analysis

In addition to the Lighthouse category analysis, a separate comparison was carried out for the proposed GEO metrics. This analysis was based on 123 pages for which both baseline and optimized GEO outputs were available. Since GEO was retained as contextual output in the tool chain and the automatic fixes were not specifically designed to target GEO-related criteria, these results are reported as a complementary evaluation rather than as part of the main paired Lighthouse comparison. Table 5.4 summarizes the average GEO values before and after optimization, together with the mean deltas for each reported measure.

Table 5.4: Average GEO metrics for baseline and optimized pages (n = 123)

Metric	Baseline mean	Optimized mean	Mean delta
Similarity	0.5947	0.6064	+0.0117
Coverage	0.5136	0.5701	+0.0565
Visibility	0.5062	0.5610	+0.0548
Positional score	0.5347	0.5669	+0.0322
Target span share	0.3598	0.4023	+0.0425
Query coverage	0.5827	0.6538	+0.0711

All reported GEO-related metrics increased on average after optimization. The largest mean increase was observed for query coverage, which increased from 0.5827 to 0.6538 (+0.0711). This was followed by coverage, which increased from 0.5136 to 0.5701 (+0.0565), and visibility, which increased from 0.5062 to 0.5610 (+0.0548). Smaller but still positive changes were observed for target span share (+0.0425), positional score (+0.0322), and similarity (+0.0117).

The page-level improvement rates also differed between the two main GEO metrics. Similarity improved on 109 of 123 pages (88.6%), whereas visibility improved on 64 of 123 pages (52.0%). This suggests that the optimization process more consistently improved

semantic alignment between page content and generated queries, while improvements in answer-level visibility were more uneven across pages, although still positive on average.

5.5 Summary of Findings

Taken together, the three-pass experiment shows positive mean movement in all four optimization categories at both page and repository levels. In the final page-level paired comparison, Accessibility, Best Practices, and SEO improved significantly, while Performance improved on average but remained more variable and did not reach statistical significance. At the repository level, all four categories still improved on average, but none of the category-level sign tests reached significance.

The experiment also exposed a robustness limitation. Only 80 of the 100 planned targets yielded complete before-and-after measurements and could, therefore, be included in the final paired comparison. The reported improvements should therefore be interpreted as results on the included subset of the benchmark rather than on the full planned target set. The 20 pages failed this test due to our limits on engine to API communication, which actually demonstrates an intentional design feature.

In the separate GEO analysis, all reported GEO-related metrics increased on average after optimization. The largest increases were observed for query coverage, coverage, and visibility, while similarity also improved across a large majority of pages. These results suggest that the optimized pages tended to perform better according to the proposed GEO metrics, even though the automatic fixes were not explicitly designed to optimize GEO-specific criteria.

6

Discussion

To analyze the impact of the Lighthouse++ workflow, this chapter evaluates its performance against the benchmark dataset, addresses the ethical aspects of automated optimization, and proposes future contributions.

6.1 Evaluation

The three-pass evaluation demonstrates that Lighthouse++ can produce measurable improvements across all four optimization categories on the included subset of the benchmark. At the page level, Accessibility showed the largest mean increase, while SEO, Performance, and Best Practices also improved. The repository-level aggregation follows the same overall direction, which indicates that the result is not caused only by a few repositories with many similar pages.

These results are also notable because the evaluation did not include manual review or manual refinement between passes. The benchmark simply applied the generated fixes automatically and then measured the resulting Lighthouse scores. In that sense, the observed improvements on the included subset are encouraging, even though real-world use should still involve human review before accepting code changes. The reason why, in the testing, we accepted all suggested changes and did not nitpick or filter the most optimal changes was to gain an honest understanding of how the agent mode performed by itself without human intervention, and to gain a non-biased result.

