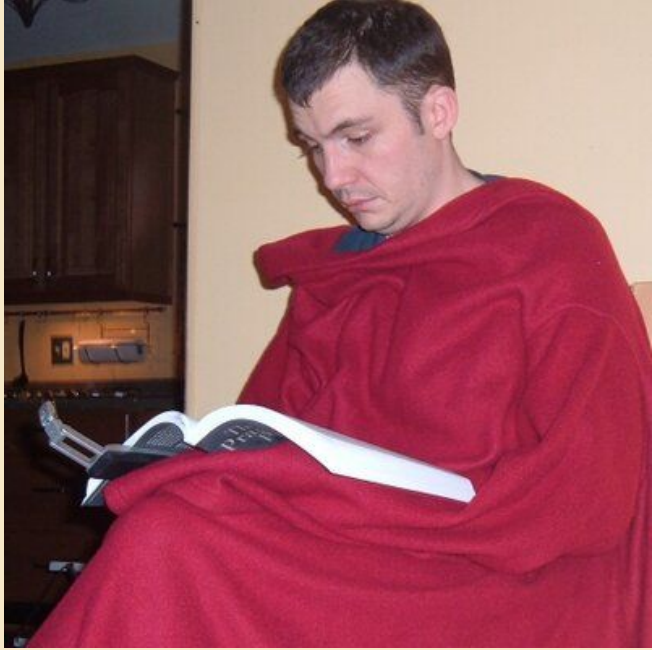
About me

- Share my learnings early
- Honesty a core value
- Still trying to figure out how search works
- Embarrassing myself as a service



http://softwaredoug.com

(Search since 2013, training + consulting)

reddit LexisNexis yelp

shopify amazon WIKIPEDIA

Come *supercharge* your AI Search skillset with us July 1 - 31!

This course is popular.

Next Cohort:
July 1 – July 31

AI-Powered Search: Modern Retrieval for Humans & Agents

Enrolled Students have access to the Recordings, Code, and Course Materials for life.

Students from 150+ companies:

Apple · TikTok · UBER · DuckDuckGo · reddit · GAP · docusign · DOORDASH

airbnb · wayfair · indeed · BEST BUY · elastic · Tripadvisor · Pinterest · amazon · fiverr.

Course Info:

$450 discount
(this week only)

Promo Code:
BERLIN

aipoweredsearch.com/live-course

Alumni Reviews:

Win a **FULL scholarship** to the course!

AI-Powered Search

Modern retrieval for humans and agents

Join the *course*, *learn with peers*!

with **Doug Turnbull** & **Trey Grainger**

René Kriegler

Kim Falk

Amélie Chatelain

Andreas Eriksen

aipoweredsearch.com/live-course

+

Women of Search

Sign up with your email address to receive news and updates.

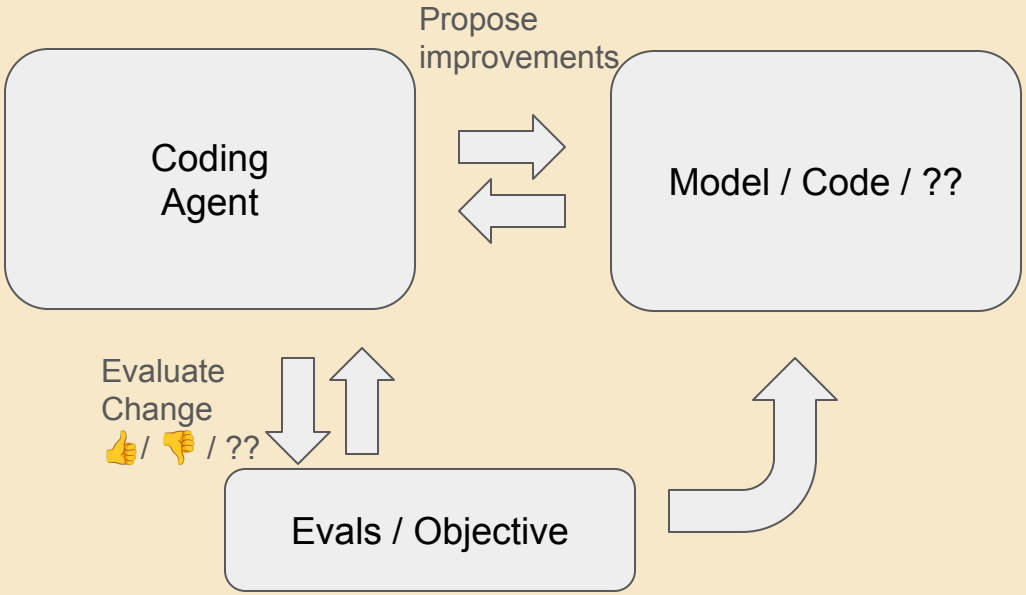
Email Address | Sign Up

women-of-search.org

Learn with **students** *from* **top companies** *like:*

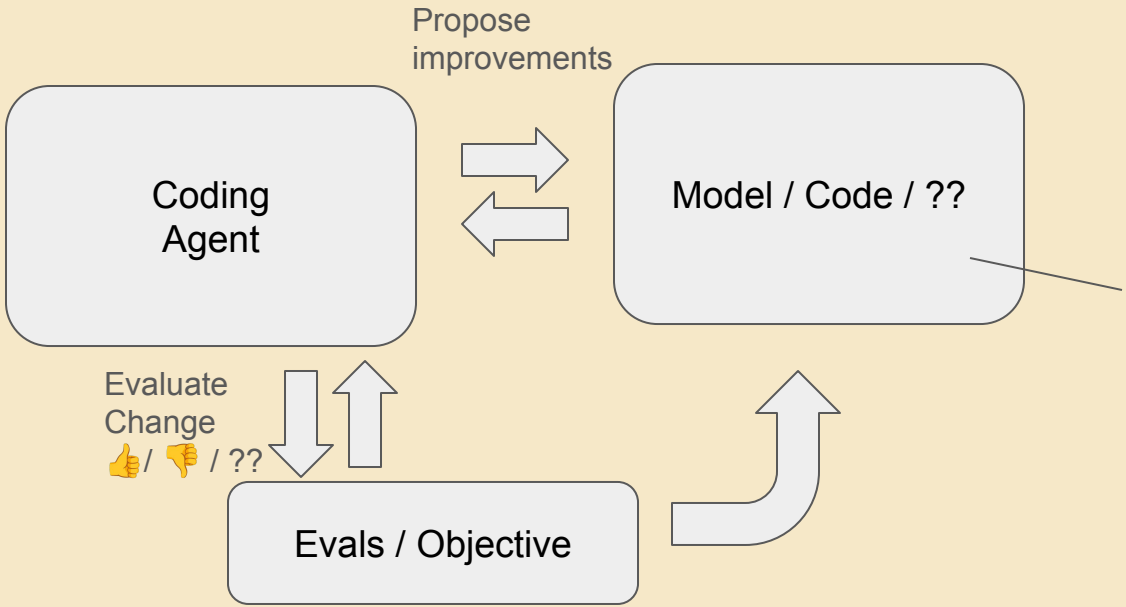
Apple · TikTok · UBER · DuckDuckGo · reddit · airbnb · Pinterest · amazon · wayfair · DOORDASH · yelp

Autoresearch - the idea



https://github.com/karpathy/autoresearch

Why? Deployment story

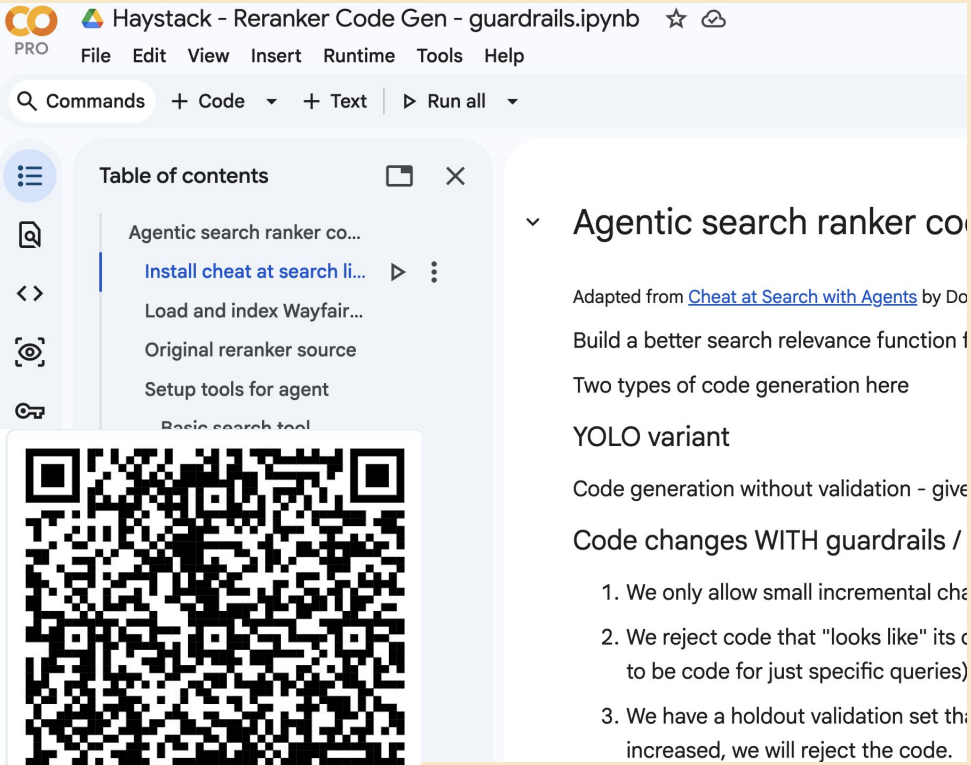


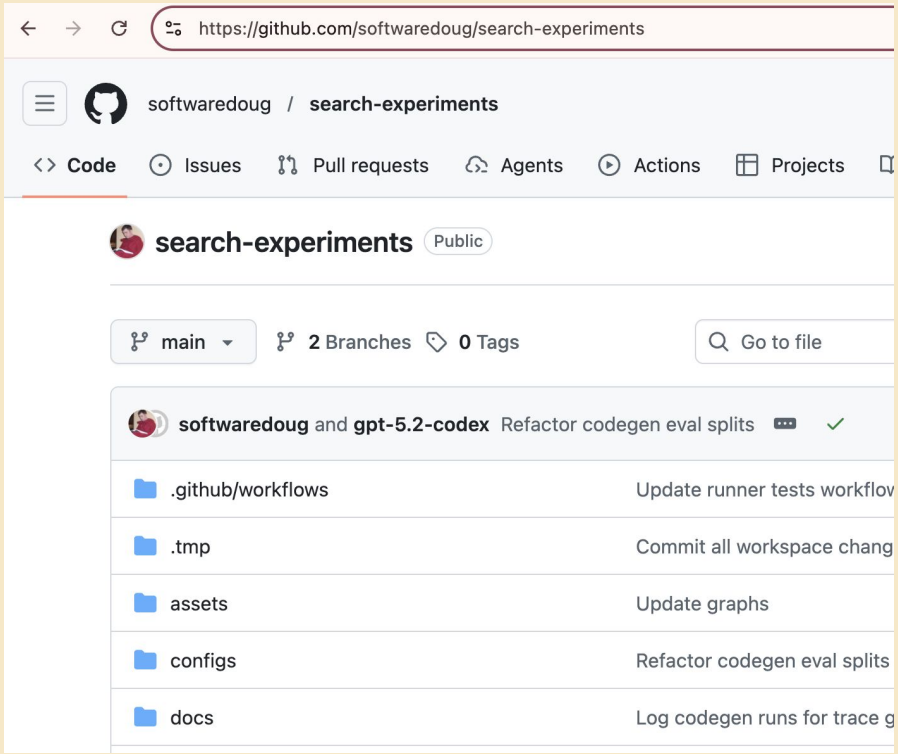
You’re already deploying this somewhere. You’re just making it better

Today: exploration of experiments

- My DIY Agent -> Autoresearch setup
- What I learned experimenting on datasets
- Sometimes backed up by data. Other times intuition.

