METEHAN.AI RESEARCH REPORT · PREPRINT COMPANION GUIDE

LLM Optimization Analyzer Test Protocol for Screaming Frog Crawls

Metehan Yesilyurt

AI Search & Information Retrieval Researcher · metehan.ai

https://metehan.ai/pdf/llm-optimization-analyzer-test-protocol.pdf

Abstract. This protocol explains how I test an LLM optimization analyzer built as a Screaming Frog custom JavaScript snippet that calls the Gemini API. It defines instrumentation, baseline and treatment runs, metrics, decision rules, and rollback conditions. The goal is to separate directional content guidance from ranking claims.

Keywords: LLM optimization, Screaming Frog, Gemini API, measurement protocol, batched pointwise, content audit

Companion to the article Is It Possible to Build? Universal LLM Optimization Analyzer with Gemini API for Screaming Frog. The article covers the topic in full; this document is a separate, practical measurement methodology and test protocol.

1. What This Protocol Measures and What It Does Not

I use this protocol to evaluate an LLM optimization analyzer that runs inside Screaming Frog and calls Gemini 1.5 Flash [3][4]. The analyzer extracts headings, paragraphs, lists, and schema, then asks the model for target queries, a 0 to 5 LLMO score, passage-level analysis, content gaps, and optimization recommendations [3][5].

The protocol measures five things: extraction fidelity, schema recall, query coverage, score stability, and agreement with human review. It does not measure external link authority, entity co-occurrence, or true multi-document ranking. The source article notes these limits explicitly [3].

Decision rule: use the analyzer to prioritize content work, not to predict citation placement. If a stakeholder asks for a ranking forecast, run a separate citation experiment. The analyzer is a diagnostic instrument, and its output is only as good as the passages it can extract and the prompt it can build.

2. Instrumentation and Run Log

Before any test, I freeze the instrument. That means recording the Gemini model name, the prompt template, the extractor version, and all heuristic weights. The source script uses MIN_WORDS = 10, MAX_CHUNK_LENGTH = 500, title weight 2.0, and list weight 1.1 [3][5]. I log those values because changing them changes the measurement.

I also record the token budget. The source notes a constraint around 4096 tokens, so long pages can lose context [3]. I log truncation events because a truncated prompt can make a page look weaker than it is. I use a Gemini API key from Google AI Studio [4]. The script itself is available in the referenced repository [5].

I use a control URL set stratified by page type: product, FAQ, editorial, technical, and local business. For each URL, I predefine target queries before scoring. This prevents post hoc query selection. I run the crawl on rendered HTML so that the custom JavaScript sees the same DOM a user would see. I store raw API responses, parsed scores, and any error messages. I keep a run ID that links each score to a specific crawl, model, and prompt version.

- Run ID and timestamp
- Model name and version
- Prompt version
- Extractor version and weights
- Token budget and truncation flag
- URL and page type
- Target queries
- Raw response and parsed score
- Human reviewer initials and notes

3. Baseline and Treatment Test Design

The baseline is a single-pass, pointwise-style run: one page per prompt, one score per page. The treatment is a batched pointwise run: multiple high-signal passages from the same page are placed in one structured prompt. This mimics the batching idea from the Batched Self-Consistency paper [1].

I also test self-consistency by repeating the same prompt multiple times and averaging the scores. The research used 15 self-consistency calls for one-by-one pointwise and batched pointwise comparisons [1]. I start with 3 repeats for a quick stability check and move to 15 when I need a publishable comparison.

1. Crawl the control set with the baseline configuration.
2. Export the baseline scores and query coverage.
3. Run the batched pointwise configuration on the same URLs.
4. Run repeated prompts for a subset of at least 20 URLs.
5. Compare mean scores, standard deviations, and query coverage.
6. Have a human reviewer rank the same pages for relevance to the predefined queries.
7. Compute rank agreement between the analyzer and the human reviewer.
8. Adopt the treatment only if it improves agreement or stability without increasing truncation.

Decision rules: If batched pointwise raises query coverage by 10 percentage points or more and keeps score standard deviation under 0.5, I adopt it. If it increases truncation above 5%, I reduce passage count or chunk more carefully. If it does neither, I keep the baseline and treat batching as unproven for that site.

4. Metrics, Thresholds, and Decision Rules

I use the following metrics to decide whether the analyzer is fit for a given audit. Each metric has a threshold. If a threshold fails, I fix the instrument before I trust the content recommendations.

Metric	Operational definition	Acceptance threshold	Action if failed
Passage extraction rate	Percentage of meaningful page passages that meet MIN_WORDS = 10 and MAX_CHUNK_LENGTH = 500 [3]	>= 80%	Adjust extractors or lower MIN_WORDS for short pages
Schema detection recall	Percentage of expected schema types found among FAQ, Article, Product, HowTo, and LocalBusiness [3]	>= 90% on pages with known schema	Fix JSON-LD parser or page markup
Query coverage	Percentage of predefined target queries with an LLMO score of 3 or higher on the 0 to 5 scale [3]	>= 60%	Run content gap analysis and add missing passages
Score stability	Standard deviation of the overall LLMO score across 3 or more runs	<= 0.5 on the 0 to 5 scale	Add self-consistency runs [1]
Rank agreement	Spearman correlation between analyzer priority and human relevance ranking	>= 0.6	Recalibrate weights or prompt
Token truncation rate	Percentage of pages where the prompt exceeds the approximate 4096 token budget [3]	<= 5%	Reduce passage count or split prompts
API failure rate	Percentage of calls that fail or return unparsable JSON	<= 2%	Add retries and validate JSON schema
Position bias	Difference in mean score between the first and last passage in the prompt	< 0.5	Randomize passage order in test runs

I treat these thresholds as starting points, not universal constants. The source article warns that heuristic weights are based on SEO intuition, not model-specific feedback [3]. That means thresholds should be recalibrated per domain and per model version. If I change the model, I rerun the stability and rank agreement tests before I compare scores across time.

5. Worked Example: Product Page and Editorial Page

I run the protocol on two pages from the same site. Page A is a product page with schema and pricing. Page B is an editorial guide with no product schema.

Page A: 12 candidate passages. 8 meet the extraction criteria. 5 predefined target queries. Mean LLMO score across 3 runs is 3.4. Query coverage is 60%. The analyzer reports missing product comparisons and no pricing section [3]. Optimization priority is high.

Page B: 20 candidate passages. 18 meet the extraction criteria. 4 predefined target queries. Mean LLMO score is 2.8. Query coverage is 50%. The analyzer reports missing FAQ schema and weak passage-level summaries. Optimization priority is medium.

Page	Passages extracted	Target queries	Mean LLMO score	Query coverage	Content gaps	Priority	Decision
Product A	8 of 12	5	3.4	60%	Missing product comparisons, no pricing section	High	Add comparison table and pricing block, then retest
Editorial B	18 of 20	4	2.8	50%	Missing FAQ schema, weak passage summaries	Medium	Add FAQ schema and rewrite two passages, then retest

Decision rules for this example: If mean score is below 3 and coverage is below 60%, I prioritize the page. If mean score is 4 or higher and gaps are empty, I monitor. If mean score is between 3 and 4, I change one element at a time and rerun the protocol. I do not deploy multiple changes at once because I cannot attribute the effect.

6. Reliability Checks: Batching, Self-Consistency, and Position Bias

The source article describes pointwise ranking as evaluating one query-document pair at a time, while batched pointwise presents multiple candidates for the same query together [3]. The Batched Self-Consistency paper found that on a legal search dataset using GPT-4o, one-by-one pointwise improved from 44.9% to 46.8% NDCG@10 with 15 self-consistency calls, while batched pointwise improved from 43.8% to 51.3% NDCG@10, a gain of 7.5 percentage points [1]. I use those numbers as a benchmark for what a well-implemented batching test can look like, not as a guarantee for my own site.

The C-SEO Bench paper found that 61% of optimized pages had no citation ranking change, 26.2% improved, and 12.8% declined [2]. That distribution is my prior for any content optimization test. If my test shows no change for most pages, that is normal. If it shows large gains everywhere, I check for measurement error.

Position in the context window matters. The source article reports that documents appearing earlier in an LLM context window receive more visibility, with the largest gains at position 1 and negative effects at positions 8 to 10 for several domains [2][3]. I test this by randomizing passage order and checking whether the analyzer score changes.

Position	Retail	Games	Books	Web	News	Debate
1	+2.77	+1.89	+1.60	+0.87	+0.70	+1.54
2	+1.78	+1.28	+1.28	+0.19	+0.45	+0.41
3	+0.67	+0.57	+0.48	-0.22	-0.01	-0.37
8 to 10	-0.76	-0.58	-0.88	-1.74	-1.15	-2.14

Decision rule: If position bias in my analyzer is 0.5 or higher, I randomize passage order for every run and average across orders. If I cannot reduce bias, I treat the score as a directional signal only. The source article also notes that the Gemini API does not natively support multi-document comparative ranking, so the script mimics batching rather than performing true batched ranking [3]. That limitation belongs in every report.

7. Reporting Template and Rollback Rules

I use a fixed report template so that every audit is comparable. The template includes:

- Run ID, date, model name, prompt version, extractor version, and weights.
- URL, page type, and predefined target queries.
- Number of runs, mean LLMO score, standard deviation, minimum, and maximum.
- Query coverage, schema types found, content gaps, and optimization priority.
- Human review notes and confidence rating.
- Recommended action and expected effect.
- Rollback condition and owner.

Rollback rules are part of the protocol. If a controlled citation test shows a ranking drop after an optimization, I revert the change. If score standard deviation is above 1.0, I do not act on the score. If token truncation is above 10%, I fix extraction before I change content. If API failure rate is above 5%, I pause the audit. The source article warns that auditing and optimizing based only on this custom JavaScript may break current rankings and that the tool is experimental [3]. I treat that warning as a hard constraint.

Final decision rule: The analyzer is useful for finding content gaps, schema omissions, and passage-level weaknesses. It is not a ranking verdict. I use it to generate hypotheses, then test those hypotheses with controlled citation measurements. If the hypothesis survives the test, I keep the change. If it does not, I roll it back.

REFERENCES & SOURCES

[1] Batched Self-Consistency paper, https://arxiv.org/abs/2505.12570

[2] C-SEO Bench paper, https://arxiv.org/abs/2506.11097v1

[3] Source article: LLM optimization analyzer with Gemini API for Screaming Frog, https://metehan.ai/blog/llm-optimization-analyzer-screaming-frog/

[4] Google AI Studio Gemini API key, https://aistudio.google.com/apikey

[5] Screaming Frog LLMO script repository, https://github.com/metehan777/llmo-optimization-screaming-frog