These findings suggest that the extension can improve Lighthouse-related outcomes when it is allowed to refine a page iteratively. At the same time, the large number of unchanged pages shows that improvements were not uniform across the benchmark. For example, 38 of the 80 paired pages were unchanged in Performance, 53 were unchanged in Accessibility, 62 were unchanged in Best Practices, and 62 were unchanged in SEO. In other words, the system produced meaningful gains on part of the included pages, but many pages were already stable enough that additional edits did not change the final category score.

One likely explanation is that many benchmark pages already started from relatively high Lighthouse values, especially for Best Practices and SEO. When initial scores are already close to the upper end of the scale, there is less room for improvement, and small implementation changes can easily have no visible effect on the final category score.

The pass-by-pass progression provides additional insight into how the gains emerged. The first pass produced the largest immediate jump in Performance, Accessibility, and SEO. Later passes still contributed, especially for Accessibility, Best Practices, and SEO, but they also showed that iterative optimization has diminishing returns and does not guarantee monotonic improvement in every category. Performance, for example, peaked after the second pass and then decreased slightly after the third, while remaining above baseline.

As a whole, the results indicate that the thesis goal was achieved. We succeeded in building and evaluating a working optimization workflow that integrates Lighthouse, custom analysis, and LLM-based suggestions into a developer-oriented tool. We also demonstrated that the workflow can improve Lighthouse-related outcomes on the included subset, particularly for Accessibility, Best Practices, and SEO at the page level. At the same time, the evaluation does not support a claim of fully robust improvement across the entire planned benchmark, because not all planned targets yielded complete before-and-after measurements.

Another limitation is that the reported benchmark depends partly on the selected LLM configuration. The study used `openai/gpt-5.4-mini` via OpenRouter, and different models could produce different suggestions and therefore different Lighthouse outcomes. Context window size is especially important for larger repositories. Models with higher context limits may be necessary to analyze bigger projects without triggering context-length failures, although this would likely increase API cost.

6.2 User experience

One of the core objectives of this project was to develop a tool that would improve the developer's workflow and user experience. Unfortunately, we did not have the time to perform user studies or empirically measure satisfaction. All testing was performed by us, who have first-hand knowledge of the system. In hindsight, user tests should have been a more prominent part of our approach from the beginning.

As a result, conducting user interviews and surveys would be the prioritized next step for the project. Gathering direct feedback from users would help identify pain points, evaluate the effectiveness of current features, and inform future improvements. This would enable the product to be refined based on real user needs and better position it for adoption by a

broader audience.

6.3 Societal and ethical aspects

There are various societal and ethical aspects we considered during the development of this tool. In general, our solution benefits society by giving website owners more tangible methods of improving their search ranking and thus visibility on the internet. This can promote small businesses that would previously have had to compete against larger ones with far greater economic leverage [47].

The research performed on GEO in particular is intended to increase the transparency of how ranking in these systems works, which is generally considered black-box information.

Since search ranking is heavily influenced by content quality [48], improving it will also increase the overall quality of websites on the internet. This benefits users in that they will find the information they seek more easily. In addition, optimizing page load speeds will also make websites more inclusive to clients with lower-end hardware or limited network capabilities.

However, any tool is only as good as its users. Our tool or the research results derived could hypothetically be used to increase the ranking of websites with malicious intent. While this is inevitable, we expect the positive use cases to outweigh the negatives.

Since several components of our system rely on Artificial Intelligence (AI), all societal and ethical aspects involved with it must also be considered. Common concerns include data privacy [49] and extensive resource usage [50]. The costly nature of running AI also favors users with more economic capacity. Since our tool leverages a provider-agnostic LLM interface, users can choose to run local models, giving far more control over data usage and power consumption [37].

6.4 Future contributions

Given that this project was conducted during a short period of time, it should be regarded as a prototype and not a finished product. There is certainly more polishing needed to be done, which is only possible for a published product where feedback can be collected from real users.

With additional time, we would have deployed a production website and empirically evaluated the impact of our tool on Google search rankings. This would require additional ethical considerations to prevent polluting search results with purposeless websites.