Repo + Notebook

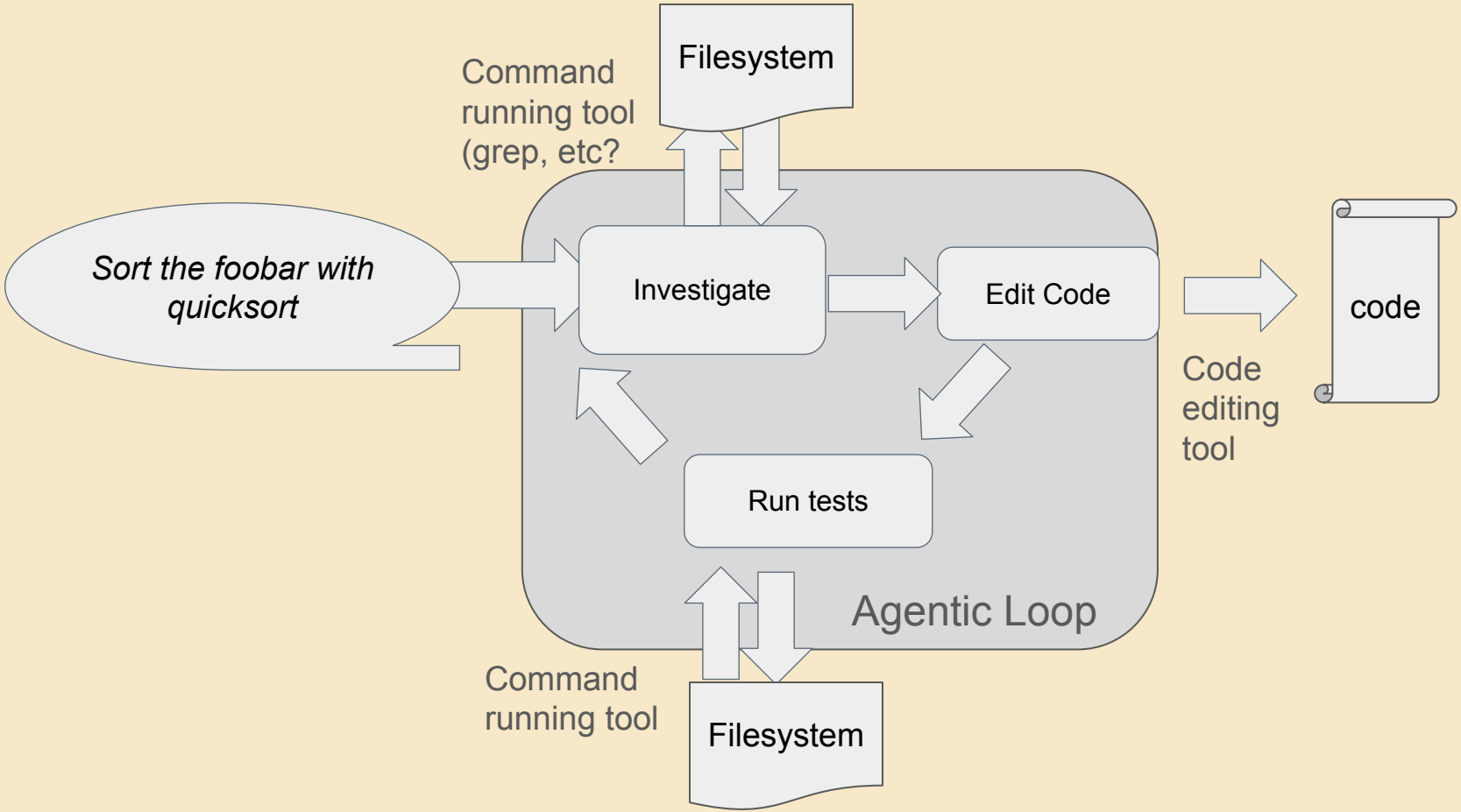




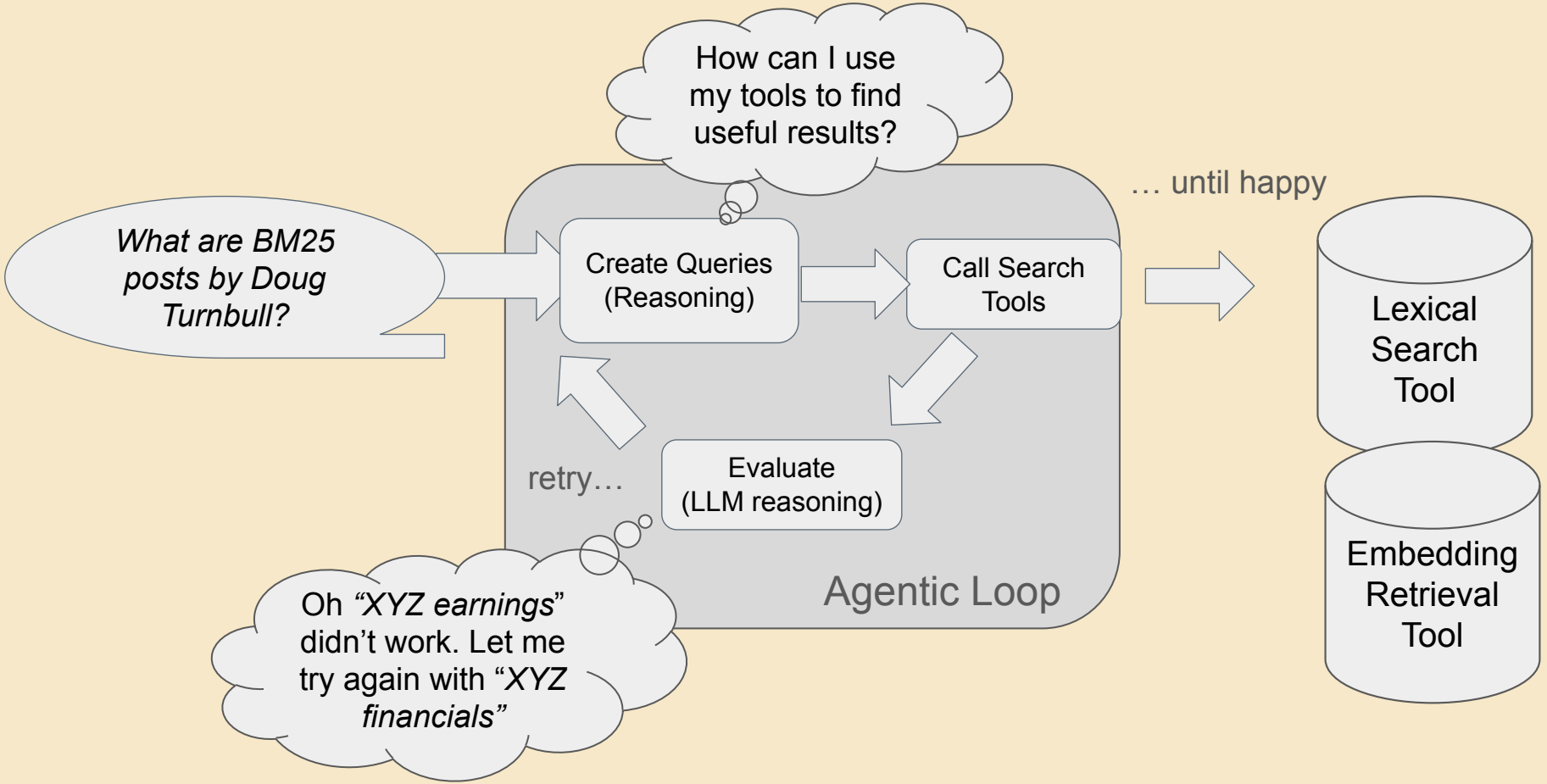
https://github.com/softwaredoug/search-experiments

Agentic Search Setup

Agents solve problems w/ tools (ie coding)



Agentic search: agents solving a *search* problem



Boring old, hallucinating AI

In:

```
inputs = [
    {"role": "system", "content": "You are a helpful assistant that helps find blog posts."},
    {"role": "user", "content": "What are some blog posts by Doug Turnbull about BM25"},
]

resp = openai.responses.create(
    model="gpt-5-mini",
    input=inputs,
)
resp.output[-1].content[-1].text
```

Out:

```
- “BM25 explained” (Elastic blog) — a straightforward explainer of the
function, its parameters (k1, b), and intuition for term frequency and
length normalization.
- “BM25 and TF saturation / parameter tuning”
- “BM25 vs. TF-IDF / Why BM25 replaced TF-IDF in Lucene” — historical/
post describing differences between classic TF-IDF and BM25 and why BM
default in Lucene/Elasticsearch.
```

Define a tool - just a python function

(In most frameworks)

In:

```
def search(keywords: str):
    """A simple BM25 keyword search of Doug Turnbull's blog posts."""
    … body omitted …

search("bm25")
```

Name + docstring + params becomes part of prompt

Out:

```
[{'id': 160,
  'title': 'Bayesian BM25 is cool',
  'score': 37.75146675109863,
  'publication_date': '2026-03-16'},
 {'id': 121,
  'title': 'Can BM25 be a probability?',
  'score': 36.11565685272217,
  'publication_date': '2026-03-06'},
```

Tell OpenAI about tools

In:

```
resp = openai.responses.create(
   model="gpt-5-mini",
   input=inputs,
   tools=... (openai tool spec) ...,
)
resp.output
```

We tell agent about a tool…

(name, description, params…)

Out:

```
[

ResponseReasoningItem(id='rs_0b3cc088788503af0069f904b2ad1c8197be9552cd0698014f',
summary=[], type='reasoning', content=None, encrypted_content=None, status=None),

ResponseFunctionToolCall(arguments='{"keywords":"BM25"}',
call_id='call_a5wFOmExw3RI1HMn9MJriMCy', name='search', type='function_call',
id='fc_0b3cc088788503af0069f904b42fd0819786283f7d06d79f18', status='completed')

]
```

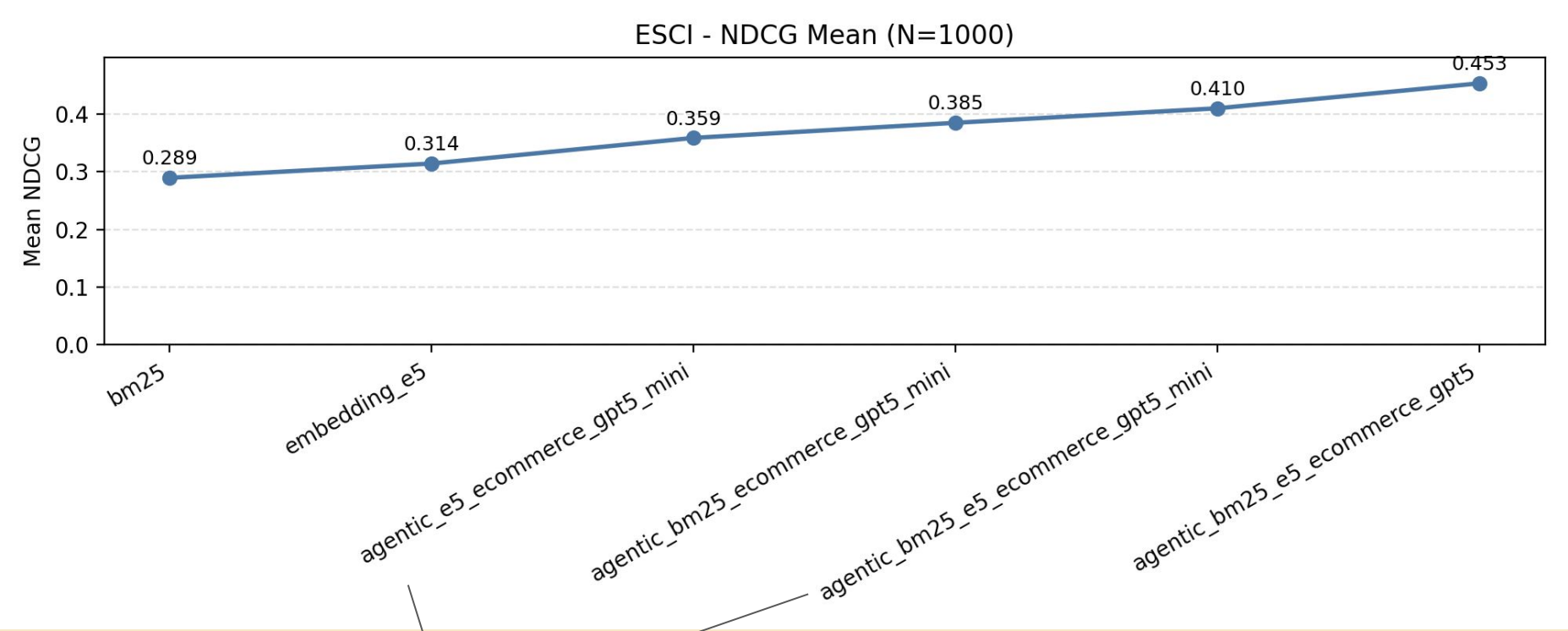
… agent requests us to call it

Loop until done calling

```
while tool_calls:
  tool_calls = False

  resp = openai.responses.create(
      model="gpt-5-mini",
      input=inputs,
      tools=... (openai tool spec) ...,
  )
  inputs += resp.output

  for func_output in resp.output:
     if func_output.type == "function_call":
        if func.name == "search”:
           tool_response = search(...)
           inputs.append(tool_response)
        tool_calls = True
```

Call tool, append the results to context

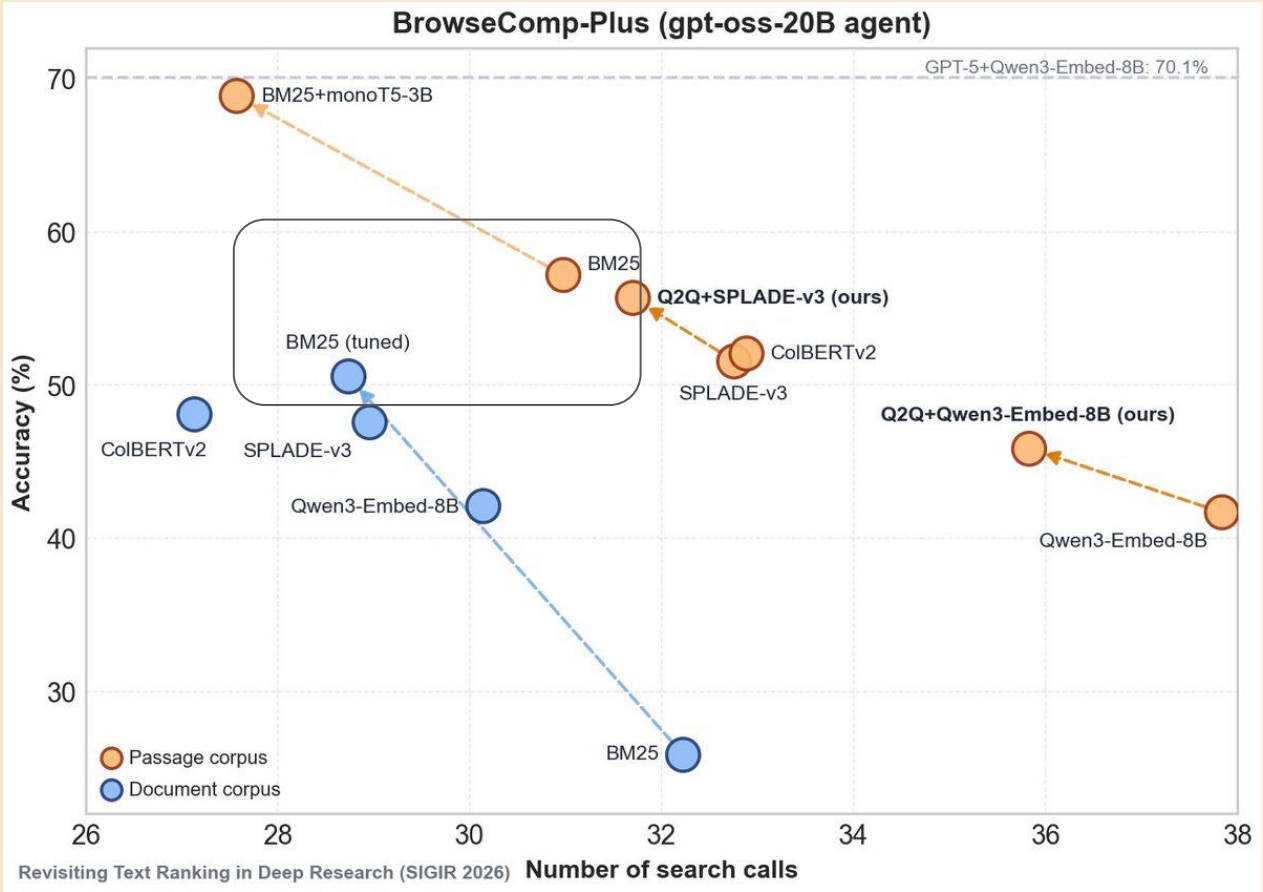
And it works!



