TrendPilot

AI-Powered LinkedIn Content Strategy Tool

Trend Finder Module -- Progress Report

Iterative Development of Trend Identification Pipeline

Author: Biraj Mishra

Date: February 16, 2026

Course: MS Capstone Project

Table of Contents

1. Objective

Build a system that takes a LinkedIn professional's profile bio and automatically identifies timely, specific, trending topics relevant to their expertise -- then recommends a concrete LinkedIn post angle backed by real-time search data from Google Trends.

The system must produce post recommendations that are specific enough to be immediately actionable (e.g., referencing a new product launch or rising search trend), not generic thought-leadership angles (e.g., "Why AI is the future"). The recommendations must be grounded in verifiable, real-time data.

2. Iteration Summary

Iter	Approach	Topic Extraction	Trend Source	Outcome
1	BuzzSumo + News	Keyword matching	News articles	Generic, noisy
2	Google Trends +	Named Entity	related_queries	Ambiguous entities
3	Google Trends +	LLM prompt	interest_over_time +	Specific, actionable
4	Iter 3 + Interactive	Same as Iter 3	Same + merged API calls	User control, factual

3. Iteration 1: BuzzSumo + News API

Approach

Used BuzzSumo API to find popular content and News API to find trending articles. Keywords were extracted from the user's LinkedIn bio using simple keyword matching, then matched against news headlines and BuzzSumo trending content.

Problems Encountered

- Generic topics: Results were dominated by broad news stories (e.g., "AI is transforming business", "Cloud computing market growth") with no specific angle for a LinkedIn post.
- Noisy results: News API returned articles about politics, sports, entertainment -- not filtered to the user's professional domain.
- Low relevance: No mechanism to score how relevant a trending topic was to the user's specific skill set.

Example Output

```
Topics found:
```

```
- "AI Revolution in Healthcare" (news article, not user's domain)
- "Cloud Computing Market Worth $1.5T by 2030" (generic market report)
- "Top 10 Programming Languages 2026" (listicle, not actionable)
```

Decision

Abandon news-based approach. Shift to Google Trends for real-time search interest data that better reflects what professionals are actually searching for.

4. Iteration 2: Google Trends + spaCy NLP

File: trend_finder.py (596 lines)

Approach

Used spaCy's en_core_web_sm NLP model to extract named entities (ORG, PRODUCT, PROPN, etc.) from the user's LinkedIn bio, then queried Google Trends related_queries API for each extracted entity.

Key Components

- Entity extraction using spaCy NER with 11 entity labels (PERSON, ORG, GPE, PRODUCT, EVENT, etc.)
- Proper noun extraction for tokens tagged as PROPN
- Noun phrase extraction for multi-word technical terms
- TrendCache class with 7-day TTL and composite cache keys (keyword|geo|timeframe)
- Retry logic with exponential backoff for Google Trends rate limiting (HTTP 429 errors)

Entity Extraction Code (spaCy NER)

```
ENTITY_LABELS = {
    "PERSON", "ORG", "GPE", "PRODUCT", "EVENT",
    "WORK_OF_ART", "LAW", "LANGUAGE", "NORP", "FAC", "LOC"
}

doc = nlp(text)  # spaCy processes text
for ent in doc.ents:
    if ent.label_ in ENTITY_LABELS:
        entities.append(ent.text)  # e.g., "Cloud", "Spring", "Kong"

for token in doc:
    if token.pos_ == "PROPN":
        entities.append(token.text)  # proper nouns
```

Problem 1: The "Cloud" Ambiguity

spaCy has no domain awareness. When a bio mentioned "cloud architecture" or "Spring Cloud", spaCy extracted "Cloud" as a generic proper noun. Google Trends then returned results for cloud weather -- not cloud computing:

```
Entity extracted: "Cloud" (PROPN)
Google Trends top queries:
  - "cloud weather today" (value: 100)
  - "cloud types" (value: 65)
```

```
  - "cumulus cloud" (value: 37)
```

Problem 2: The "Spring" Ambiguity

Similarly, "Spring" from "Spring Cloud" or "Spring Boot" was extracted standalone. Google Trends returned seasonal results:

```
Entity extracted: "Spring" (PROPN)
Google Trends top queries:
  - "spring 2026" (value: 100)
  - "spring fever" (value: 74)
  - "hot spring" (value: 49)
Rising queries:
  - "spring fever episodes" (+51500%)
  - "best places to visit in spring" (+44000%)
```