Implementing a CI/CD workflow will allow us to seamlessly deploy new features and enhancements while mitigating the risk of regression bugs.

Another interesting topic would be to compare the fuzzy-based inline suggestions with the full agentic mode, judging whether the increased analysis time and cost are warranted in an industry context. Further evaluation of the agent could be done against different web frameworks to better understand its limitations. It's however worth noting that this is highly dependent on the chosen LLM.

Additional work can also be performed in the GEO component of the project. The proposed similarity and visibility metrics provide an initial framework for evaluating how well webpage content is adapted to LLM-based retrieval and answer generation, but they require further validation. A natural next step would be to test the metrics on a larger and more diverse set of websites and examine how well the resulting scores reflect how webpages are actually retrieved, used, and cited in LLM-generated answers.

Evaluation should be done across multiple LLM providers and retrieval settings. Since GEO is closely tied to the behavior of external models and search systems, the usefulness of the metrics depends on how well they generalize across different environments. Comparing results across several systems would help determine whether the framework captures general GEO properties or is overly dependent on a specific setup.

From a tooling perspective, future work could also extend the current Lighthouse-based prototype into a more complete GEO evaluation framework. This could include better support for repeated measurements, clearer explanations of audit results, and improved optimization suggestions linked to specific content changes. Such developments would make the GEO component more useful not only as a research prototype but also as a practical tool for industry use.

7

Conclusion

To conclude, this project has successfully delivered an integrated development environment solution for lighthouse audits in VS Code, with AI analysis and code suggestions based on the audit results, using agent-based iterative optimization. With this integration into VS Code, the friction between analysis and implementation is reduced, giving developers a more seamless workflow. The project demonstrates that GEO can be approximated through retrieval-oriented metrics such as semantic similarity and visibility, despite the lack of standardized evaluation methods in the field. This contribution helps reduce uncertainty surrounding GEO for developers in this new and evolving field. However, the provided GEO framework should be viewed as exploratory rather than predictive, since modern AI retrieval systems are highly dynamic and unpredictable.

The project includes an evaluation of the tool, showing improved metrics across 80 pages from 18 different websites. The evaluation shows an increase in all metrics. The result also shows that an AI-assisted SEO and GEO optimization is possible.

This project is important because, as an AI-driven search system continues to evolve, and for the developers competing for online visibility, they will need a tool combining traditional SEO and GEO with AI-oriented optimization in the web development toolbox.

Overall, Lighthouse++ demonstrates how automated SEO and AI-assisted optimization, and exploratory GEO evaluation can be combined into a unified developer-centered workflow directly inside VS Code. While there is still a lot to explore in the GEO space as well as the projects' GEO metrics, it has shown promising results and highlighted the growing importance of AI-aware optimization in modern web development practices.

Bibliography

[1] B. J. Jansen and M. Resnick, "An examination of searcher's perceptions of nonsponsored and sponsored links during ecommerce web searching," *J. Am. Soc. Inf. Sci. Technol.*, vol. 57, no. 14, pp. 1949–1961, dec 2006.

[2] Z. Gyöngyi and H. Garcia-Molina, "Web spam taxonomy," *Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2005.

[3] Google Developers, "Web vitals," https://web.dev/articles/vitals, 2024, accessed: 2026-02-07.

[4] ——, "How google search works," 2025, accessed: Feb. 7, 2026. [Online]. Available: https://developers.google.com/search/docs/fundamentals/how-search-works

[5] ——. (2025) Introduction to lighthouse. Accessed: Feb. 9, 2026. [Online]. Available: https://developer.chrome.com/docs/lighthouse/overview

[6] T. M. Risto Ollila1, Niko Makitalo. (2022) Modern web frameworks: A comparison of rendering performance. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10243623

[7] Y. Deda. (2025, Aug. 27) Which platforms lead in ai traffic in 2025? chatgpt, perplexity, gemini more. Accessed: Feb. 10, 2026. [Online]. Available: https://seranking.com/blog/ai-traffic-research-study/?utm_source=chatgpt.com