Tool combos
(embedding + BM25)

http://github.com/softwaredoug/search-experiments

Even at hard deep research tasks



Decent on very hard questions

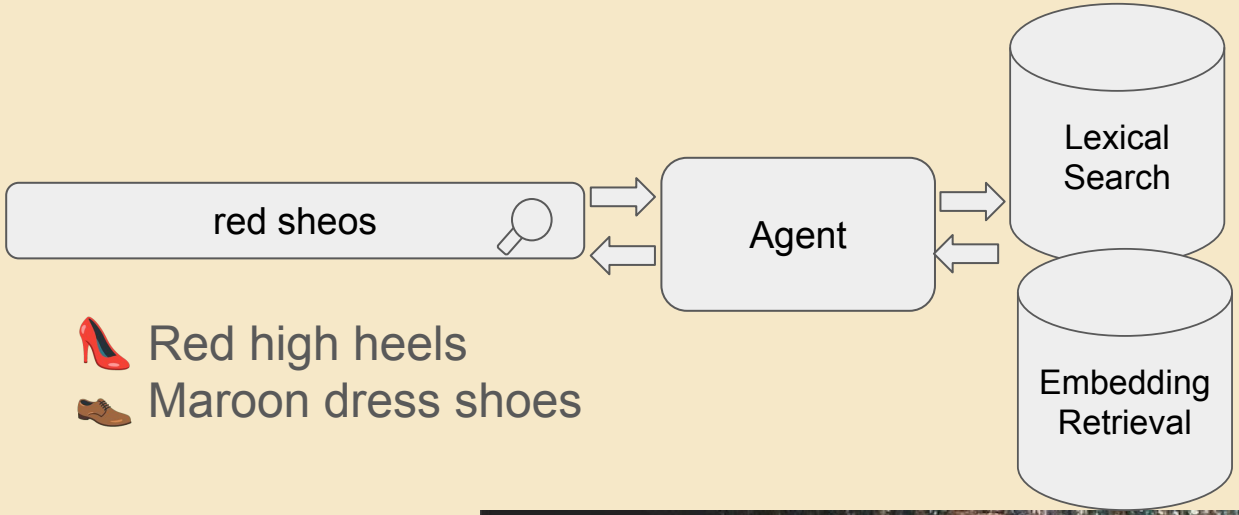
browsecomp-plus benchmark

Using weak model (gpt-4-oss)

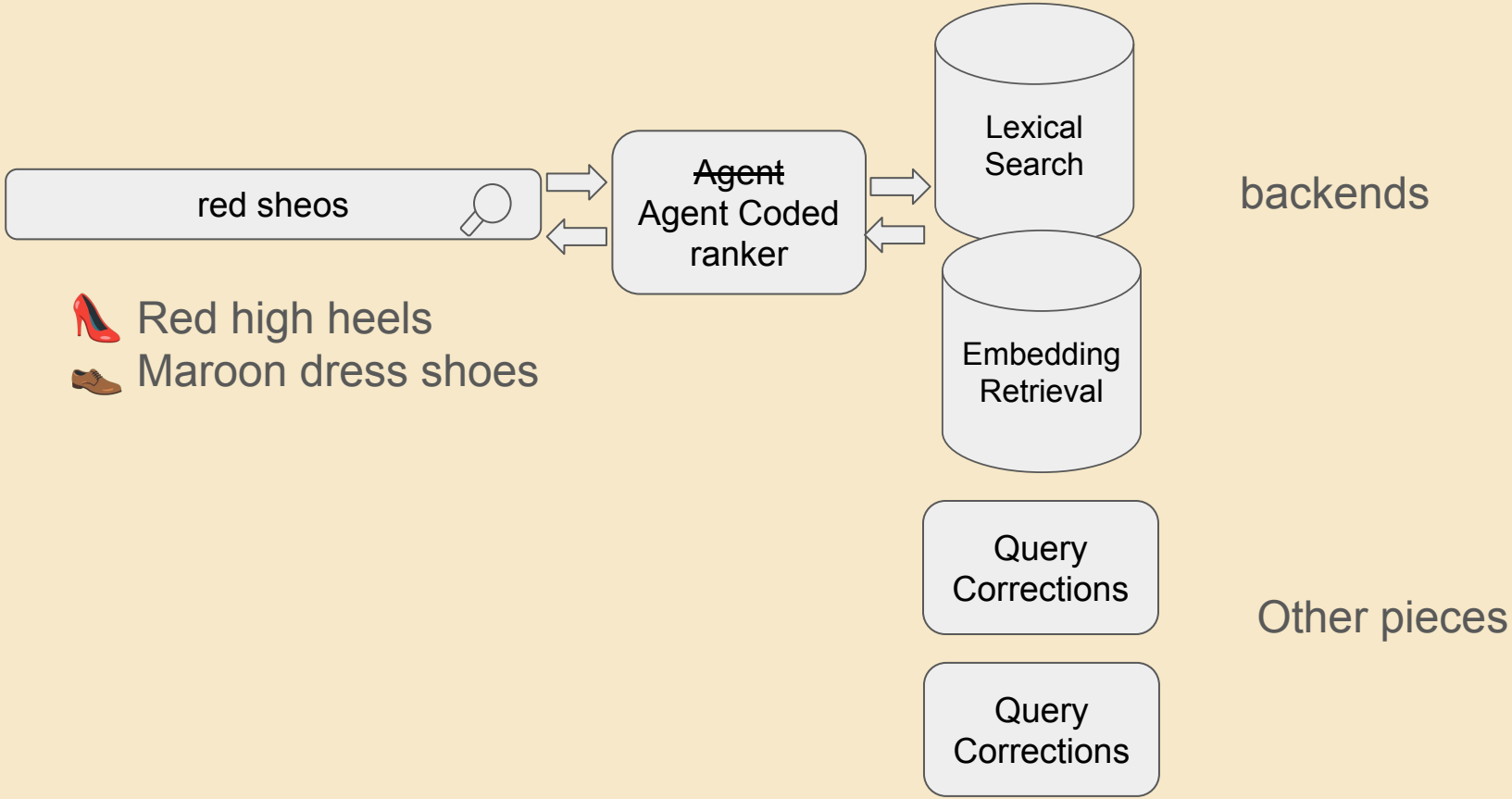
Revisiting Text Ranking in Deep Research
https://arxiv.org/pdf/2602.21456

Autoresearch

Can’t agentify all the search (yet)



Distill lessons into code?



Start with ranking code source

Inject retrieval primitives (ie BM25, etc)
call these “search tools”

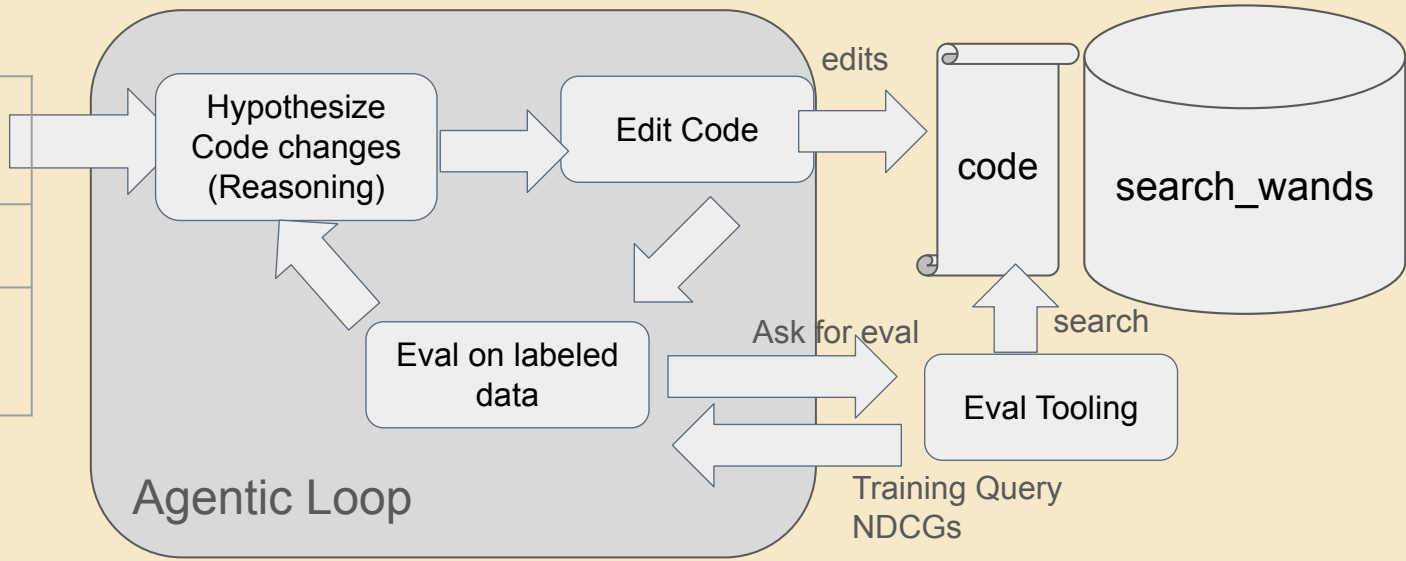
```
original_source = """
def rerank_wands(query, fielded_bm25, search_embeddings,
**kwargs):
    docs = fielded_bm25(query, fields=['title^9.3',
                        'description^4.1'],
                        operator='or', top_k=10)
    return [str(doc['id']) for doc in docs]

"""

with open("rerank_wands.py", "w") as f:
   f.write(original_source)
```

Some starter code

Searching agent -> code agent

Training set

Queries	Documents	Rel?
red shoe	1234	👍
blue shoe	5678	👎



Build your own coding tool

Tool for accepting changes:

```
def apply_patch(edit: Edit) -> EditResult:
   """Save the proposed code change to rerank_esci.py."""
   ...
```

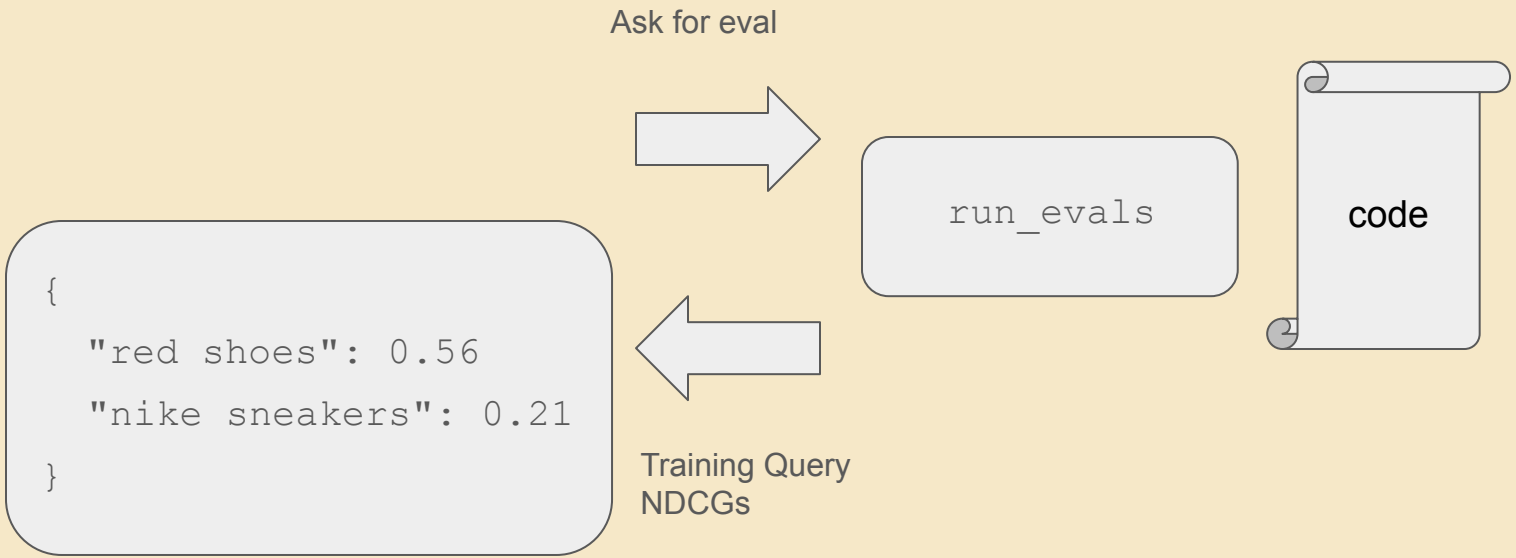
Build your own coding tool

```
class Edit(BaseModel):
    """A single edit to apply to the reranker code."""
    anchor: str = Field(
        ...,
        description="The anchor text to identify where the patch should be applied.",
    )
    block_until: str = Field(
        ...,
        description="The end of the block of text which the patch should be applied. Do not
leave blank.",
    )
    action: Literal["insert_after", "replace", "delete"] = Field(
        ..., description="The action to perform: insert_after, replace, or delete."
    )
```

Build your own coding tool

```
def apply_patch(edit: Edit) -> EditResult:
    """Save the proposed code change to rerank_esci.py."""
    block_index = code.find(edit.block_until, anchor_index)

    if edit.action == "insert_after":
        insertion_point = block_index + len(edit.block_until)
        code = (
            code[:insertion_point] + "\n" + edit.text + "\n" + code[insertion_point:]
        )
    elif edit.action == "replace":
        ...
    elif edit.action == "delete":
        ...
```

Give agent visibility into impact

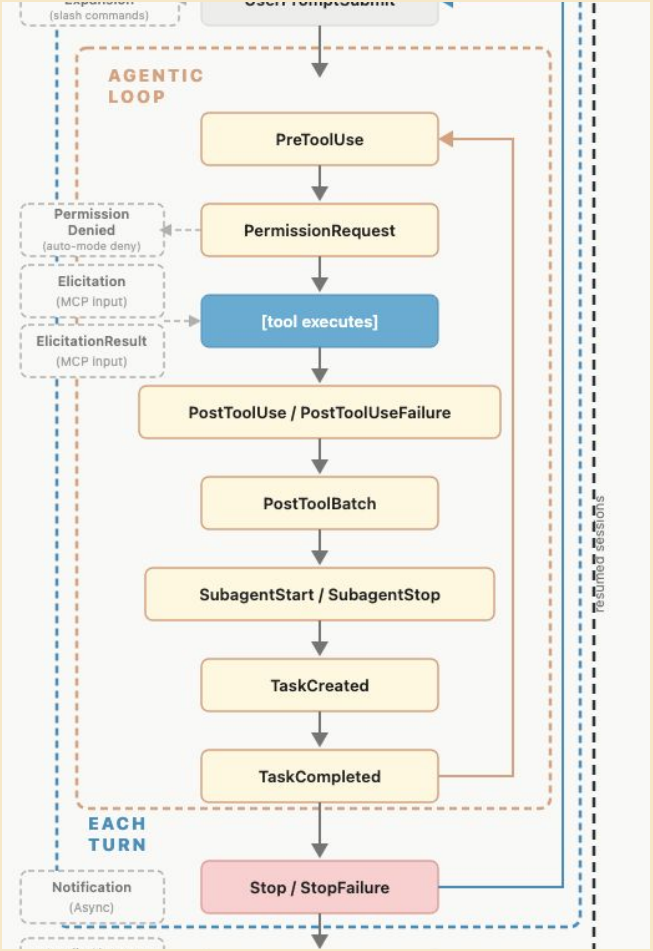


“Training Data” just means “Agent can see these queries”

Why not just /goal?

