Towards Self-Improving Retrieval Augmented Systems

Avishek Anand · TU Delft

Building retrieval systems with industry. Explaining them in academia.

My research — retrieval-augmented AI systems

Systems that retrieve documents, passages, or knowledge at query time to answer questions, verify claims, and make decisions.

Fact-checking with retrieval

Search over the web for evidence that supports or refutes a claim

CLAIM

San Francisco Bay Area contains zero towns

UNVERIFIED

RETRIEVED DOCUMENT

en.wikipedia.org/wiki/San_Francisco_Bay_Area

The San Francisco Bay Area, referred to locally as the Bay Area, is a populous region surrounding the San Francisco and San Pablo estuaries in Northern California. The region encompasses the major cities and metropolitan areas of San Jose, San Francisco, and Oakland, along with **smaller urban and rural areas.** ← *contradicts claim*
The Bay Area's nine counties are Alameda, Contra Costa, Marin, Napa, San Francisco, San Mateo, **Santa Clara, Solano and Sonoma.**
The combined statistical area of the region is the second largest in California after the Los Angeles area.

VERDICT REFUTED

Bay Area explicitly includes towns, rural areas, and nine counties — not zero towns

The retriever surfaces evidence · the verifier makes the call

Open-domain question answering

Natural language question · search millions of documents · extract the answer

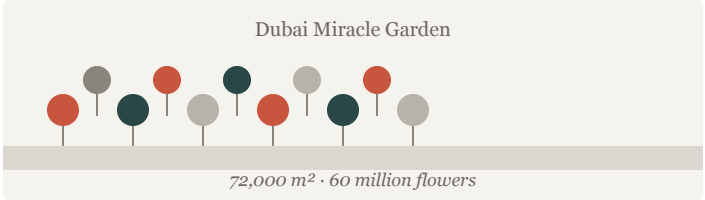
QUESTION

Where is the world's largest flower garden located?

ANSWER

Dubai, UAE

Dubai Miracle Garden



RETRIEVED DOCUMENT

travel-database.com / Wikipedia

The Dubai Miracle Garden is certainly aptly named considering that — like pretty much everything in this Middle Eastern destination — it was built on desert land. Billing itself as the **world's largest natural flower garden,** the 72,000-square-meter attraction has more than 60 million flowers on display.

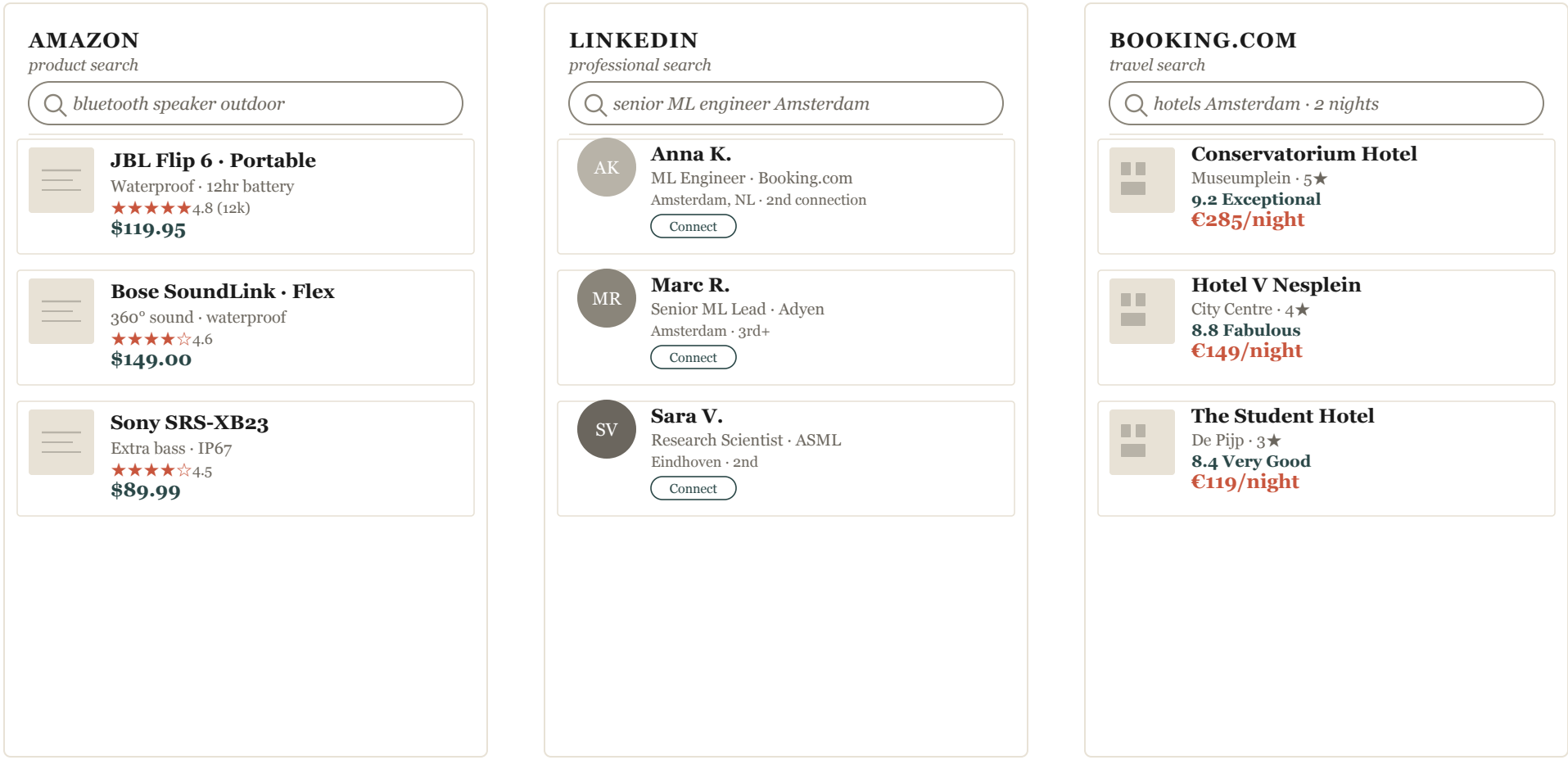
Located in Dubailand, Al Barsha South district, the garden opened in 2013. It features over 150 flower varieties arranged in themed structures.

Location: Dubai, UAE · Open season: November to April

The answer exists somewhere in the corpus. Retrieval brings it to the surface.

Search in products, people, and places

Retrieval powers every discovery surface you interact with

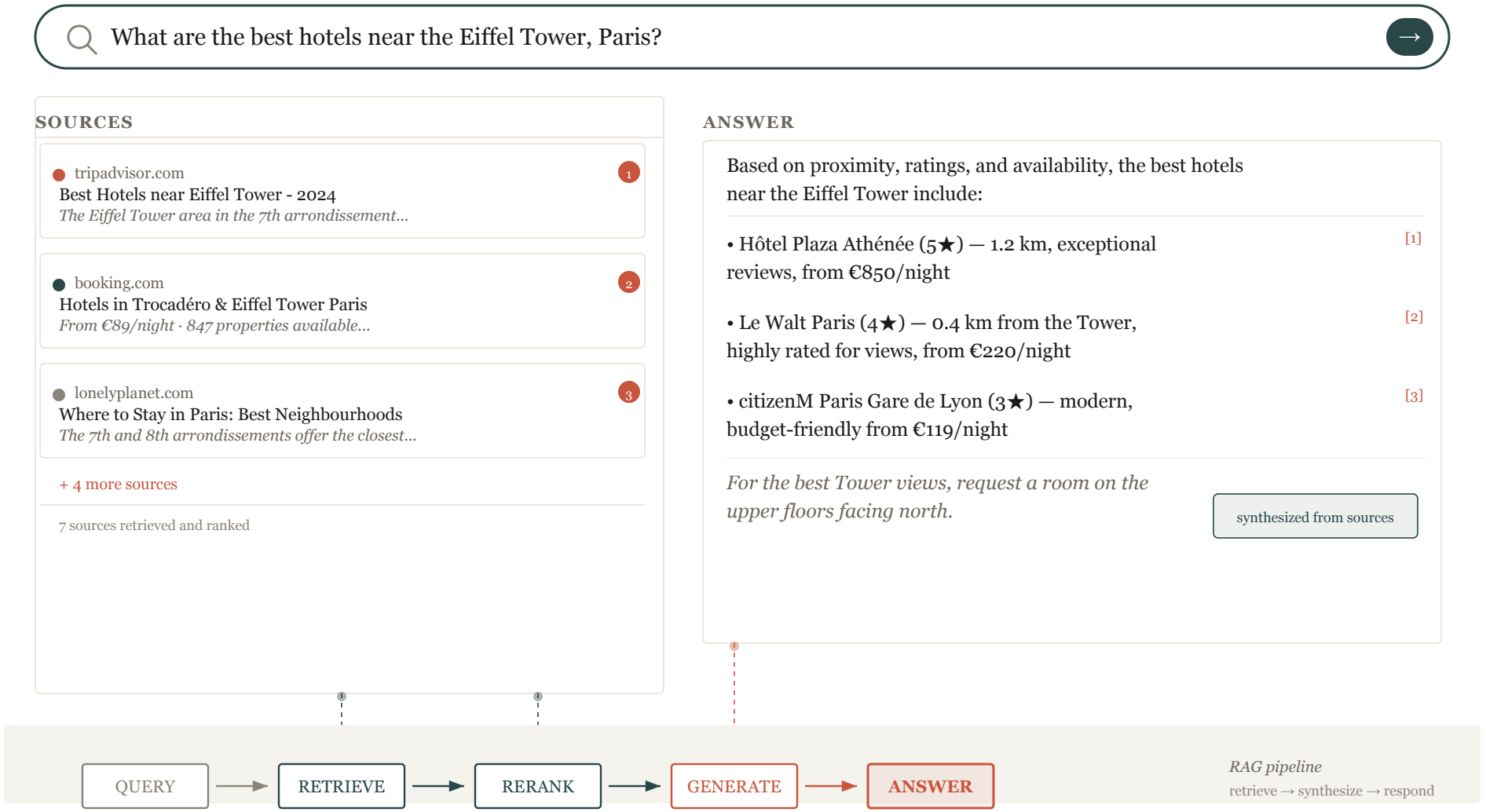


Common thread: rank millions of documents by relevance to the user's intent

E-commerce · Professional networks · Travel · News · Code — retrieval is everywhere

Modern RAG — real-time synthesis from retrieved documents

Systems like Perplexity, ChatGPT Search, Bing AI — powered by retrieval



I'm known for Explainable IR

Foundational methods for understanding *why* retrieval and ranking models decide what they decide.

Algorithms I introduced are now standard tools in the field.

But I have a split personality

15+ years alongside explainability: **scaling and efficiency of retrieval** — inside and alongside industry.

Algorithms running in production at companies you've heard of.

Amazon — first-generation embeddings-based search

2018–2021 · Semantic retrieval at the scale of a billion products

Industry collaboration · Product Search · NLP

KEYWORD MATCH

BM25 / inverted index

waterproof outdoor portable speaker

1. **Waterproof Outdoor Speaker Pro X3**
   exact keyword match
   ★★★★☆ · $78.99
2. **Outdoor Speaker Waterproof Bluetooth**
   keyword overlap
   ★★★★☆ · $65.00
3. **Portable Waterproof Speaker Case**
   partial match — wrong product!
   ★★★☆☆ · $22.00

⚠ Returns cases, not speakers (keyword collision)

VS

EMBEDDING SEARCH

Dense retrieval · semantic matching

waterproof outdoor portable speaker

1. **JBL Flip 6 — Portable Waterproof Speaker**
   semantic match · top product in category
   ★★★★★ · $119.95
2. **Bose SoundLink Flex · Outdoor Speaker**
   semantic: outdoor, durable, portable
   ★★★★☆ · $149.00
3. **Sony SRS-XB23 · Extra Bass · IP67**
   semantic: waterproof, rugged
   ★★★★☆ · $89.99

✓ Returns relevant products — no false keyword matches

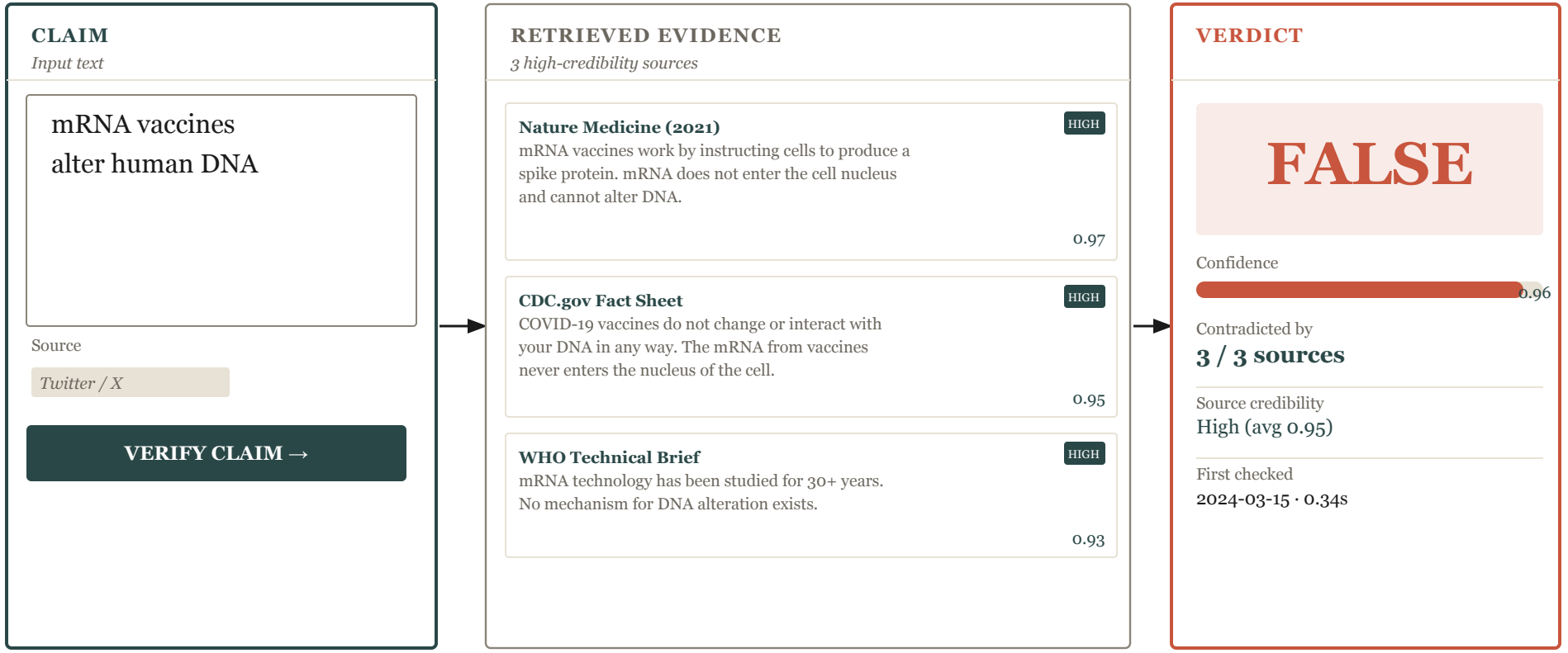
IMPACT 500M+ queries/day · 1B+ products · better recall for long-tail and semantic queries

Embeddings move keyword collision from common to rare — the long tail becomes reachable

Scaling semantic retrieval to industry demands is the engineering problem that makes the science real.

Factiverse — automated fact verification

Real-time claim checking across news, research, and social media



Retrieval is the core engine — veracity follows from which documents you retrieve.

Used by newsrooms, fact-checkers, and social platforms.

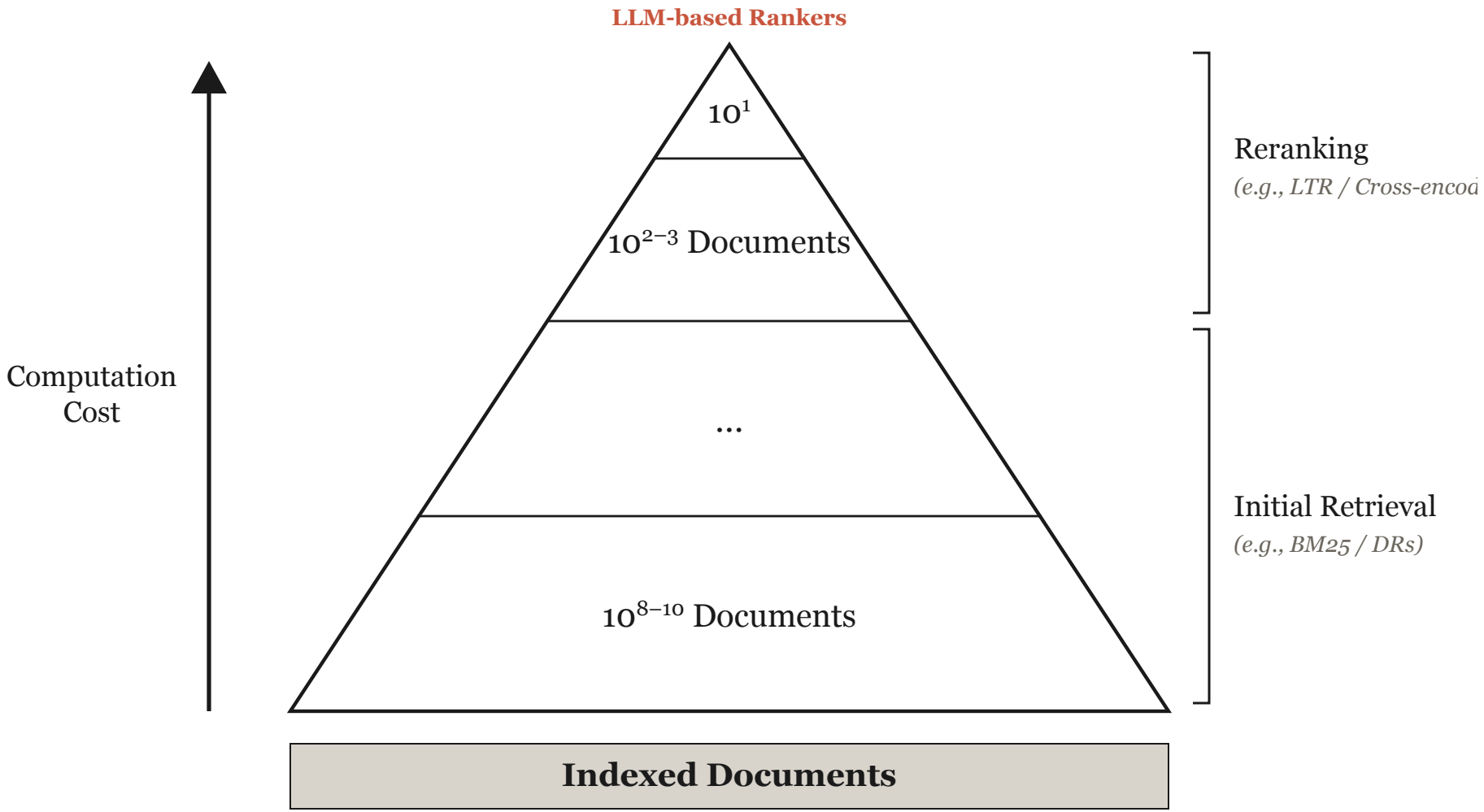
What goes wrong in production RAG today

Three failure modes — all three show up in this single session.

I'll walk through each one with a concrete example.

Cascading retrieval — narrowing from billions to a few

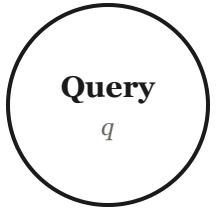
Computation cost grows as the candidate set shrinks



Each layer is more expensive per document — and reads more of the document.

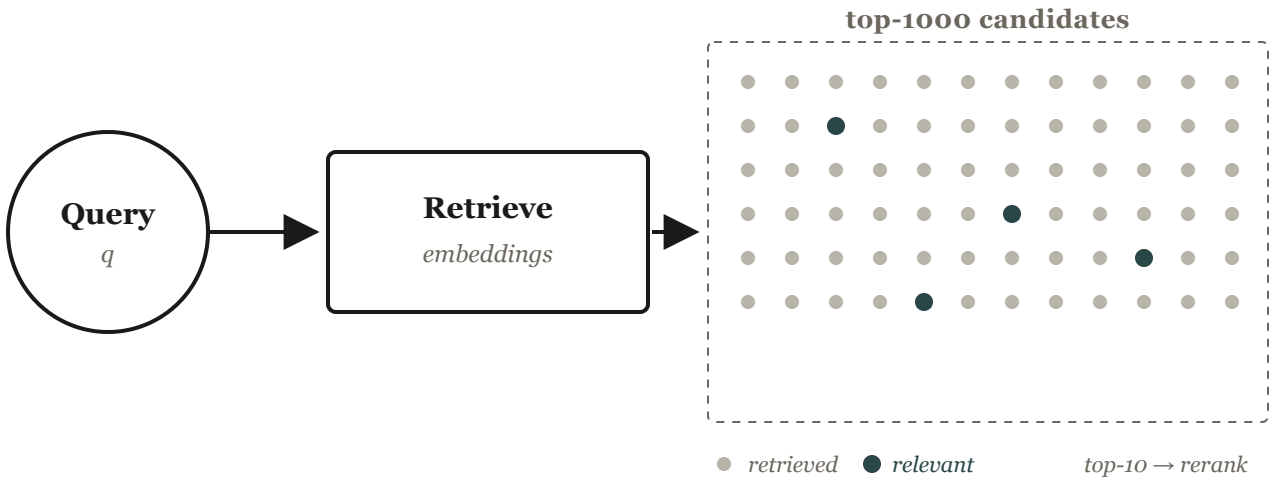
The cascading myth

The standard pipeline — and what it assumes



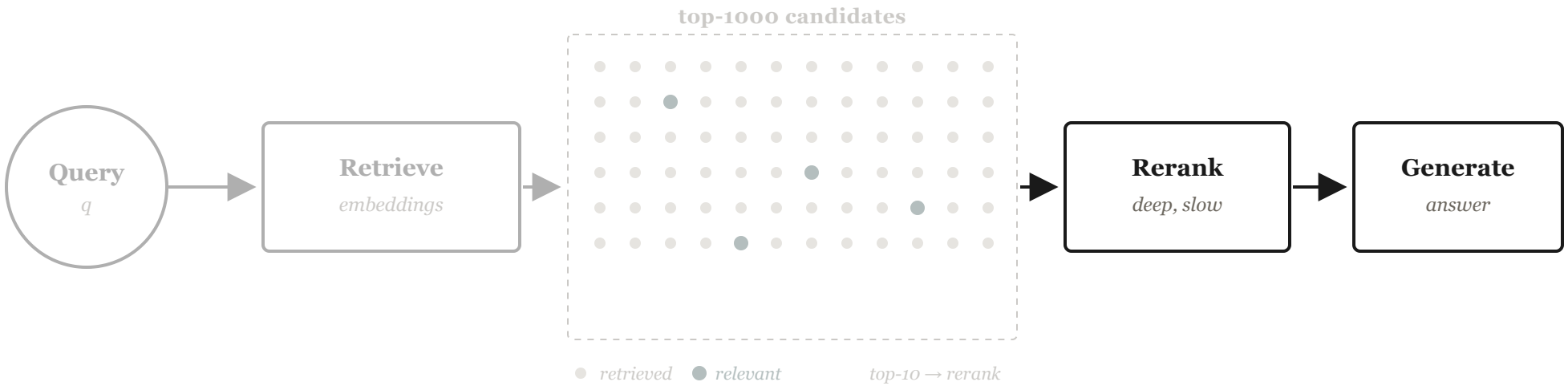
The cascading myth

The standard pipeline — and what it assumes



The cascading myth

The standard pipeline — and what it assumes



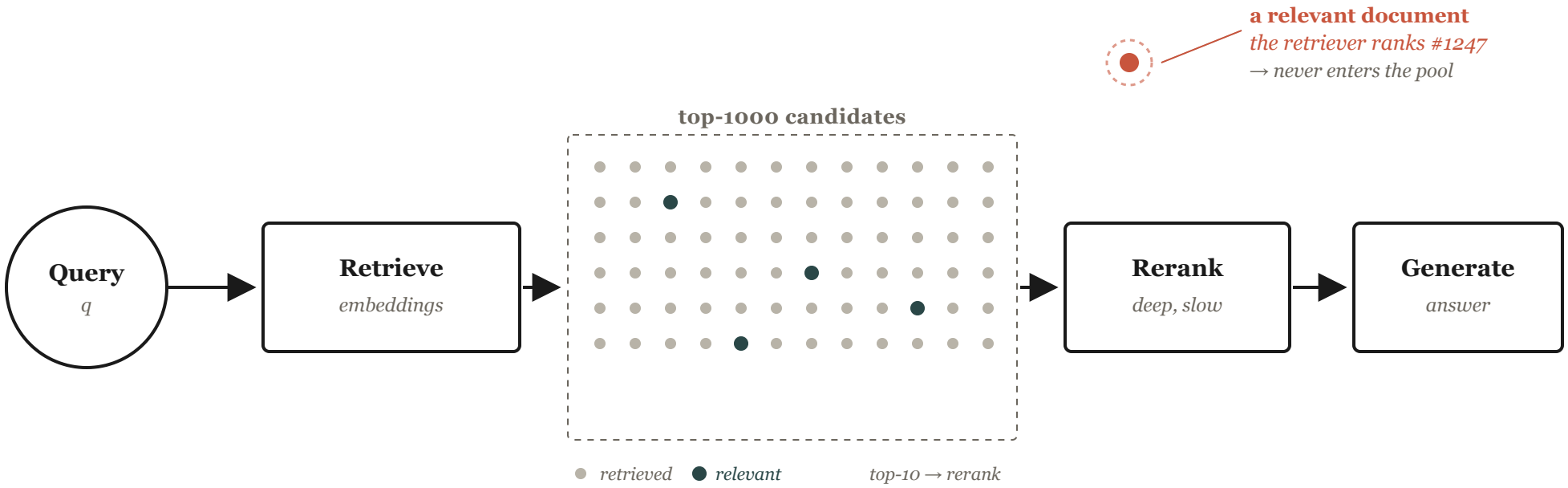
THE ASSUMPTION

The retriever's **top-1000 contains everything** the reranker would have wanted to see.

What if it doesn't?

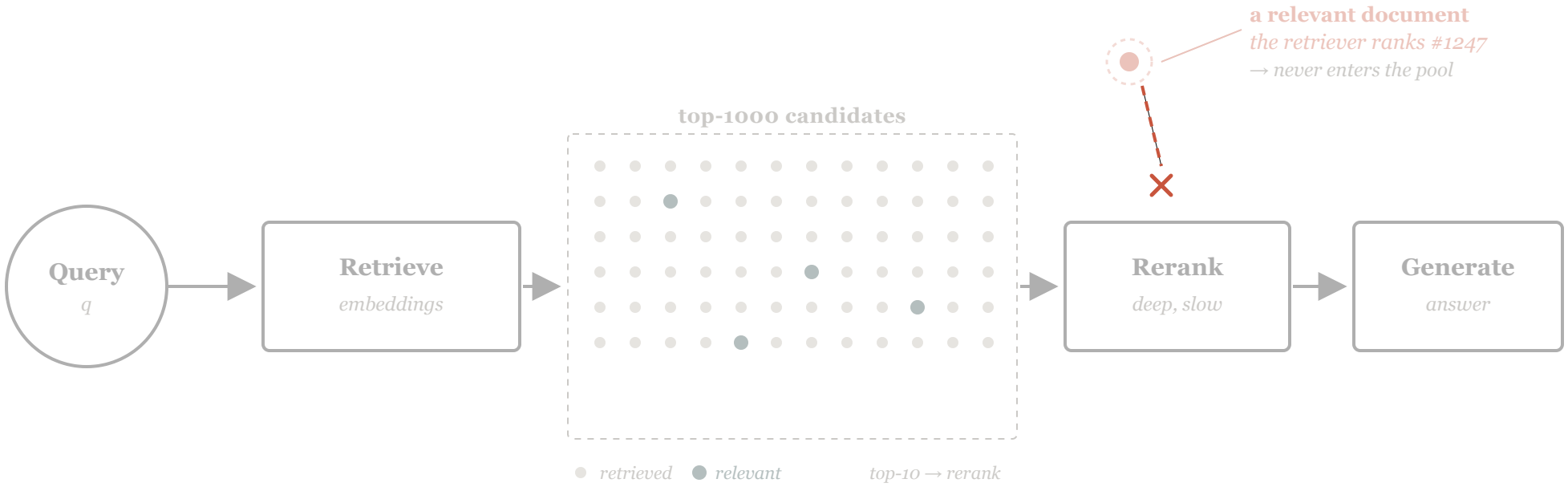
What if it doesn't?

