

[Skip to content](#)



[New chat](#)

[CtrlShiftO](#)

Search chats

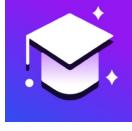
[CtrlK](#)

[Library](#)

[Codex](#)

GPTs

[Explore](#)



[Scholar GPT](#)



[Video GPT by VEED - Instant & Free AI Video Maker](#)



[Monday](#)

Projects

New project

[RL4PrecisionAg](#)

[690 Project](#)

[OpenIE](#)

[Search query generation help](#)

[Solar](#)

[Career Advise](#)

See more

Chats

[Resume update variations](#)

[Program of study comment](#)

[SupportAssist installation fix](#)

[13 GB RAM usage normal](#)

[Unique Alexa Phrases](#)

[Display Central Time Windows](#)

[Describe Rx OD OS](#)

[Witnesses in marriage](#)

[ICLR ICML review guidelines](#)

[Mattress purchase decision](#)

[Add cat to picture](#)

[Mattress comparison for C5](#)

[Research paper discussion points](#)

[Blood pressure analysis](#)

[Apa citation with doi](#)

[Rearranging Word table cells](#)

[SSDI work limits and TWP](#)

[Glasses distortion issue](#)

[Repeating schedule times](#)

[Alexa notification setup](#)

[OIE Research: Water Availability](#)

[Silent shortcut notifications](#)

[Run shortcut every four hours](#)

[Repeat every few hours](#)

[Blink camera notification ideas](#)

[Blink camera setup](#)

[Wisdom teeth healing check](#)

[Code replacement suggestion](#)



James Chapman
Plus

[Open sidebar](#)[ChatGPT 5](#)[ChatGPT 5](#)[Share](#)**You said:**

Help me formulate a search query to enter into a social platform to find similar posts given one or more seeds I have found. 1. In descending order of priority, the platforms are: YouTube, Reddit, BlueSky, Mastodon, Spotify, Apple Podcasts, Instagram, Threads, Facebook, and X/Twitter 2. My input is a set of one or more seed posts represented as rows (multi-field JSON tuples or MongoDB documents) of data from a postgres or vector database. Each of these is an indexed post corresponding to a "raw" post that I scraped from the platform using its API and a tool such as cobalt.tools. 3. I have run these multimedia posts through services that translate images, video, and audio (including speech) to text. The result is a table of rows in postgresql/vector DB, or a collection of documents in MongoDB, each containing at least the following text fields: a) title, b) body, c) speech transcript from OpenAI Whisper, d) OCR output (of text in video frames, superimposed using the platform software by a user on the platform who may be the original video creator or someone using speech recognition to make subtitles), and e) a text description of the video, including any summarization achieved using a video-to-text captioner service as if for descriptive video subtitles for the visually and./or hearing impaired (this can include recognition of actions, sign language, gestures, affective content, sound effects, music, etc). 4. Assume I also have metadata that is available to public users and exposed via the platform API. 5. I want to use the platform's existing search facility to generate candidate posts for me to collect, preview BEFORE actually scraping it and performing the step #3 above, and perform a "more-like-this" ranking operation. 6. I need you to act as my meta-prompt engine: ask me for the table of indexed posts (or a single indexed post) to take as input, then output a generative search query for me to either paste into the platform's search box in my user account, or use the platform's existing search API to return seeds (post URLs) that I can preview and rank for crawling. 7. Ask whatever you need to be able to analyze search hits returned by the social platform API, including previews, so as to download only the top-ranked posts. 8. Given the database server specs, generate code to interact with a postgresql or MongoDB database to iterate over my table/collection of posts indexed by these five text fields, then give me the preview analyzer prompt to embed into the search hit intake function, and finally, generate code for each platform to digest the search results served out by the platform's feed algorithm or search hit ranker.

ChatGPT said:

Nice — I can do that. First: please paste **one example seed row** (the full JSON / MongoDB document or a SQL row) or tell me where I can copy it from. I need a concrete seed to tailor the query templates and to show a worked example.