[8] Google ai overviews. Accessed: Feb. 13, 2026. [Online]. Available: https://search.google/ways-to-search/ai-overviews/

[9] McKinsey Company. (2025) New front door to the internet: Winning in the age of ai search. Accessed: 2026-03-10. [Online]. Available: https://www.mckinsey.com/capabilities/growth-marketing-and-sales/our-insights/new-front-door-to-the-internet-winning-in-the-age-of-ai-search

[10] Q. Chen, J. Chen, H. Huang, Q. Shao, J. Chen, R. Hua, H. Xu, R. Wu,

R. Chuan, and J. Wu, “Cc-gseo-bench: A content-centric benchmark for measuring source influence in generative search engines,” 2025. [Online]. Available: https://arxiv.org/abs/2509.05607

[11] Semrush. (2026). [Online]. Available: https://www.semrush.com/

[12] ahrefs. (2026). [Online]. Available: https://ahrefs.com/

[13] seranking. (2026). [Online]. Available: https://seranking.com/

[14] Lighthouse. Accessed: Feb. 13, 2026. [Online]. Available: https://github.com/googlechrome/lighthouse

[15] Search engine optimization (seo) starter guide. Accessed: Feb. 23, 2026. [Online]. Available: https://developers.google.com/search/docs/fundamentals/seo-starter-guide

[16] Bing webmaster guidelines. Accessed: Feb. 23, 2026. [Online]. Available: https://www.bing.com/webmasters/help/webmaster-guidelines-30fba23a

[17] Website optimization. Accessed: Feb. 23, 2026. [Online]. Available: https://webmaster.yandex.com/site/optimization/seo-guide/?section=OPTIMIZATION_ADVICE&menu=ADVANCED

[18] (2026) search-engine-market-share. Accessed: 4. 26, 2026. [Online]. Available: https://gs.statcounter.com/search-engine-market-share

[19] N. V. Pattedar, C. Yoshitha, J. K. A., G. V. Prakashini, U. S., and V. K.V., “Search engine optimization (seo) auditing tool,” in *2024 8th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS)*, 2024, pp. 1–6.

[20] Seonaut. Accessed: May. 15, 2026. [Online]. Available: https://github.com/stjudewashere/seonaut

[21] Google Developers, “Crawl budget management for large sites,” https://developers.google.com/search/docs/crawling-indexing/large-site-managing-crawl-budget, 2025, accessed: 2026-02-07.

[22] ——, “Overview of crawling and indexing topics,” https://developers.google.com/search/docs/crawling-indexing, 2025, accessed: 2026-02-07.

[23] L. Page, “Method for node ranking in a linked database,” U.S. Patent U.S. Patent 7,058,628, Jun. 6, 2006.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[25] Lighthouse. Accessed: June. 1, 2026. [Online]. Available: https://developer.chrome.com/docs/lighthouse

[26] Lighthouse performance scoring. Accessed: June. 1, 2026. [Online]. Available: https://developer.chrome.com/docs/lighthouse/performance/performance-scoring

[27] Lighthouse accessibility score. Accessed: June. 1, 2026. [Online]. Available: https://developer.chrome.com/docs/lighthouse/accessibility/scoring

[28] Page lacks the html doctype, thus triggering quirks mode. Accessed: June. 1, 2026. [Online]. Available: https://developer.chrome.com/docs/lighthouse/best-practices/doctype

[29] Document does not have a meta description. Accessed: June. 1, 2026. [Online]. Available: https://developer.chrome.com/docs/lighthouse/seo/meta-description

[30] Information retrieval. Accessed: June. 2, 2026. [Online]. Available: https://en.wikipedia.org/wiki/Information_retrieval

[31] Vector embeddings. Accessed: June. 2, 2026. [Online]. Available: https://developers.openai.com/api/docs/guides/embeddings#use-cases

[32] Cosine similarity. Accessed: June. 2, 2026. [Online]. Available: https://en.wikipedia.org/wiki/Cosine_similarity