Some relevant documents never enter the top-1000



What if it doesn't?

Some relevant documents never enter the top-1000



THE FAILURE MODE

The reranker can **only re-order what it's given.**
If the right answer isn't in the top-1000 — it can't recover it.

Industry has accepted this as a cost of doing business.

Failure 1 — the recall ceiling

The reranker can't fix what it never sees

QUERY

"romantic agriturismo
near Florence with a vineyard"

THE PERFECT LISTING

exists in inventory · ranked #1,400 by first-stage

Casa di Lucia

listing description (truncated)

"family-run estate in the
Chianti hills, working winery,
two-bedroom suite, terrace
overlooking the vines…"

— never says "romantic"
— never says "agriturismo"

vocabulary mismatch · BM25 misses · dense ranks #1,400

Failure 1 — the recall ceiling

The reranker can't fix what it never sees

QUERY

"romantic agriturismo
near Florence with a vineyard"

THE PERFECT LISTING

exists in inventory · ranked #1,400 by first-stage

Casa di Lucia

listing description (truncated)

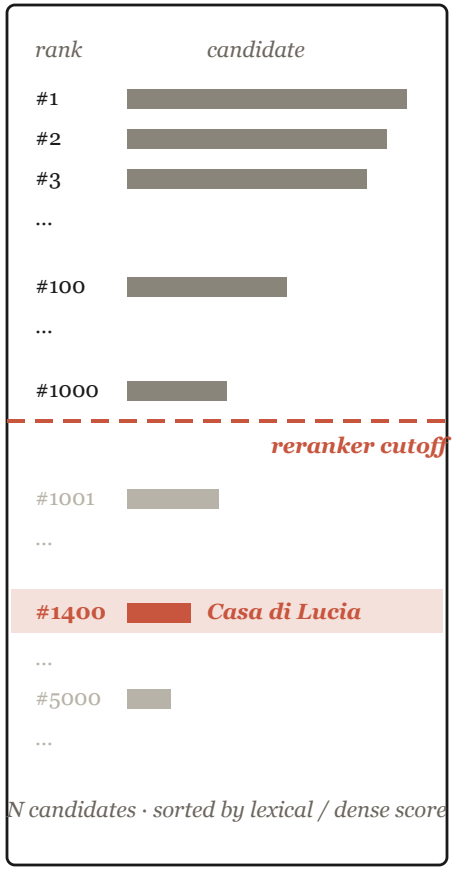
"family-run estate in the
Chianti hills, working winery,
two-bedroom suite, terrace
overlooking the vines…"

— never says "romantic"
— never says "agriturismo"

vocabulary mismatch · BM25 misses · dense ranks #1,400

FIRST-STAGE RANK

BM25 / dense over corpus



Failure 1 — the recall ceiling

The reranker can't fix what it never sees

QUERY

"romantic agriturismo near Florence with a vineyard"

THE PERFECT LISTING

exists in inventory · ranked #1,400 by first-stage

Casa di Lucia

listing description (truncated)

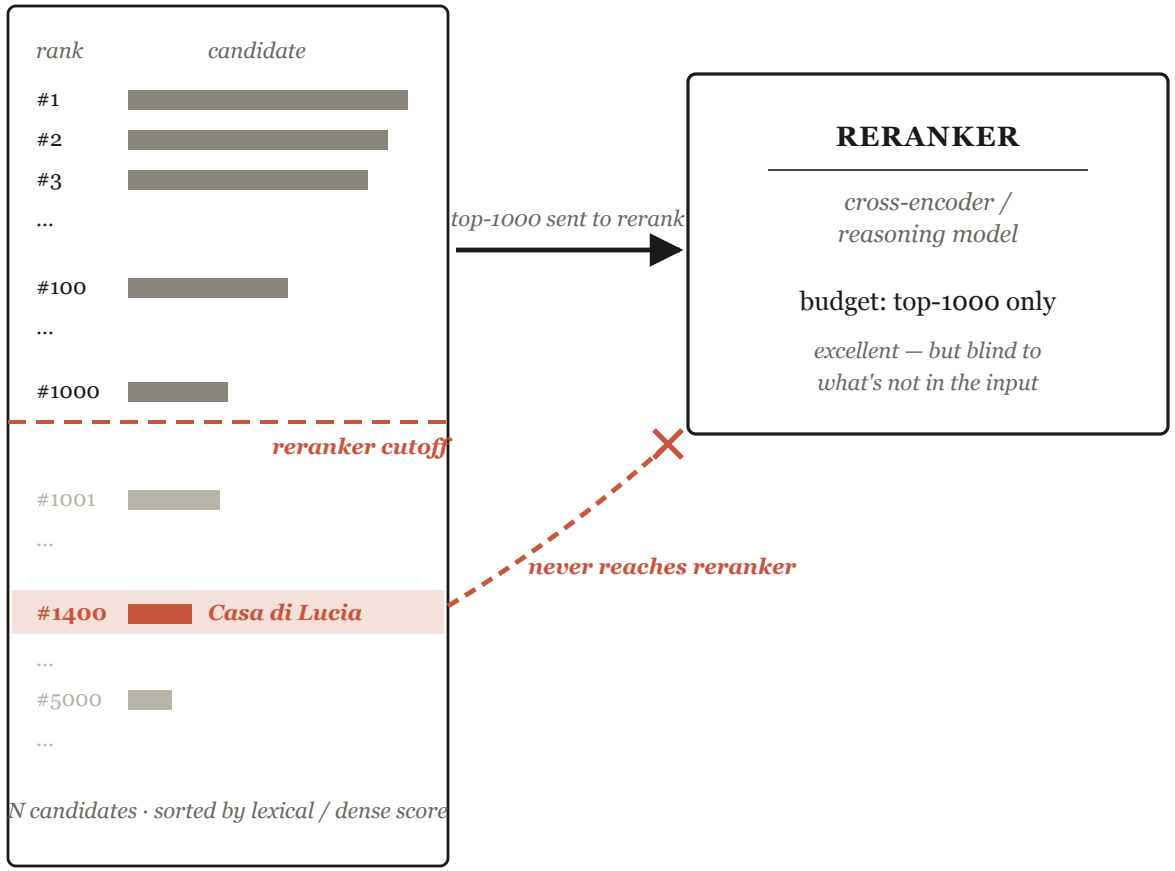
"family-run estate in the Chianti hills, working winery, two-bedroom suite, terrace overlooking the vines…"

— never says "romantic"
— never says "agriturismo"

vocabulary mismatch · BM25 misses · dense ranks #1,400

FIRST-STAGE RANK

BM25 / dense over corpus



A perfect document, lost forever between the retriever and the reranker.

A better reranker doesn't help. The relevant doc was never on the table.

> *The reranker can't fix what it never sees.*

Failure 2 — The drift problem

Modern systems try to fix recall with **query reformulations**.

Generate several rephrasings of the original query. Retrieve for each. Merge.

The intuition: more queries, more recall.

The reality: most reformulations drift.

Reformulating *"good hotels in Tokyo"*

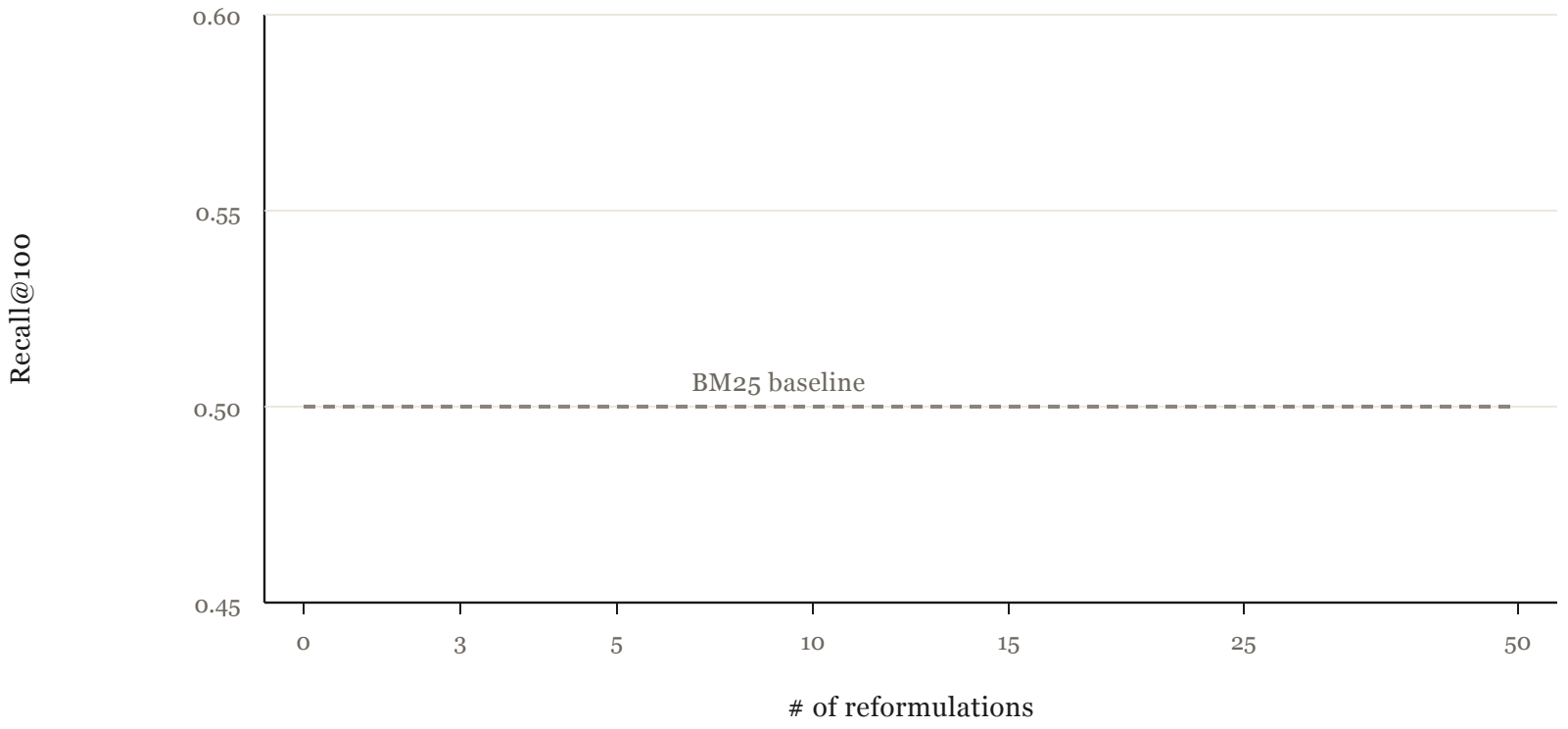
REFORMULATION	*WHAT IT ACTUALLY RETRIEVES*
"luxury hotels in Tokyo"	drifts toward expensive — wrong for most users
"highly rated hotels in Tokyo"	usually most useful
"popular hotels in Tokyo"	drifts toward tourist traps
"hotels near Tokyo Disneyland"	drifts away from the actual intent

Concatenate or rank-fuse all of them, and noise dominates signal.

More reformulations → more drift

Classical reformulation degrades past ~10 reformulations · ReformIR stays stable

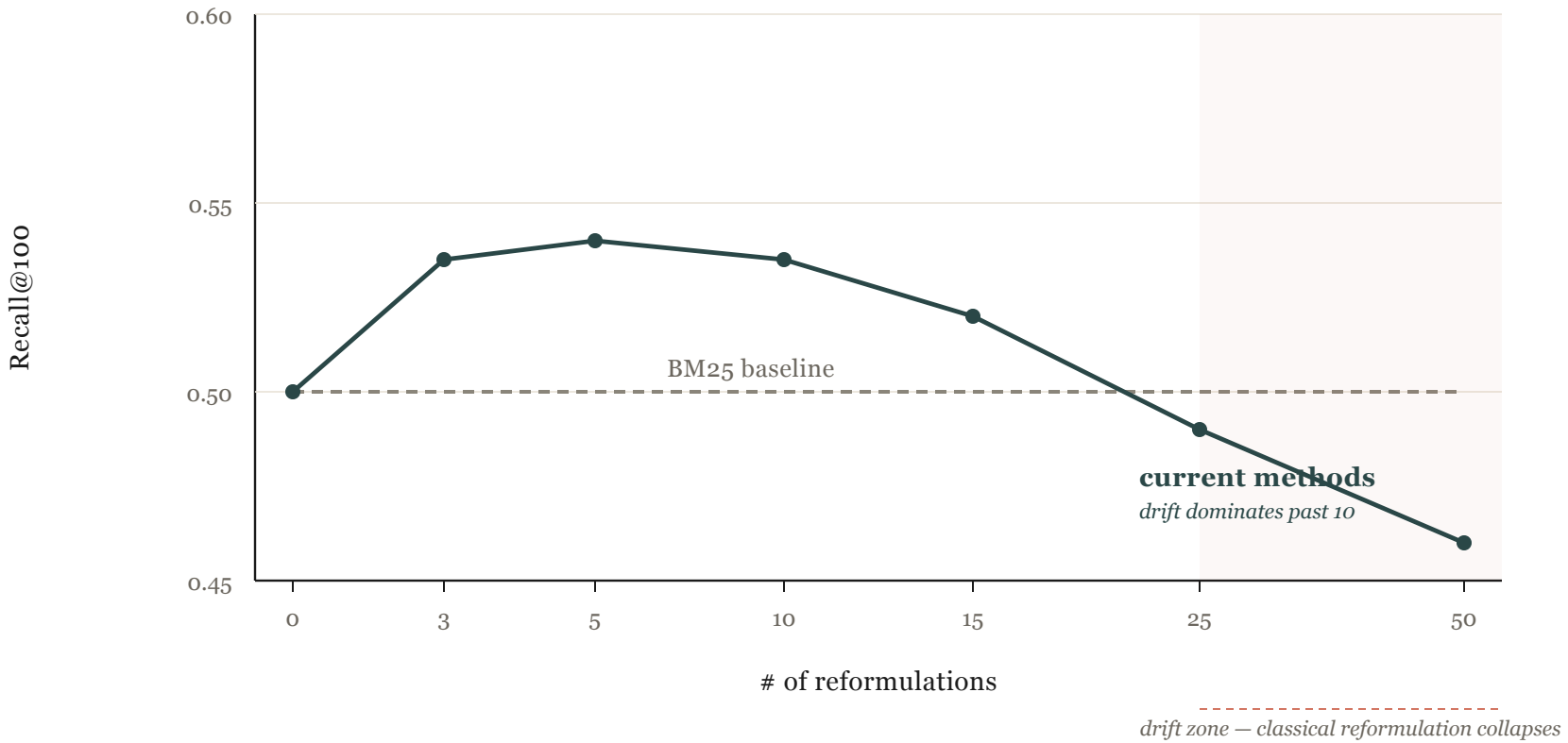
TREC DL19 · BM25 → MonoT5 (budget c=100)



More reformulations → more drift

Classical reformulation degrades past ~10 reformulations · ReformIR stays stable

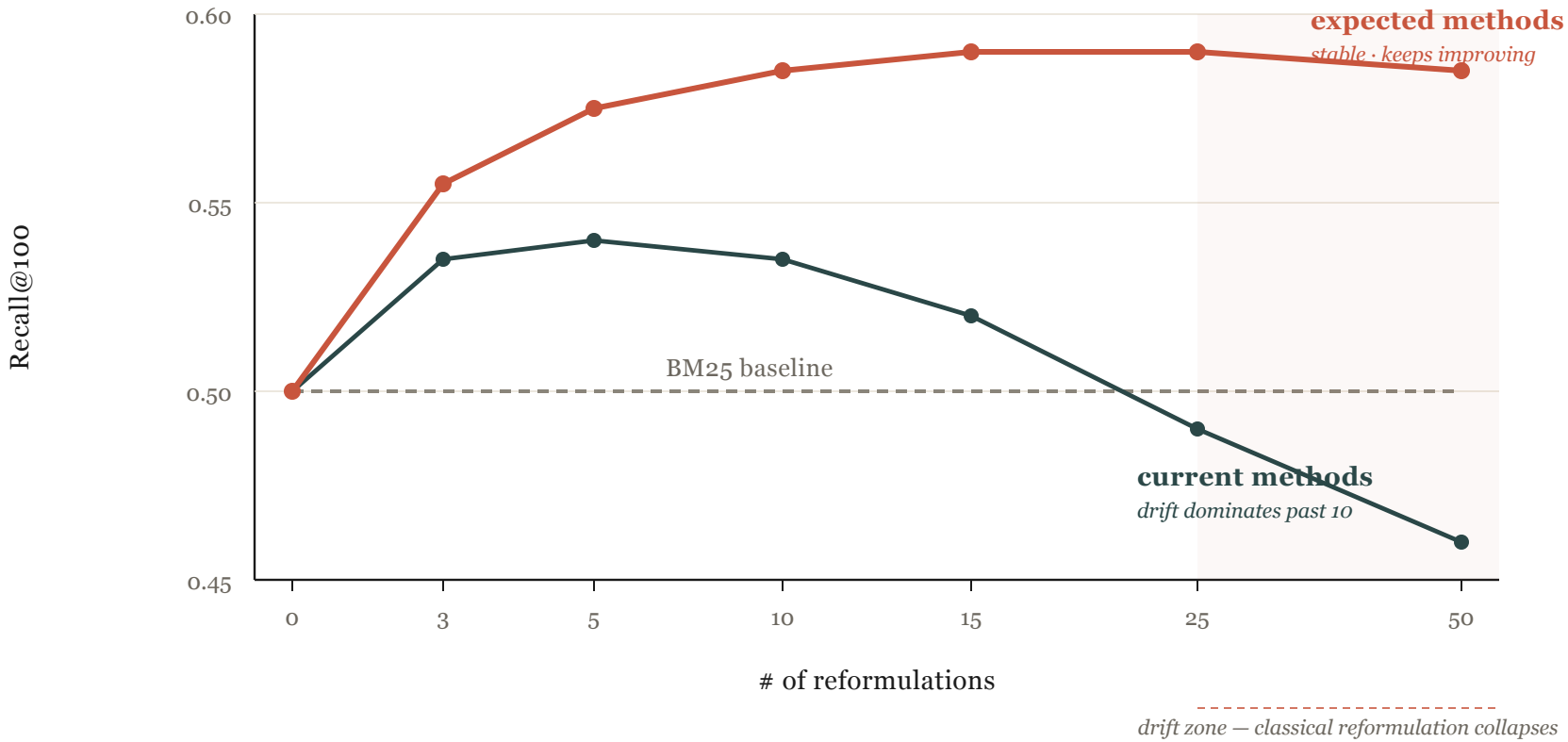
TREC DL19 · BM25 → MonoT5 (budget c=100)



More reformulations → more drift

Classical reformulation degrades past ~10 reformulations · ReformIR stays stable

TREC DL19 · BM25 → MonoT5 (budget c=100)



Adding more reformulations doesn't fix recall — **picking the right ones does.**

expected methods uses ranker feedback to up-weight useful reformulations and down-weight drifty ones.

> *More reformulations → more drift — current query reformulation methods are not the answer.*

Failure 3 — The cost wall

The obvious response to failures 1 and 2: use a stronger model.

A reasoning-model reranker reads the document, the query, the constraints — and judges relevance like a human would.

It works. It's also **prohibitively expensive.**

So you're back to the recall ceiling — gated by whatever the cheap retriever surfaces.

Failure 3 — the cost wall

Better rerankers exist. You can't afford to run them.

LATENCY TAX

40×

slower per query

cross-encoder → LLM reranker

Failure 3 — the cost wall

Better rerankers exist. You can't afford to run them.

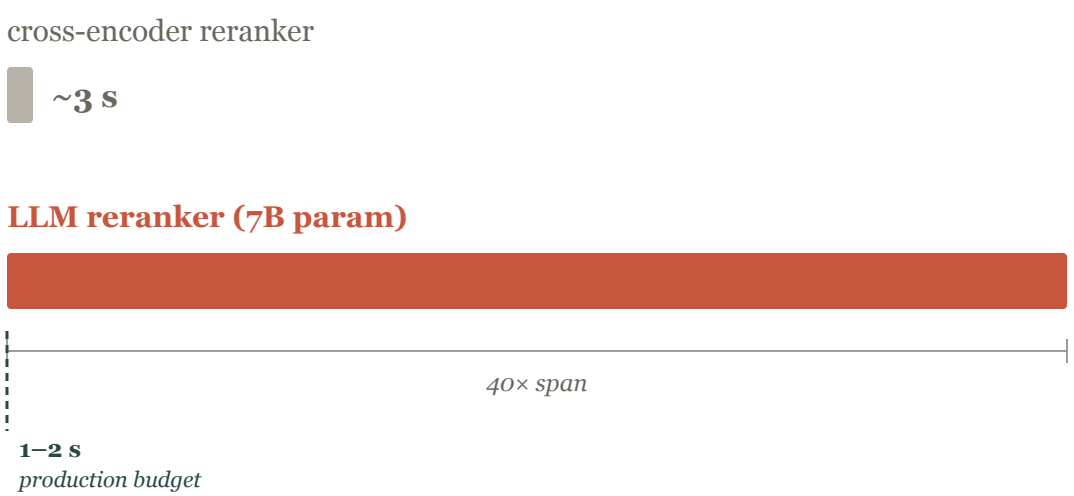
LATENCY TAX

40×

slower per query

cross-encoder → LLM reranker

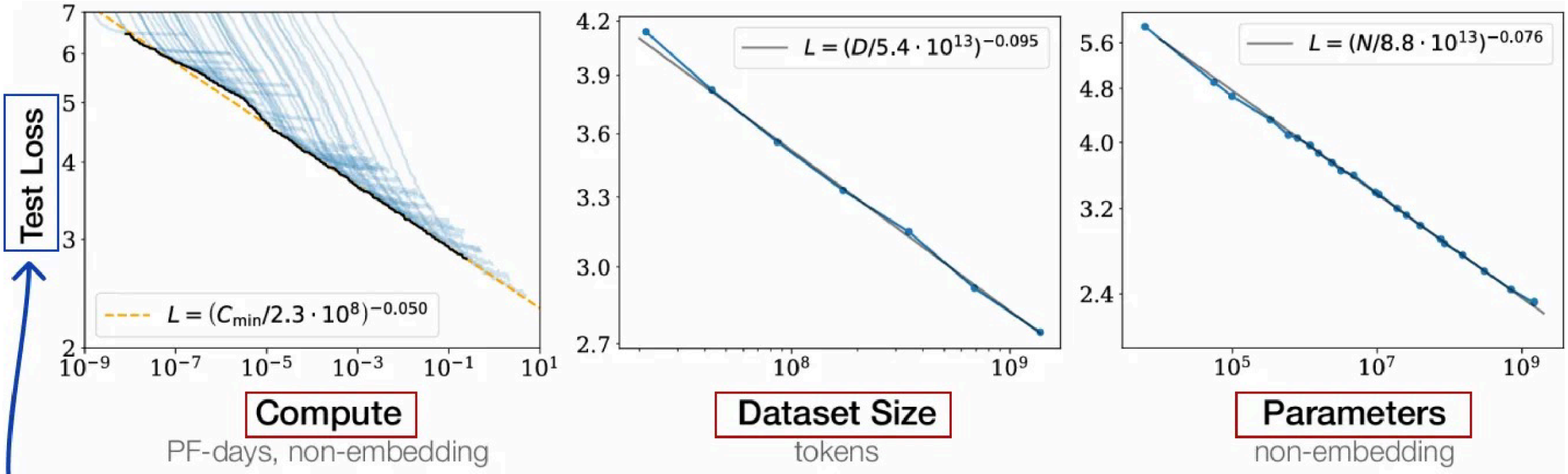
TIME TO RANK 1,000 DOCUMENTS



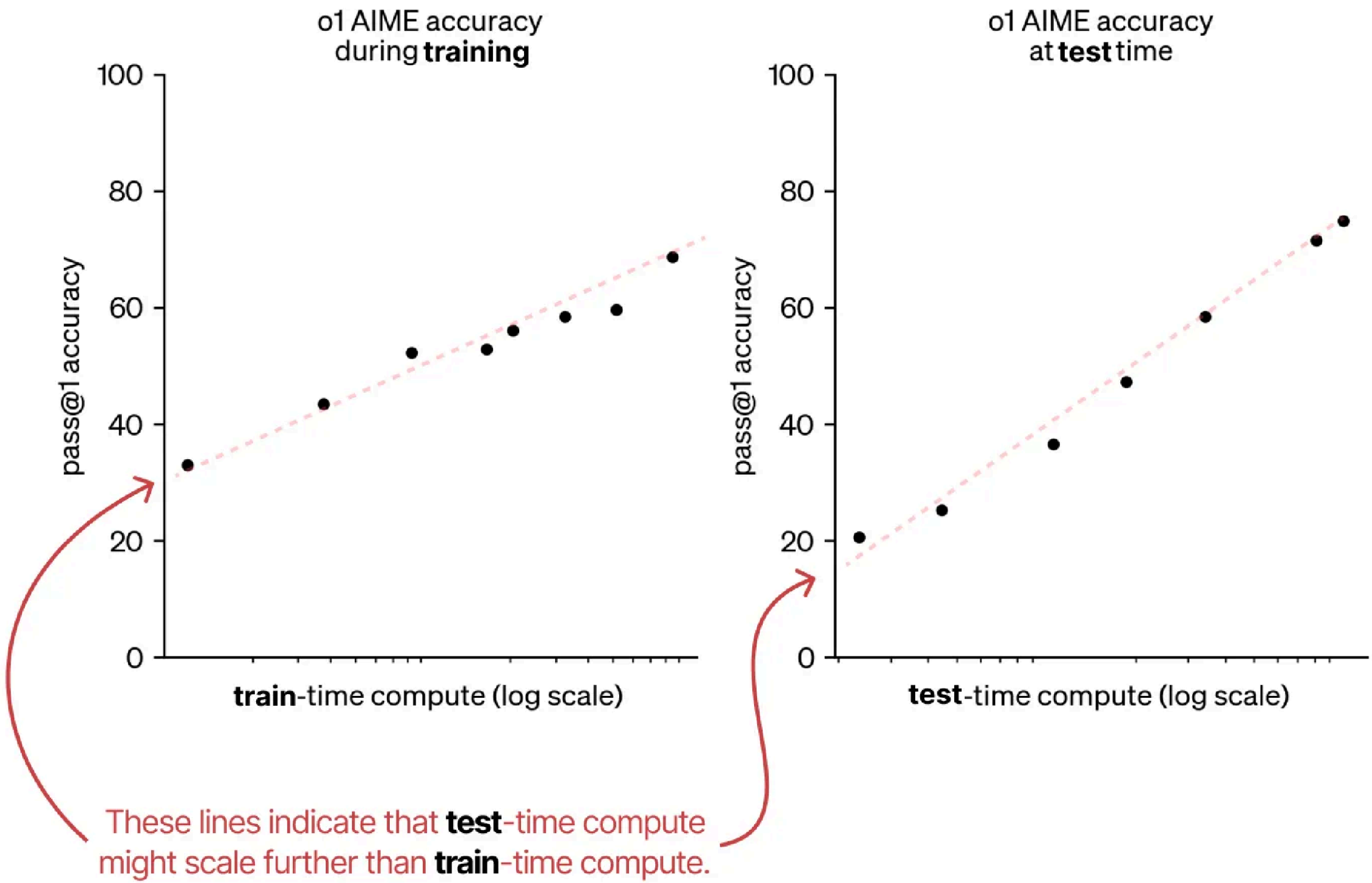
Production search has a second-scale latency budget, not a minute-scale one.

You can afford the LLM reranker on 10–20 candidates. Not on 1,000.

Better rerankers don't fix bad retrievers — they make the gap more expensive to ignore.