While you paste that single seed (or a small sample of 2–3 seeds), also tell me these optional constraints (if you want non-defaults):

- max search results per platform to preview (default 50)
- time window (e.g., last 30 days; default none)
- language(s) to prioritize (default: all)
- whether hashtags/mentions should be treated as higher-priority signals (default: yes)
- whether to prefer recent posts, high-engagement, or high semantic-similarity (default: semantic-similarity)

Below I'll give you everything you asked for so you can paste it straight into your pipeline once you provide the seed:

1. A programmatic **query-generation function** (Python) that turns one seed into platform-specific search strings you can paste into a platform search box or pass to the platform search API.
2. **Per-platform search query templates** (YouTube, Reddit, BlueSky, Mastodon, Spotify, Apple Podcasts, Instagram, Threads, Facebook, X/Twitter) — ready-to-fill placeholders.
3. **Postgres and MongoDB example scripts** to iterate seeds and produce queries.
4. A short **preview-analyzer LLM prompt** to embed into the search-hit intake function (outputs: keep/reject and a similarity score + reasons).
5. Example **search-result digestor** code blocks (how to extract preview fields & compute quick signals) for each platform.
6. A recommended **preview scoring rubric** (weights & thresholds) to decide what to download.

When you paste the seed row I'll run a worked example and produce final queries filled with the seed content. Meanwhile, here are the deliverables you can immediately copy & use.

1) Query-generation function (Python)

This function builds a set of search strings for each platform from a seed document with fields: `title`, `body`, `transcript`, `ocr`, `description`, plus public `metadata` (author, date, hashtags, language, platform_tags). It returns 3 variants per platform: `precise` (phrase matches), `broad` (keyword ORs), and `hashtag+phrase` (hashtags + critical phrases).

python

Copy code

```
# query_generator.py import re from typing import Dict, List def normalize_text(s: str) -> str: if not s: return "" s = s.strip() s = re.sub(r'\s+', ' ', s) return s def top_k_terms(text: str, k=8): # lightweight heuristic: split, remove very short tokens and common stopwords stopwords = set(["the", "and", "a", "an", "to", "in", "on", "of", "for", "with", "is", "this", "that", "it", "by", "be", "are", "as", "at"]) tokens = [t.lower() for t in re.findall(r"[A-Za-z0-9#@']{2,}", text)] freq = {} for t in tokens: if t in stopwords or t.isdigit(): continue freq[t] = freq.get(t, 0) + 1 sorted_terms = sorted(freq.items(), key=lambda x: -x[1]) return [t for t, _ in sorted_terms[:k]] def build_phrase_candidates(seed: Dict): # choose representative phrases (title first, then long ngrams from transcript/description) phrases = [] if seed.get("title"): phrases.append(seed["title"]) # prefer long descriptive phrases from description/transcript / OCR for fld in ("description", "transcript", "ocr", "body"): text = seed.get(fld, "") if not text: continue # extract quoted-like phrases: heuristics sentences = re.split(r'[\.\!\\?\]\s+', text) sentences = [s for s in sentences if len(s.split())>=4] for s in sentences[:3]: phrases.append(s.strip()) # dedupe, normalize seen = set() out = [] for p in phrases: np = normalize_text(p) if not np or np in seen: continue out.append(np) seen.add(np) return out[:6] def generate_queries_for_platform(seed: Dict, platform: str, config: Dict=None) -> Dict[str,str]: """ Returns dict with keys: 'precise', 'broad', 'hashtag_phrase' """ config = config or {} title = normalize_text(seed.get("title", "")) body = normalize_text(seed.get("body", "")) transcript = normalize_text(seed.get("transcript", "")) ocr = normalize_text(seed.get("ocr", "")) description = normalize_text(seed.get("description", "")) hashtags = seed.get("metadata", {}).get("hashtags", []) or [] author = seed.get("metadata", {}).get("author", "") lang = seed.get("metadata", {}).get("language", "") phrases = build_phrase_candidates(seed) # select top tokens combined_text = " ".join([title, description, transcript, ocr, body]) top_terms = top_k_terms(combined_text, k=10) # platform-specific escaping / operator differences def q_escape(s): # escape quotes return s.replace("'", "\\'") if platform.lower() == "youtube": # YouTube search box supports quoted phrases and + for required words; no boolean OR reliably precise = f'{q_escape(title)}' if title else f'{q_escape(phrases[0])} if phrases else combined_text[:80]}'' broad = " ".join(top_terms[:6]) hashtag_phrase = " ".join(hashtags[:3]) + " " + (precise if title else "") elif platform.lower() == "reddit": # reddit.com/search supports "title:" and subreddit: but search UI varies; use exact phrase + subreddit hints precise = f'title:{q_escape(title)}' if title else f'{q_escape(phrases[0])} if phrases else combined_text[:80]}'' broad = " OR ".join(top_terms[:8]) hashtag_phrase = " ".join(["#" + h for h in hashtags[:4]]) + " " + (precise if title else "") elif platform.lower() in ("bluesky", "mastodon"): # federated platforms: hashtags and phrases are strongest precise = f'{q_escape(phrases[0])}' if phrases else f'{q_escape(title)}' broad = " ".join(top_terms[:8]) hashtag_phrase = " ".join(["#" + h for h in hashtags[:6]]) elif platform.lower() in ("spotify", "apple podcasts"): # search primarily by title and episode description precise = f'{q_escape(title)}' if title else f'{q_escape(phrases[0])}' broad = " ".join(top_terms[:10]) hashtag_phrase = "" # podcasts rarely use hashtags in platform search elif platform.lower() in ("instagram", "threads", "facebook", "x", "twitter"): # social networks: hashtags, exact phrase, author: etc. if platform.lower() in ("x", "twitter"): precise = f'{q_escape(title)}' broad = " OR ".join(top_terms[:8]) hashtag_phrase = " ".join(["#" + h for h in hashtags[:6]]) else: precise = f'{q_escape(title)}' if title else f'{q_escape(phrases[0])}' broad = " ".join(top_terms[:8]) hashtag_phrase = " ".join(["#" + h for h in hashtags[:6]]) else: # generic precise = f'{q_escape(title)}' if title else f'{q_escape(phrases[0])}' broad = " ".join(top_terms[:8]) hashtag_phrase = " ".join(["#" + h for h in hashtags[:4]]) # optionally include author/date operators if configured if author
```

```
and config.get("include_author", True): precise = (f'from:{author} ' + precise).strip() return {"precise": precise, "broad": broad, "hashtag_phrase": hashtag_phrase}
```