[33] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023. [Online]. Available: https://arxiv.org/abs/2312.10997

[34] (2021, Nov) Visual studio code. Accessed: Feb. 13, 2026. [Online]. Available: https://code.visualstudio.com/

[35] Stack Overflow. (2025) 2025 developer survey. [Online]. Available: https://survey.stackoverflow.co/2025/technology

[36] (2021, Nov) Extension api. Accessed: Feb. 13, 2026. [Online]. Available: https://code.visualstudio.com/api

[37] I. O'Byrne. (2024) The case for running ai/ml models locally. [Online]. Available: https://wiobyrne.com/running-models-locally/

[38] T. Joachims, L. Granka, B. Pan, and H. Hembrooke, "Accurately interpreting clickthrough data as implicit feedback," *ACM SIGIR Forum*, vol. 51, pp. 4–11, 08 2017.

[39] (2025, 10) E-e-a-t for geo: How to build trust signals that win ai citations. Accessed: 2026-05-14. [Online]. Available: https://agenxus.com/blog/eeat-for-geo-trust-framework-generative-engine-optimization

[40] E. Karpas, O. Abend, Y. Belinkov, B. Lenz, O. Lieber, N. Ratner, Y. Shoham, H. Bata, Y. Levine, K. Leyton-Brown, D. Muhlgay, N. Rozen, E. Schwartz, G. Shachaf, S. Shalev-Shwartz, A. Shashua, and M. Tenenholtz, "Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning," 2022. [Online]. Available: https://arxiv.org/abs/2205.00445

[41] S. Chen, U. Thaduri, and V. Ballamudi, "Front-end development in react: An overview," *Engineering International*, vol. 7, pp. 117–126, 12 2019.

[42] (2026) Staging and committing changes. Accessed: 2026-05-14. [Online]. Available: https://code.visualstudio.com/docs/sourcecontrol/staging-commits#_review-changes-with-the-diff-editor

[43] L. Dinesjö and E. Floreteng, "Efficacy of context summarization techniques on large language model chatbots : Balancing compression and recollection," p. 29, 2024.

[44] H. Elahi, N. Liu, J. Chen, and F. Zhang, "Toward a secure framework for regulating artificial intelligence systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 23, no. 1, pp. 1373–1389, 2026.

[45] Terminal | cursor docs. Accessed: May. 17, 2026. [Online]. Available: https://cursor.com/docs/agent/tools/terminal

[46] N. Zahan, T. Zimmermann, P. Godefroid, B. Murphy, C. Maddila, and L. Williams. (2022) What are weak links in the npm supply chain? [Online]. Available: https://doi.org/10.1145/3510457.3513044

[47] P. X. McCarthy, X. Gong, S. Eghbal, D. S. Falster, and M.-A. Rizoiu, "Evolution of diversity and dominance of companies in online activity," 2020. [Online]. Available: https://arxiv.org/abs/2003.07049

[48] C. Ziakis, M. Vlachopoulou, T. Kyrkoudis, and M. Karagkiozidou, "Important factors for improving google search rank," *Future Internet*, vol. 11, no. 2, 2019. [Online]. Available: https://www.mdpi.com/1999-5903/11/2/32

[49] M. Ghosh, "Artificial intelligence (ai) and ethical concerns: a review and research agenda," *Cogent Business amp; Management*, vol. 12, 2025. [Online]. Available:

https://doi.org/10.1080/23311975.2025.2551809

[50] A. de Vries, “The growing energy footprint of artificial intelligence,” *Joule*, vol. 7, pp. 2191–2194, 2023. [Online]. Available: https://doi.org/10.1016/j.joule.2023.09.004