Problem 3: The "Kong" Ambiguity

The API gateway tool "Kong" was extracted correctly, but Google Trends returned entertainment results:

```
Entity extracted: "Kong" (ORG)
Google Trends top queries:
  - "king kong" (value: 100)
  - "hong kong china" (value: 54)
  - "donkey kong switch" (value: 17)
  - "godzilla x kong" (value: 14)
```

Problem 4: Rate Limiting

The error log (trend_finder.log) shows aggressive 429 errors from Google Trends API:

```
2026-02-07 13:45:37 - WARNING - Retry 1/2 for 'java programming':
  Google returned a response with code 429
2026-02-07 13:45:40 - WARNING - Retry 2/2 for 'java programming':
  Google returned a response with code 429
2026-02-07 13:45:45 - ERROR - Could not fetch trends:
  TooManyRequestsError: code 429
```

Decision

spaCy NER is not suitable for technical topic extraction from professional bios. The model lacks domain context and cannot distinguish "Cloud (technology)" from "Cloud (weather)". Replace NLP entity extraction with LLM-based prompt engineering.

5. Iteration 3: Google Trends + GPT-4o Prompt Engineering

File: trend_identification_v2.py (initial version)

Approach

Replaced spaCy NER entirely with a GPT-4o prompt that understands professional context. Added interest_over_time API for trend scoring alongside related_queries (top + rising) for search context. The LLM both extracts topics AND recommends a post angle.

Key Changes from Iteration 2

- LLM-driven topic extraction instead of spaCy NER
- Trend scoring via interest_over_time (mean score over 3 months)
- Top + rising queries fed into the post recommendation prompt
- Two-stage LLM pipeline: extract topics -> score via Google Trends -> recommend post

Topic Extraction Prompt (GPT-4o)

```
You are analyzing a LinkedIn professional bio.

Extract 8-12 specific, post-worthy technical topics
suitable for LinkedIn.
Only include:
- Technologies, Platforms, Tools, Frameworks,
  Engineering practices

Exclude:
- Job titles
- Generic words (e.g., cloud, data, software)
- Soft skills
- Single generic nouns

Return the result as a comma-separated list.
```

Why This Prompt Works

- Explicitly excludes generic words like "cloud", "data", "software" -- solving the Iteration 2 ambiguity problem.
- Asks for "specific, post-worthy technical topics" -- the LLM understands that "Spring Cloud" is a technology while "Spring" alone is ambiguous.
- Filters out job titles and soft skills that polluted Iteration 2 results.

Post Recommendation Prompt (Key Instructions)

```
Instructions:
1. Focus on SPECIFICITY over generality. Look at the
   rising/breakout searches - these reveal new releases,
   announcements, hot debates, or breakthroughs.
2. The post title MUST reference a specific thing (a new
   feature, release, tool, comparison, migration,
   controversy, or real-world use case).
3. If rising queries mention a specific product launch,
   version, integration, or comparison - use that as
   the post hook.
4. The post should feel like the author is reacting to
   something happening NOW.
```

Example Output (Real Cached Data)

For a LinkedIn bio mentioning API gateways, microservices, AWS, and integration architecture:

```
Topic Extraction (LLM):
  AWS, OpenShift, Spring Cloud, Microservices, Apigee,
  Kong, REST APIs, Integration Architecture, API Gateways,
  Zuul, WSO2, Axway

Top Trending Topics (sorted by trend score):
  1. AWS              (score: 55.2)
  2. OpenShift        (score: 51.7)
  3. Spring Cloud     (score: 50.8)
  4. Integration Arch (score: 39.2)
  5. Microservices    (score: 31.2)

AWS - Rising Searches:
  aws devops agent (+1600%)
  aws transform custom (+1300%)
  aws security agent (+300%)

Apigee - Rising Searches:
  apigee news (+250%)
  apigee pricing (+200%)
  kong (+190%)
```

Remaining Issues

- Blank related queries: Some topics (OpenShift, Spring Cloud, Zuul) returned empty top_queries and rising_queries. Root cause: the code called build_payload() twice per keyword -- once for interest_over_time, again for related_queries -- and the second call was rate-limited.
- LLM hallucinated version numbers: When recommending a post about OpenShift, the LLM produced:

```
TOPIC: OpenShift
POST TITLE: "OpenShift 4.12: What Its New GitOps Features
  Mean for Streamlined Cloud-Native Deployments"

Problem: OpenShift 4.12 is outdated (current is 4.21).
The LLM fabricated the version from its training data -
it was NOT in the search data.
```