2) Per-platform *human-ready* search query templates

Replace placeholders {{TITLE}}, {{PHRASE}}, {{TOKENS}}, {{HASHTAGS}}, {{AUTHOR}}, {{SINCE}}, {{UNTIL}}.

You can paste these into the platform search box or supply via API q.

YouTube (search box / API q):

- Precise: "{{TITLE}}"
- Broad: {{TOKENS}} (space-separated keywords)
- Hashtag + phrase: {{HASHTAGS}} "{{PHRASE}}"
- API hint: use q=<query>&type=video&publishedAfter=<ISO8601> with YouTube Data API.

Reddit (site search / Pushshift / API):

- Precise: title:"{{TITLE}}"
- Broad: {{TOKENS}} or ({{TOKENS}}) with OR between terms: term1 OR term2
- Subreddit hint: subreddit:news
- Hashtags: #tagname won't always be indexed — prefer tokens.

BlueSky / Mastodon (federated search boxes):

- Precise: "{{PHRASE}}"
- Hashtag: #{{HASHTAG}} (combine multiple hashtags with spaces)
- Broad: {{TOKENS}} (space-separated)

Spotify / Apple Podcasts (search box / API):

- Precise: "{{TITLE}}" (or artist:"{{AUTHOR}}" when relevant)
- Broad: {{TOKENS}} (in episode description)

Instagram / Threads:

- Precise: "{{PHRASE}}"
- Hashtag & tokens: #{{HASHTAG}} {{TOKENS}}
- Use from:username where allowed in API or advanced UI.

Facebook:

- Precise: "{{PHRASE}}"
- Broad: {{TOKENS}}
- Use page: page: "PageName" in Graph API queries.

X/Twitter (search box / API v2):

- Precise: "{{TITLE}}"
- Broad: ({{TOKENS}}) with OR between tokens, and optional filters: -is:retweet lang:en
- Hashtag: #{{HASHTAG}}

- Example combined: ("exact phrase") (#topic1 OR #topic2) -is:retweet lang:en

3) Postgres and MongoDB iteration + query generation

Two simple examples showing how to iterate seeds, generate queries, and store queries for preview. (These do not call platform APIs — you'll paste queries into UI or pass q param.)