- I want tight, programmatic control over accepting changes
- Easy reproducibility with few dependencies
- It’s not that hard, this is simple
- I’m already building harnesses for other things

(But yes you can do this with Claude Code + Hooks)



All the code editing + eval tools

```
tools = [# -------
        # CODE EDITING tools
        apply_patch,      # Edit the reranker with a patch
        revert_changes,   # Restore the reranker to the last version

        # -------
        # INSPECT + EVAL current code
        search_wands,   # The raw search tool (from earlier)
        run_reranker,   # Run on one query (optionally label results)
        run_evals,      # Run on training set, getting per-query NDCG / mean]
```

Editing

All the code editing + eval tools

```
tools = [# -------
        # CODE EDITING tools
        apply_patch,     # Edit the reranker with a patch
        revert_changes,  # Restore the reranker to the last version

        # -------
        # INSPECT + EVAL current code
        search_wands,  # The raw search tool (from earlier)
        run_reranker,  # Run on one query (optionally label results)
        run_evals,     # Run on training set, getting per-query NDCG / mean]
```

Run evals

Use the search primitives directly

Now I can just configure what I need

```yaml
strategy:
  name: codegen_no_guards
  type: codegen
  params:
    train:
      model: gpt-5
      refresh_every: 5
      search_tools:
        - fielded_bm25
        - e5_base_v2
      edit:
        guards: []
      eval:
        train_query_fraction: 0.20
        training_seed: 5678
        validation_seed: 1234
        eval_margin: 0.003
      system_prompt: |
        Your task is to improve the reranker code so it returns more relevant results.

        Use apply_patch to edit the reranker module.
        Use run_reranker to inspect single queries and run_evals for NDCG.
        If NDCG does not improve, revert with revert_changes.

        DO NOT rename the function in the code. You MUST keep the signature the same.

    run:
      top_k: 10
```

Setup task with config

```
strategy:
 name: codegen_no_guards
 type: codegen
 params:
   train:
     model: gpt-5
     search_tools:
       - fielded_bm25
       - e5_base_v2
   ...
```

Primitives available to the code generation

- **fielded_bm25** - does a BM25 search on specified field
- **e5_base_v2 -** embedding model

System prompt

```
...
system_prompt: |
      Your task is to improve the reranker code so it returns more
relevant results.

      Use apply_patch to edit the reranker module.
      Use run_reranker to inspect single queries and run_evals for
NDCG.
      If NDCG does not improve, revert with revert_changes.

      DO NOT rename the function in the code. You MUST keep the
signature the same.
```

With system prompt

```
inputs = [
   {"role": "system", "content": "Your task is to improve the reranker
code so it returns more relevant results….

...
Here’s the current code:

def rerank_wands(query, fielded_bm25, e5_base_v2, ...):
   docs = fielded_bm25(keywords=query,
                       field_to_search='product_name',
                       operator='and',
                       top_k=10)
   return [doc['id'] for doc in docs]

"},
```

This is just:

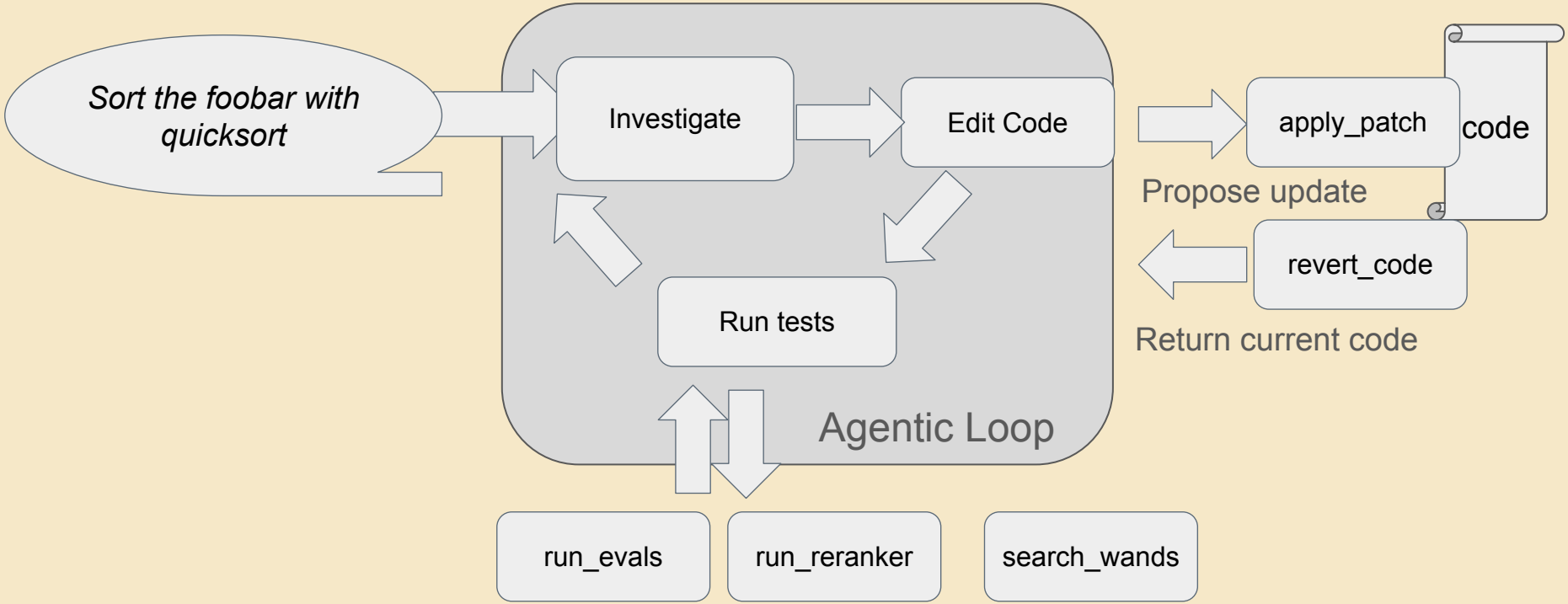
```
tool_calls = True
while tool_calls:
  tool_calls = False

  resp = openai.responses.create(
      model="gpt-5-mini",
      input=inputs,
      tools=... (openai tool spec) ...,
  )
  inputs += resp.output

  for func_output in resp.output:
     if func_output.type == "function_call":
         ... call eval + editing tools ...
```

(Pretty much same agentic loop…

…
But prompts asking agent to edit code + run evals)

Or in pretty picture form

Added:

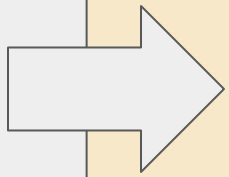


After one run

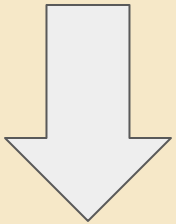
```
def rerank_wands(query, fielded_bm25, search_embeddings, **kwargs):
    """
    Hybrid reranker for the wands dataset.

    Strategy:
    - Gather a broad candidate set using multiple retrieval modes:
      * BM25 (OR) for recall
      * BM25 (PHRASE) to capture exact-phrase matches
      * BM25 (AND) for precise multi-term intent
      * Embedding search for semantic recall
    - Compute lightweight features per candidate (normalized scores, token overlap,
      title boosts, and presence of product-category head terms).
    - Combine with a weighted score and return top results by final score.
    """

  ...
```

Start again w/ this code

```
def rerank_wands(query, fielded_bm25, search_embeddings, **kwargs):
    """
    Hybrid reranker for the wands dataset.

    Strategy:
    - Gather a broad candidate set using multiple retrieval modes:
      * BM25 (OR) for recall
      * BM25 (PHRASE) to capture exact-phrase matches
      * BM25 (AND) for precise multi-term intent
      * Embedding search for semantic recall
    - Compute lightweight features per candidate (normalized scores, token overlap,
      title boosts, and presence of product-category head terms).
    - Combine with a weighted score and return top results by final score.
    """

  ...
```



Autoresearch run



Even better ranker

Run it... over multiple rounds

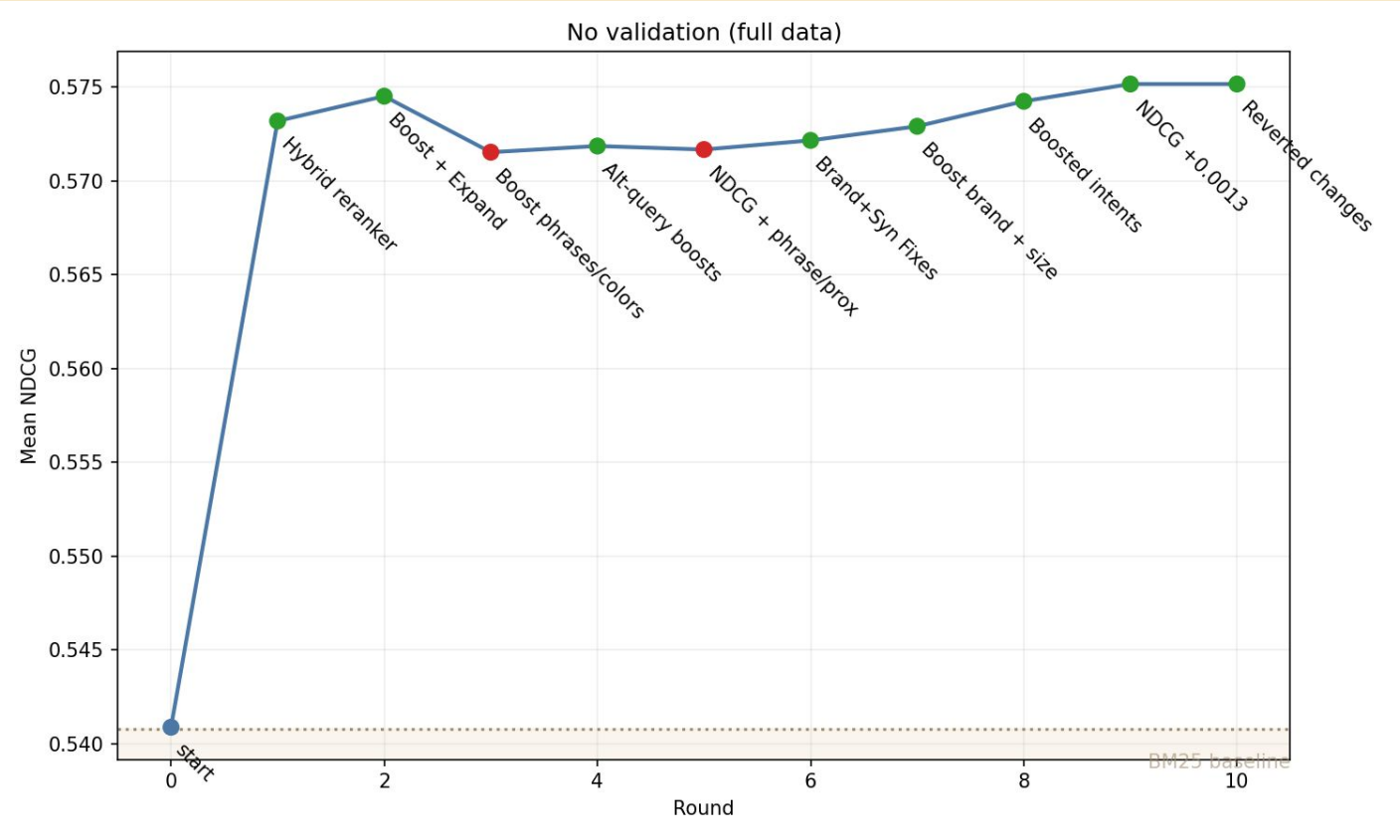
```
uv run train \
    --strategy configs/codegen/codegen_no_guards.yml  \
    --dataset wands \
    --rounds 10
```

Yml file controlling agentic proce

Judgments + corpus from here

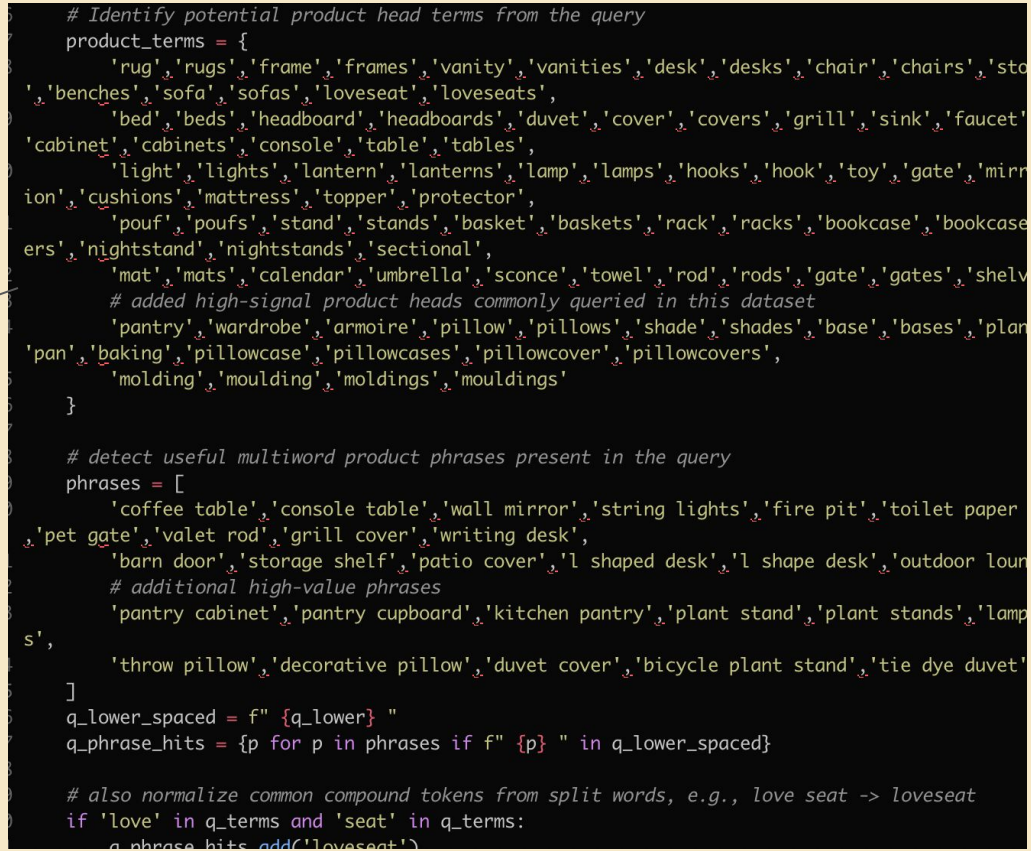
Restart fresh round w/ output of last

The results


No validation (full data)
Mean NDCG
Round
0.575
0.570
0.565
0.560
0.555
0.550
0.545
0.540
0
2
4
6
8
10
start
Hybrid reranker
Boost + Expand
Boost phrases/colors
Alt-query boosts
NDCG + phrase/prox
Brand+Syn Fixes
Boost brand + size
Boosted intents
NDCG +0.0013
Reverted changes
BM25 baseline

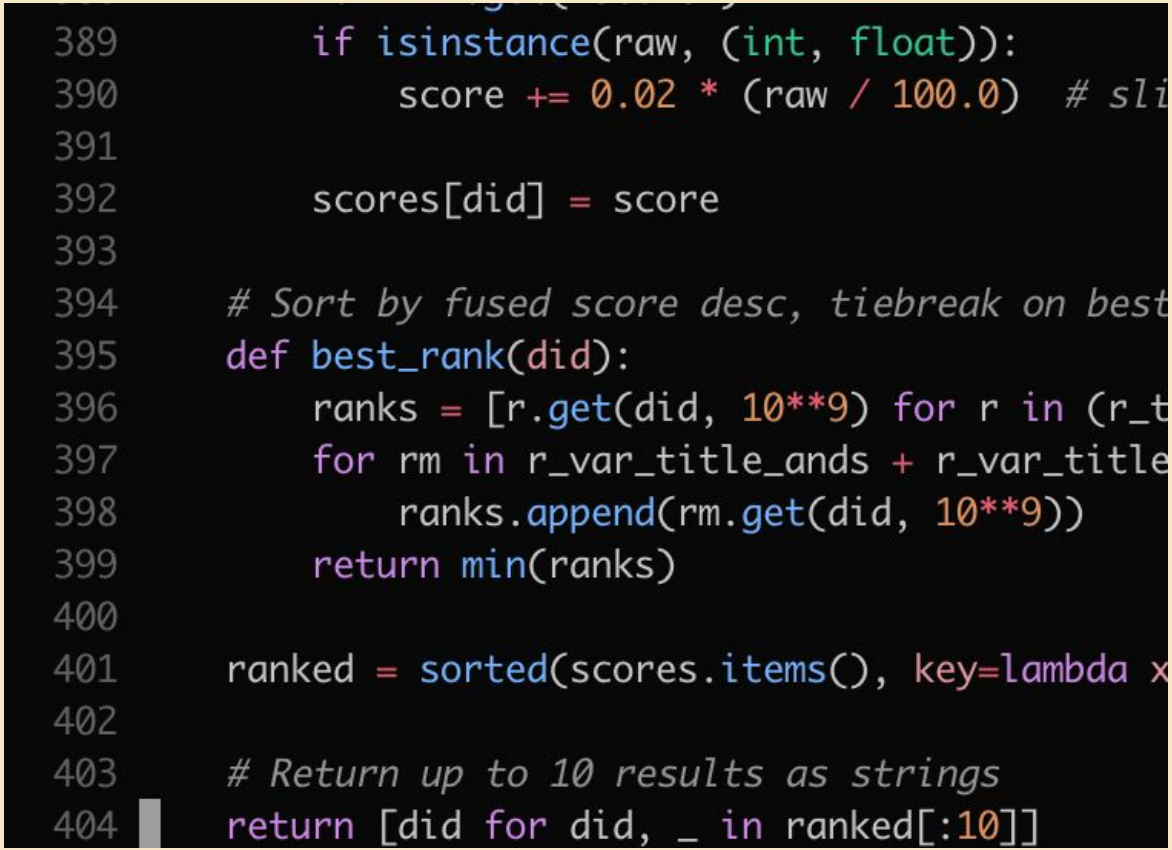
Code looks sus

Holy overfit batman!



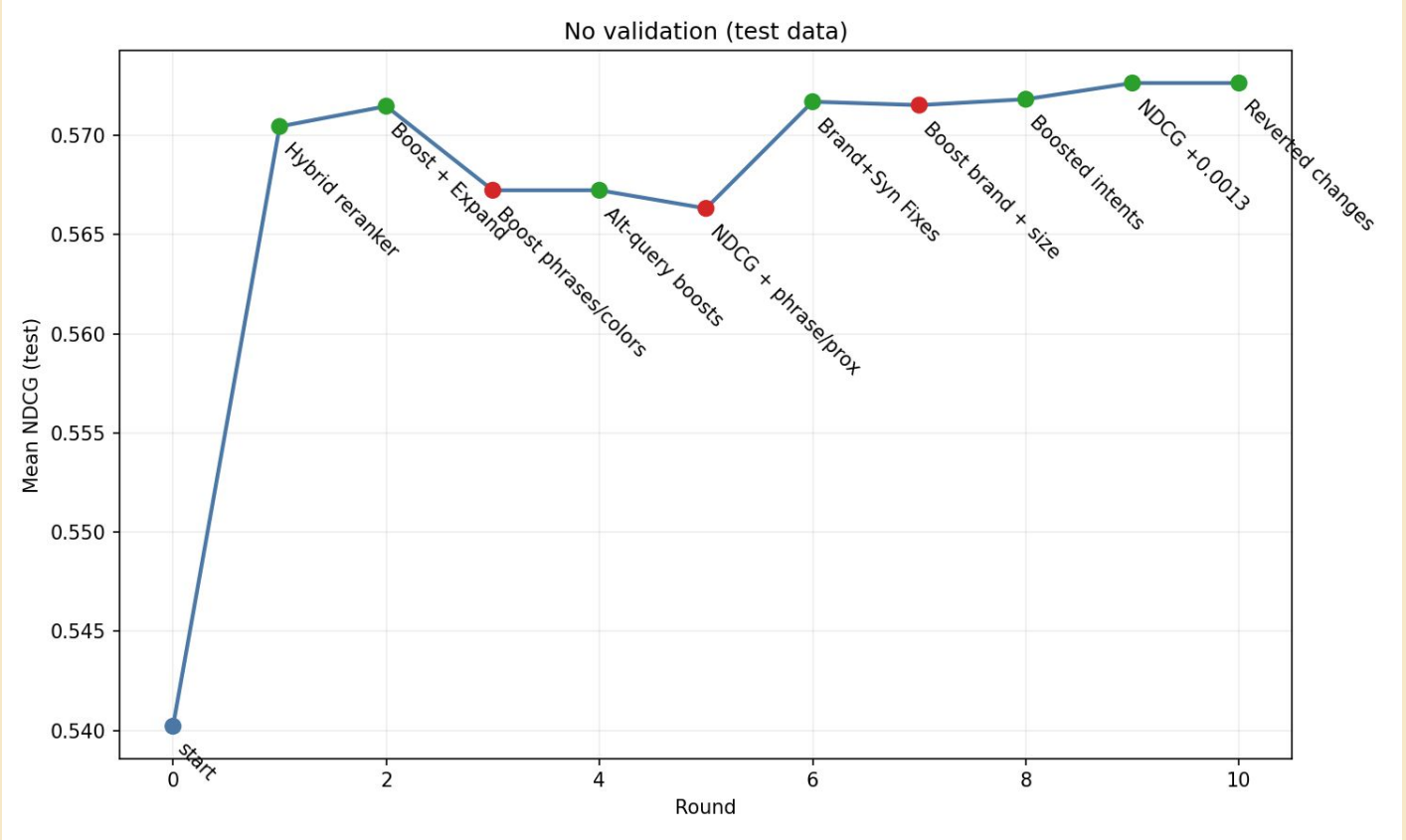
```
    # Identify potential product head terms from the query
    product_terms = {
        'rug','rugs','frame','frames','vanity','vanities','desk','desks','chair','chairs','sto
','benches','sofa','sofas','loveseat','loveseats',
        'bed','beds','headboard','headboards','duvet','cover','covers','grill','sink','faucet'
'cabinet','cabinets','console','table','tables',
        'light','lights','lantern','lanterns','lamp','lamps','hooks','hook','toy','gate','mirr
ion','cushions','mattress','topper','protector',
        'pouf','poufs','stand','stands','basket','baskets','rack','racks','bookcase','bookcase
ers','nightstand','nightstands','sectional',
        'mat','mats','calendar','umbrella','sconce','towel','rod','rods','gate','gates','shelv
        # added high-signal product heads commonly queried in this dataset
        'pantry','wardrobe','armoire','pillow','pillows','shade','shades','base','bases','plan
'pan','baking','pillowcase','pillowcases','pillowcover','pillowcovers',
        'molding','moulding','moldings','mouldings'
    }

    # detect useful multiword product phrases present in the query
    phrases = [
        'coffee table','console table','wall mirror','string lights','fire pit','toilet paper
','pet gate','valet rod','grill cover','writing desk',
        'barn door','storage shelf','patio cover','l shaped desk','l shape desk','outdoor loun
        # additional high-value phrases
        'pantry cabinet','pantry cupboard','kitchen pantry','plant stand','plant stands','lamp
s',
        'throw pillow','decorative pillow','duvet cover','bicycle plant stand','tie dye duvet'
    ]
    q_lower_spaced = f" {q_lower} "
    q_phrase_hits = {p for p in phrases if f" {p} " in q_lower_spaced}

    # also normalize common compound tokens from split words, e.g., love seat -> loveseat
    if 'love' in q_terms and 'seat' in q_terms:
        q_phrase_hits.add('loveseat')
```

The other problem: size



```
389         if isinstance(raw, (int, float)):
390             score += 0.02 * (raw / 100.0)  # sli
391
392         scores[did] = score
393
394     # Sort by fused score desc, tiebreak on best
395     def best_rank(did):
396         ranks = [r.get(did, 10**9) for r in (r_t
397         for rm in r_var_title_ands + r_var_title
398             ranks.append(rm.get(did, 10**9))
399         return min(ranks)
400
401     ranked = sorted(scores.items(), key=lambda x
402
403     # Return up to 10 results as strings
404     return [did for did, _ in ranked[:10]]
```

Non training data tells messy story…



Slightly less bad Autoresearch

Add overfit guard before accepting change

```yaml
strategy:
 name: codegen_guarded
 type: codegen
 params:
   train:
     model: gpt-5
     search_tools:
       - fielded_bm25
       - e5_base_v2
     edit:
       guards:
         - validation
         - overfit:
             model: openai/gpt-5-mini
```

1. Force check on validation set before accepting edits
2. Ask another LLM if these “look overfit” per our requirements

Reject code failing check

```
def apply_patch(edit: Edit) -> EditResult:
    """Save the proposed code change to rerank_esci.py."""

    # Validate code
    for guardrail in guardrail_fns:
        error_message = guardrail(edit.text)
        if error_message is not None:
            raise ValueError(error_message)

    block_index = code.find(edit.block_until, anchor_index)

    if edit.action == "insert_after":
        ...
```

Run checks.

Reject code that fails

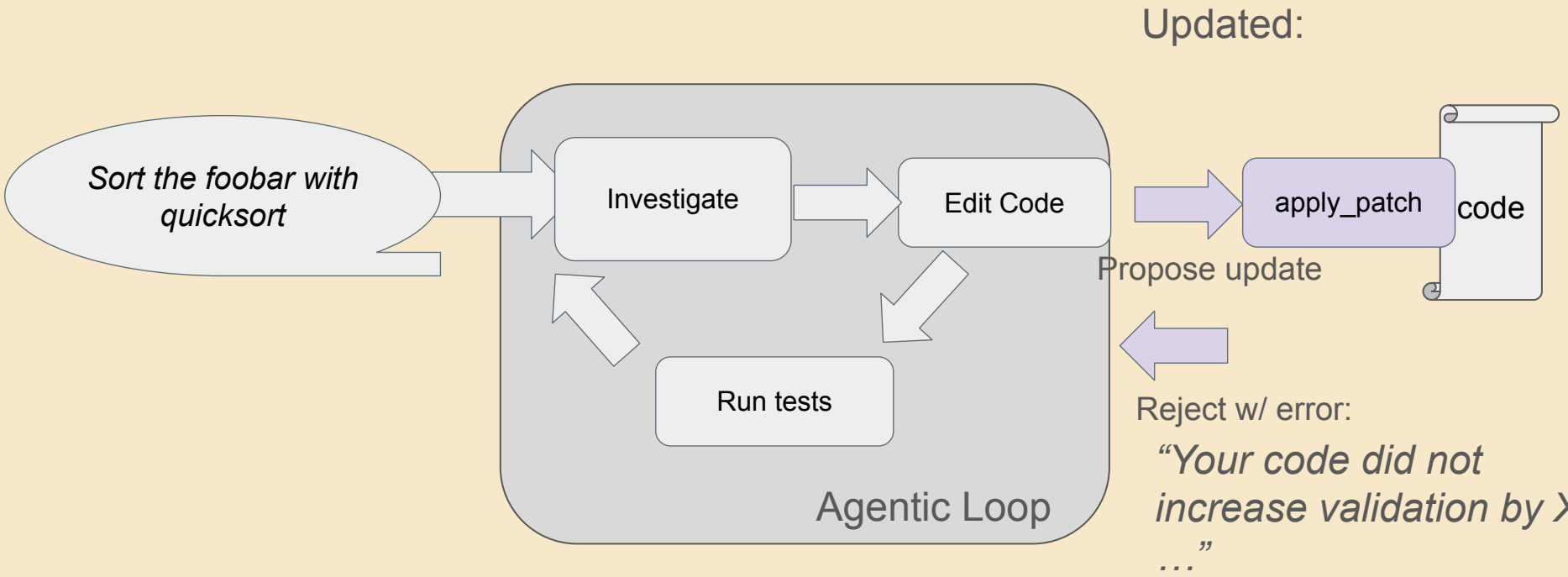
Shrink the decision space

```yaml
strategy:
 name: codegen_guarded
 type: codegen
 params:
   train:
     model: gpt-5
     search_tools:
       - fielded_bm25
       - e5_base_v2
     edit:
       guards:
         - length:
             max_lines: 10
             max_cols: 120
```