6. Iteration 4: Interactive Loop + Version Safety + Query Fix

File: trend_identification_v2.py (current version, 272 lines)

Fix 1: Merged API Calls (Blank Queries)

Before (Iteration 3) -- Two separate functions, two build_payload calls per keyword:

```
def fetch_trend_score(keyword):
    pytrends.build_payload(...)  # 1st API call
    df = pytrends.interest_over_time()
    return float(df[keyword].mean())

def fetch_related_queries(keyword):
    pytrends.build_payload(...)  # 2nd call (RATE-LIMITED!)
    related = pytrends.related_queries()
    ...
```

After (Iteration 4) -- Single function, one build_payload call:

```
def fetch_trend_data(keyword):
    pytrends.build_payload(...)  # Single API call

    # Trend score
    df = pytrends.interest_over_time()
    score = float(df[keyword].mean())

    # Related queries (same payload, no extra call)
    related = pytrends.related_queries()
    ...
    return score, top_queries, rising_queries
```

Cache validation was also updated to re-fetch entries saved without query data:

```
# Old: accepted stale entries without queries
if cache_entry and is_cache_valid(timestamp):

# New: re-fetches if queries are missing
if cache_entry and is_cache_valid(timestamp)
   and cache_entry.get("top_queries") is not None:
```

Fix 2: Version Number Hallucination Guard

Added a critical rule to the post recommendation prompt to prevent fabricated version numbers:

```
CRITICAL RULE - Version numbers and factual accuracy:
- NEVER invent or guess version numbers, release names,
```

```
  or feature names. Your training data may be outdated.
- ONLY mention a specific version/release if it explicitly
  appears in the search query data above.
- If no version is mentioned in the search data, write
  the post title WITHOUT a version number. Use phrasing
  like "latest release", "new update", or focus on the
  concept/trend instead.
- NEVER fabricate announcements, launches, or features
  that are not evidenced by the search data.
```

Added explicit bad/good examples in the prompt:

```
BAD:  "OpenShift 4.12: What Its New GitOps Features Mean..."
       (version not from search data = hallucinated)

GOOD: "OpenShift's latest update doubles down on GitOps -
       here's what changed for cloud-native teams"
```

Fix 3: Interactive Topic Selection Loop

Before (Iteration 3): Program ran once, LLM auto-picked a topic, then exited. After (Iteration 4): User sees top 5 topics and can interactively choose:

```
--- Top 5 Trending Topics (Profile-Driven) ---

  1. AWS  (trend score: 55.2)
     Top searches: what is aws, aws news, aws ai
     Rising: aws devops agent (+1600%)

  2. OpenShift  (trend score: 51.7)
  3. Spring Cloud  (trend score: 50.8)
  4. Integration Architecture  (trend score: 39.2)
  5. Microservices  (trend score: 31.2)

  0. Let AI pick the best topic automatically
  q. Quit

Choose a topic number (1-5), 0 for AI pick, or q to quit:
```

After each recommendation, the menu loops back -- the user can explore multiple topics without restarting the program.

Fix 4: Error Visibility

```
# Before: Silent exception swallowing
except Exception:
    score = 0.0

# After: Errors are printed
```

```
except Exception as e:
    print(f"[warn] Failed to fetch: {e}")
    score = 0.0
```