Performance increases with more **compute**, **tokens**, and **parameters**.


o1 AIME accuracy
during **training**
100
80
60
40
20
0
pass@1 accuracy
train-time compute (log scale)
o1 AIME accuracy
at **test** time
100
80
60
40
20
0
pass@1 accuracy
test-time compute (log scale)
These lines indicate that **test**-time compute
might scale further than **train**-time compute.

> *Better rerankers don't fix bad retrievers. They make the gap more expensive to ignore.*

What these failures share

All three come from the same root cause:

The system makes **one-shot decisions**.

The retriever decides once. Reformulation happens once. The reranker reads what it's given.

No stage's signal is fed back to revise the stages before it.

What these failures share

Three failure modes · one root cause

FAILURE 1

The recall ceiling

retriever decides once →
reranker can't see what was dropped

FAILURE 2

The drift problem

reformulations decided once →
drift accumulates with each one

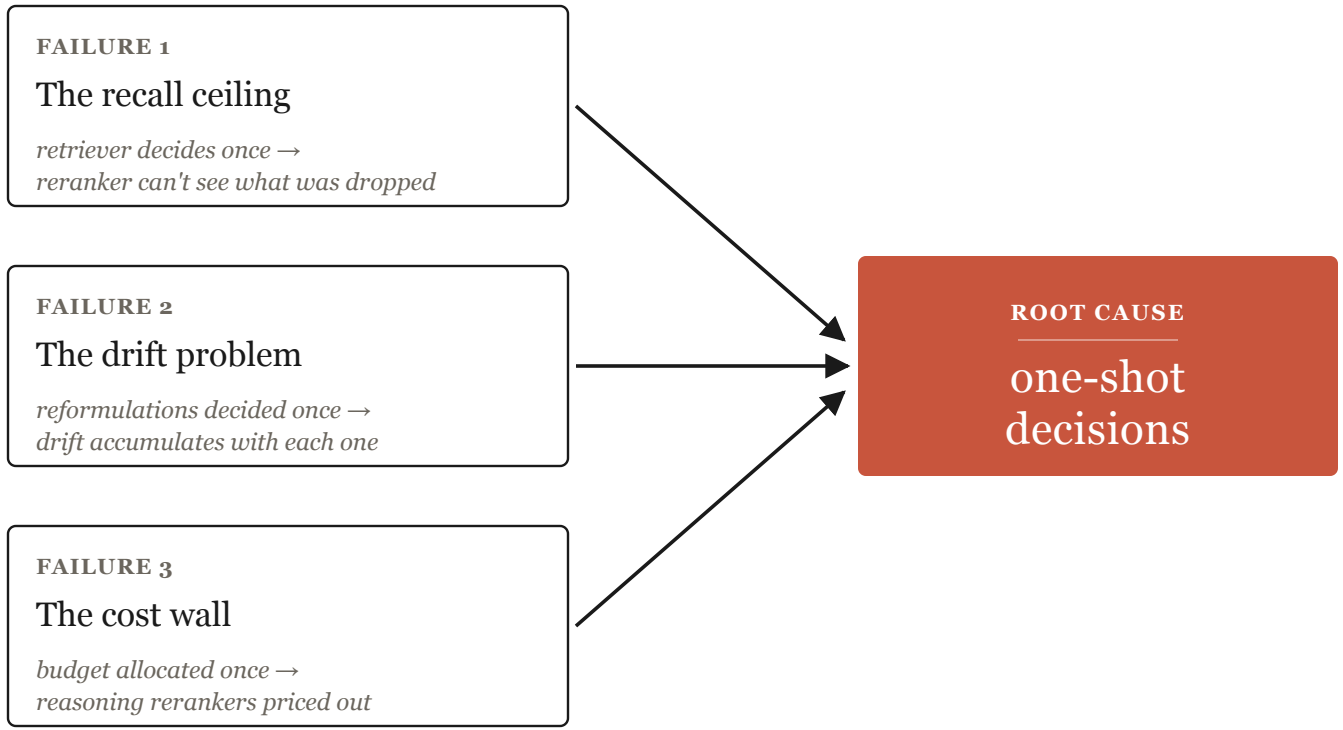
FAILURE 3

The cost wall

budget allocated once →
reasoning rerankers priced out

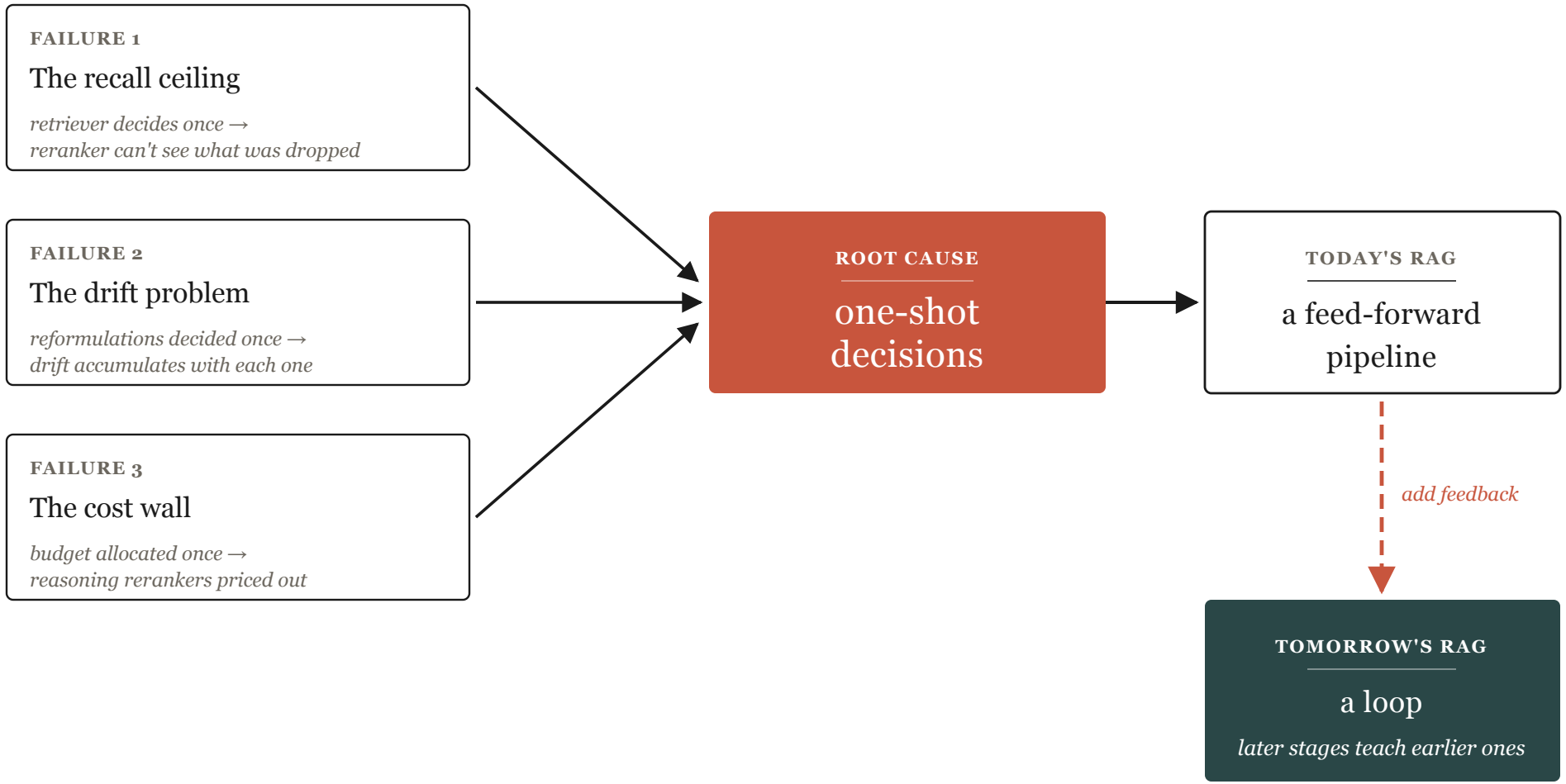
What these failures share

Three failure modes · one root cause



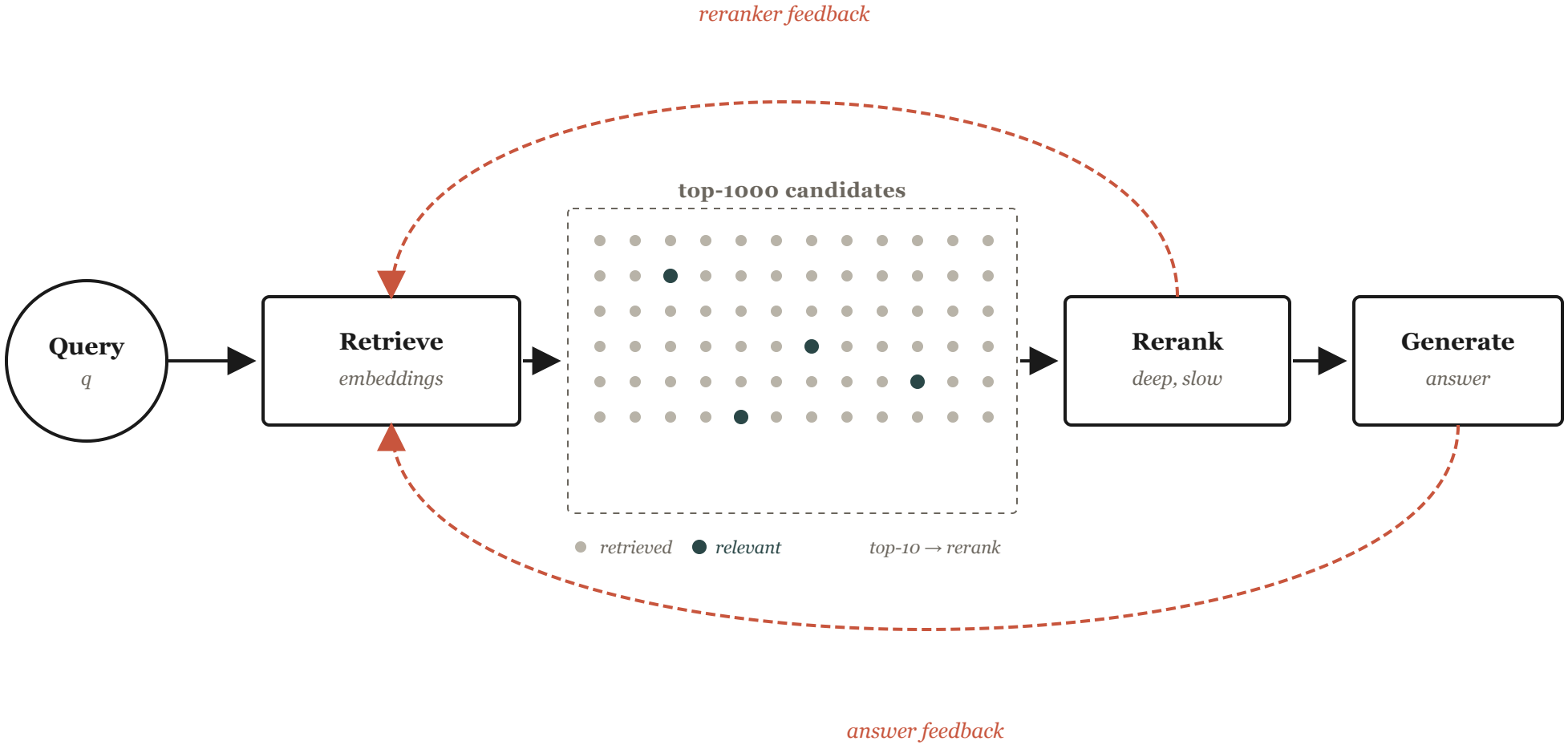
What these failures share

Three failure modes · one root cause



No stage's signal flows back. Every decision is locked in at the moment it's made.
Every algorithm I'm about to show you breaks that constraint.

Today's RAG is a feed-forward pipeline.

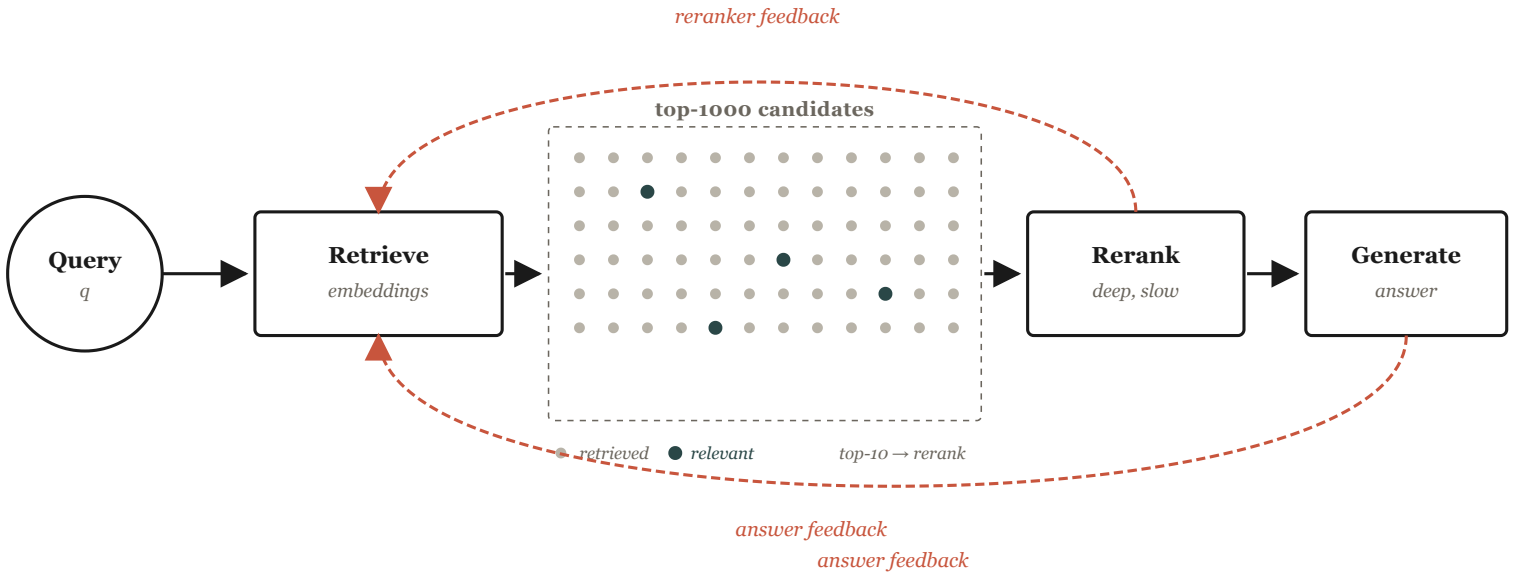


Tomorrow's RAG needs to be a loop.

The principle

Feedback from later stages can — and should — drive decisions in earlier stages.

The reranker's relevance estimates can re-prioritize the retriever's candidates.
The downstream generator's confidence can score upstream evidence.
The cost of judgments can budget the depth of search.



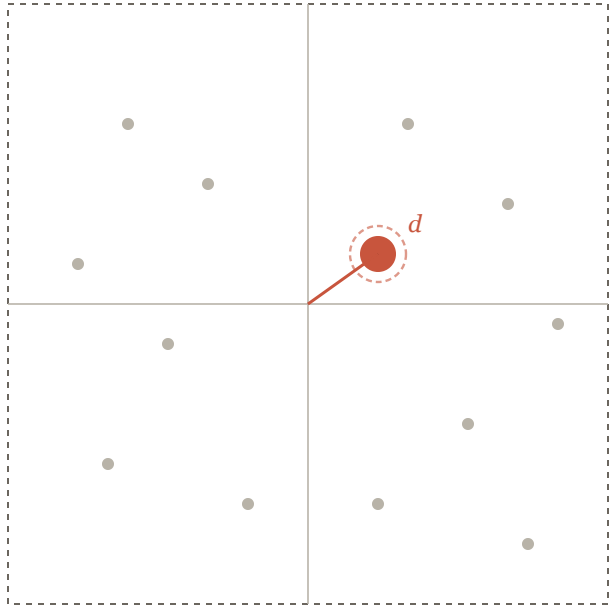
> *In the remaining slides: what feedback to use, how to use feedback, and what to optimize using feedback.*

The power of feedback

The reranker knows things the retriever doesn't

THE RETRIEVER SEES

a vector

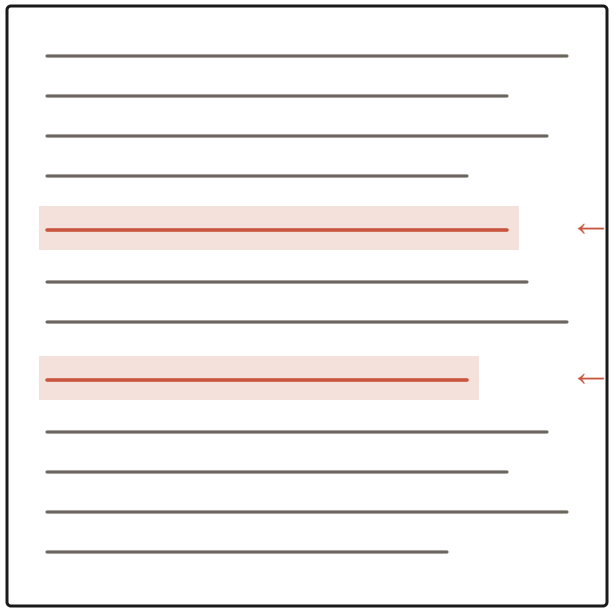


a 768-dim point

fast · shallow · approximate

THE RERANKER READS

the document



every word, in context

slow · deep · precise

That depth is where **the relevance signal lives**.

The retriever can't see it. The reranker can — but only on the docs it's given.

The reranker is a better estimator of relevance

Learning to Rank has formalized this for over 20 years.

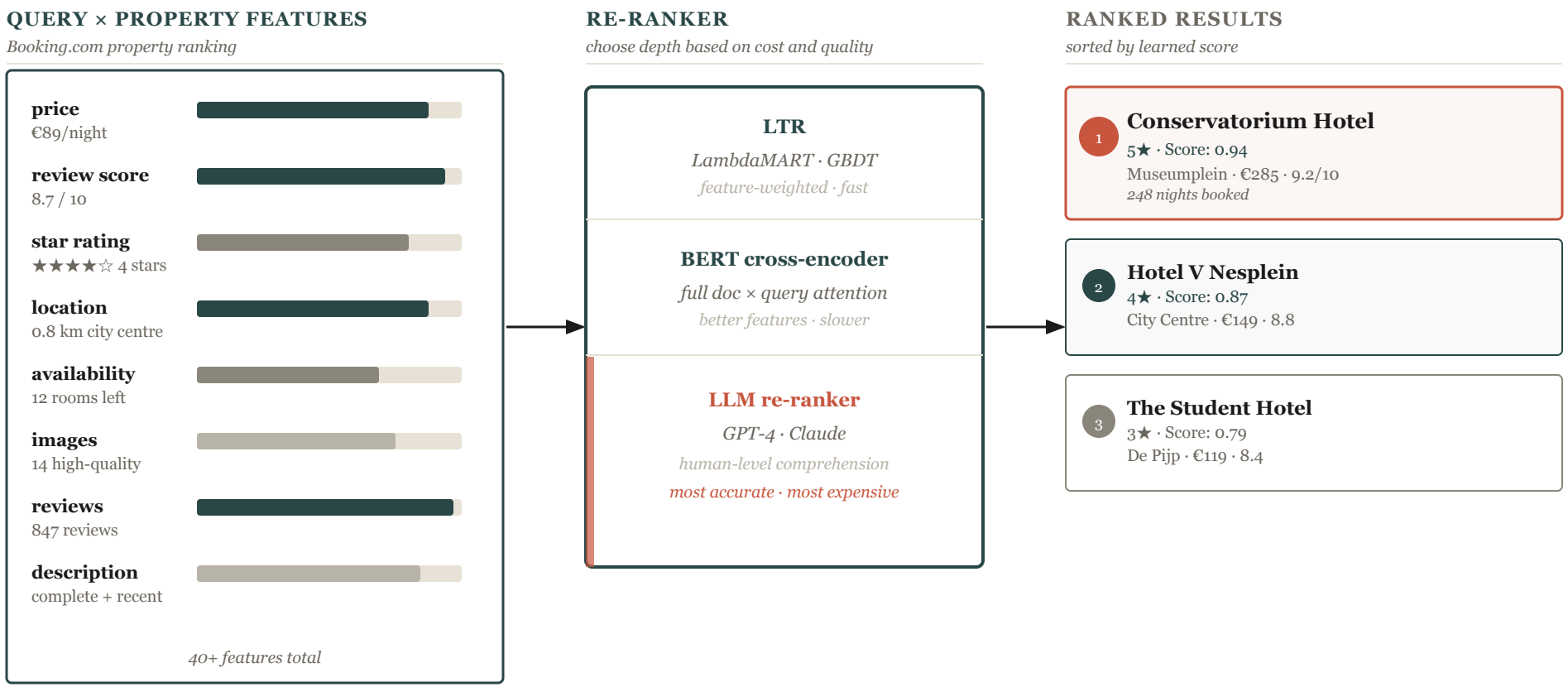
The reranker weights dozens of signals — query–title match, semantic similarity, price, location, review count, image quality — instead of one BM25 score.

Modern era: BERT reads the full document with cross-encoder attention. LLMs judge relevance with human-level reading comprehension.

The reranker is not a different problem from retrieval — it is a richer estimator of the same thing. Better features, more compute, fewer documents.

Learning to Rank — the reranker as a scoring function

20 years of building better estimators of relevance



The reranker is not magic — it has better features. The choice is cost vs. quality.

> *What feedback should we use — and how do we reorganize our indexes to react to it?*

> *Use the re-ranking signal as feedback.*

Static batching — one signal, fixed order

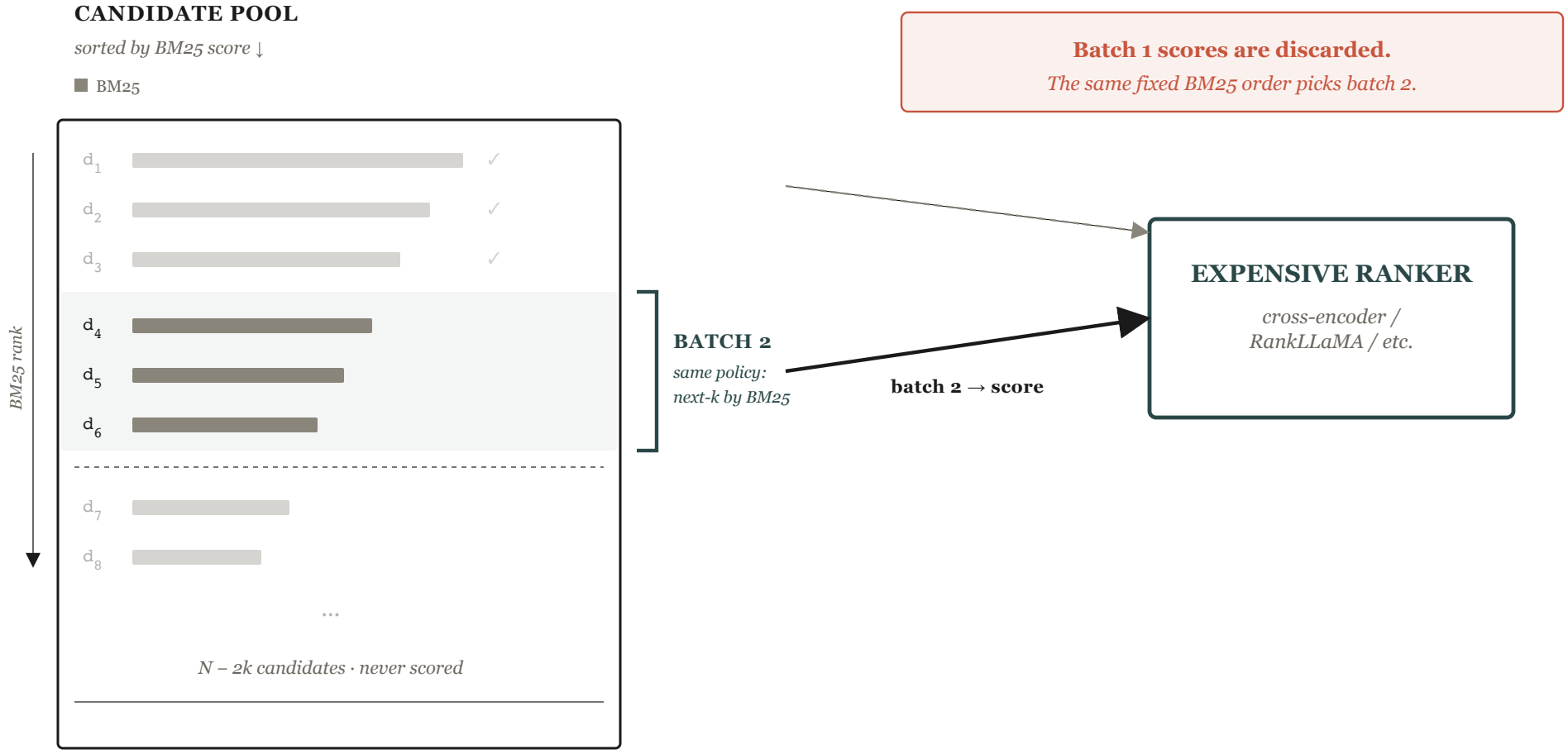
Sorted by BM25 · peel off batches in order · never look back

Reranker scores order the output — but never decide what comes next.

The information from batch 1 is discarded. What if we kept it?

Static batching — one signal, fixed order

Sorted by BM25 · peel off batches in order · never look back

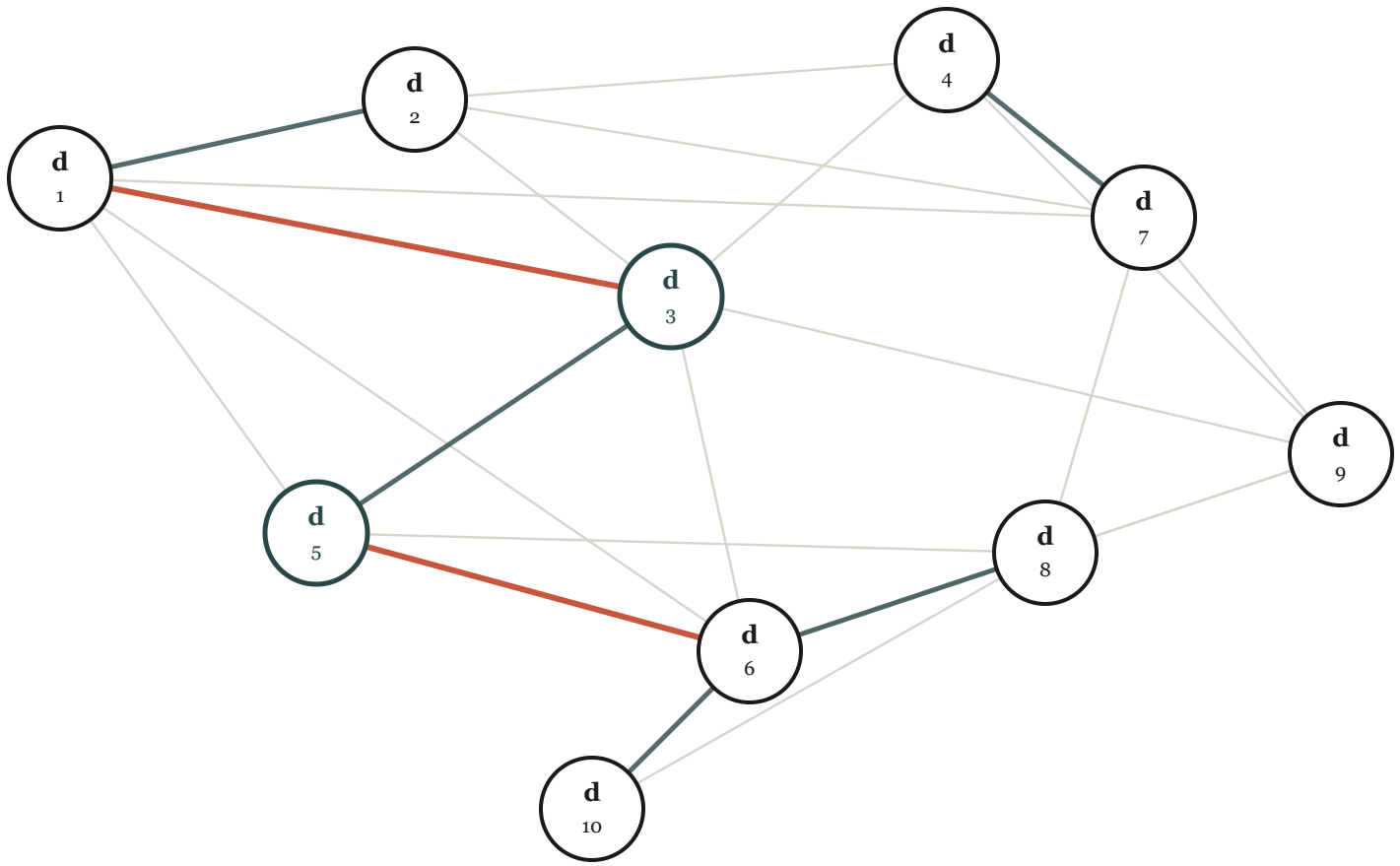


The reranker's feedback on batch 1 could have told us which neighbors to try next.