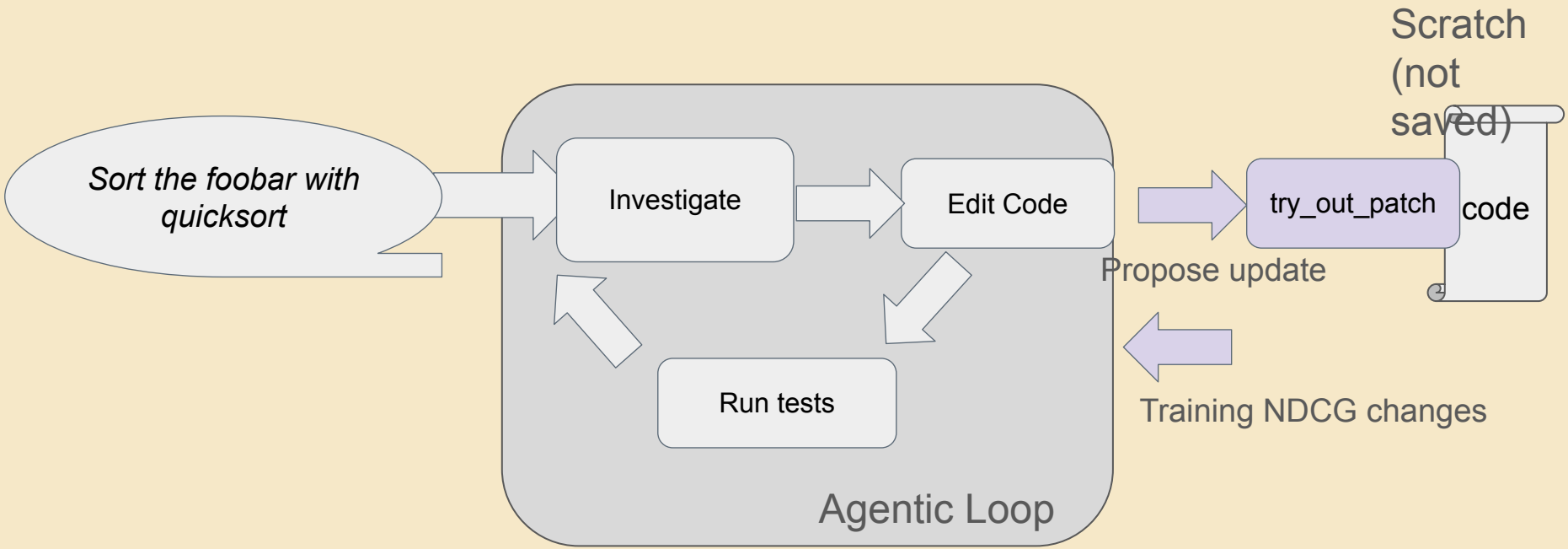
Disallow edits over a certain size

- More targeted / incremental changes
- Cause + effect in reasoning

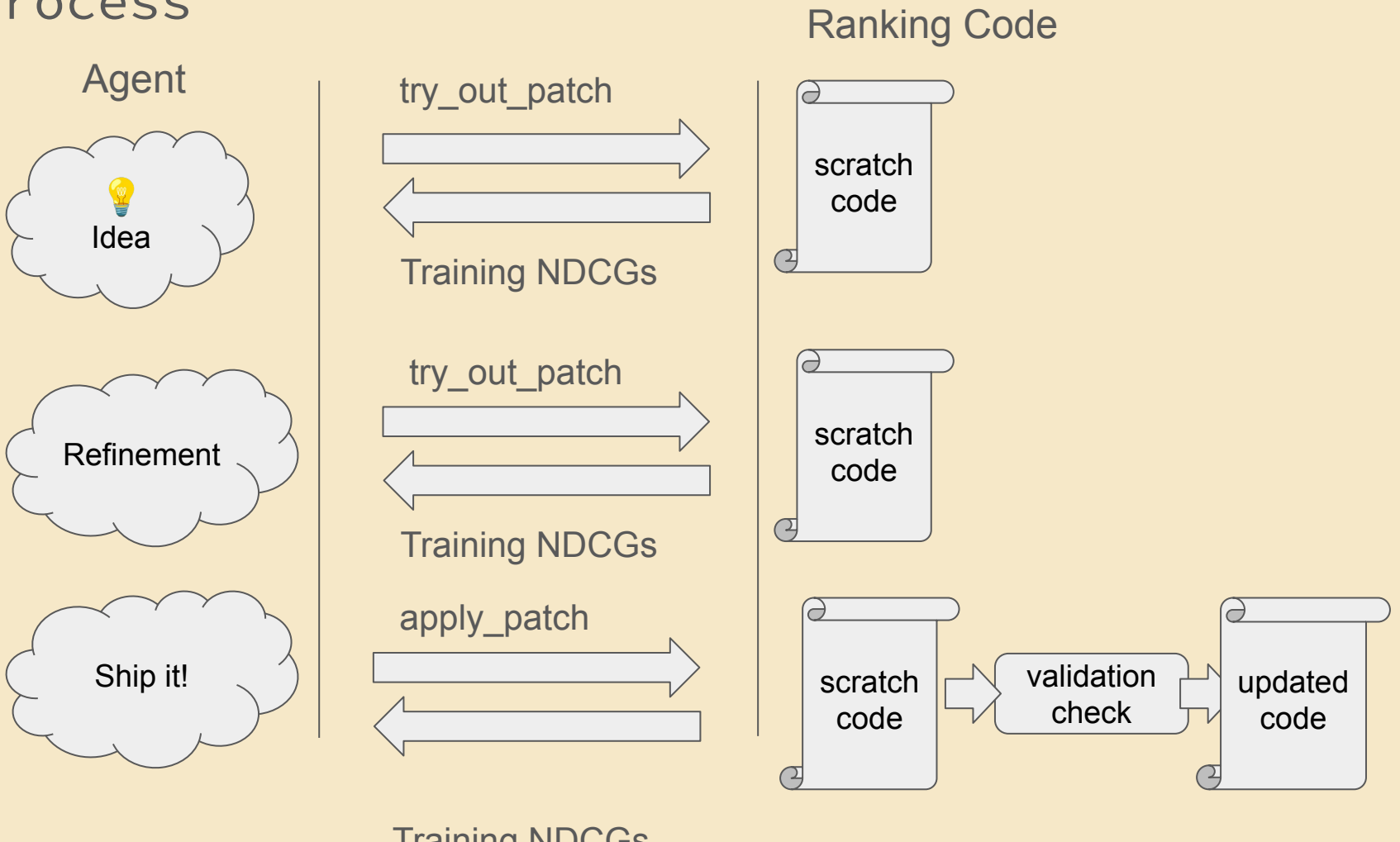
Autoresearch w/ validation



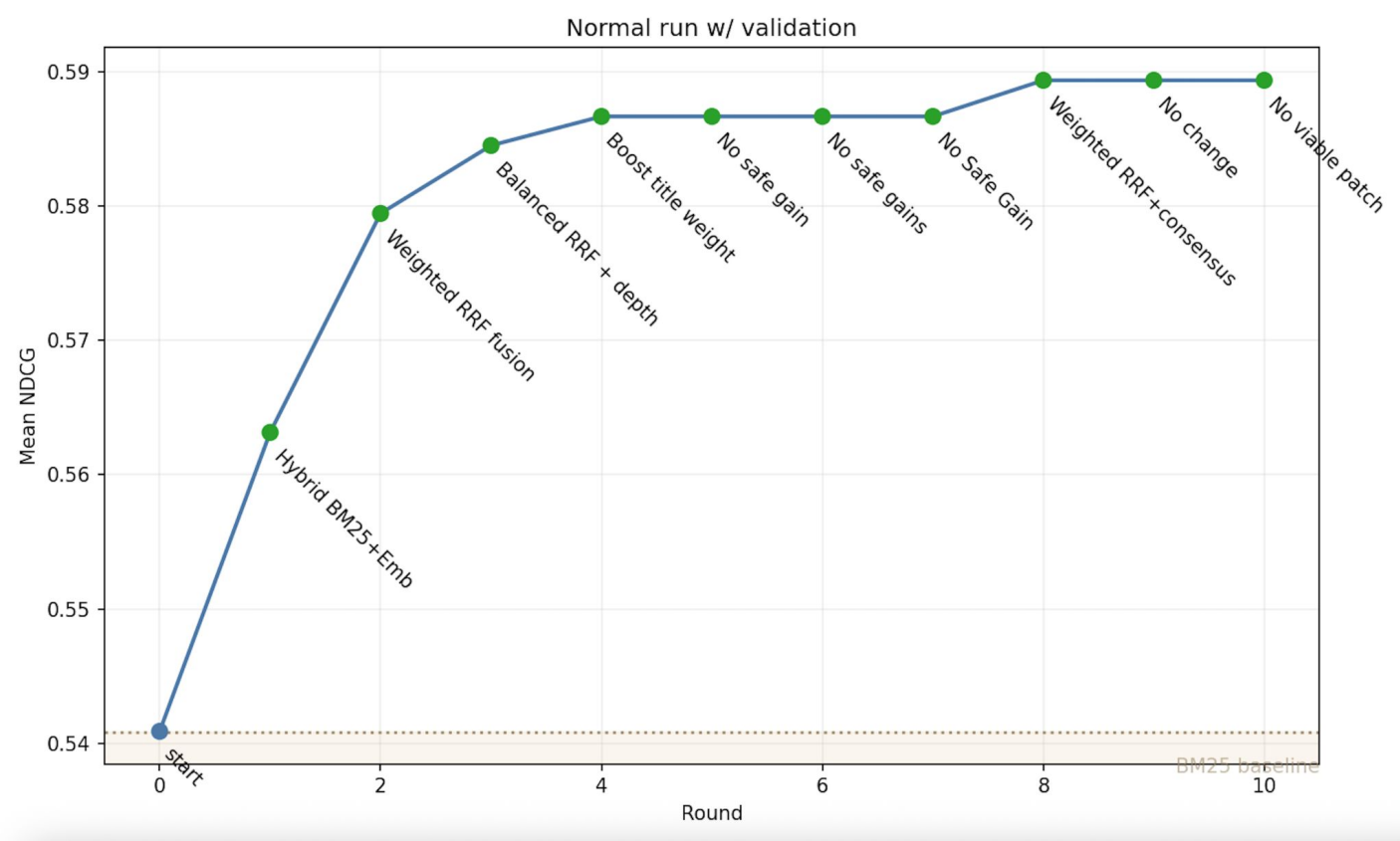
Add another tool - ‘try out’ patch



New process

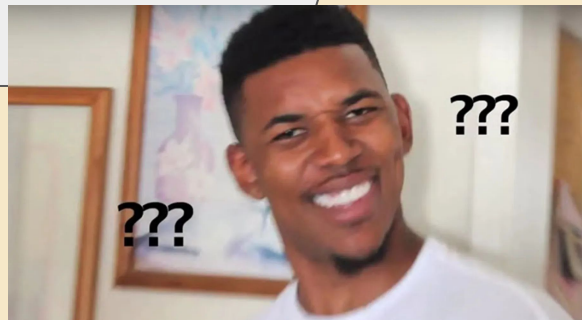

Agent
Ranking Code
Idea
try_out_patch
Training NDCGs
scratch code
Refinement
try_out_patch
Training NDCGs
scratch code
Ship it!
apply_patch
scratch code
validation check
updated code
Training NDCGs

Ah better. Higher plateau ~0.59



Final code

```
def rerank_wands(query, fielded_bm25, search_embeddings, **kwargs):
   b = fielded_bm25(query, fields=['title^10.5','description^3.9'], operator='or',
top_k=95, k1=1.2, b=0.6)
   e = search_embeddings(query, top_k=40)
   s = {}; k = 50.0
   for i,d in enumerate(b): s[str(d['id'])] = s.get(str(d['id']),0.0)+0.45/(k+i+1.0)
   for i,d in enumerate(e): s[str(d['id'])] = s.get(str(d['id']),0.0)+0.55/(k+i+1.0)
   ib = {str(d['id']) for d in b}; ie = {str(d['id']) for d in e}
   for i in ib & ie: s[i] = s.get(i,0.0)+0.01
   return [k for k,_ in sorted(s.items(), key=lambda x:x[1], reverse=True)][:10]
```



As explained by ChatGPT

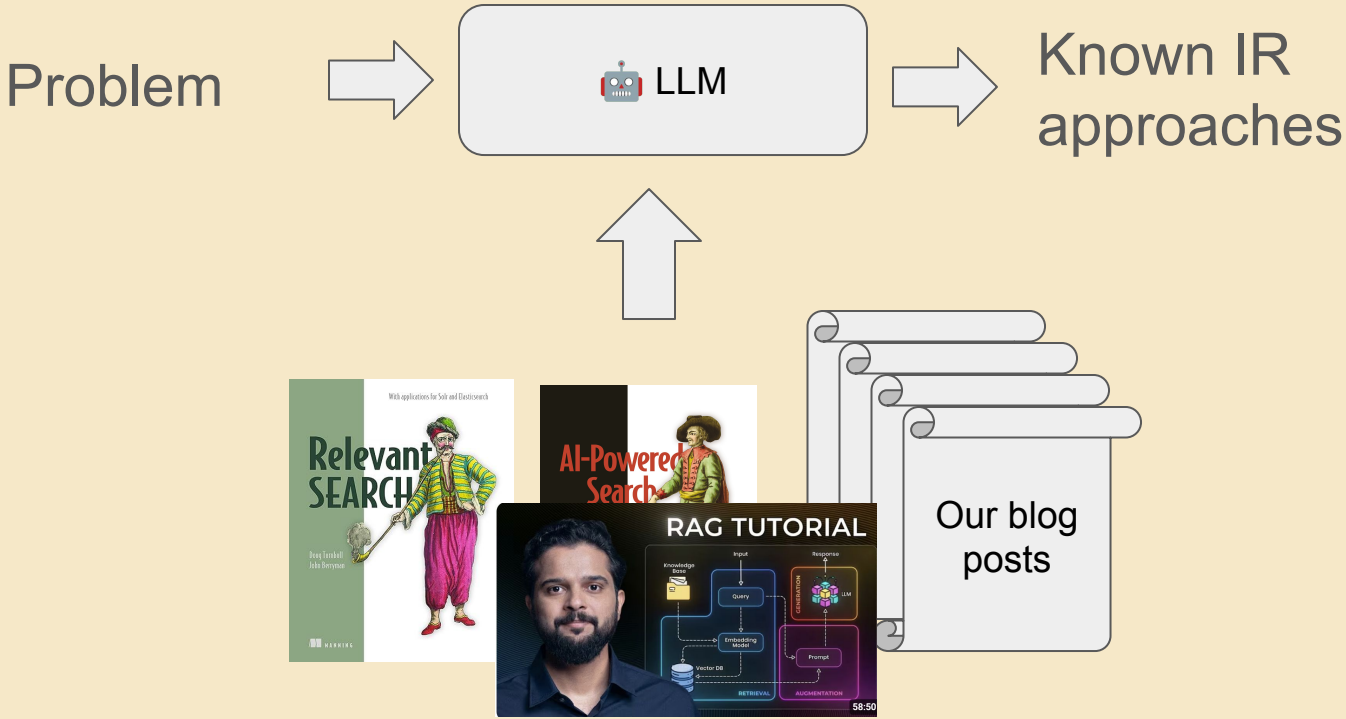
Step 1: Run fielded BM25 search. (modified b, stronger title)

Step 2: Embedding search

Step 3: Weighed RRF

Step 4: Bonus for overlapping in both retrievers

LLMs pick unsurprising solutions



Exploring novel solutions

First thought - control ingredients

```
def rerank_wands(query,
                 fielded_bm25,
                 search_embeddings,
                 rewrite_query,
                 get_commute_distance,
                 categorize_queries
                 ...
                 **kwargs):
    …
```

Give agent many prepared primitives?