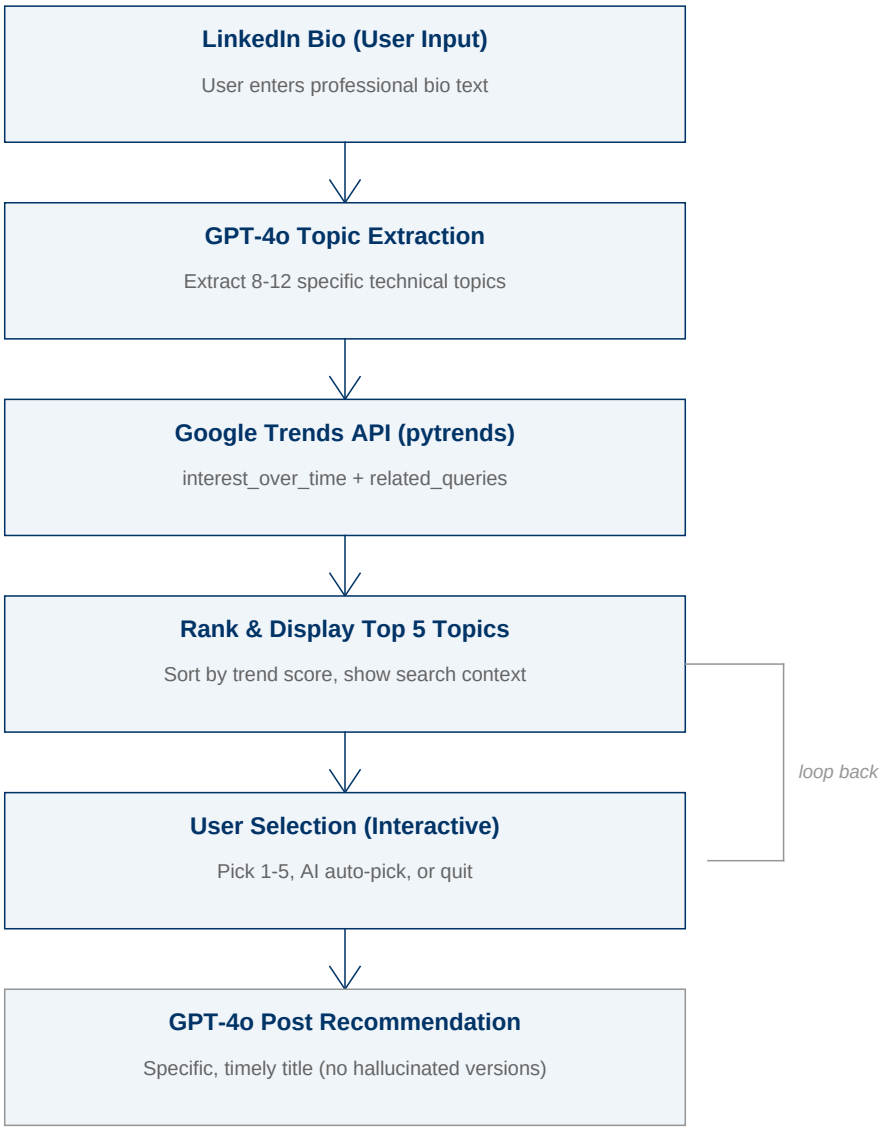
Fix 5: JSON Serialization Safety

Added explicit type conversion for query values to prevent numpy int64 serialization errors when saving to the JSON cache:

```
top_queries = [
    {"query": row["query"], "value": int(row["value"])}
    for _, row in top_df.head(10).iterrows()
]
rising_queries = [
    {"query": row["query"], "value": str(row["value"])}
    for _, row in rising_df.head(10).iterrows()
]
```

7. System Architecture (Current -- Iteration 4)

The current pipeline follows a two-stage LLM architecture with Google Trends data enrichment:



8. Key Learnings

Problem	Root Cause	Solution
Generic topics (Iter 1)	News APIs return broad content, not professional context	Switched to Google Trends for real search interest

Problem	Root Cause	Solution
"Cloud" = weather (Iter 2)	spaCy NER has no domain	Replaced with GPT-4o prompt that excludes
"Spring" = season (Iter 2)	NLP model lacks tech context	LLM extracts "Spring Cloud" as a whole term
Blank related queries (Iter 3)	Double build_payload() caused	Merged into single fetch_trend_data() call
Hallucinated versions (Iter 3)	LLM invents versions from stale	Prompt rule: only cite versions from search
One-shot exit (Iter 3)	Program terminated after one	Added interactive loop with re-selection
Silent failures (Iter 3)	except Exception: pass hid errors	Now prints [warn] with error message

9. File Inventory

File	Lines	Purpose
trend_finder.py	596	Iteration 2 - spaCy NER + Google Trends (deprecated)
trend_identification_v2.py	272	Iteration 3 & 4 - GPT-4o + Google Trends (current)
trend_cache.json	~785	Shared cache file (7-day TTL)
trend_finder.log	79	Error logs from Iteration 2 (429 rate-limit errors)
requirements.txt	3	Dependencies: spacy, pytrends, pandas

10. Next Steps

- Integrate trend finder output with the post generation agent (streamlit_poc/agents/post_generator.py)
- Add geographic customization (currently hardcoded to US)
- Explore caching related queries more aggressively to reduce API calls
- Add support for multi-profile batch analysis
- Evaluate prompt variations for topic extraction quality