Instead, it is thrown away. That is the gap QUAM closes.

Co-Relevance in the Representation Space

Documents close in embedding space are not necessarily co-relevant — and vice versa.



weak / no co-relevance

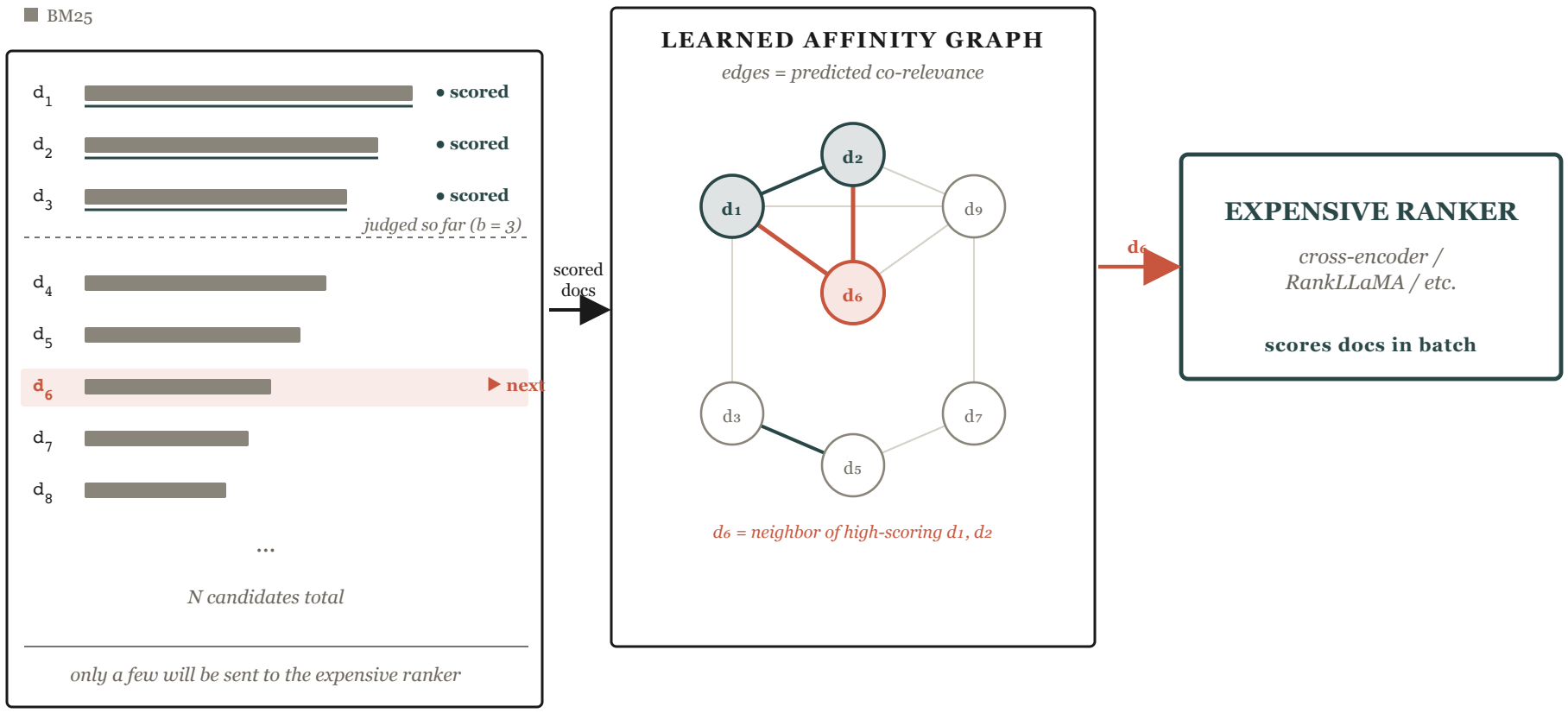
co-relevant (often judged together)

strongly co-relevant

Co-relevance ≠ textual similarity.
d_1 and d_3 may look different but answer the same query.
QUAM learns this structure from reranker feedback.

QUAM — reranker feedback guides the next batch

Scored docs reveal neighbors · neighborhood lookup selects next candidates

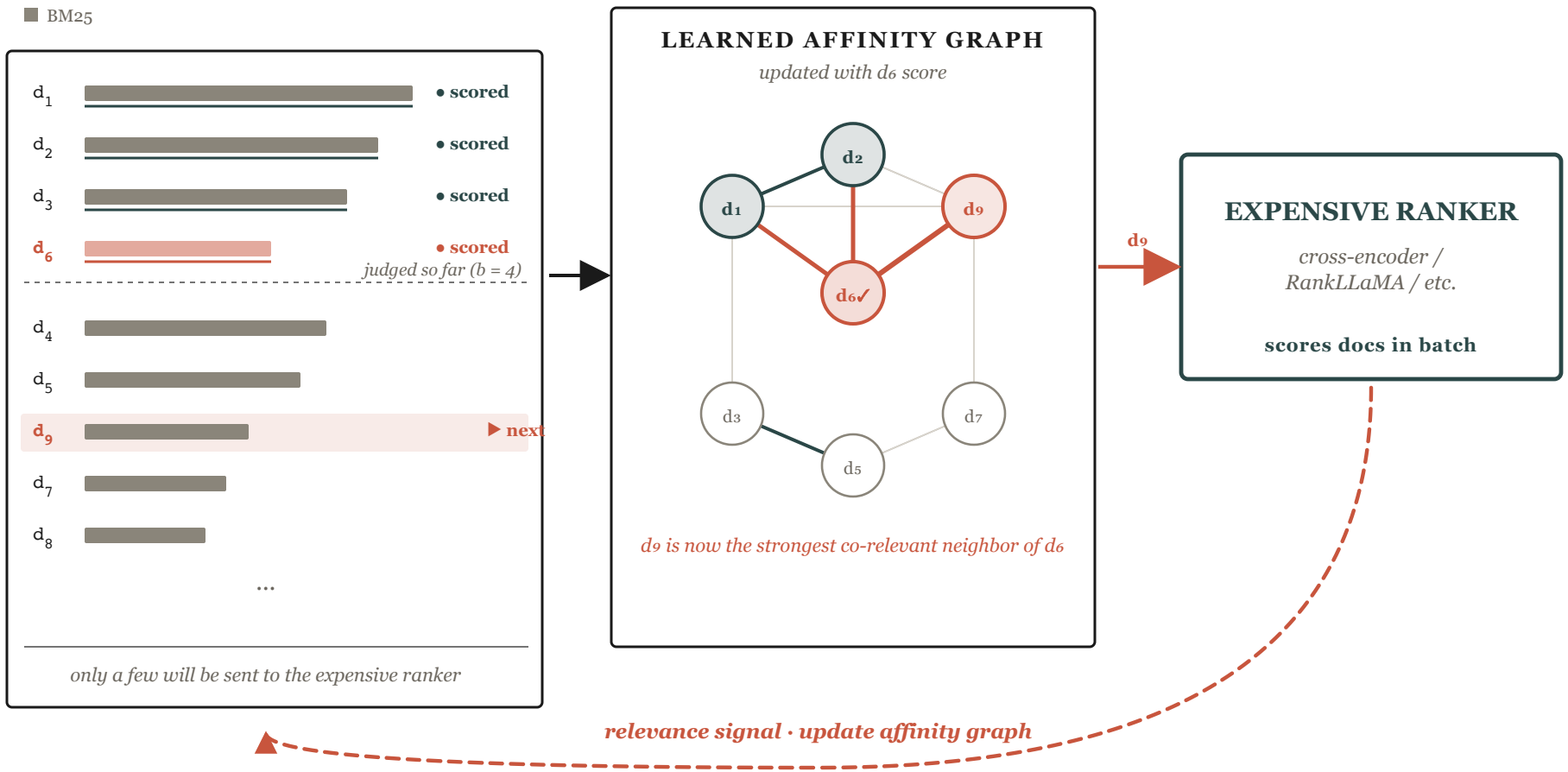


The affinity graph tells us: *given that d1 and d2 scored well, d6 is likely to be relevant too.*

Next doc chosen by co-relevance proximity — not raw BM25 rank.

QUAM — feedback updates the affinity lookup

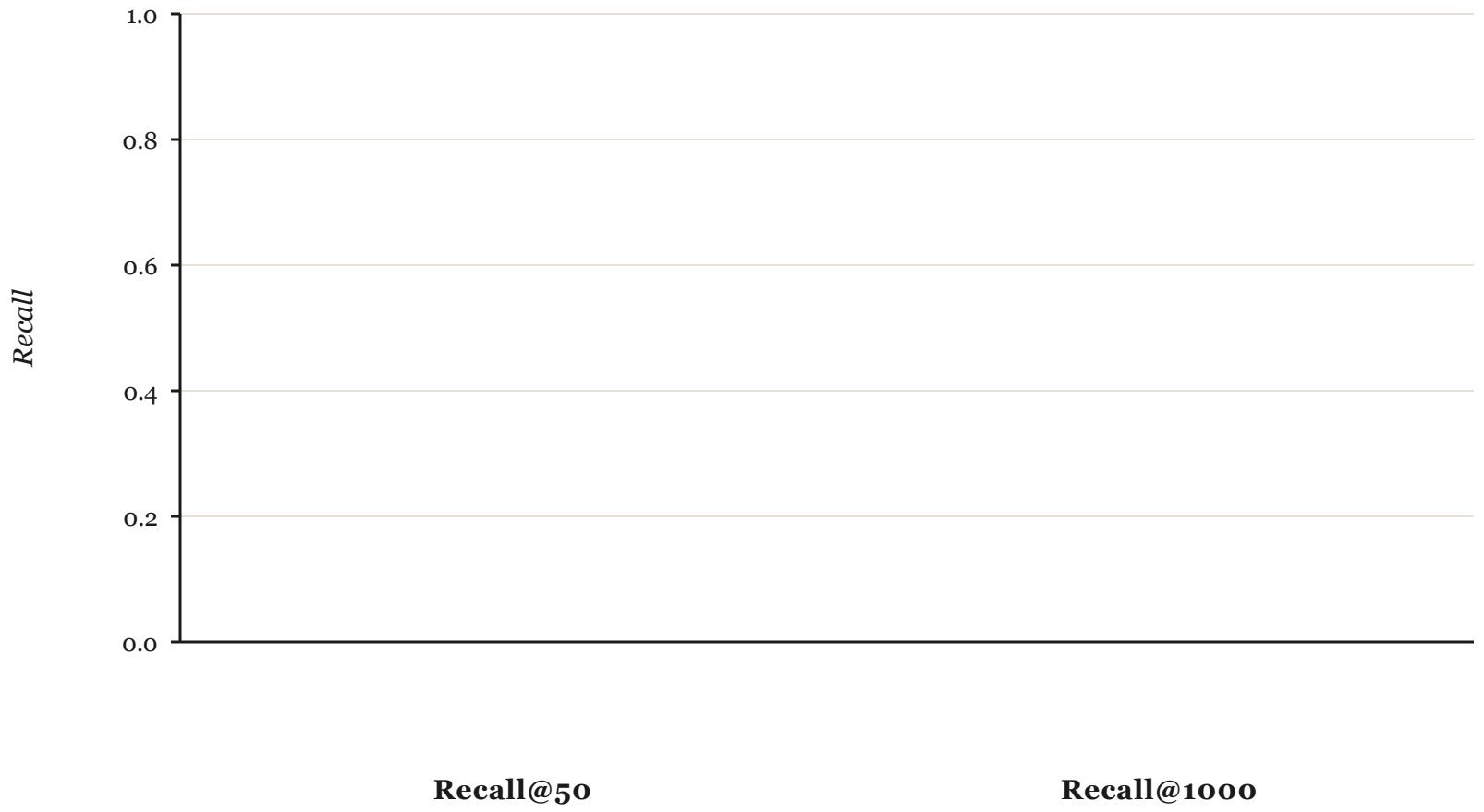
d6 has been scored — the next pick d9 comes from the updated neighborhood



Each scored document updates the graph — the next pick follows the strongest co-relevant edge.

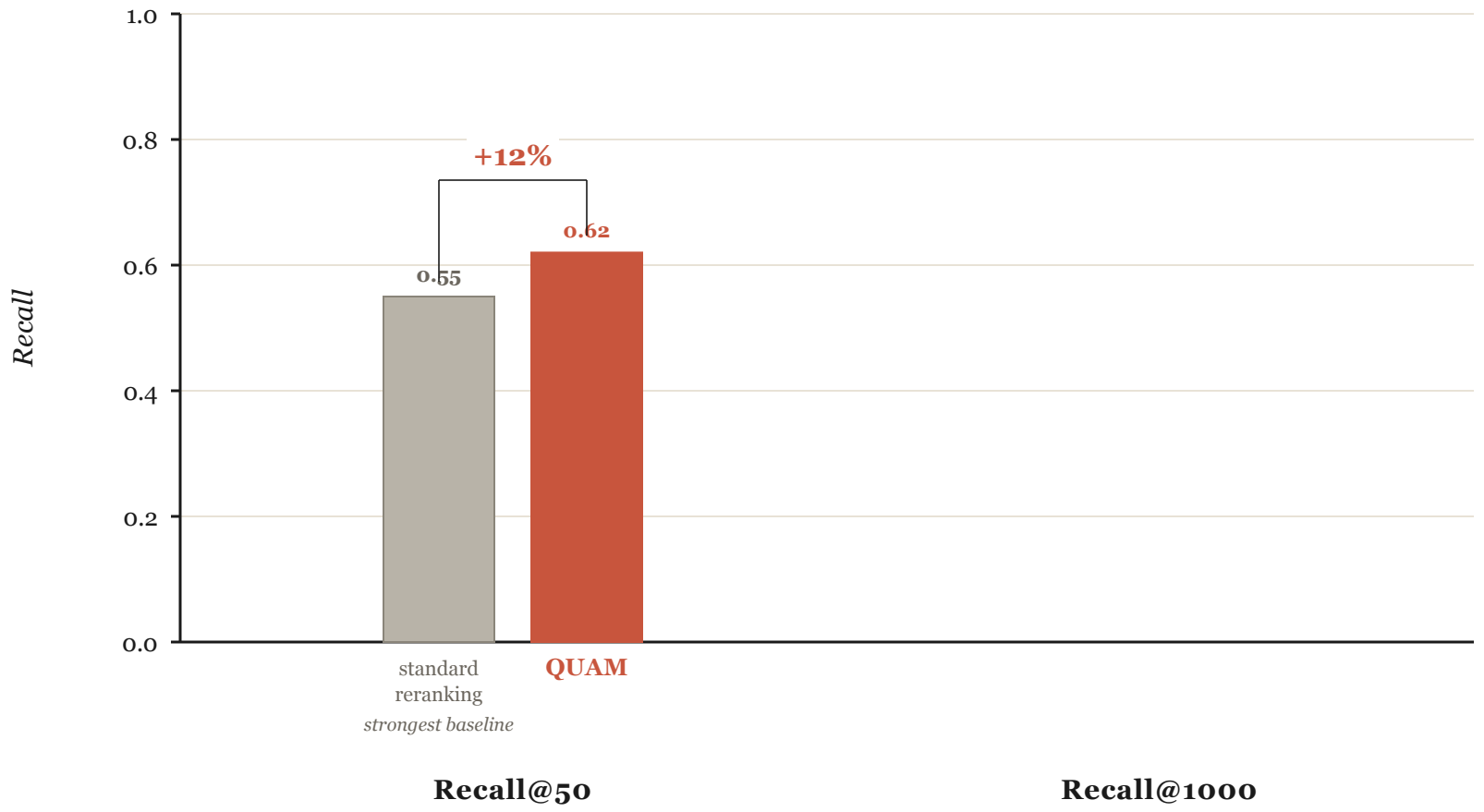
QUAM — TREC DL19 / DL20

Recall improvements over the strongest adaptive-retrieval baseline



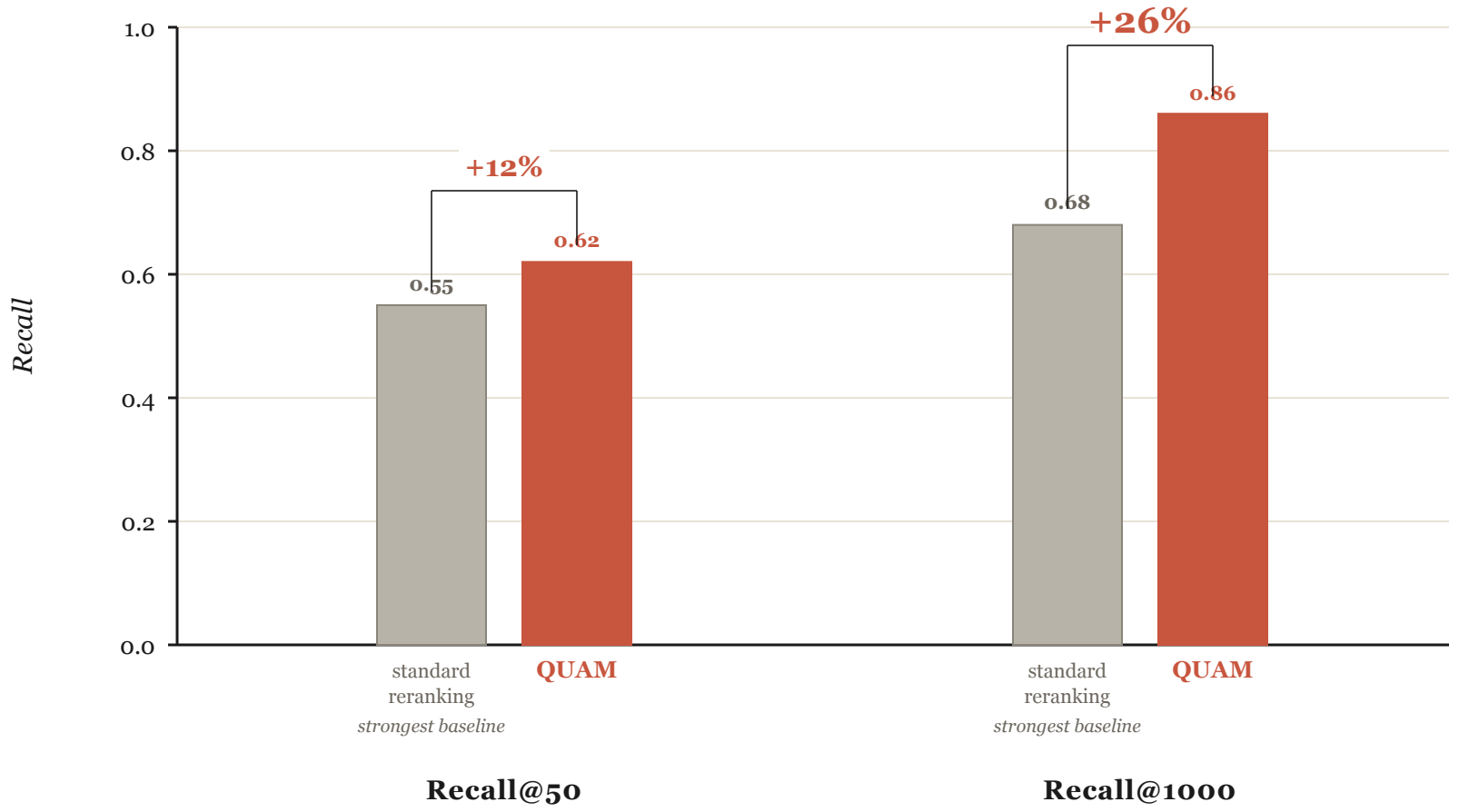
QUAM — TREC DL19 / DL20

Recall improvements over the strongest adaptive-retrieval baseline



QUAM — TREC DL19 / DL20

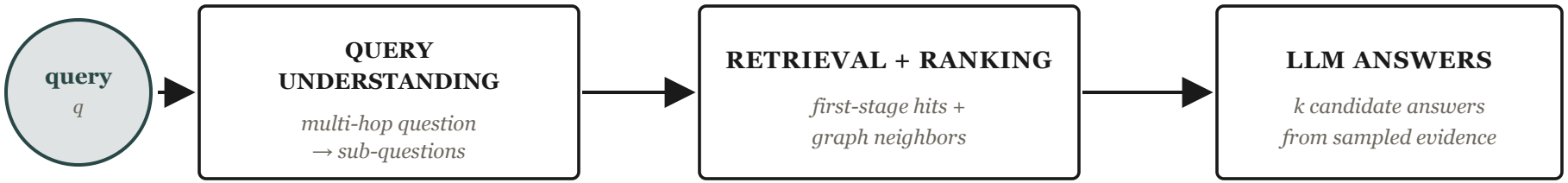
Recall improvements over the strongest adaptive-retrieval baseline



Latency overhead: ~3% of total reranking cost. The retriever isn't fixed — it learns from the reranker, query by query.

What if the feedback signal was not a point-wise score?

From point-wise reranker scores to richer feedback signals



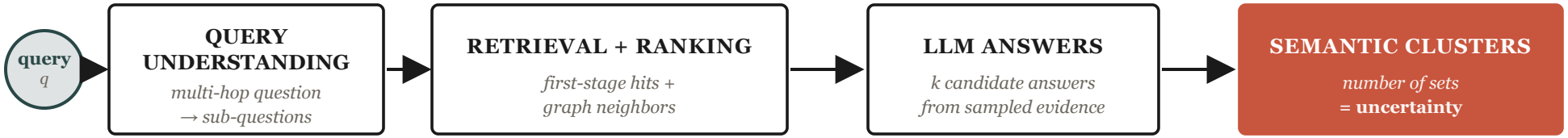
What if the output was just an answer —
what feedback signal could you use?

SUNAR — uncertainty as the feedback signal

LLM answer consistency drives next-iteration document retrieval

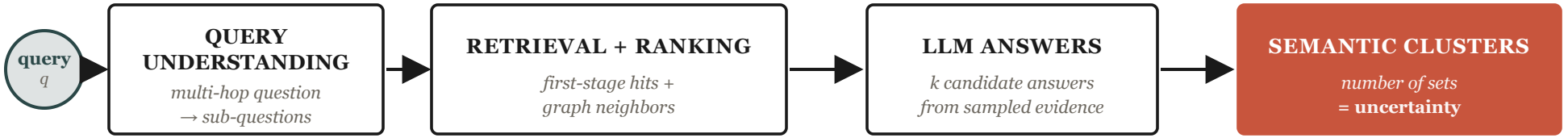
SUNAR — uncertainty as the feedback signal

LLM answer consistency drives next-iteration document retrieval



SUNAR — uncertainty as the feedback signal

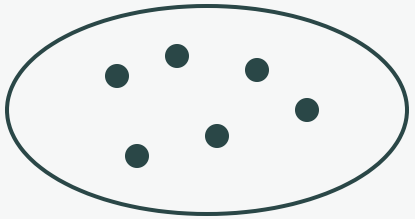
LLM answer consistency drives next-iteration document retrieval



HOW UNCERTAINTY IS COMPUTED

Consistent answers

1 semantic cluster · low uncertainty



evidence is good

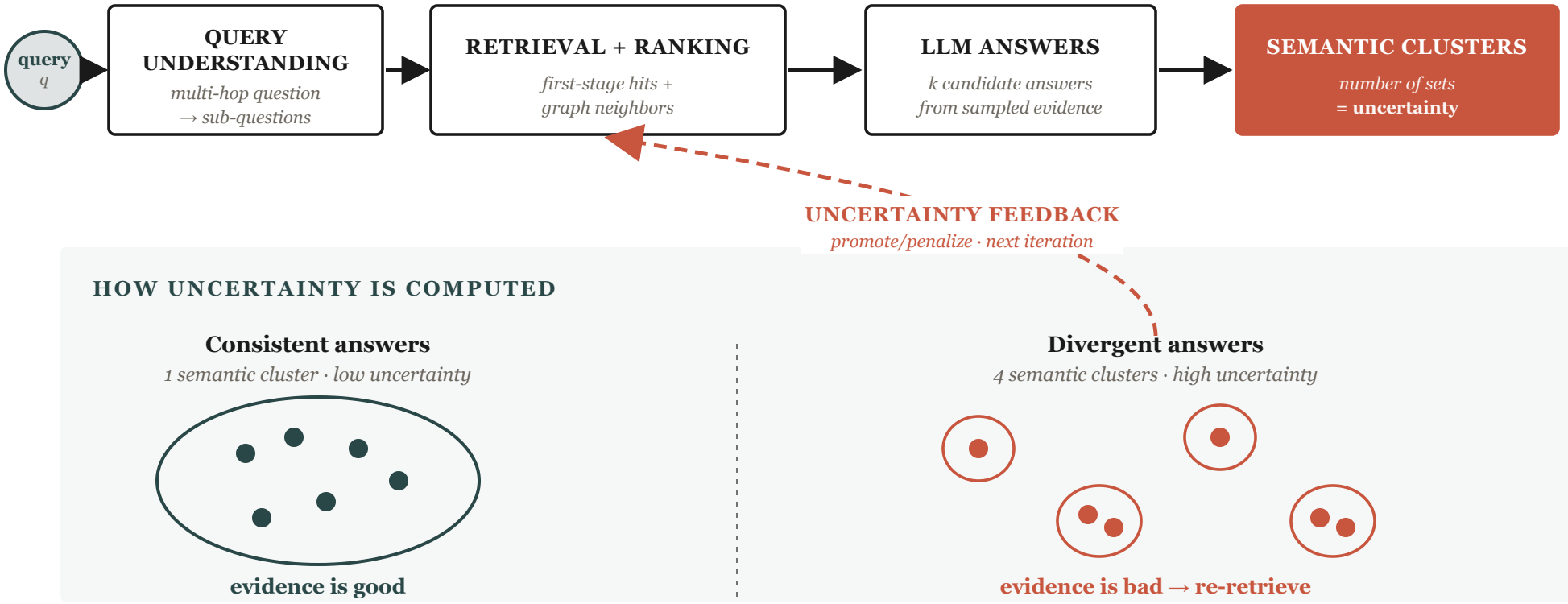
Divergent answers

4 semantic clusters · high uncertainty

evidence is bad → re-retrieve

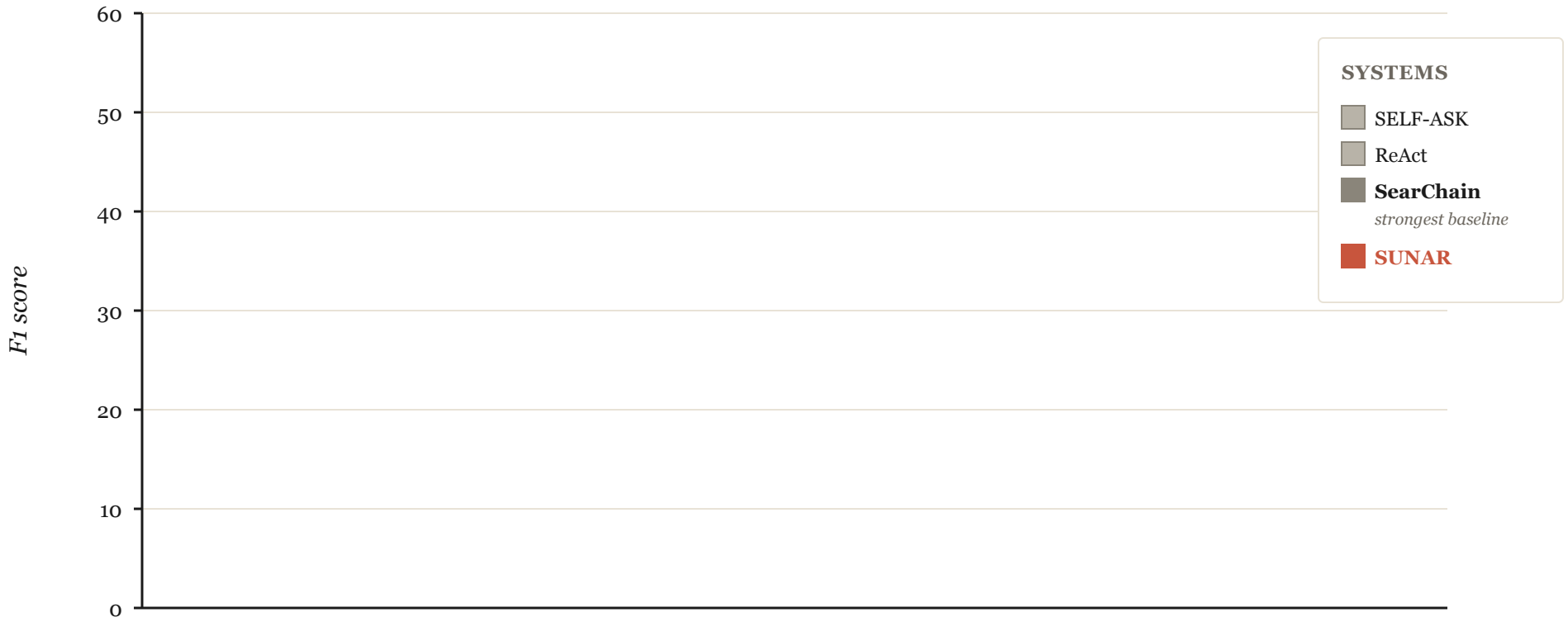
SUNAR — uncertainty as the feedback signal