A

CodeSuggestion

The following interface describes the properties for code suggestions, used by Inline (sec 4.3.1) and Lightweight (sec 4.3.2) mode.

```
interface CodeSuggestion {
    /** Unique identifier */
    id: string,
    /** Workspace-relative file path (e.g. "src/index.html") */
    file: string;
    /** 1-based line number where the issue is */
    line: number;
    /** 1-based line where the affected code block ends (inclusive) */
    endLine: number;
    title: string;
    /** Severity: "error" | "warning" | "info" */
    severity: 'error' | 'warning' | 'info';
    /** Which Lighthouse audit triggered this (e.g. "meta-description") */
    auditId: string;
    /** Short description of the problem */
    problem: string;
    /** Explanation of how to fix it */
    suggestion: string;
    /** The existing code that should be changed (optional) */
    existingCode?: string;
    /** The replacement code (optional) */
    fixedCode?: string;
}
```

B

Query Generation Prompt

The following prompt is used to generate primary and candidate queries used in GEO evaluation (sec 4.5).

Return ONLY valid JSON (no markdown, no extra text).

Goal: Generate search queries that a user might use to find this webpage in a search engine or LLM search system.

Rules:

- Create 1 primaryQuery representing the main topic of the page.
- Create 10 candidateQueries closely related to the page content.
- Queries must be 2–8 words.
- Queries should reflect information explicitly present in the page.
- Avoid speculative or niche queries not strongly supported by the content.
- Avoid repeating the same query with minor wording changes.

Webpage signals:
title: ${title}
metaDescription: ${description}
h1: ${h1}
h2: ${h2}

Main text snippet: ${snippet}

Return JSON EXACTLY like:

```
{"primaryQuery":"...","candidateQueries":["...","...","..."]}
```

C

Lightweight System Prompt

The following prompt is used by Lightweight mode (sec 4.3.2) to propose code suggestions for inline problems.

You are an expert web developer and SEO specialist. You will receive:

- A JSON array of code snippets and a suggestion to improve them.

Your job is to improve the code snippet to your best ability.

Example input:

```
[
    {
        "id": "e1c212a4-2a51-44dd-b22f-af6a5cd3d9be",
        "code": "<img src='cat.png'>",
        "suggestion": "Informative elements should aim for..."
    }
]
```

Each suggestion object must have the following fields:

```
{
  "id": "e1c212a4-2a51-44dd-b22f-af6a5cd3d9be",
  "fixedCode": "<img src='cat.png' alt='Picture of a cat'>",
}
```

The id should be mapped to the corresponding problem. Respond ONLY with a JSON array.

D

Agent System Prompt

The following system prompt is used in Agent mode (sec 4.3.3) to state the workflow and boundaries of the LLM.

You are an expert web developer and SEO specialist. You will receive:

1. A JSON summary of a Lighthouse audit with category scores and failed audits.
2. Access to tools to search and read files in the user's workspace.

Your job is to:

1. First explore the codebase using the available tools (grep, glob, read_file) to understand the structure
2. Find files that are relevant to the failed Lighthouse audits
3. Propose fixes to the code with the "edit" tool

Important:

- Use tools to explore the codebase BEFORE making fixes
- For each failed audit, grep/search for relevant patterns (e.g., for meta-description audit, search for "<head>" patterns)
- Read the actual files to see the current code before suggesting fixes
- Prioritize by impact: errors first, then warnings, then info

Iteratively apply the "edit" tool to propose fixes to the code. When you are finished with ALL edits, reply with a SHORT summary and any suggestions that are not code-related.

E

Visibility Retrieval Prompt

The following prompt is used by the Visibility metric (sec 4.5.4) to simulate a retrieval scenario.

You are an assistant answering the following question:

Question: ${query}

Rules:

- Use ONLY the following documents to answer.
- Cite the URL of any information you use.
- For each fact or statement you mention, include the URL of the document it came from in square brackets, e.g., [https://example.com].
- Do NOT use numeric references like [1], [2]. DO NOT create a reference list.
- If a fact appears in multiple documents, pick the one with the highest relevance.
- Do not invent URLs. If you cannot answer from the documents, respond: "I cannot answer."

Documents: ${documents}

Answer:

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden
www.chalmers.se

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY