AI Detection Report for The Problem Synaptic maintains a database of compa...

Aug 17, 2026 | 1,976 Words | Pangram 4.0

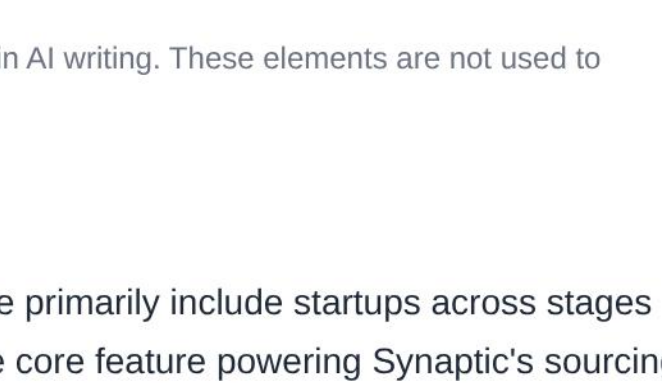
Summary

We believe that this entire text is human-written.

Authorship Breakdown

0 % of the document	AI Generated
0 % of the document	Moderately AI Assisted
0 % of the document	Lightly AI Assisted
100 % of the document	Human Written

Authorship Progression



Analyzed Text

Words that are underlined indicate AI phrases, while the numbers represent how frequently they are likely to appear in AI writing. These elements are not used to determine the results but are displayed separately as patterns commonly found in AI-generated text.

Human Written | 1524 Words | High Confidence

The Problem Synaptic maintains a database of companies that VCs might be interested in. These primarily include startups across stages operating in high-growth spaces like Consumer Tech, Enterprise Tech, AI, Crypto, and so on. The core feature powering Synaptic's sourcing tool is the Company Screener. The screener lets users apply filters on company attributes (founding year, headcount, industry, HQ location, growth rate, alternative metrics, etc.) to get a resultant list of companies. While most company attributes took on values from a predefined set (e.g. HQ Location) or were numeric (e.g. Headcount), a lot of the times users wanted to filter for a sector, business model, or an obscure technology. One way the screener achieved this was through a keyword search filter on company descriptions. An example search could be: "B2B" AND "payment gateway solution" — and Razorpay would be one of the results. This keyword search was powered through a hybrid combination of BM25 (lexical search) and embedding-based vector search. The document set was company descriptions sourced from 3rd party data sources, primarily Crunchbase, LinkedIn, Pitchbook, and Glassdoor. This search suffered from several critical issues: Lossy intent translation. The user had to translate their query intent into a bunch of keywords, and the tool had to decipher user intent back from the set of keywords in order to give back a relevant set of results. This process was extremely lossy. It was very often the case that a small set of keywords returned too many results, and a larger set of keywords returned irrelevant or partially relevant intersections improperly ranked. Non-exhaustive, error-prone

Pangram.

Human Written | 1524 Words | High Confidence

documents. Company descriptions were generally not comprehensive enough, along with containing outdated and irrelevant information, so many relevant companies did not show up in search results if not given enough keywords. Complicated weighting in BM25/Vector. Calibrating the weights of lexical and semantic search was also an art that needed a lot of care and caution by the DS team; and even then it frequently failed for long-tail keywords. Vector search compressed too aggressively. Representing a document as a single vector flattened multiple concepts into one point in space; and in cases where a company offered multiple products/services, the final document vector might not achieve a high cosine similarity with a relevant query vector. Overall, the search performed predictably well for simple and broad queries, but for niche queries (in terms of industry/sector/tech) and/or multi-product companies, it failed quite often. Key Principles I came into this project with learnings from a previous one where we followed an over-ambitious all-or-nothing approach, and we'd over-scoped and failed big too late. This time instead, I held steadfastly to three operating principles. The first one was to build fast, demo fast, and fail fast. Whatever approach we followed had to necessarily produce something visible and measurable inside a short cycle. No building in the dark for multiple months only for the foundations to fail at the end of it. Second, the evaluation criteria had to exist before any modeling did. I locked in a baseline dataset of 17 queries. For each of these, the Top 1000 results using the existing search system were saved, before any of us touched a new model. Trained analysts hand-labelled the top 500 results per query. This was the dataset against which any future system would be compared. Third, using primary sources of information as the truth set. An ongoing project (described in a separate case study) was the generation of company factsheets using information summarized straight from the company websites using an agentic system. I realized this would be a more comprehensive, more truthful document set to power our search model compared to the existing set composed of 3rd party information. Our targets were: demo and internal release within the first 3 months; refining, benchmarking, and final customer-facing release by another 3 months. What Didn't Work Before the system that worked, two approaches taught us what the real problem was. Reweighting the existing hybrid. We tried everything from pure BM25, pure vector search, and several weighted combinations in between. This didn't significantly affect accuracy. This encouraged us to try to look for a different frame. A Knowledge-Graph approach. We planned to build a graph of company tags using terms extracted from factsheets and Wikipedia/dbpedia. Companies would then be indexed using these tags. During retrieval time, an LLM would expand the user query with related tags using the graph, and relevant companies would then be retrieved against expanded tags. This one was again too over-ambitious an approach with no intermediate reward till the entire KG was complete. We spent ~3 weeks on it before we realized we couldn't afford this approach and abandoned it. Knowledge Graph approach Looking back, both approaches suffered from the same problem -- they merged retrieval and judgement into the same system. It was like trying to ensure both precision and recall with one system. (Will be explained better in the next section) Key Breakthrough After several experiments and realizations, we stumbled upon a deceptively simple conceptual framework: having two models instead of one. The retriever's job is to show every plausible candidate. It manages recall, and the more generous it is — i.e. "this seems relevant" — the more successful it is. A judge's job is to decide whether a candidate actually matches. It manages precision, and it succeeds by being strict. Trying to solve for both using one model is trying to battle inherent trade-offs. So we split the problem into two layers: A retriever that returned top K results for any given query, optimized for recall. A verifier — a simple LLM call per candidate that marks each candidate as relevant (1) or not relevant (0) — optimized for precision. The immediate win

Pangram.