LLM answer consistency drives next-iteration document retrieval



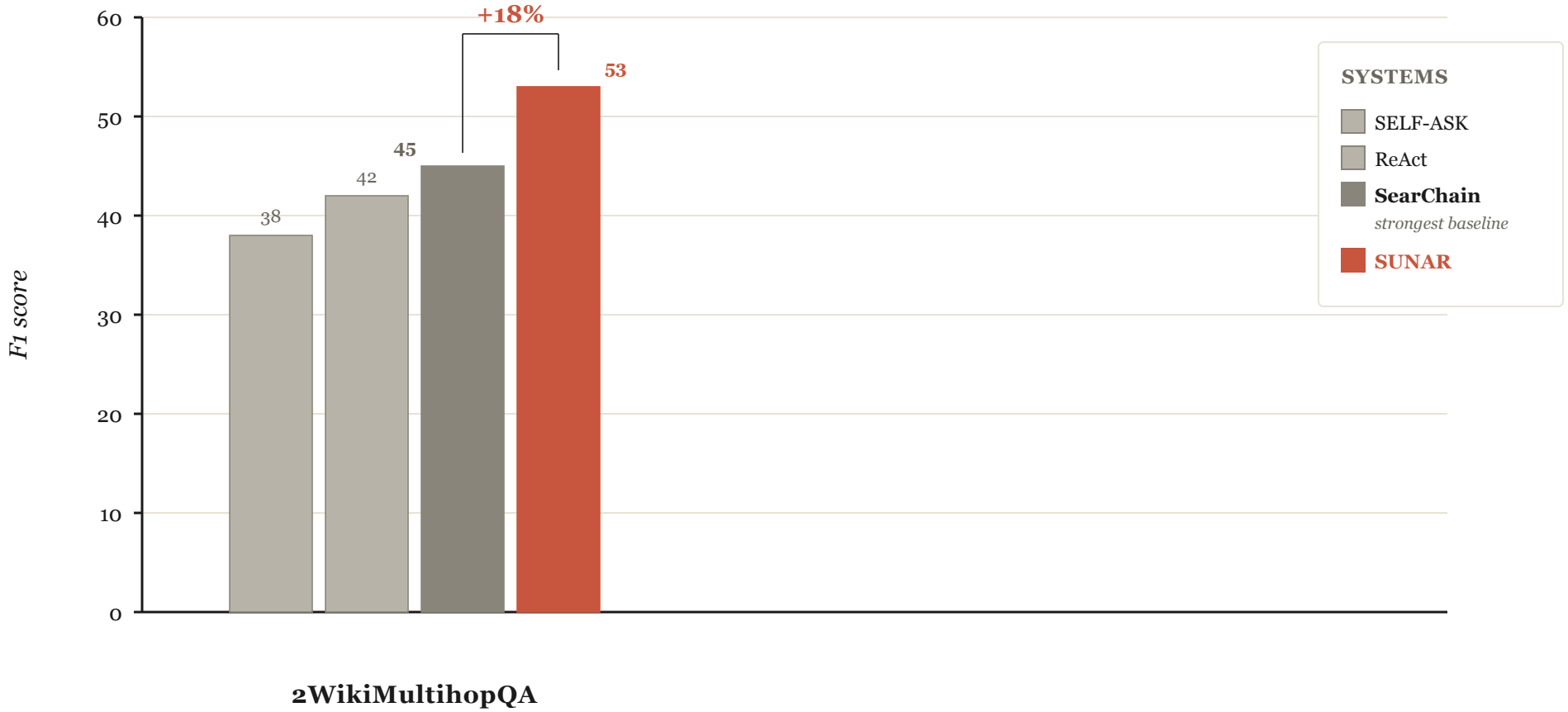
SUNAR — multi-hop QA

F1 score vs the strongest decomposition baselines · LLM-agnostic



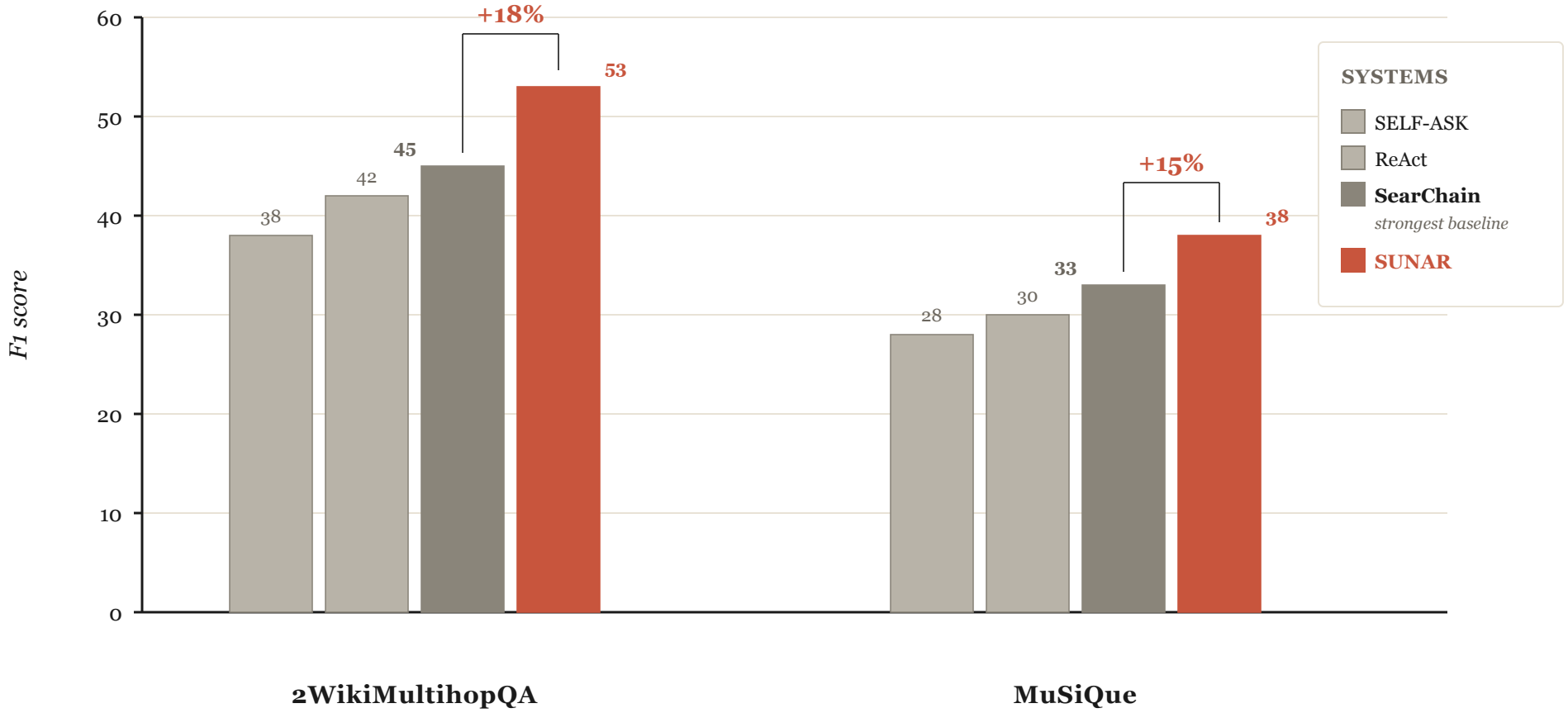
SUNAR — multi-hop QA

F1 score vs the strongest decomposition baselines · LLM-agnostic



SUNAR — multi-hop QA

F1 score vs the strongest decomposition baselines · LLM-agnostic



LLM-agnostic — gains hold across GPT-3.5, GPT-4o-mini, Llama, and Mistral.

Uncertainty itself becomes the relevance signal.

> *With QUAM and SUNAR, we understood what kind of feedback signals could be used — and how to organize a representation space and explore it cleverly to improve retrieval.*

> *But we are too tied to the representation space. What if there are different aspects of relevance, just like in Learning to Rank?*

Different queries need different signals

"Marriott London Bridge" — exact match. Lexical / BM25 wins.
Neighborhood signals just add noise.

"romantic weekend getaway near Amsterdam" — vague. Lexical fails.
Embedding similarity and document neighborhood matter.

"kid-friendly hotel with pool, walking distance to Sagrada Familia, under €200" — multi-constraint. No single signal is enough.

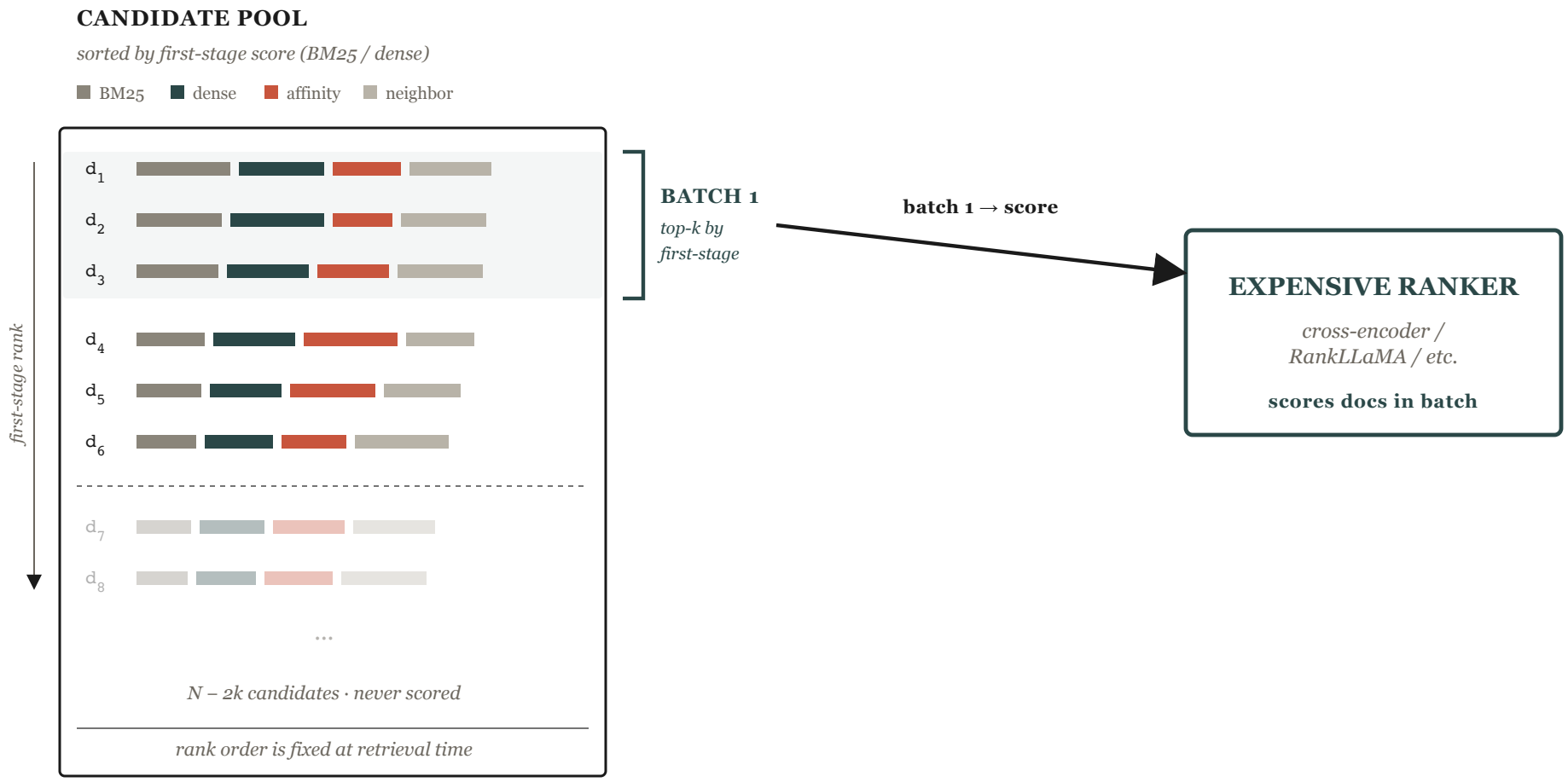
The standard pipeline picks **one weighting** of these signals at training time.

That weighting is wrong for most queries.

What if the system picked the weighting per query, online, from reranker feedback?

Static batching — the baseline

Rank by first-stage score · peel off batches in order · never look back

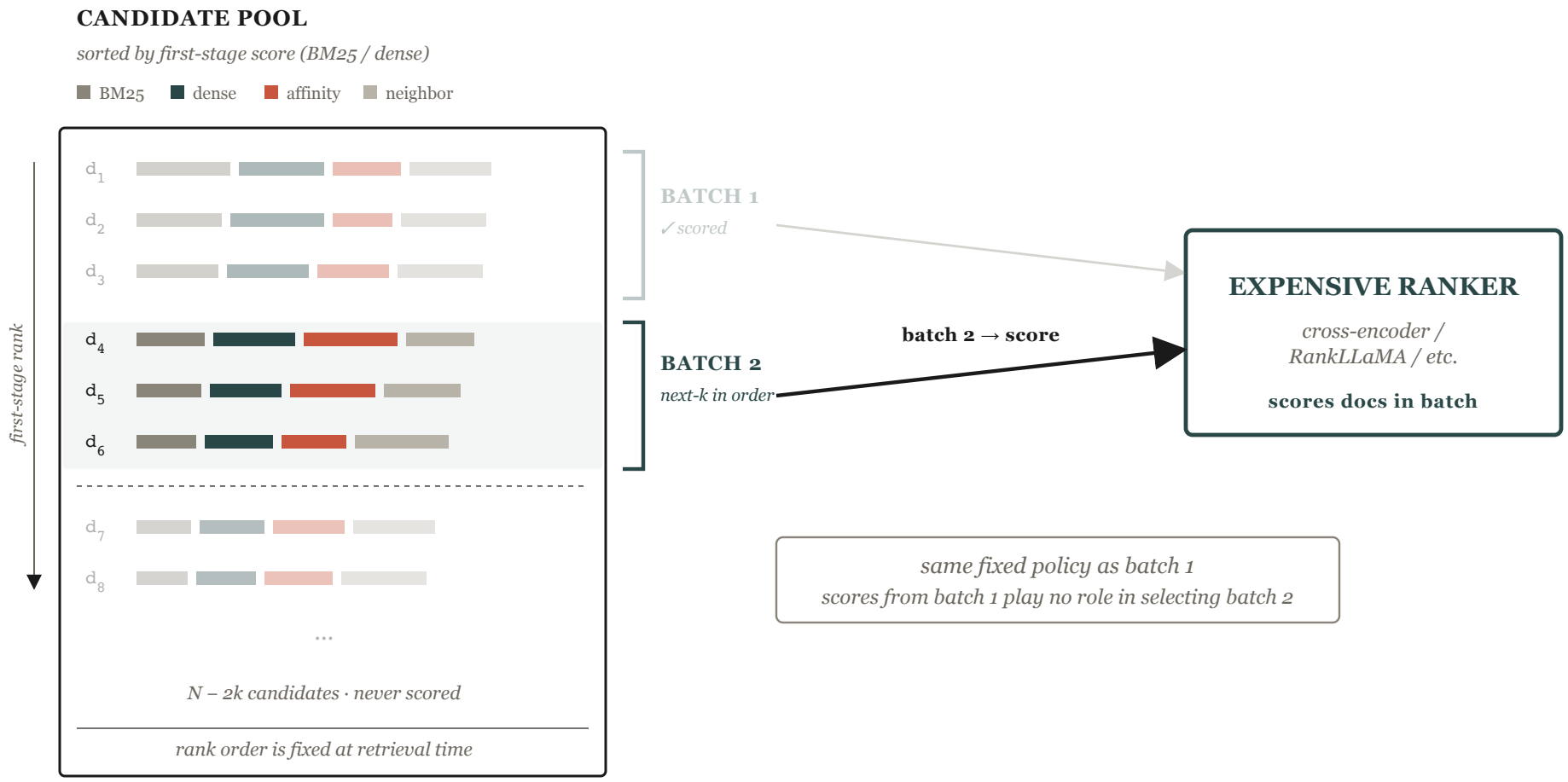


Reranker scores *order the output* — but never decide what comes next.

The information from batch 1 is thrown away. ORE is what happens when you keep it.

Static batching — the baseline

Rank by first-stage score · peel off batches in order · never look back



Reranker scores *order the output* — but never decide what comes next.

The information from batch 1 is thrown away. ORE is what happens when you keep it.

Algorithm 3 — ORE — Online Relevance Estimation

QUAM and SUNAR pick the next batch using a **fixed** affinity structure.

ORE goes further: it learns *online*, per query, **which signals matter for this query**.

A **linear stochastic bandit** over the candidate pool. Each document is an arm; features are simple, well-known relevance factors; rewards are the expensive ranker's scores.

[16] Rathee, Venktesh V, MacAvaney, Anand. Breaking the Lens of the Telescope: Online Relevance Estimation over Large Retrieval Sets. SIGIR 2025.

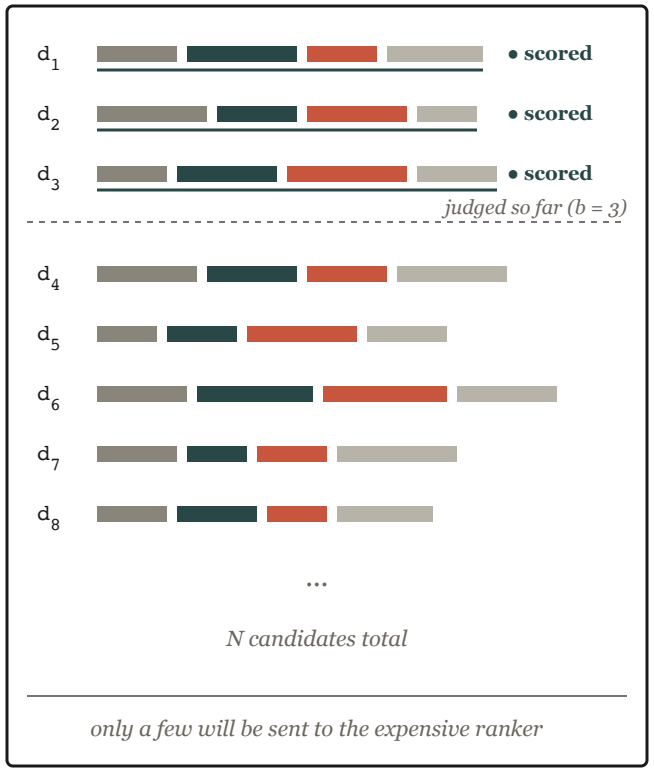
ORE — Online Relevance Estimation

A linear bandit learns which signals matter — per query, online

CANDIDATE POOL

each doc carries a feature vector

BM25 dense affinity neighbor



LINEAR BANDIT

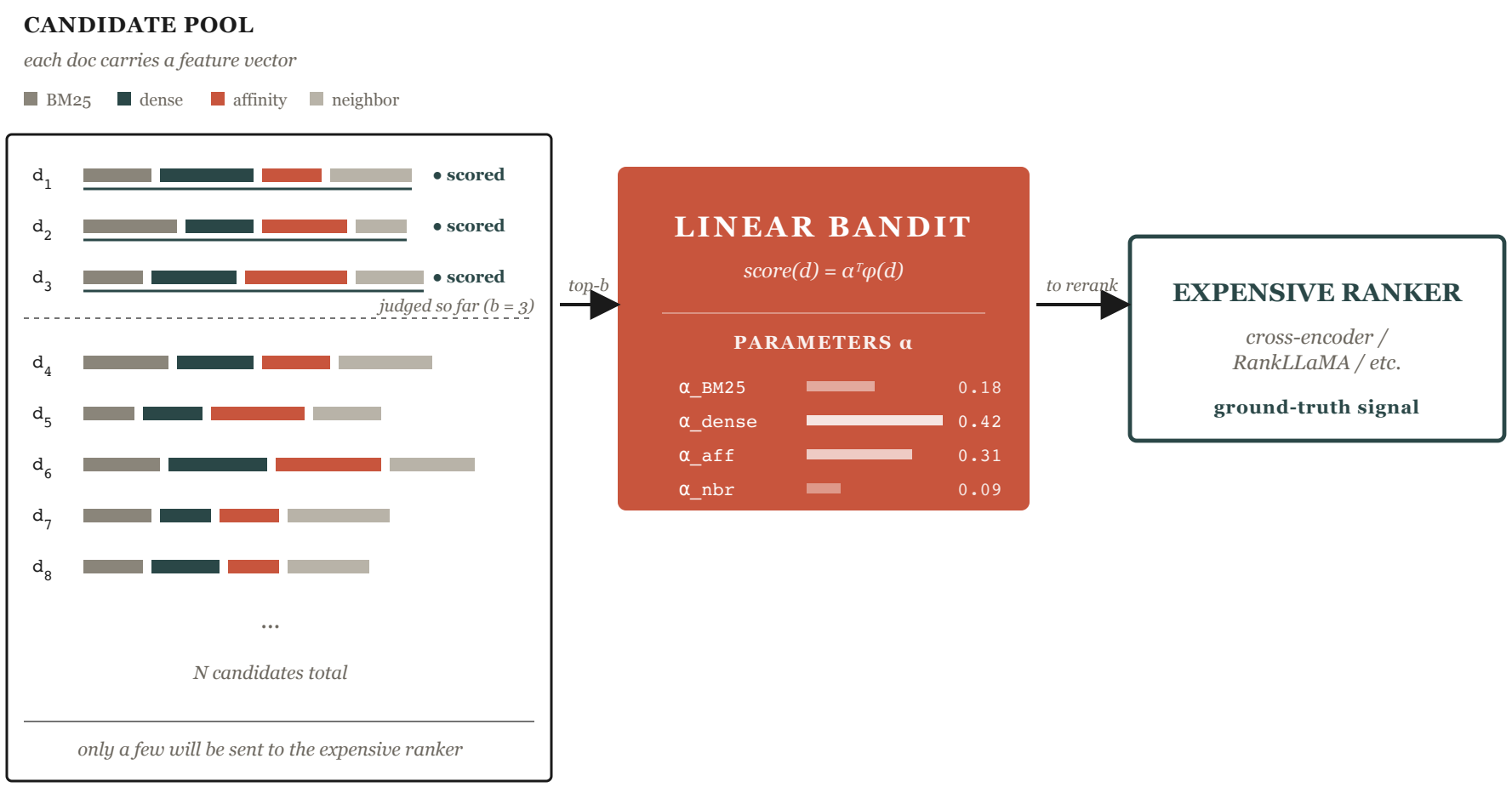
$score(d) = \alpha^T \varphi(d)$

PARAMETERS α

α_BM25		0.18
α_dense		0.42
α_aff		0.31
α_nbr		0.09

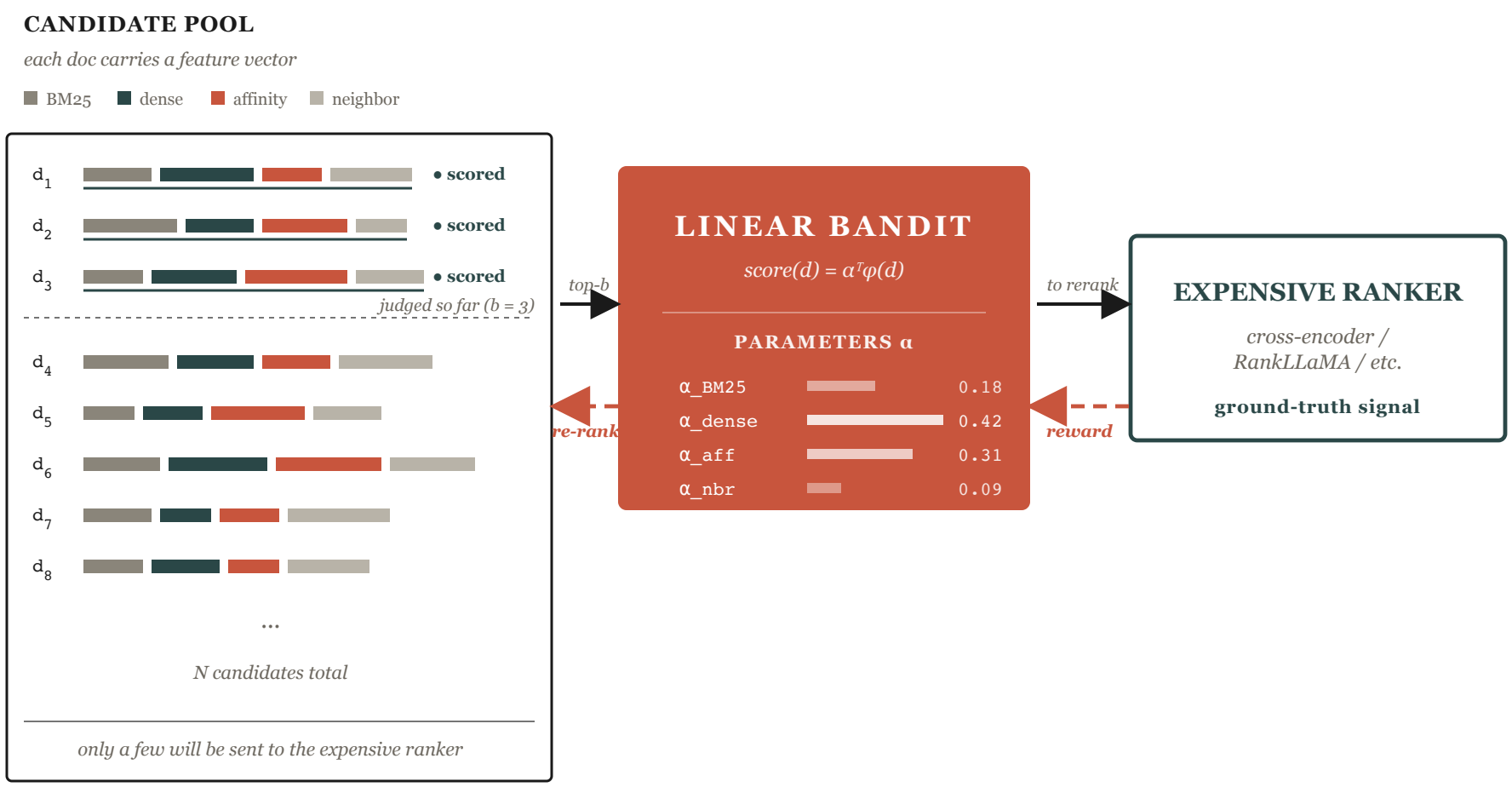
ORE — Online Relevance Estimation

A linear bandit learns which signals matter — per query, online



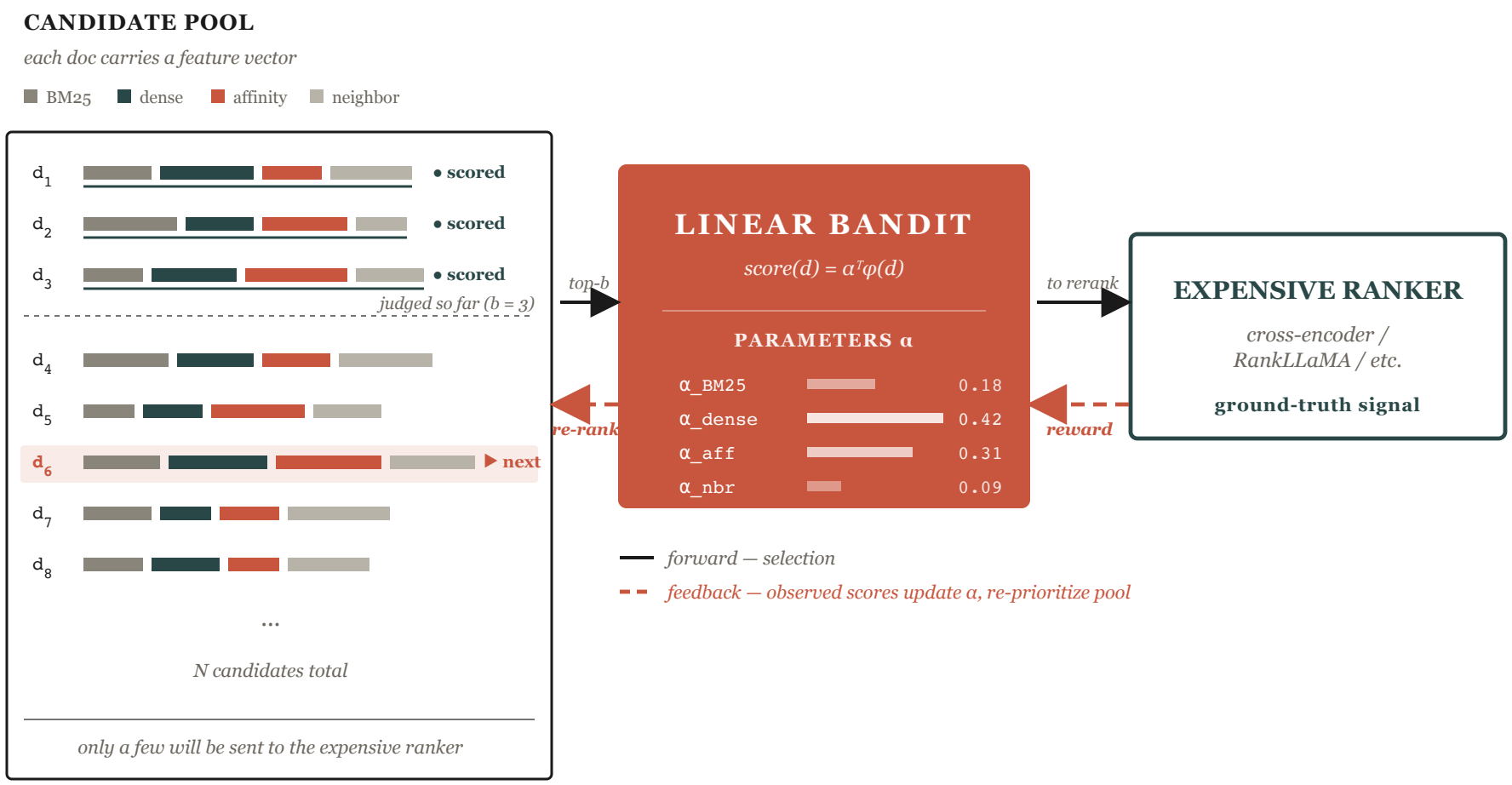
ORE — Online Relevance Estimation

A linear bandit learns which signals matter — per query, online



ORE — Online Relevance Estimation

A linear bandit learns which signals matter — per query, online



The bandit doesn't just *pick* documents —
it learns **which features predict relevance for this query**.

QUAM, SUNAR: fixed signal structure
ORE: signal weights learned online, per query

ORE — speedup and recall, both at once

Hybrid retrieval and adaptive retrieval, in one framework

SPEEDUP OVER QUAM

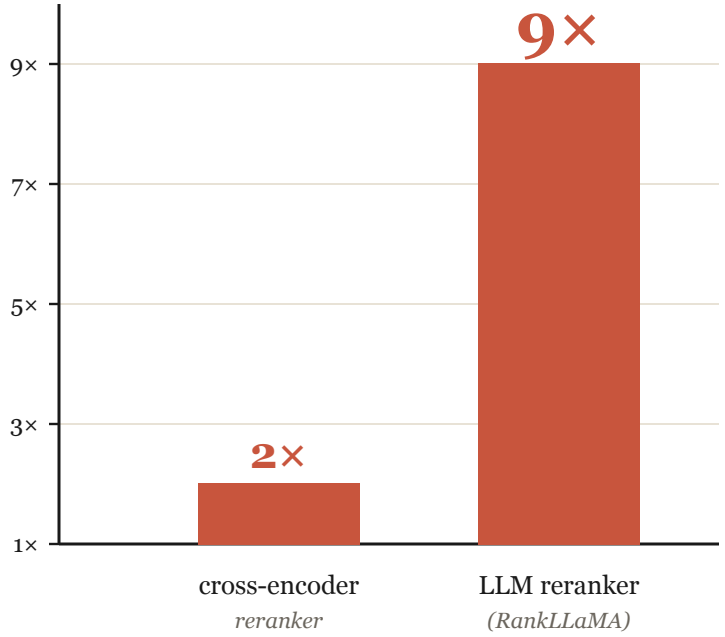
same recall, fewer reranker calls

ORE — speedup and recall, both at once

Hybrid retrieval and adaptive retrieval, in one framework

SPEEDUP OVER QUAM

same recall, fewer reranker calls

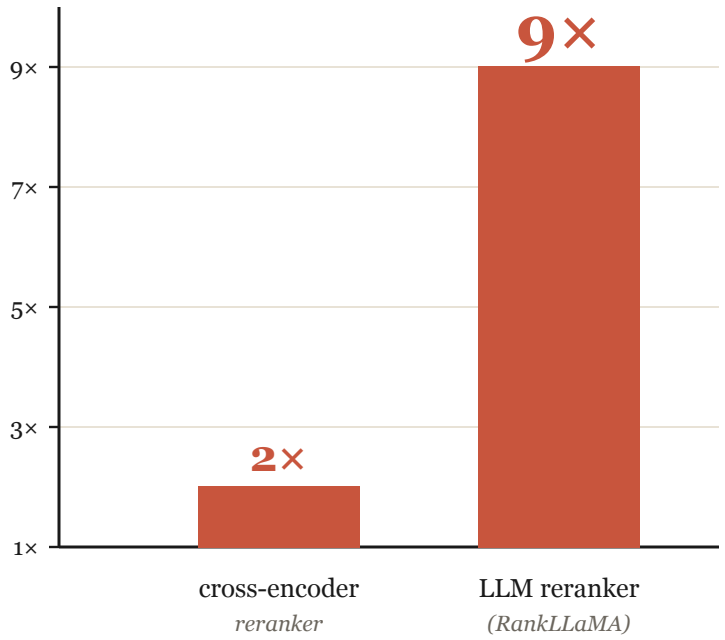


ORE — speedup and recall, both at once

Hybrid retrieval and adaptive retrieval, in one framework

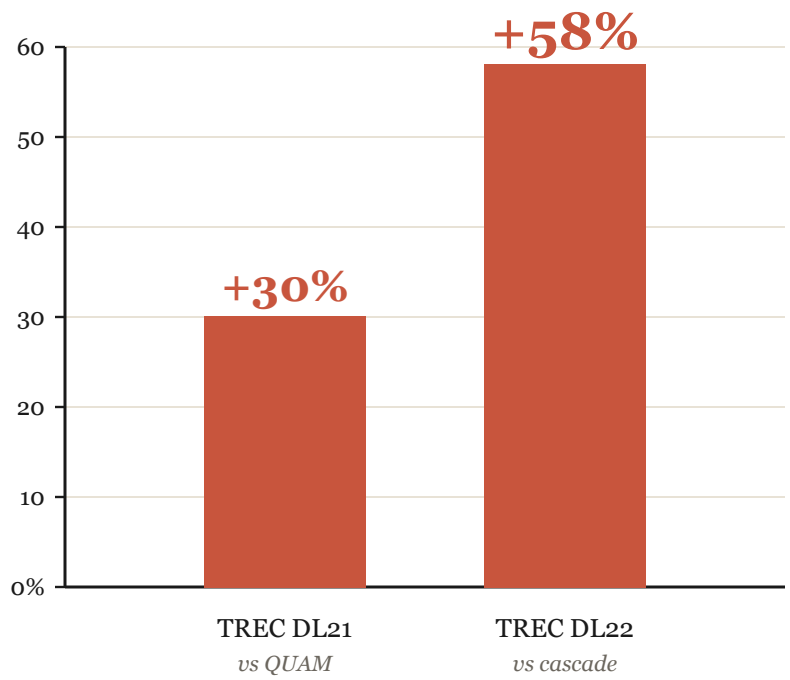
SPEEDUP OVER QUAM

same recall, fewer reranker calls



RECALL@50 IMPROVEMENT

same compute, more relevant candidates



ORE pushes **both frontiers at once** — quality and cost.

The bandit doesn't just pick documents. It learns which features predict relevance for this query.

But signals aren't the only thing we can pick per query

ORE asks: *given this query, which **relevance signals** should I trust?*

Now ask the same question one level up.

ReformIR — picking reformulations per query

Failure 2: more reformulations → more drift.

Remember the Tokyo example? Four reformulations, one of them useful, three of them drifty.

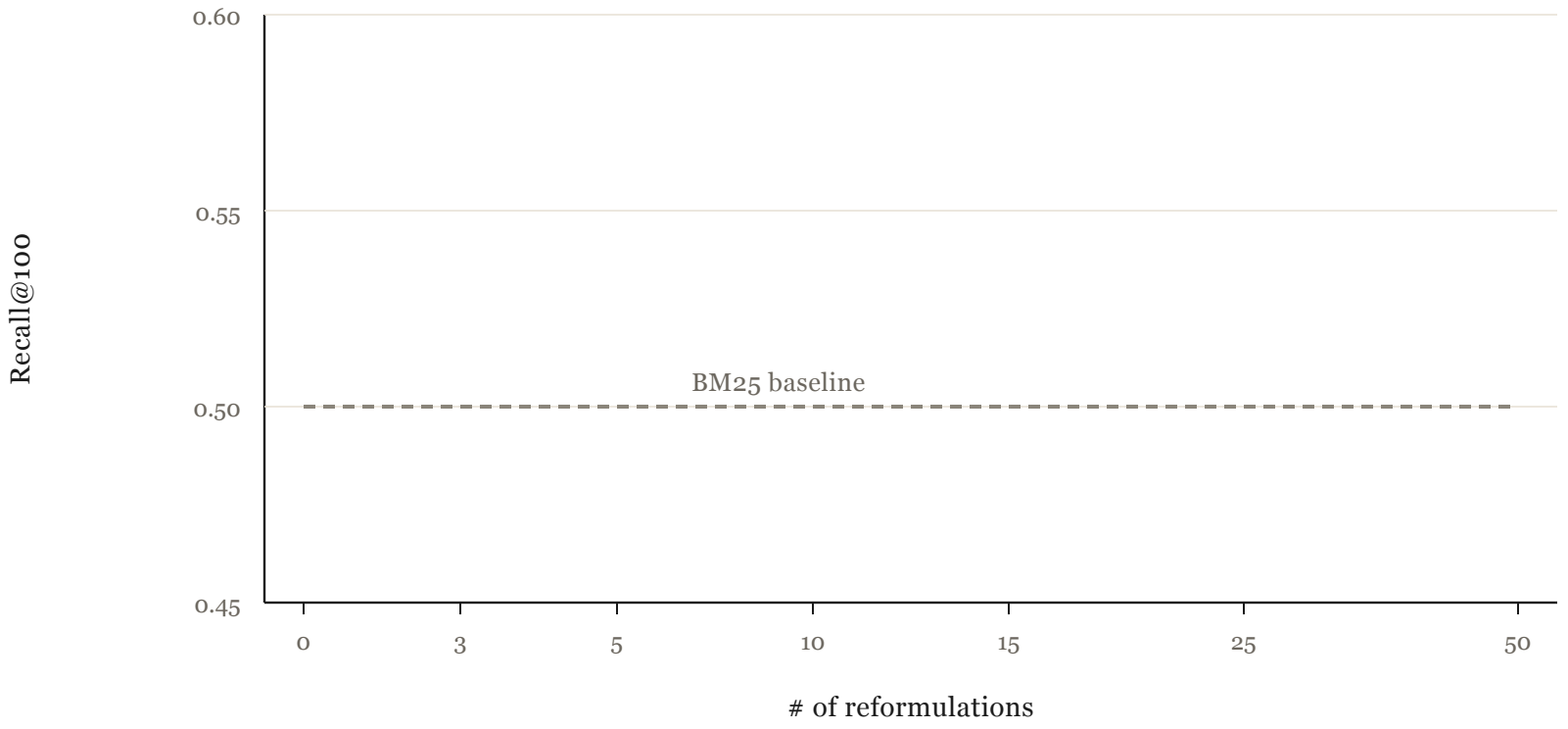
The fix isn't fewer reformulations. It's **picking which ones to trust, per query, online, from reranker feedback**.

[5] Venktesh V, Rathee, Anand. When More Reformulations Hurt: Avoiding Drift using Ranker Feedback. SIGIR 2026.

More reformulations → more drift

Classical reformulation degrades past ~10 reformulations · ReformIR stays stable

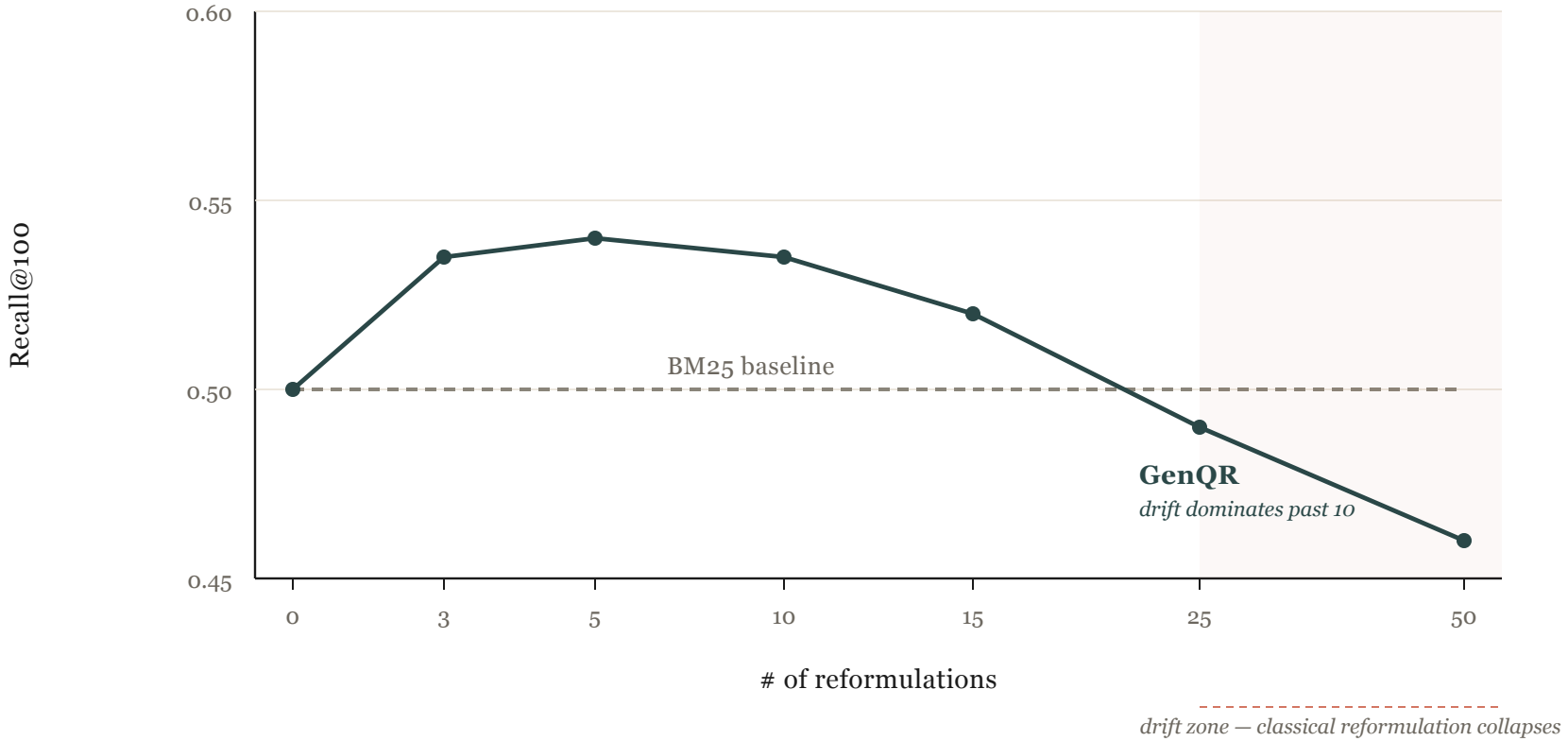
TREC DL19 · BM25 → MonoT5 (budget c=100)



More reformulations → more drift

Classical reformulation degrades past ~10 reformulations · ReformIR stays stable

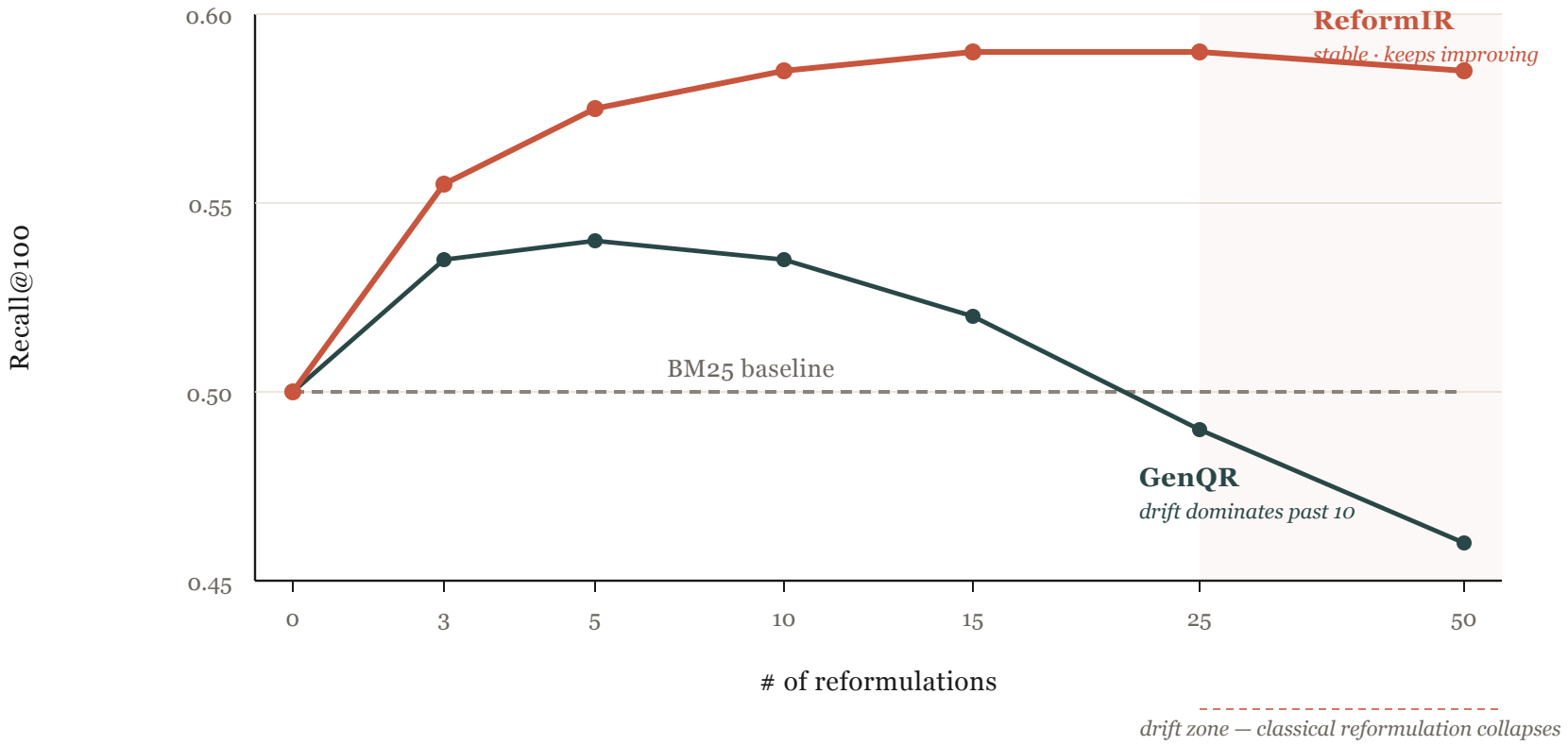
TREC DL19 · BM25 → MonoT5 (budget c=100)



More reformulations → more drift

Classical reformulation degrades past ~10 reformulations · ReformIR stays stable

TREC DL19 · BM25 → MonoT5 (budget c=100)



Adding more reformulations doesn't fix recall — **picking the right ones does.**

ReformIR uses ranker feedback to up-weight useful reformulations and down-weight drifty ones.

ReformIR — same idea as ORE, one level up

ORE puts **relevance signals** in the bandit's feature vector.

ReformIR puts **reformulations** in the bandit's feature vector.

The reranker always scores against the **original query**, anchoring the system against drift.

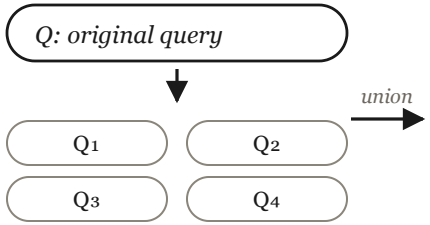
Drifty reformulations get downweighted. Useful ones get upweighted.

ReformIR — picking reformulations per query

Same bandit machinery as ORE — but the features are reformulations, not signals

REFORMULATOR

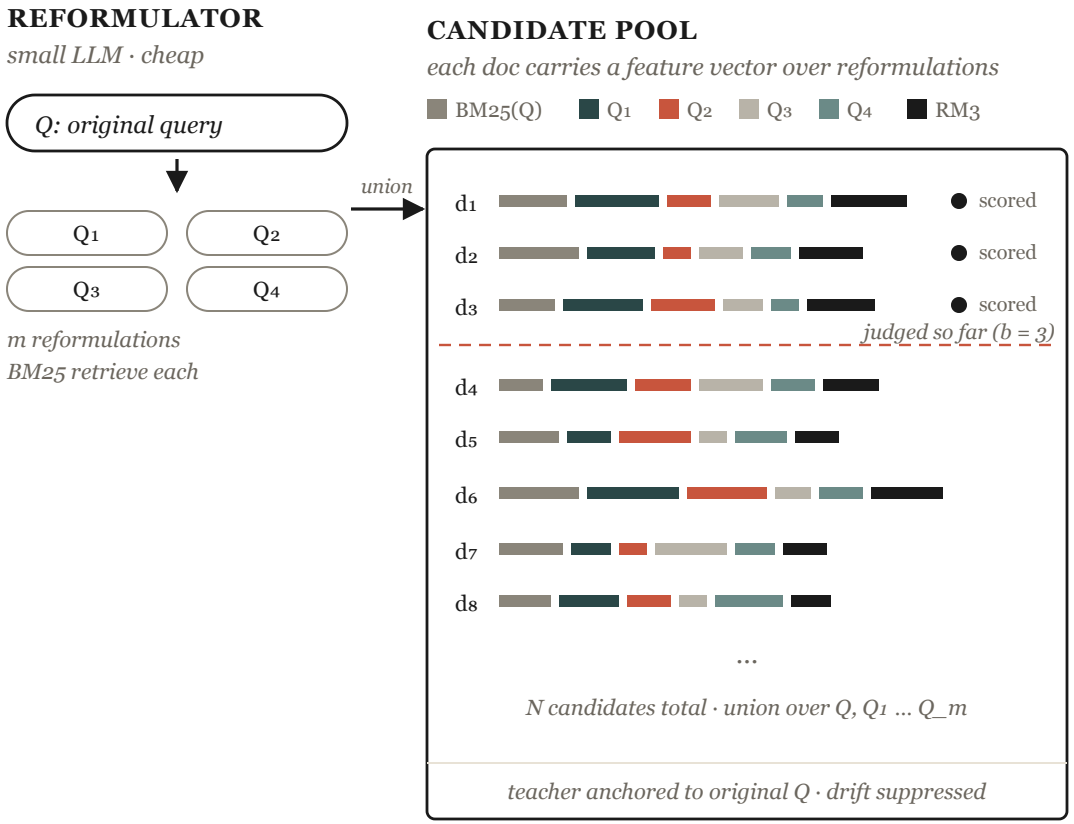
small LLM · cheap



m reformulations
BM25 retrieve each

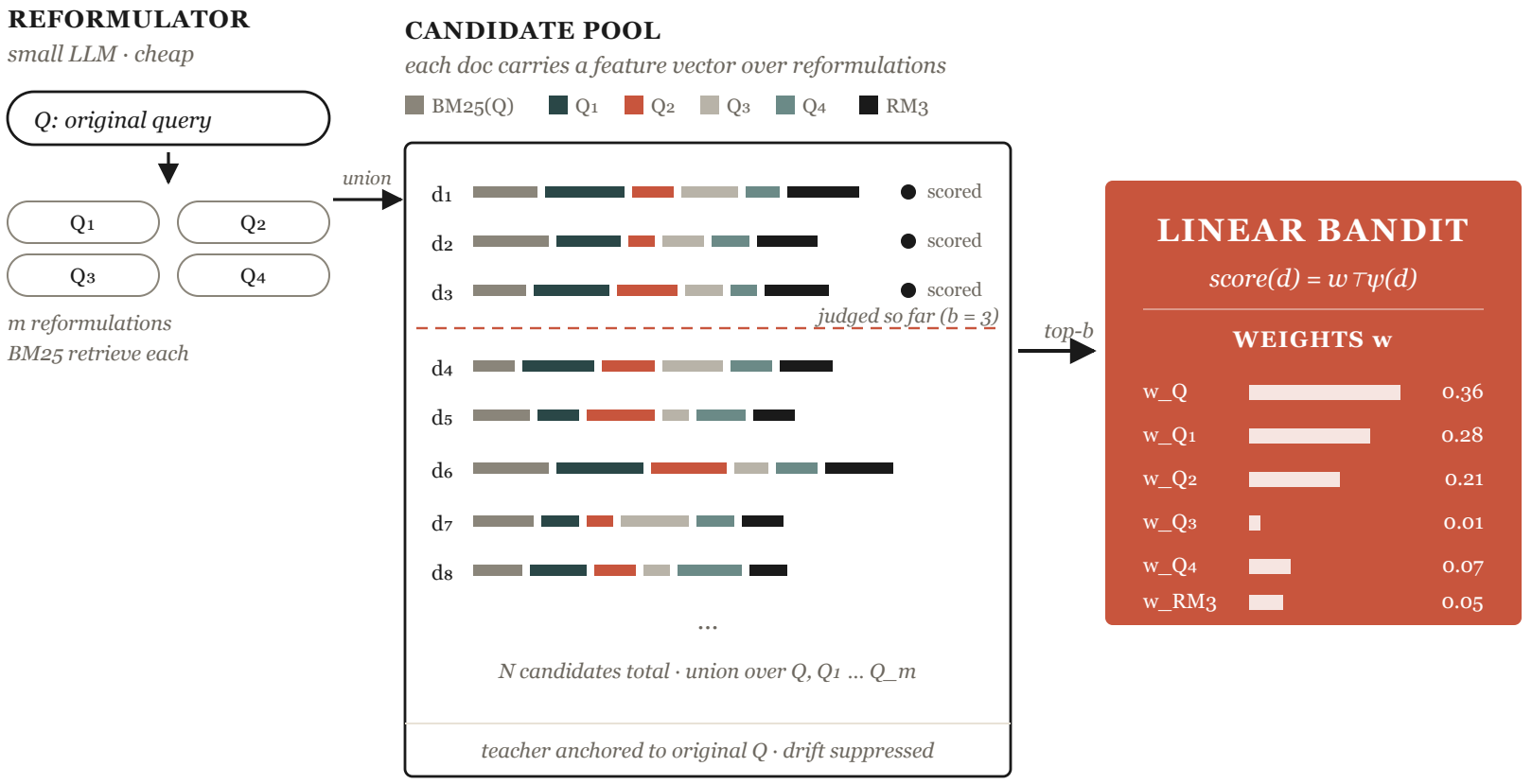
ReformIR — picking reformulations per query