(~feature engineering)

Or we could give something open-ended

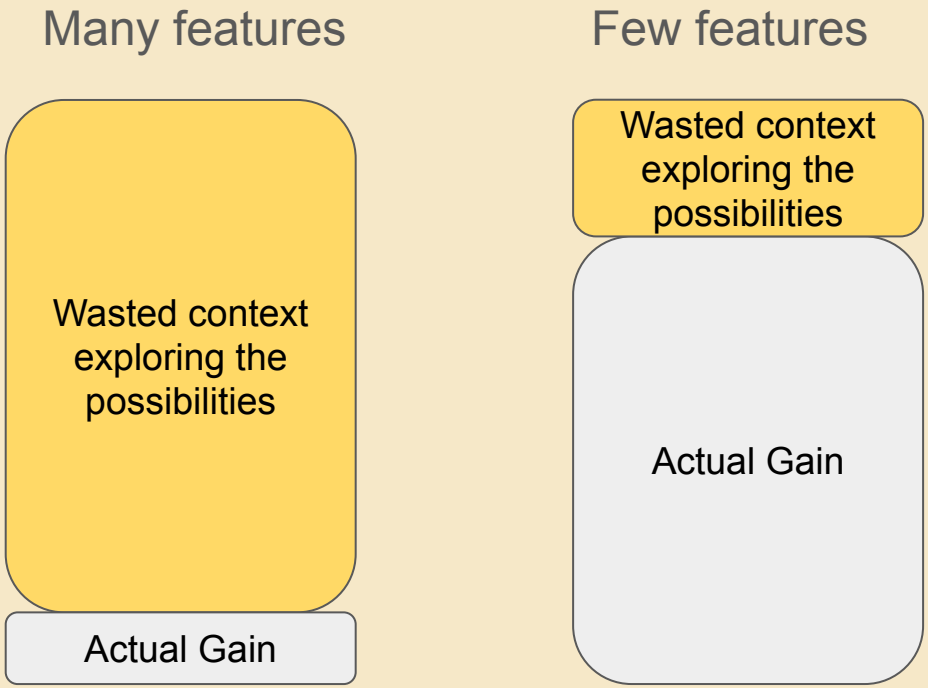
```
def rerank_wands(query,
                 fielded_bm25,
                 search_embeddings,
                 call_llm,
                 **kwargs):
    …
```

Or let agent figure things out from first principles?

~guide with skills to figure it out

Really this is a **curse of dimensionality** problem: explore given limited context

Curse of dimensionality



(Using up context trying all the options)

One approach - focus on part of the problem

```
def rerank_wands(query,
                 search,
                 rewrite_query
                 ...
                 **kwargs):
    …
```

Result from previous run - treated like a black box

New tool being used

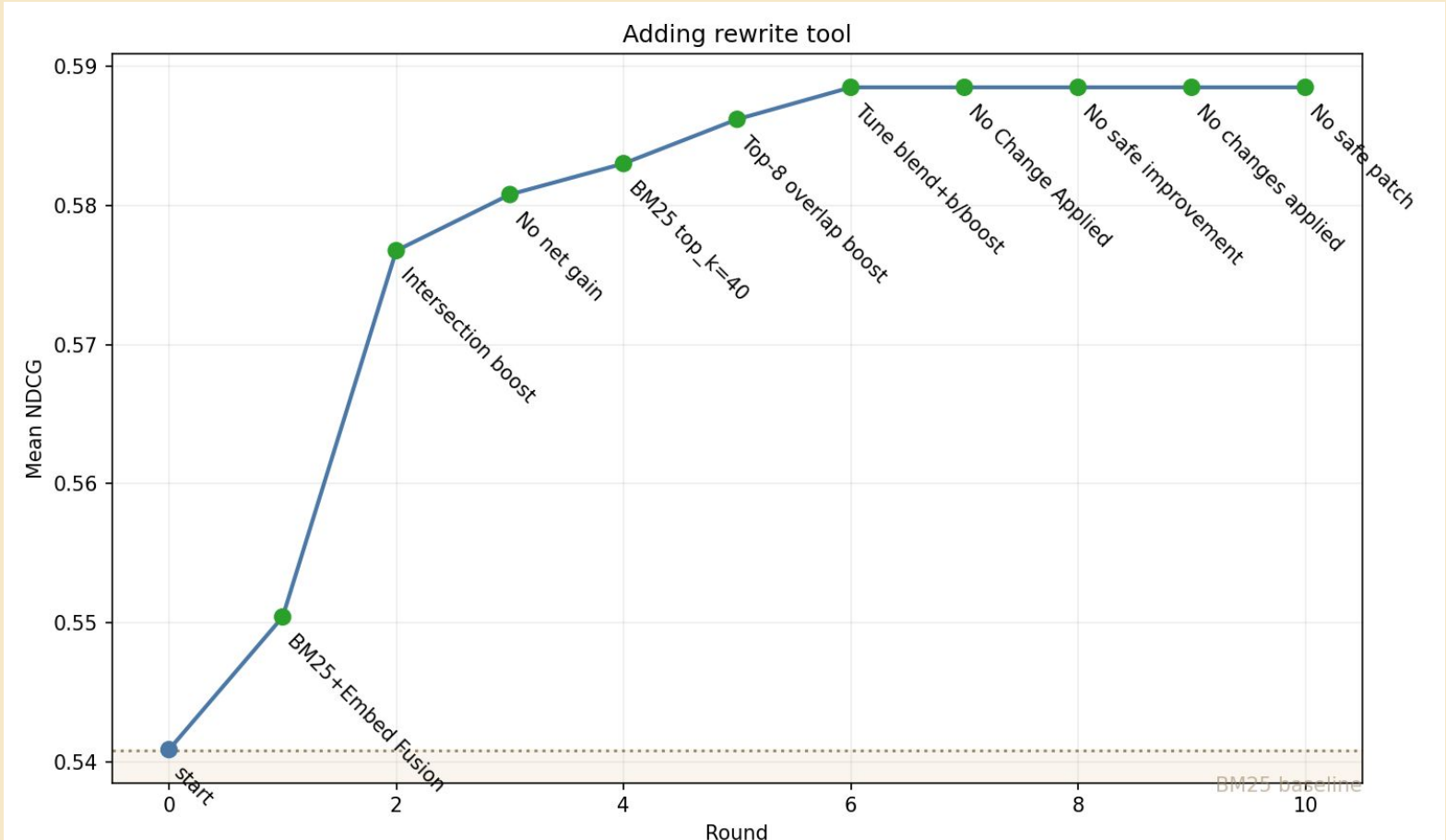
Think in terms of ensemble / hiding

```
search_tools:
  - codegen:
      path: runs/codegen/codegen_guarded/20260502_025238
      name: search
      dependencies:
        - fielded_bm25
        - e5_base_v2
  - query_rewrite:
      model: gpt-5-mini
      max_alternatives: 5
```

Agent just sees “search” doesn’t know details

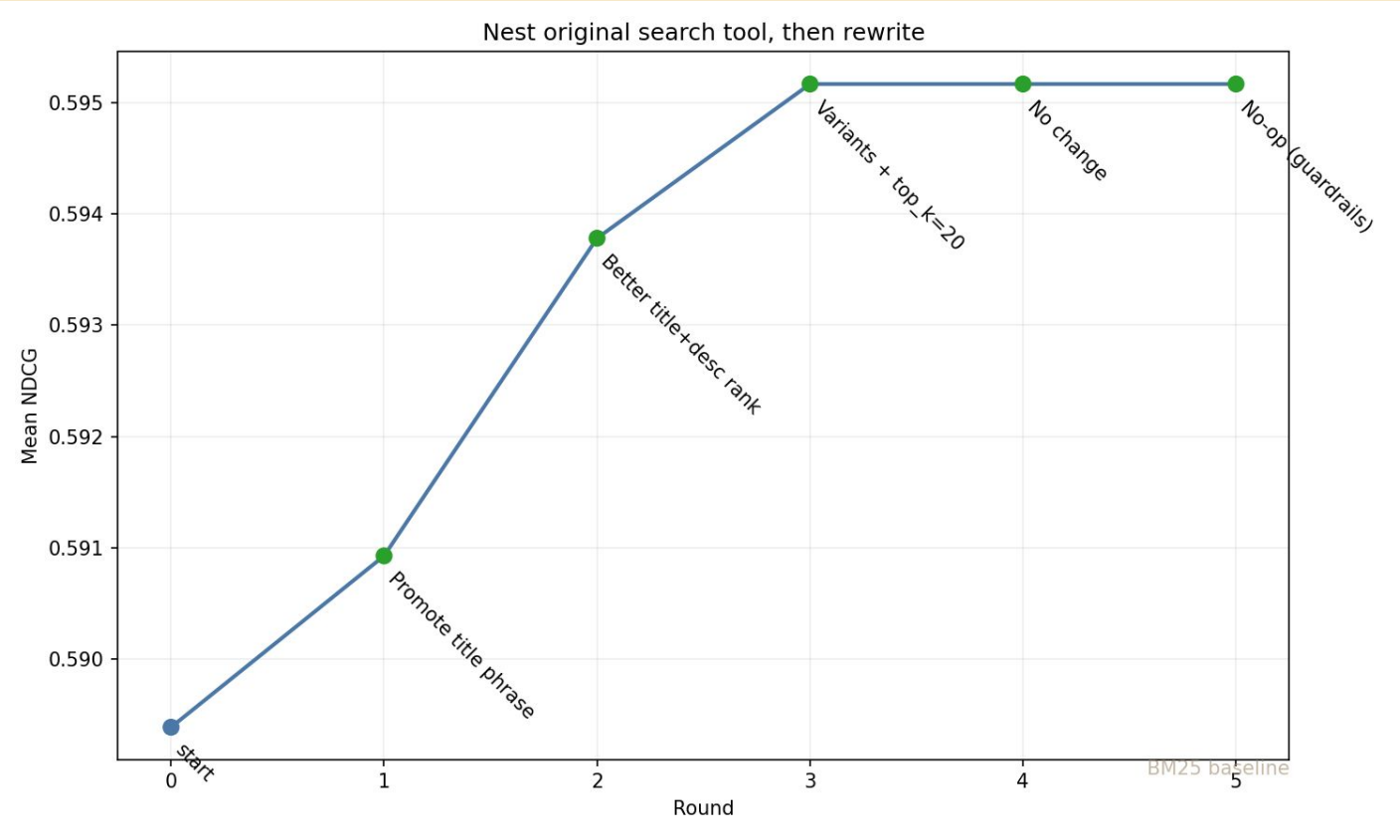
Add “query rewrite”

Train w/ rewrite + two retrieval



~0.586

Hide the retrieval behind “search”

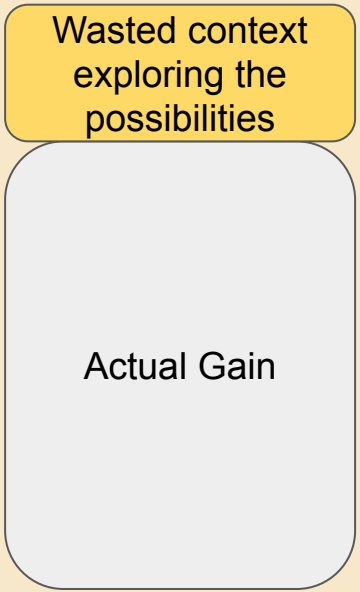


~0.595

Fewer distracting decisions

Giving many options

Fewer options, more gain



(Using up context trying all the options)

What if we give raw ingredients (tf, df, etc)?

Start with BM25 baseline code

Access raw term stats

```
def rerank_minimarco(query, fielded_bm25, get_corpus, **kwargs):
    corpus = get_corpus()
    snowball = corpus["description_snowball"].array
    ...
```

```
for term in terms:
    term_freqs = snowball.termfreqs(term)
    doc_freq = snowball.docfreq(term)
    if doc_freq == 0:
        continue
    idf = np.log(1.0 + (n_docs - doc_freq + 0.5) / (doc_freq + 0.5))
    denom = term_freqs + k1 * (1.0 - b + b * (doc_lengths / avg_dl))
    scores += idf * (term_freqs * (k1 + 1.0)) / np.where(denom == 0, 1.0, denom)
```

Start with BM25?

Try on MSMarco dataset

(Passage ranking, just the text)

Classic BM25 baseline: ~0.189 MRR (anserini)

Question

how much do rear brakes and rotors cost?

Answer

$105 for brake pads (front and rear) and $170 for labor. I was quoted $932 for front and rear breaks and rotors. It was steep, but I needed it done and didn't want to shop it around. When I drove out, my breaks were squeeking worse than when I went in.

I’m so smart. Can I beat BM25?