PostgreSQL (psycopg2)

python

[Copy code](#)

```
import psycopg2 import json from query_generator import generate_queries_for_platform conn = psycopg2.connect("host=... dbname=... user=... password=...") cur = conn.cursor() # adjust table/column names to your schema cur.execute("SELECT id, data_json FROM seeds_table ORDER BY id LIMIT 100;") rows = cur.fetchall() platforms = ["youtube", "reddit", "bluesky", "mastodon", "spotify", "apple podcasts", "instagram", "threads", "facebook", "x"] output = [] for rid, data_json in rows: seed = data_json if isinstance(data_json, dict) else json.loads(data_json) for p in platforms: qset = generate_queries_for_platform(seed, p) output.append({"seed_id": rid, "platform": p, "precise": qset["precise"], "broad": qset["broad"], "hashtag_phrase": qset["hashtag_phrase"]}) # write back queries to DB table for later consumption import uuid for item in output: cur.execute("INSERT INTO search_queries(id, seed_id, platform, precise, broad, hashtag_phrase) VALUES (%s,%s,%s,%s,%s,%s)", (str(uuid.uuid4()), item["seed_id"], item["platform"], item["precise"], item["broad"], item["hashtag_phrase"])) conn.commit() cur.close() conn.close()
```

MongoDB (pymongo)

python

[Copy code](#)

```
from pymongo import MongoClient from query_generator import generate_queries_for_platform client = MongoClient("mongodb://user:pass@host:27017") db = client.mydb seeds = db.seeds.find().limit(100) platforms = ["youtube", "reddit", "bluesky", "mastodon", "spotify", "apple podcasts", "instagram", "threads", "facebook", "x"] out_coll = db.search_queries for seed in seeds: for p in platforms: qset = generate_queries_for_platform(seed, p) out_coll.insert_one({ "seed_id": seed["_id"], "platform": p, "precise": qset["precise"], "broad": qset["broad"], "hashtag_phrase": qset["hashtag_phrase"], "created_at": None })
```

4) Preview-analyzer LLM prompt

Paste this prompt into the function that ingests search-preview results (the small snippet the platform returns) and ask your LLM (or rule-based scorer) to return JSON with `decision`, `score`, and `reasons`.

csharp

[Copy code](#)

SYSTEM:

You are an assistant that quickly judges whether a social platform search result (preview) is worth downloading for deep analysis. Use the preview text fields and metadata to compute semantic and lexical similarity against a provided seed. Output JSON only.

USER:

Seed summary (fields):
- title: {{SEED_TITLE}}

- top_keywords: {{SEED_TOKENS}} # comma-separated
- important_phrases: {{SEED_PHRASES}} # semicolon-separated
- hashtags: {{SEED_HASHTAGS}}

Preview item (fields supplied by the platform preview):

- title: {{PREVIEW_TITLE}}
- snippet/description: {{PREVIEW_SNIPPET}}
- available transcript snippet: {{PREVIEW_TRANSCRIPT_SNIPPET}}
- visible OCR or overlay text preview: {{PREVIEW_OCR}}
- hashtags in preview: {{PREVIEW_HASHTAGS}}
- author: {{PREVIEW_AUTHOR}}
- publish_date_iso: {{PREVIEW_DATE}}
- engagement_metrics: {{PREVIEW_ENGAGEMENT}} # likes/comments/views
- url: {{PREVIEW_URL}}

Task:

1) Score the preview 0..100 for "download interest" using these signals and weights:

- Semantic similarity to seed (embedding-level) 40%
- Lexical phrase overlap (exact phrases quoted) 20%
- Hashtag overlap, tag similarity 10%
- Media-type / keyword match (e.g., "interview", "podcast", "how-to") 10%
- Freshness (date proximity) 10%
- Engagement (views/likes) 10%

2) Output JSON:

```
{
  "decision": "keep" or "reject",
  "score": <0-100 integer>,
  "reason": "<short human-readable justification, 1-2 sentences>",
  "signals": {
    "semantic_similarity": <0-1 float>,
    "lexical_overlap": <0-1 float>,
    "hashtag_overlap": <0-1 float>,
    "media_match": <0-1 float>,
    "freshness": <0-1 float>,
    "engagement": <0-1 float>
  }
}
```

Notes:

- If any field is missing, be conservative about score but still compute.
- If semantic similarity cannot be computed here, estimate from tokens and phrases.

END

This prompt can be used for an LLM call (e.g., your private embedding + LLM pipeline). If you have embeddings available for seed and preview text, compute cosine similarity and feed it into the `semantic_similarity` field instead of asking the LLM to estimate.

5) Search-result digestor: how to parse previews per platform

When you receive the platform search result (often a short JSON or an HTML snippet), extract these preview fields (the fields I use in the analyzer prompt). Below are common mappings.