Same bandit machinery as ORE — but the features are reformulations, not signals



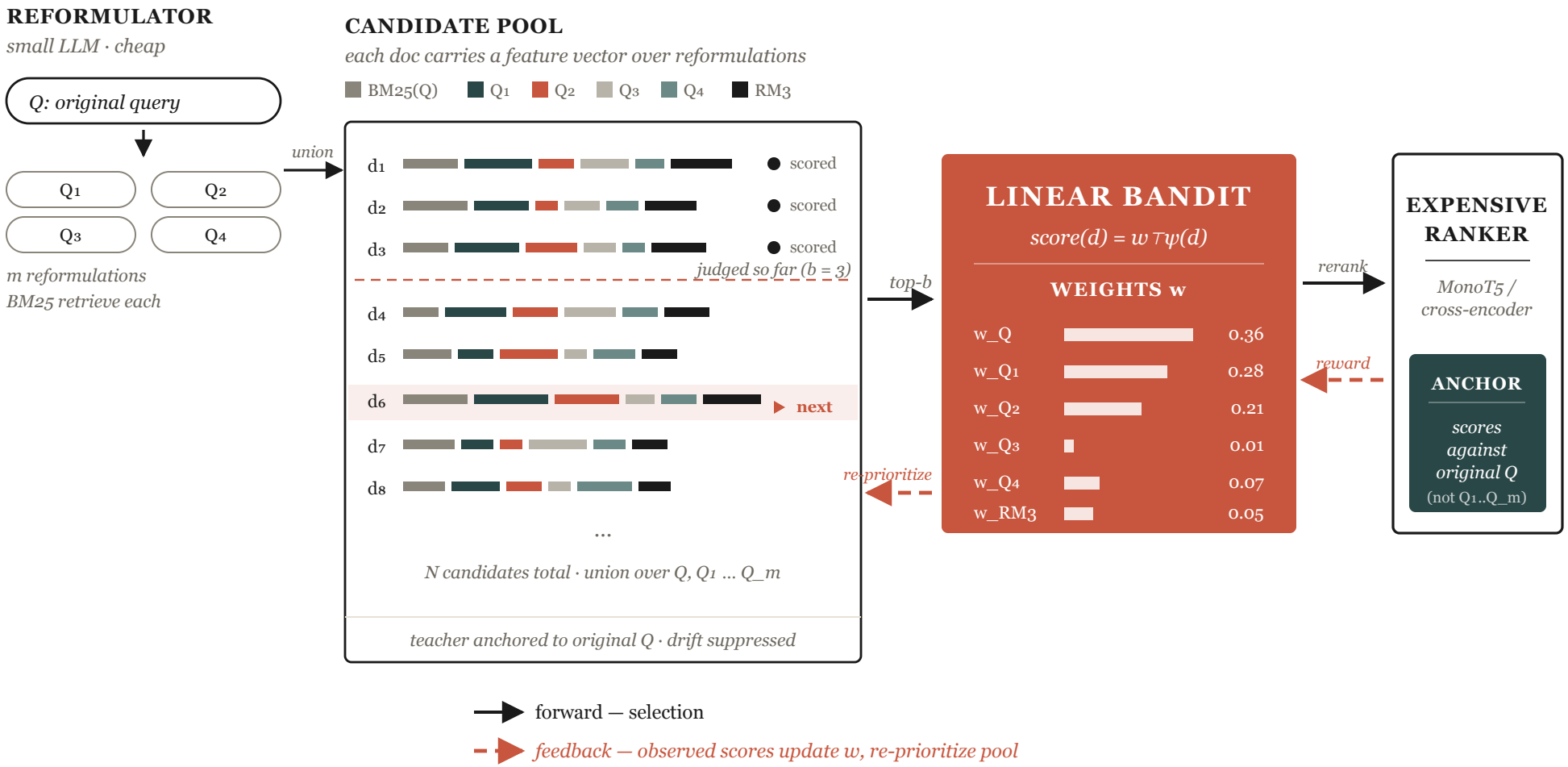
ReformIR — picking reformulations per query

Same bandit machinery as ORE — but the features are reformulations, not signals



ReformIR — picking reformulations per query

Same bandit machinery as ORE — but the features are reformulations, not signals

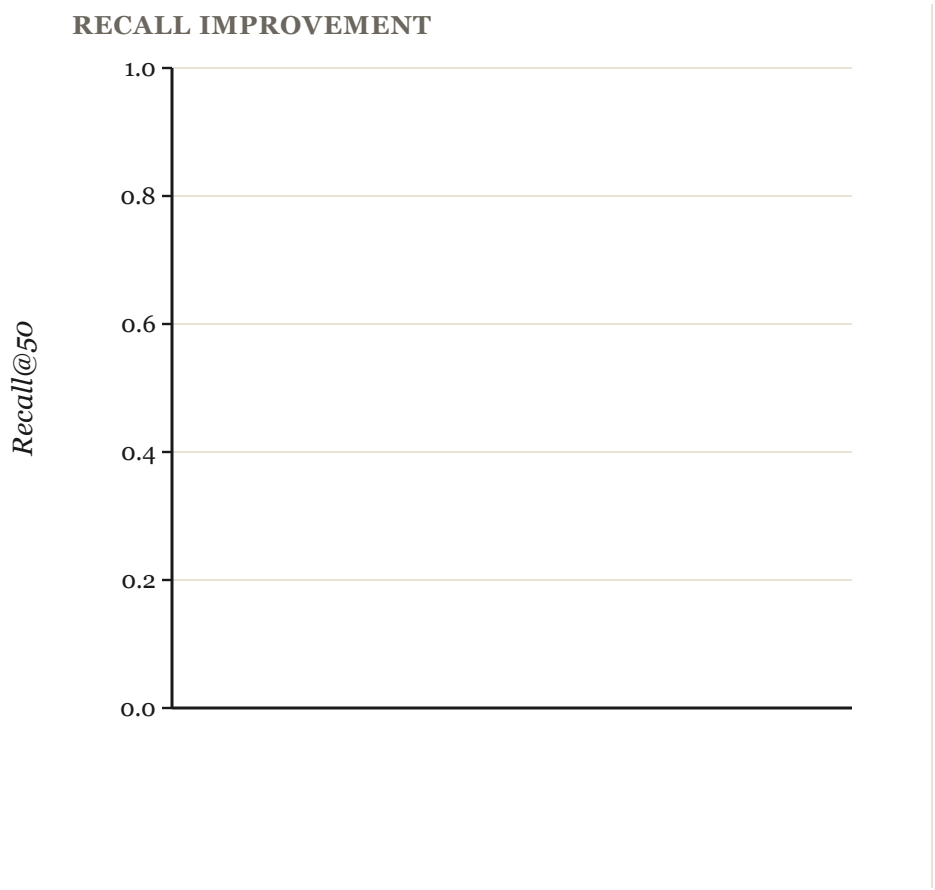


ORE puts *signals* in the feature vector. ReformIR puts *reformulations* in the feature vector.

Drifty reformulations get downweighted. Useful ones get upweighted. Per query. Online.

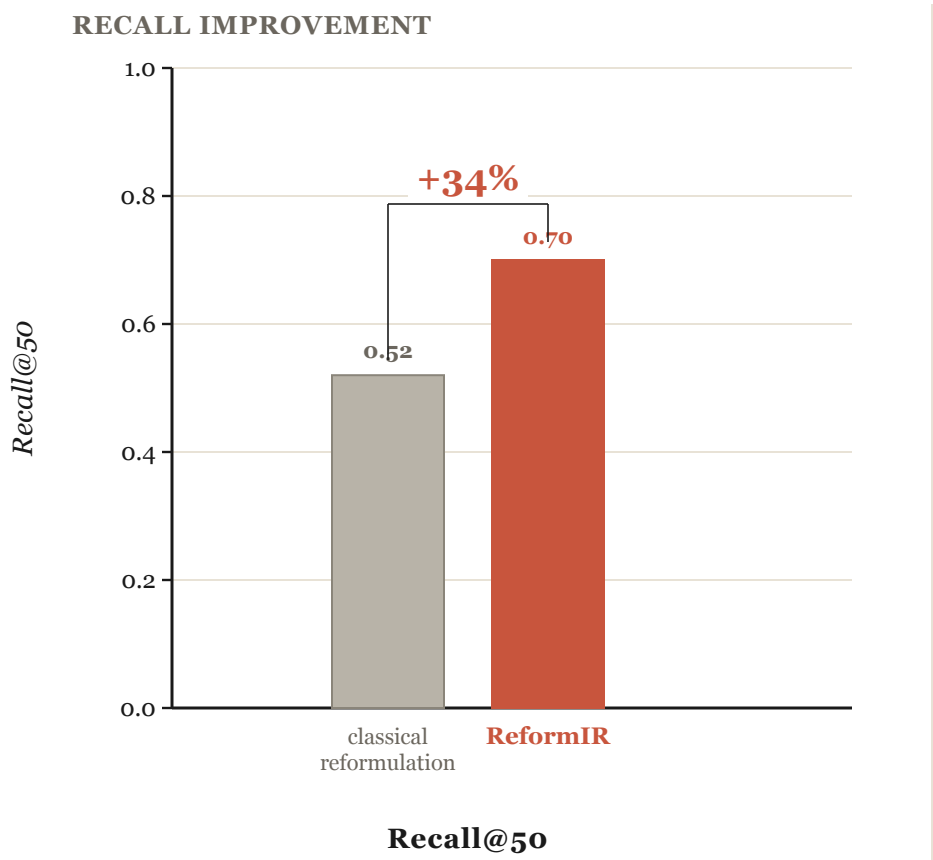
ReformIR — TREC DL19–DL22

Training-free adapter over any reformulation method



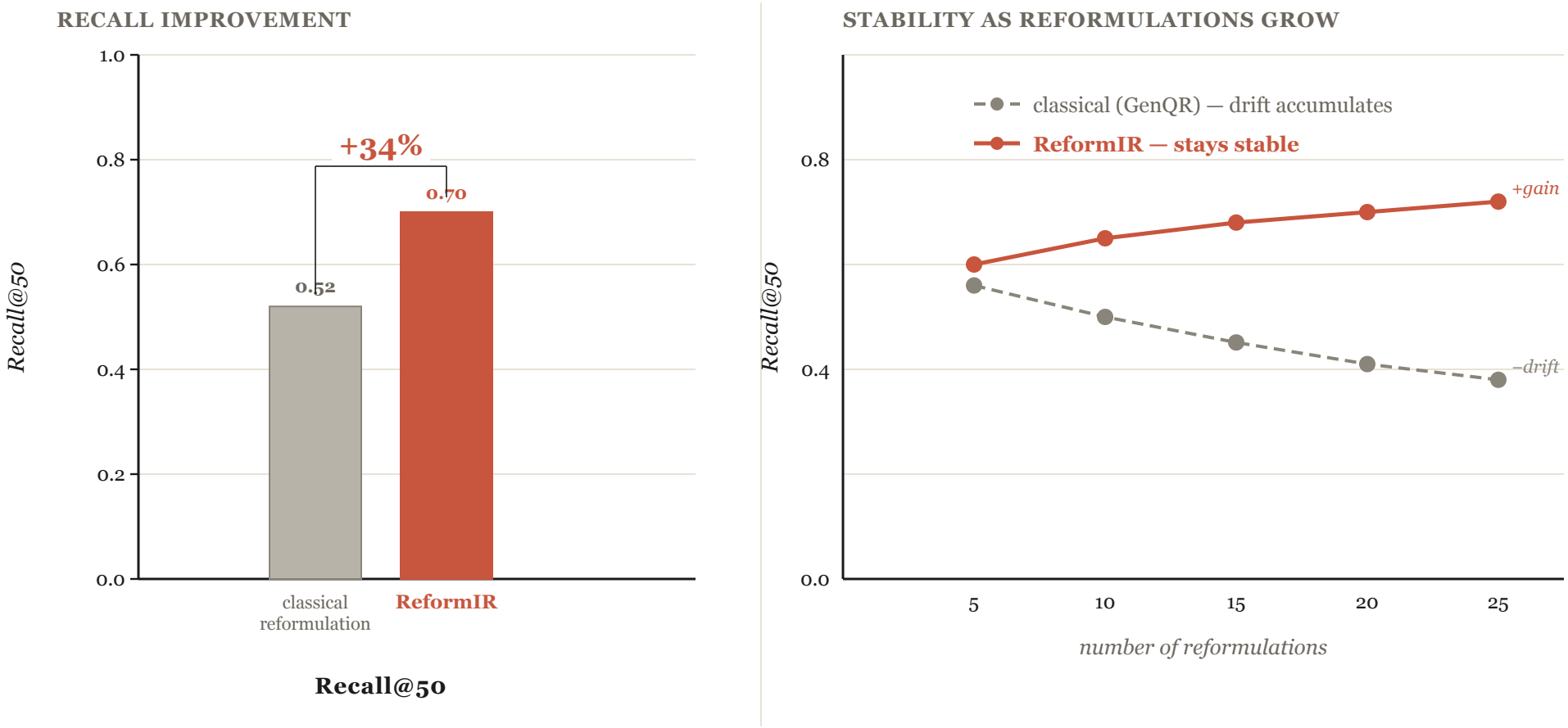
ReformIR — TREC DL19–DL22

Training-free adapter over any reformulation method



ReformIR — TREC DL19–DL22

Training-free adapter over any reformulation method



3.3×–4.5× more efficient than LLM-based reranking · better recall quality.

Training-free · works on GenQR, GEns, QA-Expand, Query2Doc · stable across reformulation count.

This is the paradigm shift

Stop fixing the system at training time.

PER-QUERY, ONLINE, FROM RERANKER FEEDBACK:
Which relevance signals to trust → ORE
Which reformulations to keep → ReformIR
Which neighbors to expand → QUAM, SUNAR

[15] QUAM (WSDM 2025) · [19] SUNAR (NAACL 2025) · [16] ORE (SIGIR 2025) · [5] ReformIR (SIGIR 2026)

> *Retrieval-augmented AI systems need to be feedback-aware and learn on a per-query basis.*

Bandits give us the language for all of it

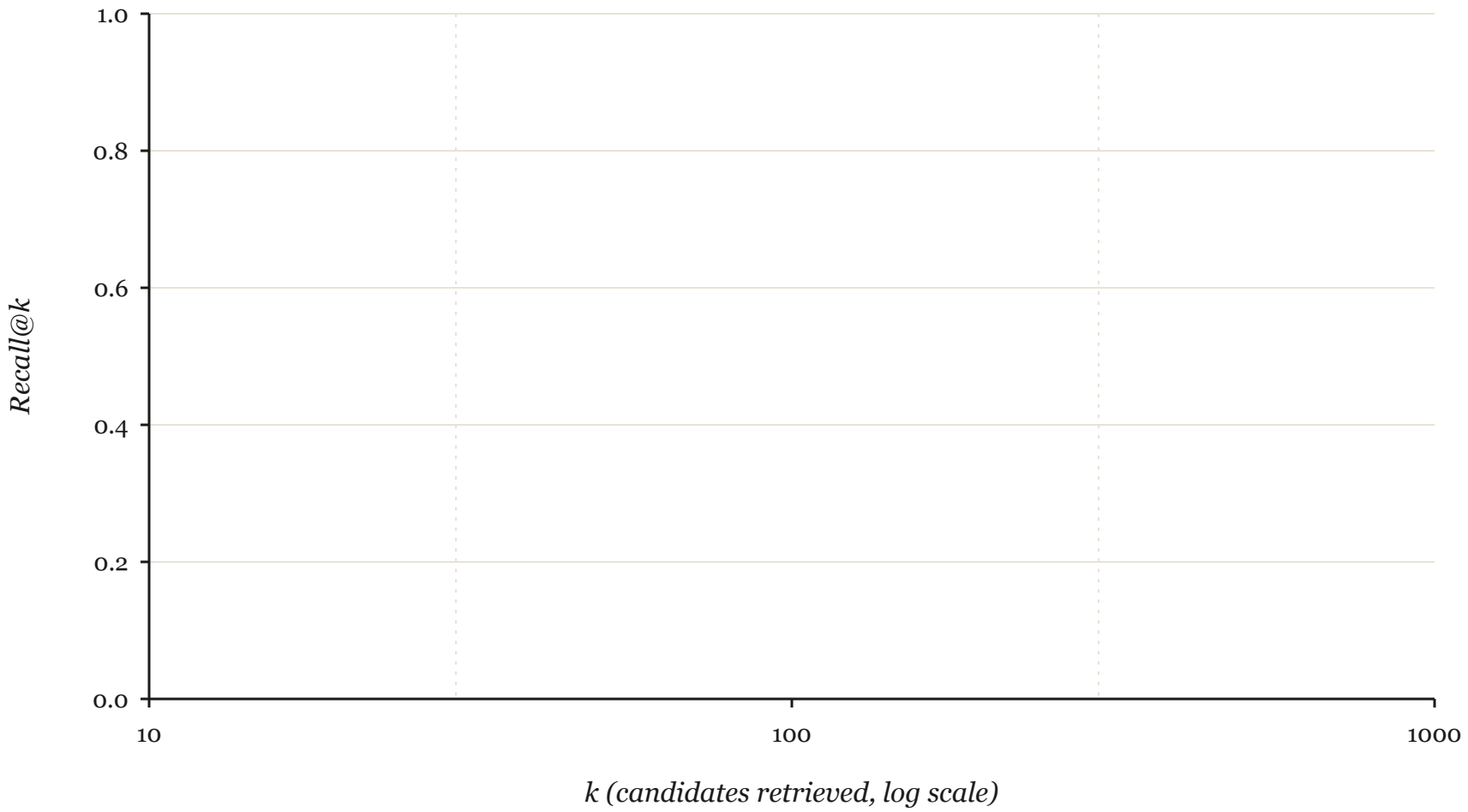
Same principle, different decisions, query by query:

- Which **query formulation** to issue → ReformIR
- Which **documents** to score → ORE, QUAM, SUNAR
- Which **judgments** to spend budget on → all of the above

> *Not "how do I train a better retriever" — but "how do I deploy compute optimally, query by query?"*

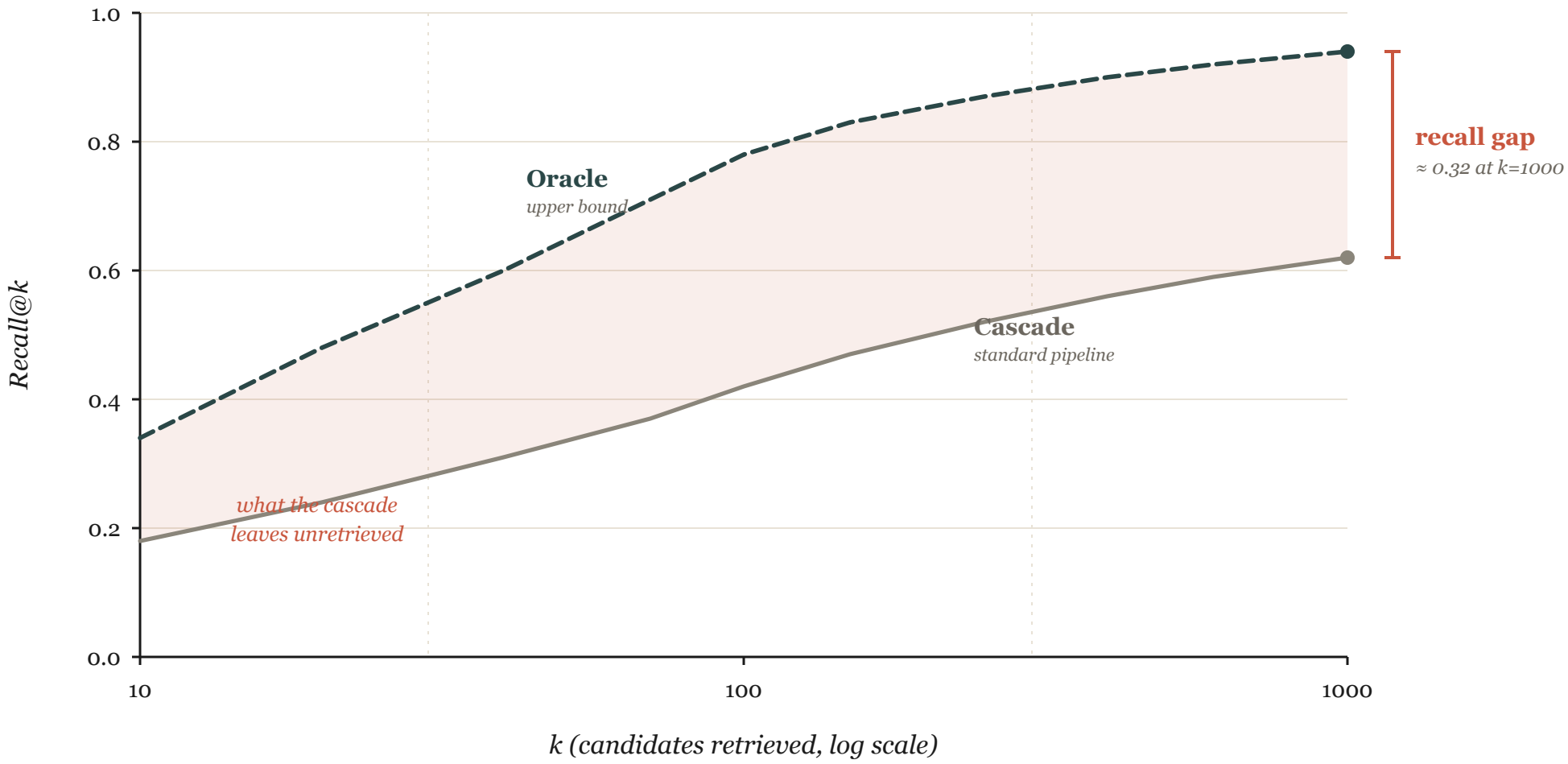
The recall gap

What the cascade leaves on the table



The recall gap

Cascade pipeline vs. oracle — what is left on the table

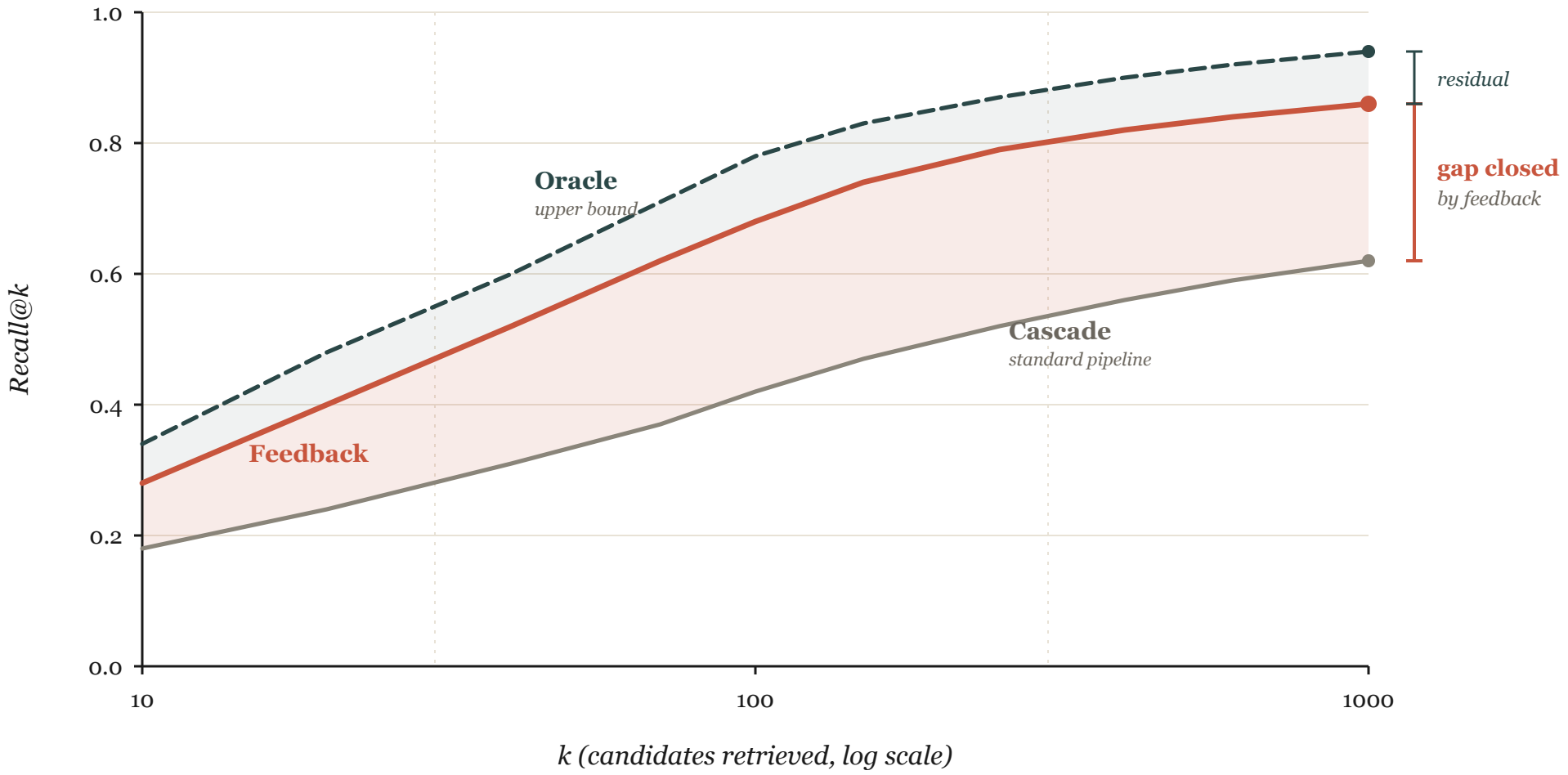


The cascade's recall is bounded by what the retriever surfaces.

The oracle shows how much is left — regardless of how good the reranker is.

Feedback closes the recall gap

Recall@k across the retrieval candidate pool



The reranker's relevance signal, fed back to the retriever, recovers most of the gap to oracle — without changing the retriever architecture.

The science: subset selection under uncertainty

Every algorithm I just showed reduces to the same question:

Given a large pool and a tiny budget, which items deserve the expensive call?

This is **top-m arms identification in stochastic linear bandits**.

The bottleneck: the candidate pool is exponentially large. Classical bandit algorithms compare every arm against every other — infeasible at retrieval scale.