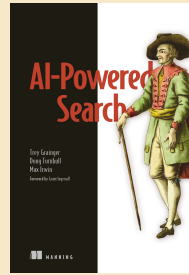
Add human guidance - what to try

System Prompt

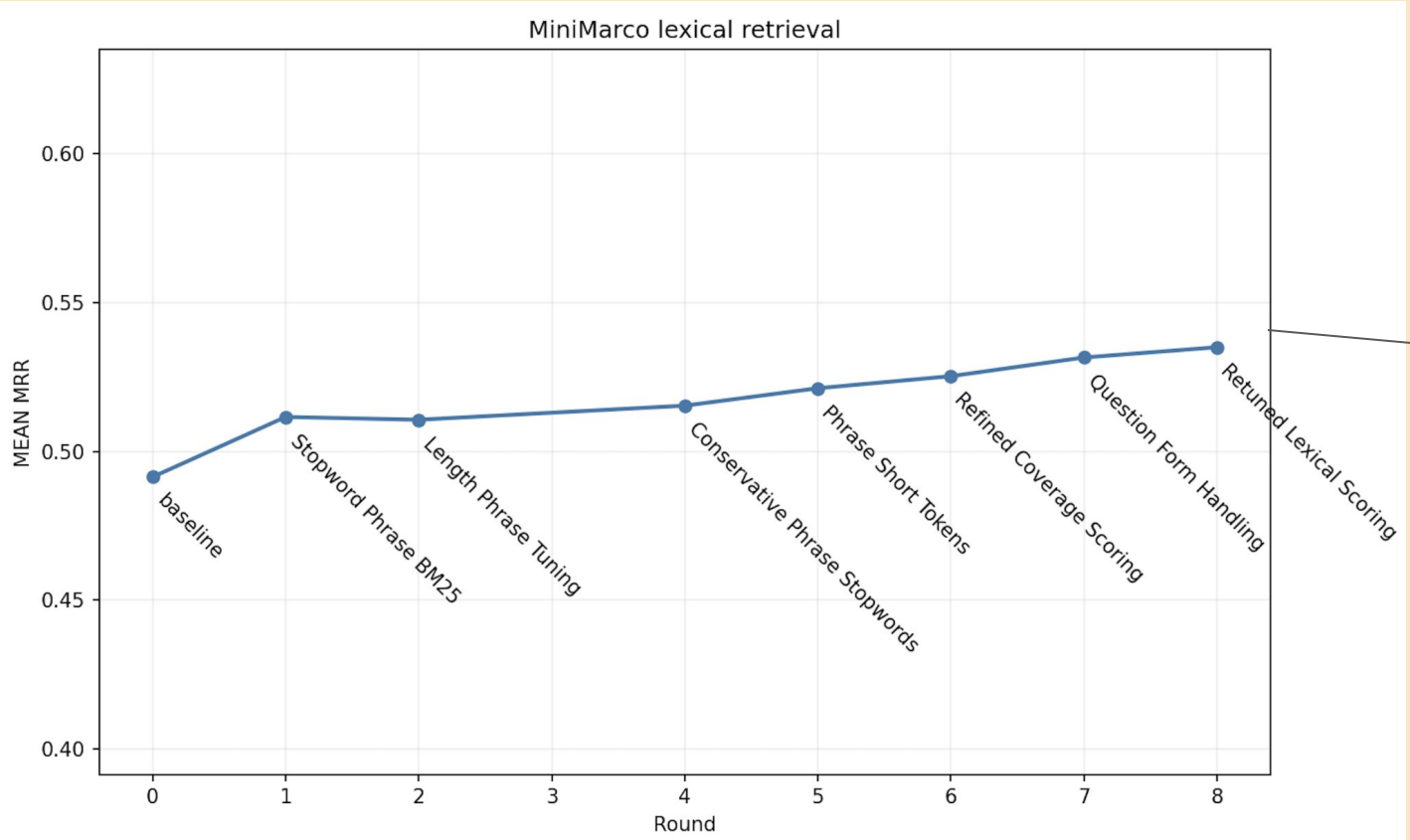
HINTS - what to explore

- Find statistically significant bigrams (colocations) using the phrase term frequency stats
- Use the fielded BM25 scores as a signal in your reranker
- Try reranking on exact phrase matches in the retrieved candidates
- Use term length as another measure of specificity (ie longer terms are more specific user intent)
- Try different stopword removal opproaches (but don't apply it to short queries)
- Look in logs where there was ALMOST a 0.002 improvement and see what the code changes re. Try to understand why they were close but not quite there - maybe you can mitigate the wnsides to get a net improvement?

LLM



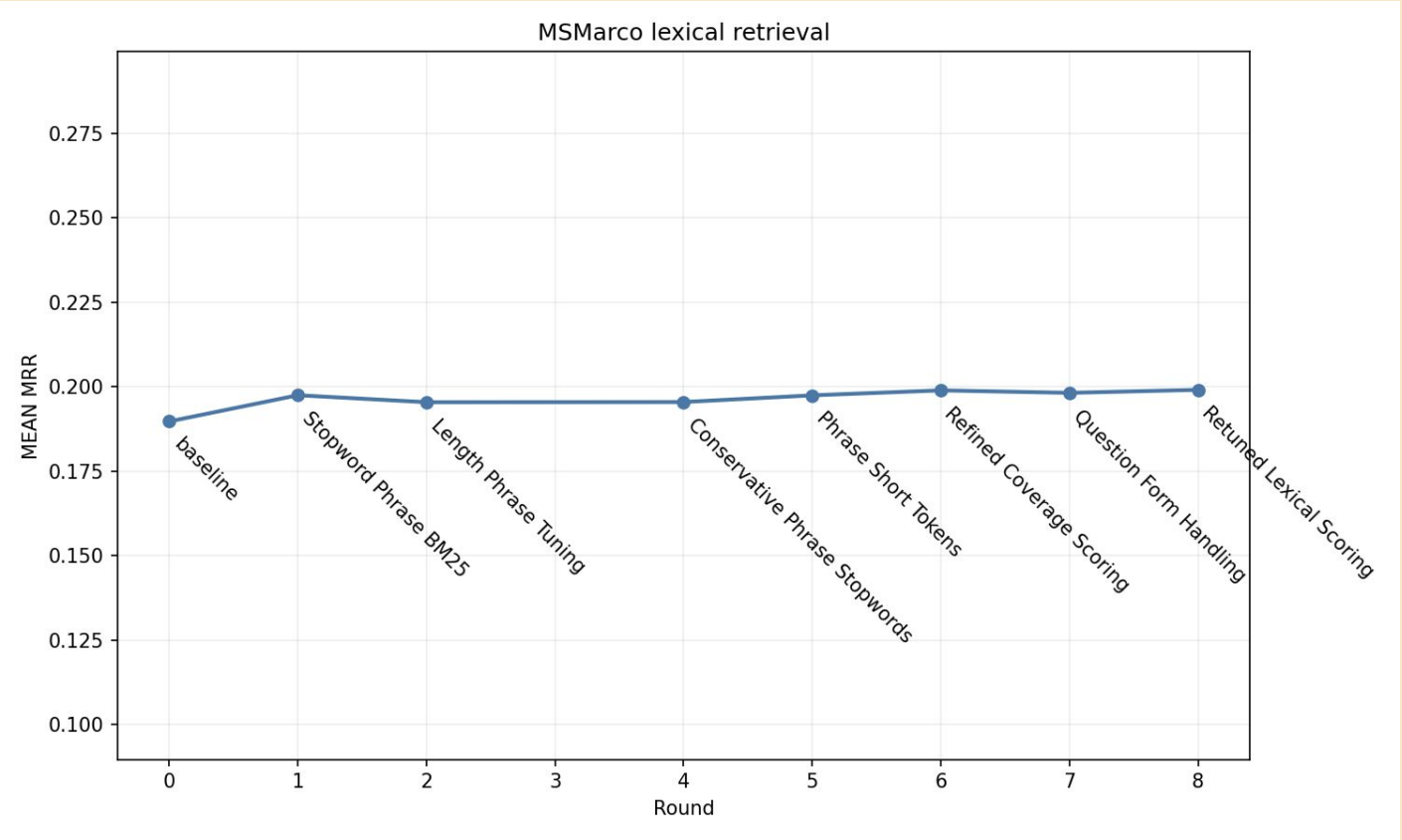
On minimarco (smaller msmarco)



Whoah! It's getting better

(gpt-5.5 xhigh)

On whole MSMarco, suspicious...



Flatten after initial stopword removal

(gpt-5.5 xhigh)

Overfit to validation

```
def rerank_minimarco(query, fielded_bm25, get_corpus, **kwargs):
    import numpy as np; c=get_corpus(); a=c["description_snowball"].array; toks=[t for t in a.tokenizer(que
t]
    sw=set(("what is are was were be as a an the in to for do doe did can you i me there where when who why
            "which consid achiev mani some word need and or on with from that call place medicin vacat").sp
```

(Notice mention of very specific terms in stopwords list)

Important to monitor dataset NOT in validation or training set

Still interesting results

stopword removal -> BM25 -> bigram boost

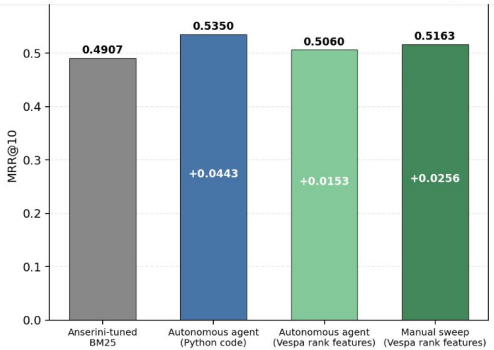
```
# The boost is intentionally small compared with the BM25 score.
bigram_boost_weight = 0.08

bigram_scores = sum(
    (
        bigram_boost_weight
        * description_index.termfreqs(scoring_terms[i : i + 2])
        for i in range(len(scoring_terms) - 1)
    ),
    np.zeros(num_documents),
)

scores += bigram_scores
```

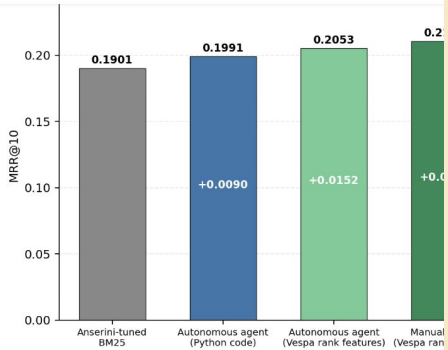
Shoutout to Vespa

Vespa

Search blog posts

Share





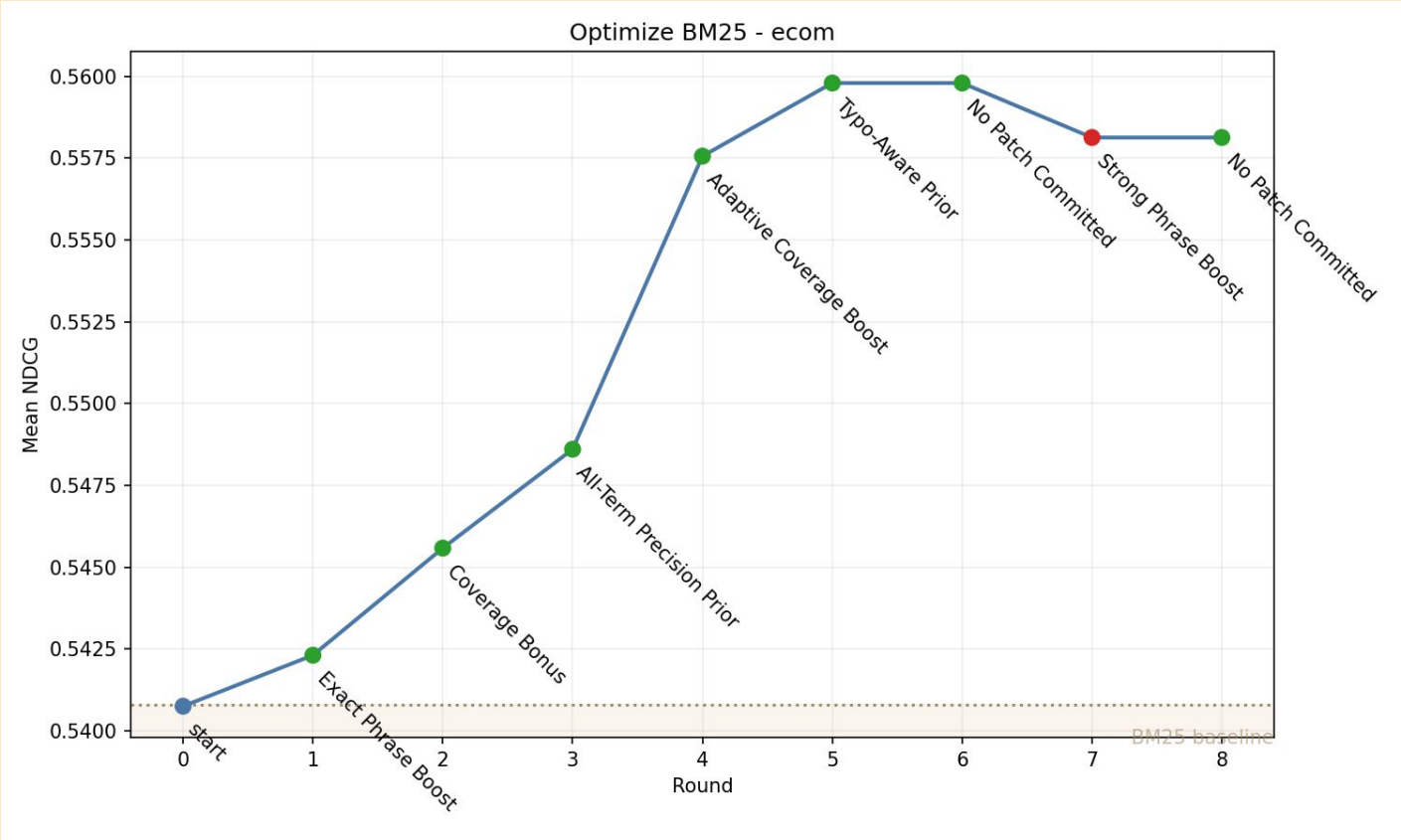
How well do our results transfer to the full dataset?

	minimarco	full MSMARCO	retention
BM25	0.4907	0.1901	N/A
Free-form-Python agent	0.5350 (+0.044)	0.1991 (+0.009)	**21%**
Our agent (Vespa rank features)	0.5060 (+0.015)	0.2053 (+0.015)	**99%**
Our manual sweep (Vespa rank features)	0.5163 (+0.026)	0.2106 (+0.021)	**80%**

Most of our minimarco lift from both attempts survives the jump to full MSMARCO - these are all features wit
generalizable signal, re-tuning the weights on the full dataset might squeeze out more.

https://blog.vespa.ai/re-autoresearching-msmarco-bm25-on-vespa/

Trying to find a better BM25 for ecom...



Not magic, but data-driven collaboration

Inspiration + Guidance

Ideas outside the norm
Manage Agent focus
Evals + Guardrails

Exhaustive, Rote, Obvious.

Existing knowledge
Tireless trial + error
Try the obvious

Analogy: Erdos Problems

The story of Erdős problem #1026

8 December, 2025 in expository, math.CA, math.CO | Tags: AI, AlphaEvolve, Erdos | by Terence Tao

Problem 1026 on the Erdős problem web site recently got solved through an interesting combination of existing literature, online collaboration, and AI tools. The purpose of this blog post is to try to tell the story of this collaboration, and also to supply a complete proof.

The original problem of Erdős, posed in 1975, is rather ambiguous. Erdős starts by recalling his famou
of k^2+1 distinct rea
which is either increas
k^2+1 to k^2, by con

conjecture that $c(k^-) = 1/k$, which would also imply that $\sqrt{n}c(n) \to 1$ as $n \to \infty$. (EDIT: as later located by an AI deep research tool, this conjecture was also made in Section 12 of this 1980 article of Steele.) Stijn also described the extremizing sequences for this range of n, but did not continue the calculation further (a naive computation would take runtime exponential in n,

(Should we add search to our deep research?)

Terence Tao
https://terrytao.wordpress.com/2025/12/08/the-story-of-erdos-problem-126/