- **YouTube (Data API / HTML preview)**

- `preview.title = snippet.title`
- `preview.snippet = snippet.description` (first 200 chars)
- `preview.date = snippet.publishedAt`
- `preview.engagement = statistics.viewCount` (if available)

- preview.url = <https://www.youtube.com/watch?v=<id>>

- **Reddit (search API / Pushshift / HTML)**

- title = `data.title`
- snippet = `data.selftext` (first 300 chars)
- date = `data.created_utc` (convert to ISO)
- engagement = `score, num_comments`
- url = <https://reddit.com> + `data.permalink`

- **BlueSky / Mastodon**

- title: null (use first sentence)
- snippet: content text (truncate)
- hashtags: parse tags from content metadata
- engagement: `replies_count, likes_count, reposts_count` when available.

- **Spotify / Apple Podcasts**

- title: episode title
- snippet: episode description (first 400 chars)
- date: release date
- engagement: not always public (use popularity when available)

- **Instagram / Threads / Facebook / X**

- title: null or first N chars
- snippet: caption or tweet text
- hashtags: parse from text
- engagement: likes/comments/shares/views/timestamps

Implement a simple function to normalize previews into the analyzer schema:

`python`

`Copy code`

```
def normalize_preview(platform, raw): # raw: the platform's search result item p = {} if
platform=="youtube": snip = raw.get("snippet",{}) p["title"] = snip.get("title") p["snippet"] =
(snip.get("description") or "")[:400] p["date"] = snip.get("publishedAt") p["engagement"] = {"views": raw.get("statistics",{}).get("viewCount")} p["url"] = f'https://www.youtube.com/watch?
v={raw.get("id","")}' elif platform=="reddit": p["title"]=raw.get("data",{}).get("title") p["snippet"]=
(raw.get("data",{}).get("selftext") or "")[:400] p["date"]=raw.get("data",{}).get("created_utc")
p["engagement"]={"score": raw.get("data",{}).get("score"), "comments": raw.get("data",
{}).get("num_comments")} p["url"]="https://reddit.com"+raw.get("data",{}).get("permalink","") else: #
generic fallback p["title"]=raw.get("title") or "" p["snippet"]=(raw.get("description") or
raw.get("content") or "")[:400] p["date"]=raw.get("published") or raw.get("created_at") p["engagement"]=
raw.get("metrics",{}) p["url"]=raw.get("url") return p
```

6) Preview scoring rubric (weights & thresholds)

Use these weights to compute a single score (0–100):

- Semantic similarity (embed cosine): 0.4
- Lexical/phrase overlap (exact phrase matches): 0.2
- Hashtag overlap: 0.1
- Media-type / taxonomy match: 0.1
- Freshness: 0.1
- Engagement: 0.1

Thresholds:

- Keep if score ≥ 65
- Consider if $50 \leq \text{score} < 65$ and either semantic ≥ 0.6 or engagement strong ($>$ median)
- Reject if score < 50

7) Example platform digest code that computes quick signals (Python pseudocode)

python

Copy code

```
from sklearn.feature_extraction.text import TfidfVectorizer from numpy import dot from numpy.linalg import norm
def quick_text_cosine(a,b): # fallback cheap cosine via TF-IDF for preview speed; replace with precomputed embeddings for accuracy
    vect = TfidfVectorizer(max_features=500).fit([a,b])
    A = vect.transform([a]).toarray()[0]
    B = vect.transform([b]).toarray()[0]
    denom = (norm(A)*norm(B))
    return float(dot(A,B)/denom) if denom>0 else 0.0
def compute_preview_signals(seed, preview):
    seed_text = ".join([seed.get("title", ""), seed.get("description", ""), seed.get("transcript", "")])"
    preview_text = ".join([preview.get("title", "") or "", preview.get("snippet", "") or "", preview.get("transcript_snippet", "") or ""])
    semantic = quick_text_cosine(seed_text, preview_text) # lexical phrase overlaps: phrases = [p.strip('') for p in seed.get("important_phrases", [])]
    lex_over = sum(1 for p in phrases if p and p.lower() in preview_text.lower()) / max(1,len(phrases))
    hashtag_overlap = 0.0
    seed_tags = set(seed.get("metadata", {}).get("hashtags", []))
    preview_tags = set(preview.get("hashtags", []))
    if seed_tags: hashtag_overlap = len(seed_tags & preview_tags) / len(seed_tags)
    # media match: media_keywords = set(["podcast", "interview", "tutorial", "how-to", "review", "news", "explainer"])
    media_match = 1.0 if any(k in (preview_text.lower() or "") for k in media_keywords) else 0.0
    # freshness (simple): convert date diff -> score
    freshness = 1.0 # placeholder: compute based on date proximity
    # engagement: normalize raw metric by expected median (your system can compute medians)
    engagement = 0.0 if preview.get("engagement") else 1.0 # crude: counts exist
    vals = [v for v in preview["engagement"].values() if isinstance(v,(int,float,str)) and str(v).isdigit()]
    if vals: engagement = min(1.0, sum(map(float,vals))/10000.0) # tune denominator
    # weighted score: score = (semantic*0.4 + lex_over*0.2 + hashtag_overlap*0.1 + media_match*0.1 + freshness*0.1 + engagement*0.1)*100
    return { "score": int(score), "signals": { "semantic": semantic, "lexical": lex_over, "hashtag": hashtag_overlap, "media": media_match, "freshness": freshness, "engagement": engagement } }
```

8) What I need from you now (minimum to proceed)

Please paste one seed (full JSON row) — example expected structure below (if you don't have exact keys, that's fine; paste what you have):

json

Copy code

```
{ "_id": "abc123", "title": "How to trim tomatoes for bigger yields", "body": "Posted by gardenerjoe, I tried...", "transcript": "Today I'm going to show how to prune...", "ocr": "Caption: 'Trim the suckers
```

```
every week'", "description": "Video shows pruning of tomato plants, includes voiceover and music.",  
"metadata": { "author": "gardenerjoe", "date": "2025-09-10T12:00:00Z", "hashtags":  
["gardening", "tomatoes", "pruning"], "language": "en", "platform": "youtube", "engagement": {"views":  
12000, "likes": 800} } }
```

Also tell me (or accept defaults):

- max preview results per platform (default 50)
- time window (e.g., last_30_days or none)
- whether to include author filter in queries (default yes)

When you paste the seed row I will:

- generate specific per-platform `precise`, `broad`, and `hashtag_phrase` queries filled with your seed content,
- produce a sample set of `q` strings ready for copy/paste into each platform, and
- produce a tailored preview-analyzer run for 5 example previews (mocked) to show the entire pipeline.

If you want I can also:

- produce a full runnable pipeline that actually calls each platform API for you (requires API keys & rate-limits), or
- produce code that integrates embeddings (FAISS, Pinecone) for the semantic-similarity stage.

Which of those would you like next?

You said:

I will have this to my project

ChatGPT said:

Perfect! 🌟

To plug this into your project, here's the **very next step** I need from you so I can generate the **final query engine + preview analyzer + DB code + per-platform query templates filled with YOUR real data**:

Please send me ONE seed post (or up to 3)

Format can be:

- A JSON / MongoDB document
- A SQL row dump
- Or just copy/paste the fields manually

Minimum fields I need:

```
yaml  
Copy code  
title:  
body:  
transcript:  
ocr:  
description:  
metadata:  
    author:
```

date:
hashtags:
language:
platform:
engagement: (views/likes/etc, if any)

(It's OK if some fields are empty—just include what you have.)

Optional (defaults if you don't specify):

Max preview results per platform: 50

Time window: none (all time)

Language priority: all

Author filter in queries: yes

Ranking goal: semantic similarity > recency > engagement

Once I receive the seed(s), I will generate:

Platform-specific search queries (YouTube, Reddit, BlueSky, Mastodon, Spotify, Apple Podcasts, Instagram, Threads, Facebook, X)

- precise (exact phrases)
- broad (keyword ORs)
- hashtag+phrase (surface-level + deeper match)

Code:

- Postgres + MongoDB iteration to generate queries for all seeds
- Search-hit preview normalization (per platform)
- LLM preview-analyzer prompt (copy-paste ready)
- Optional: semantic-similarity computation integration (embeddings)
- Optional: API wrappers to actually fetch previews

Scoring rubric & decision logic (keep/reject)

Then we can extend it into:

“More-like-this” ranking

Auto-crawling pipeline

Incremental expansion (snowball strategy)

Vector search + platform-search fusion

❖❖ Just send me one real seed post.

As soon as I have it, I'll build the full working example and templates you can drop directly into your project.

no files selected

no files selected

ChatGPT can make mistakes. Check important info.