Human Written | 1524 Words | High Confidence

was that precision and recall could now be improved independently. We had a clean future path too: making the verifier finer-grained — distinguishing between relevant (2), related (1), and irrelevant (0) — without touching the retriever at all. The trade-off we accepted was that the per-query compute (and cost) would go up. A verifier that runs an LLM call per candidate is significantly more compute-heavy than a cosine similarity between two vectors. Looking at the big picture trend, we bet that cheap models would keep the cost under reasonable budget and that precision gains would be worth it. This definitely panned out. The Solution Retriever We finetuned a Reason-ModernColBERT model with 512 token-windows on both query and document side. ColBERT, a late-interaction model, was the right choice because it preserves the token-level structure that a single vector flattens. This was one significant failure of the previous vector search. Finetuning used triplet loss against the curated dataset. Retrieval and indexing ran on FAISS. At query time, the retriever pulled the top 20K candidates and passed them to the verifier. Verifier Gemini-2.5-flash-lite with a prompt iteratively optimized against the eval dataset. At first, a detailed prompt was manually written and manually optimized; after this, it was optimized using DSPy with MIPROv2. At the end of the process, a Cohen's kappa of ~0.75 between human and LLM judge was achieved. Other models were evaluated too, but Gemini-2.5-flash-lite was the model chosen after considering agreement and cost. At runtime, 20 parallel Gemini accounts each verified ~1K candidates, which kept latency within the acceptable threshold. First-page median load-time was ~1.2 seconds; cached queries were faster. Evaluation Methodology Data curation for evaluating these models proved to be just as big a challenge. A set of 17 queries with Top 1000 results using the existing system was chosen. These queries were chosen to span broad and niche queries using a combination of industry, business model, offering, and technology terms, mirroring the patterns of real user queries. Trained human analysts evaluated the Top 500 results for each query using information provided only in the factsheet. The next step was the prompt optimization of the LLM judge / verifier as previously described. The curated dataset was used to finetune a ColBERT model using triplet loss. The LLM judge was used to mark the performance improvements of the subsequent step models on the same dataset. All future models showed substantial improvements. Since the LLM verifier used the same prompt as the LLM judge, it could not have been used to judge its own performance. Technically, using the judge to judge itself would yield an accuracy of ~100% (almost a tautological necessity), thus practically meaningless for us. Therefore, the numbers used in this case study only refer to the performance of the retriever. The full production system is better than these numbers suggest because the verifier adds another high precision layer on top, but I won't claim a definite number because we can't cleanly measure this without extensive and independent human effort. Infrastructure GPU hosting the retriever on Runpod. Cost was dominated by Gemini API calls. A full batch of 20K candidates cost ~$1 — which was within the budget we'd assigned. Outcomes Adoption & Business Impact. This search filter was critical in ensuring a step-up in Screener usage, ensuring a 65% customer adoption for the Screener 2.0 feature which offered this search. Customer Feedback. Explicit positive feedback by power users on the AI search tool. Weekly search complaints by customers dropped by 50%+.

Human Written | 439 Words | High Confidence

Quality (Precision). Result accuracy improved from 77% to 98% (Precision@Top 100); and from 55% to 85% (Precision@Top

Pangram.

Human Written | 439 Words | High Confidence

1000). These numbers just refer to the improvements because of the search model, not the underlying document improvements; with document improvement, the step-up is much larger — ~48% to 98% Precision@Top 100, i.e. 40 pp. The precision drop-off at lower ranks is much steeper for the older models, while the newer ones still gave a higher proportion of relevant results even at later ranks. This mattered a lot for VC analyst workflows where users scroll well past the first page. Quantity (Recall). Since the retrieval was performed on factsheets instead of 3rd party data sources, almost all of the ~5M companies were part of the document set; instead of just ~2M when considering 3rd party data sources. Speed. Using a combination of parallelization and lazy loading, the performance was kept within acceptable user levels (first page of results median load time ~1.2 seconds, faster with caching). Learnings Simplifying systems versus model improvements When one model is doing two jobs badly, the better move is to split the jobs and employ two models that do their respective one well. We followed a relatively dumb (read: simple) approach: one AI model (ColBERT) to retrieve results, and one smart one (full LLM) that just marks relevant or irrelevant. Stack this and you've got a system that is cheap and works very well. Looking at the data / making high-quality training data Despite all the hullabaloo around architecture improvements, the brunt of the work was actually curating the 17 query dataset, understanding implicit query intent, manually going through and understanding factsheets, labelling each candidate result as relevant, related, or irrelevant. This involved a lot of back-and-forth, setting constraints on definitions and possibilities, peer reviews, and arguments among subject matter experts. Most of the "build fast, demo fast and fail fast" was downstream of creating this curated dataset. Accepting ‘The Bitter Lesson’ In the words of Richard Sutton, “One thing that should be learned from the bitter lesson is the great power of general purpose methods, of methods that continue to scale with increased computation even as the available computation becomes very great.” Our approach was simple, yes, but given enough computation, it would eventually beat any system we tried to build that has hard built-in assumptions/knowledge about the structure of the task. For instance, the KG approach was one such traditional style system that relied on connections between tags that were defined from Wikipedia (human knowledge), instead of connections being discovered by an LLM. This seems to me like yet another demonstration of the bitter lesson in action.