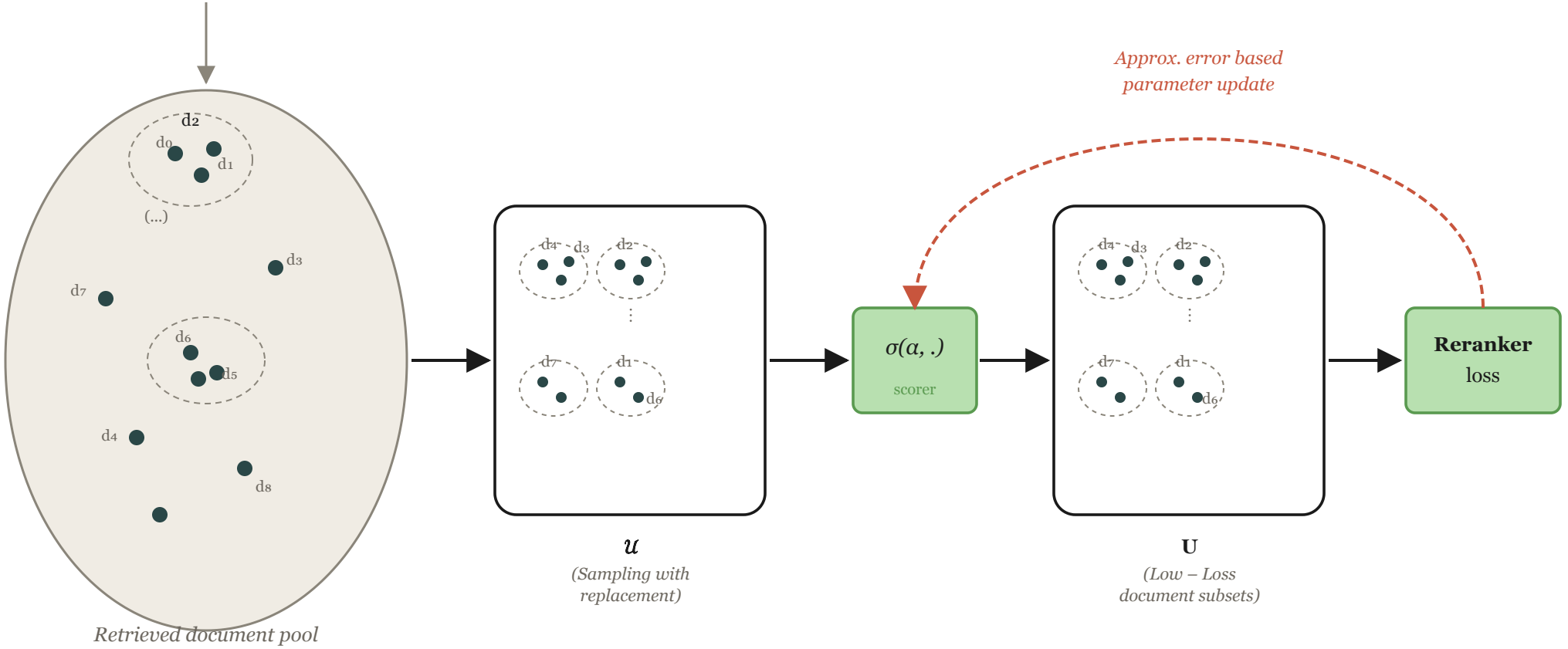
Challenger-arm sampling — choosing a small subset of docume

Pick the most promising k-document subsets from a large retrieved pool

Retrieved Documents

*d₁ : "Marriott London Bridge offers **city-centre rooms** …"*

*d₂ : "Boutique hotels with **romantic suites** near canals …"*



Retrieved document pool

$\mathcal{U}$

(Sampling with replacement)

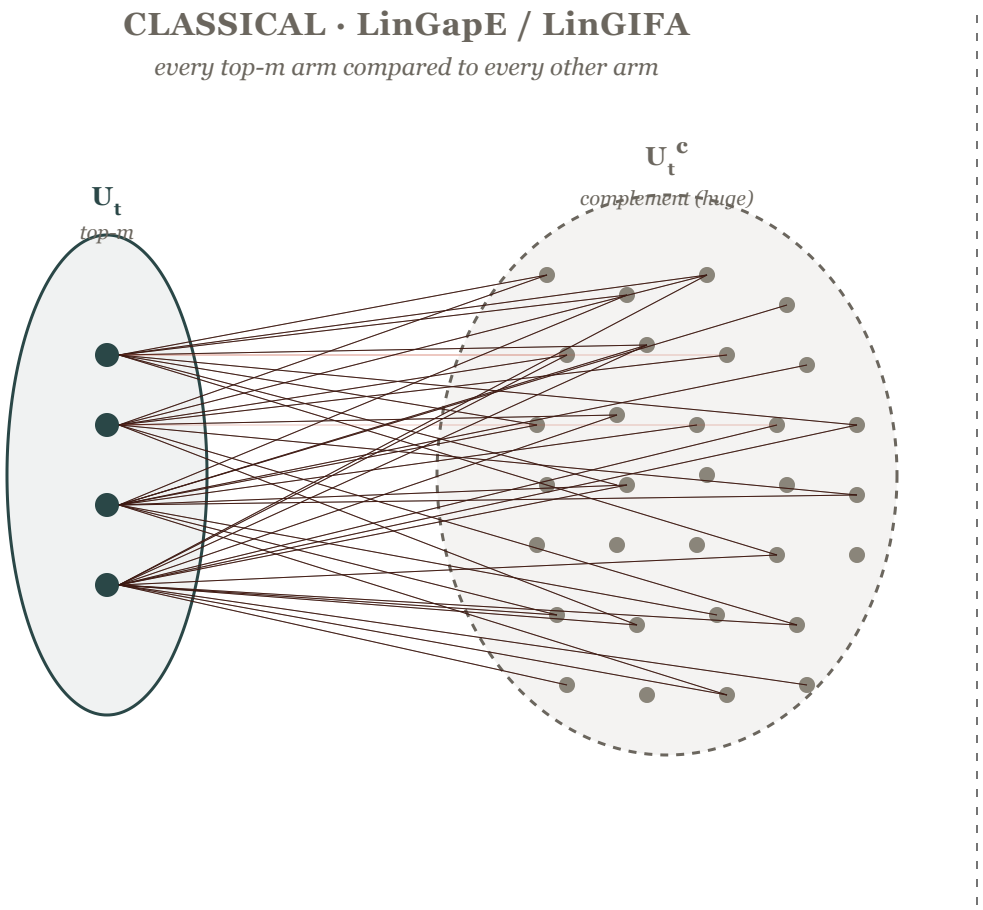
U

(Low – Loss document subsets)

Challenger-arm sampling efficiently picks the most promising k-document subsets — without scoring every subset.

CASE — pruning the comparison space

Why the same statistical guarantees can be achieved with far less work

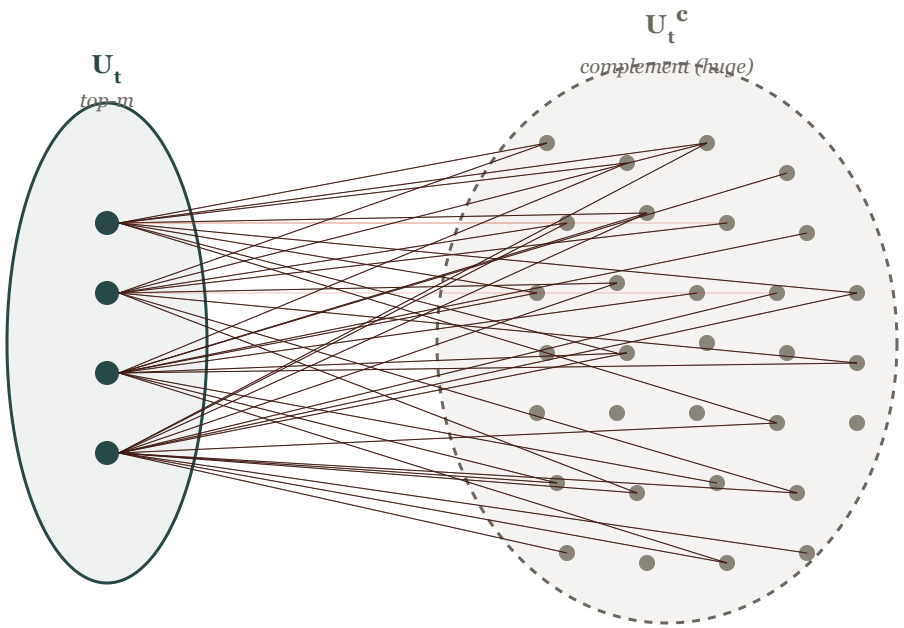


CASE — pruning the comparison space

Why the same statistical guarantees can be achieved with far less work

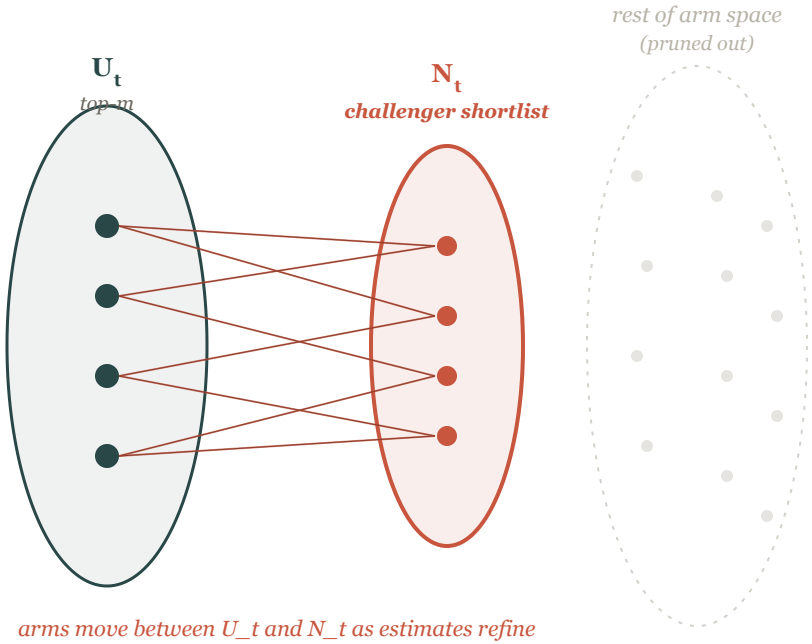
CLASSICAL · LinGapE / LinGIFA

every top-m arm compared to every other arm



CASE · OUR APPROACH

each top-m arm compared only to a challenger shortlist



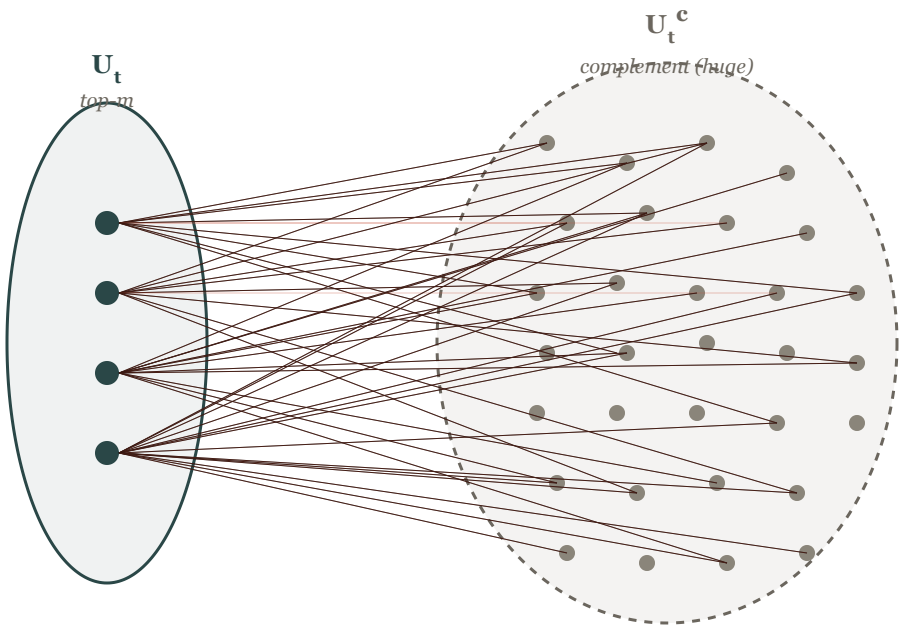
arms move between U_t and N_t as estimates refine

CASE — pruning the comparison space

Why the same statistical guarantees can be achieved with far less work

CLASSICAL · LinGapE / LinGIFA

every top-m arm compared to every other arm

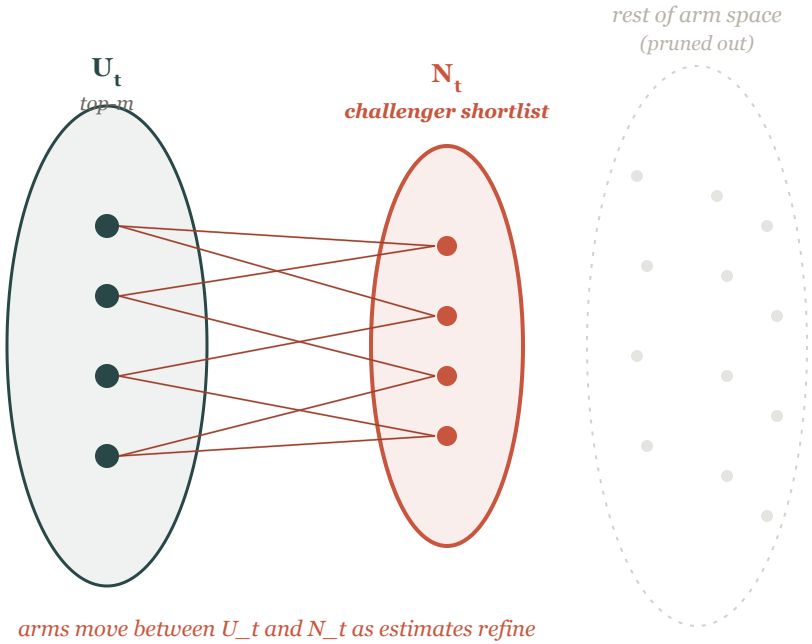


COMPARISON COST

$O(m \times |U_t^c|)$ *comparisons per round*

CASE · OUR APPROACH

each top-m arm compared only to a challenger shortlist



arms move between U_t and N_t as estimates refine

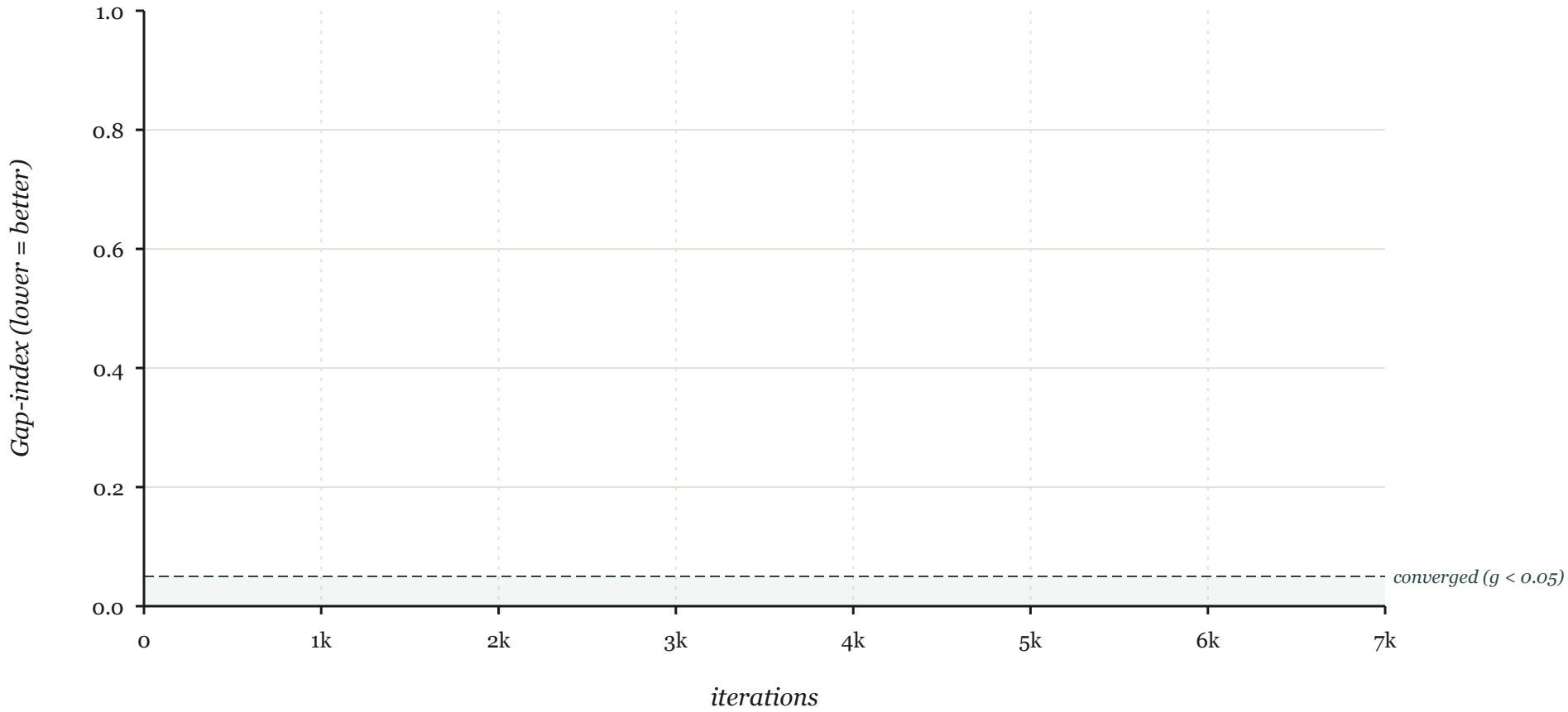
COMPARISON COST

$O(m \times |N_t|)$ *comparisons* · $|N_t| << |U_t^c|$

The shortlist is maintained adaptively. CASE preserves convergence guarantees of LinGapE while pruning out the comparisons that don't matter.

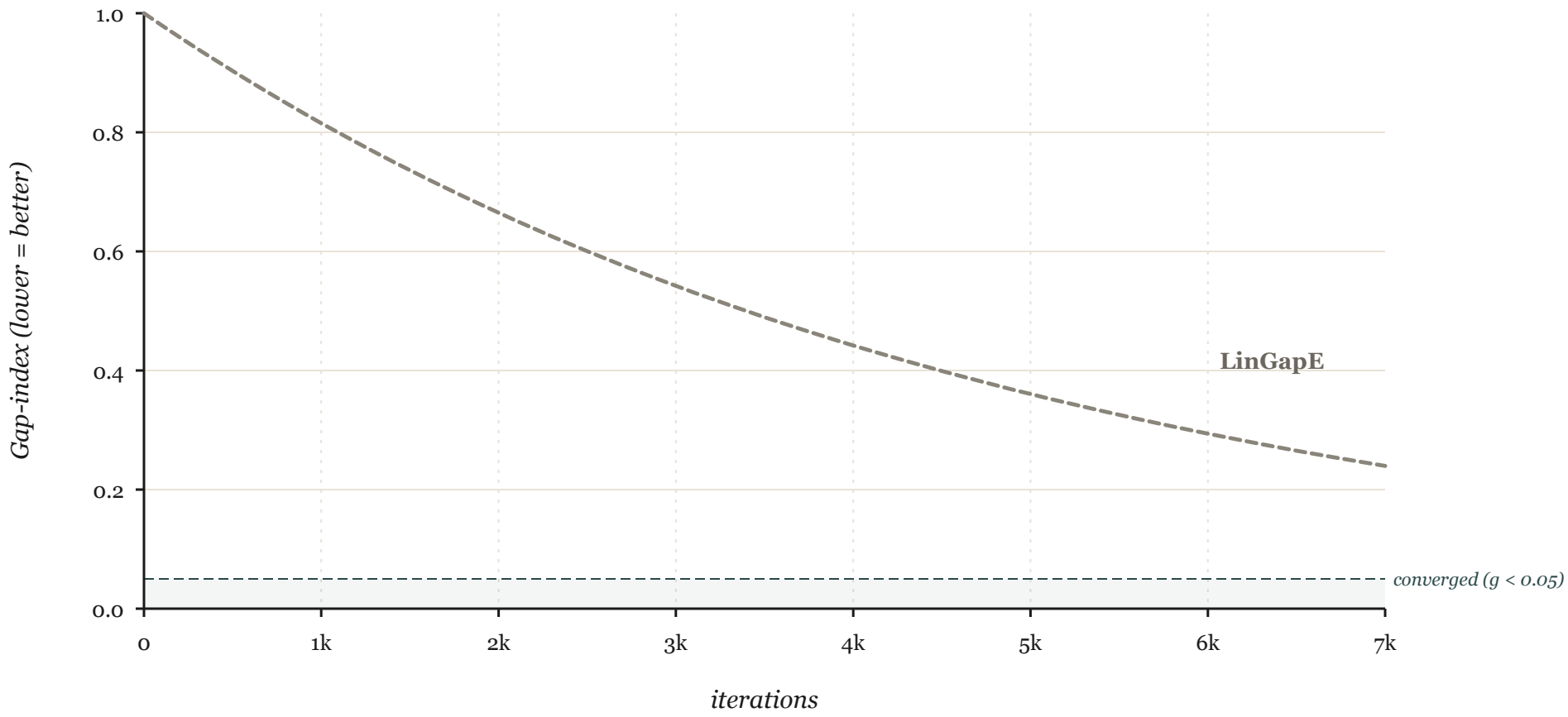
CASE converges in fewer iterations

Gap-index decay vs iteration count



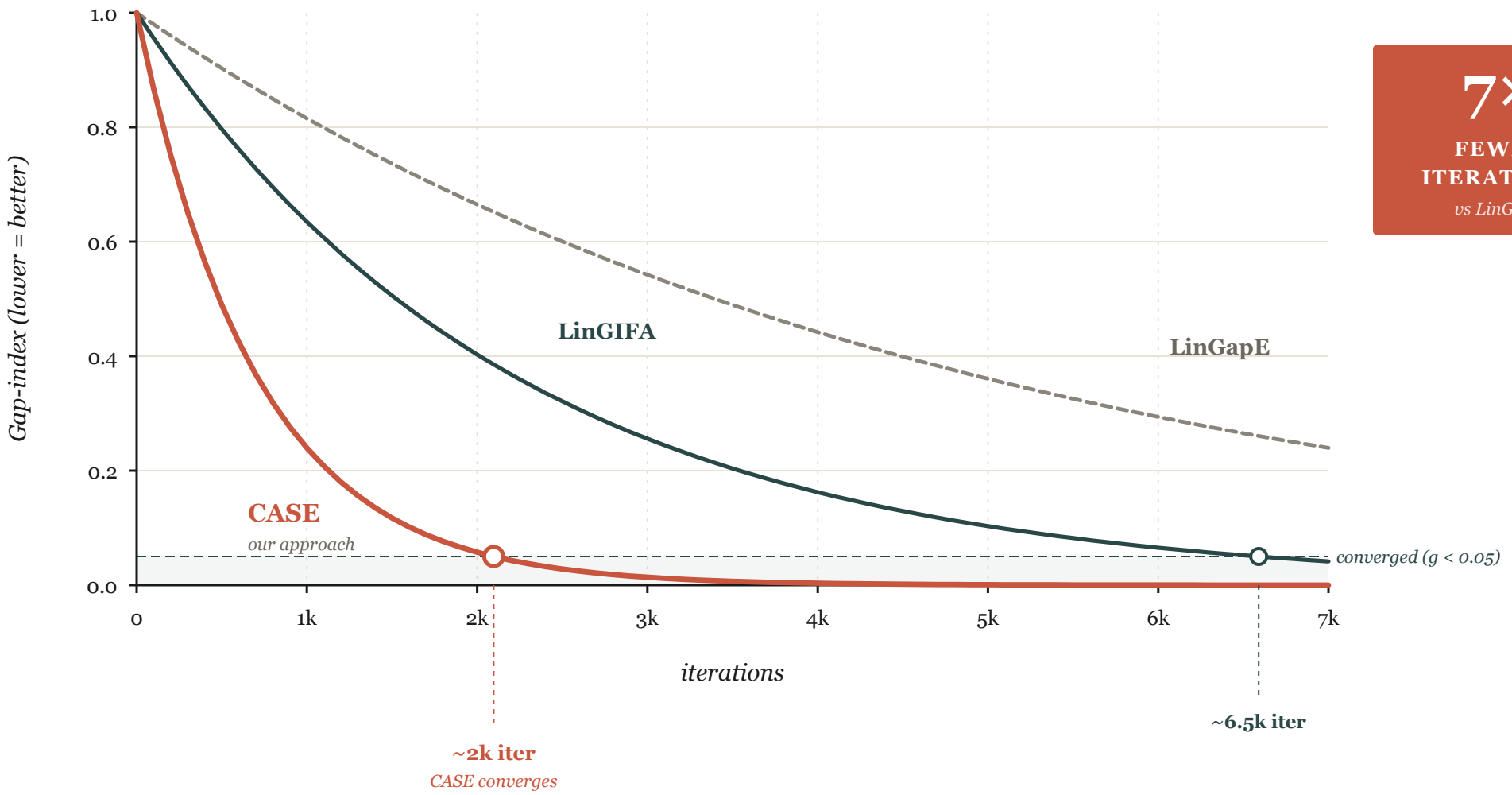
CASE converges in fewer iterations

Gap-index decay vs iteration count



CASE converges in fewer iterations

Gap-index decay vs iteration count



7×
FEWER ITERATIONS
vs LinGapE

Why this matters

Comparable task performance, with far fewer expensive calls

HEADLINE

7×

fewer LLM calls

same task performance

Why this matters

Comparable task performance, with far fewer expensive calls

HEADLINE

7×

fewer LLM calls

same task performance

LLM CALLS PER QUERY

each ● = 10 expensive calls · same task, same final accuracy

classical baseline

~700 *calls*

CASE

~100 calls

The same algorithm picks reranker-bound documents, in-context exemplars, evaluation samples — anywhere you choose a subset under a budget.

Better retrieval is useless if it's too expensive.

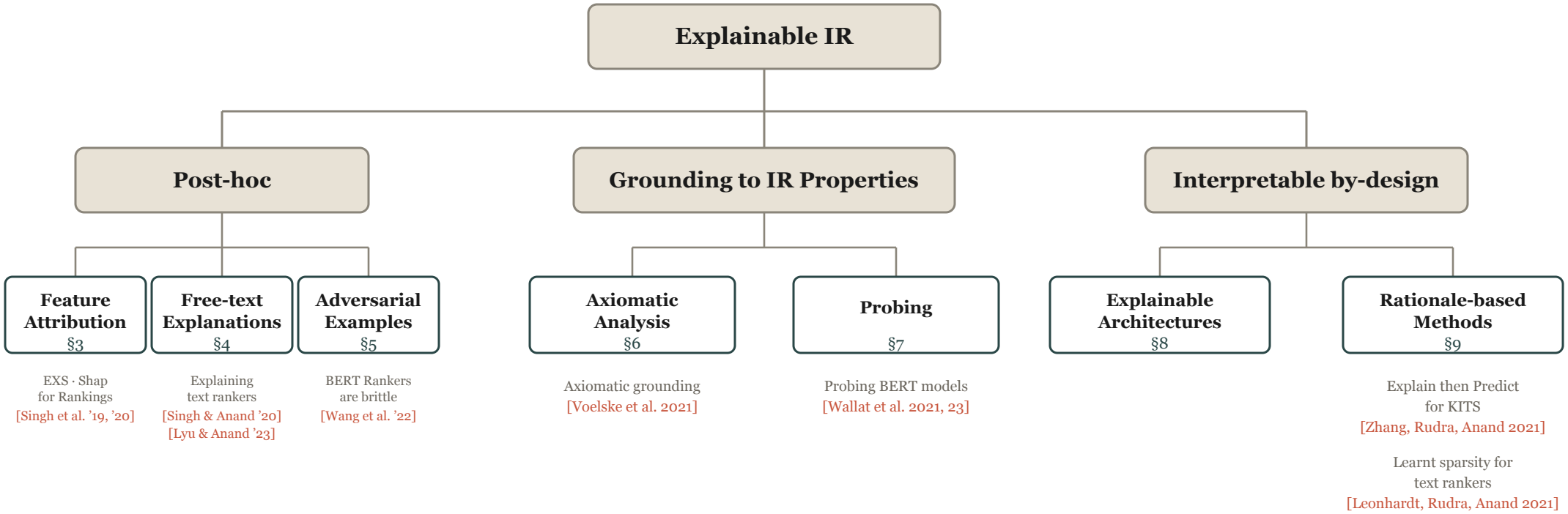
We're pushing the cost frontier, not just the quality frontier.

Explainable Information Retrieval

Explainable Information Retrieval: A Survey

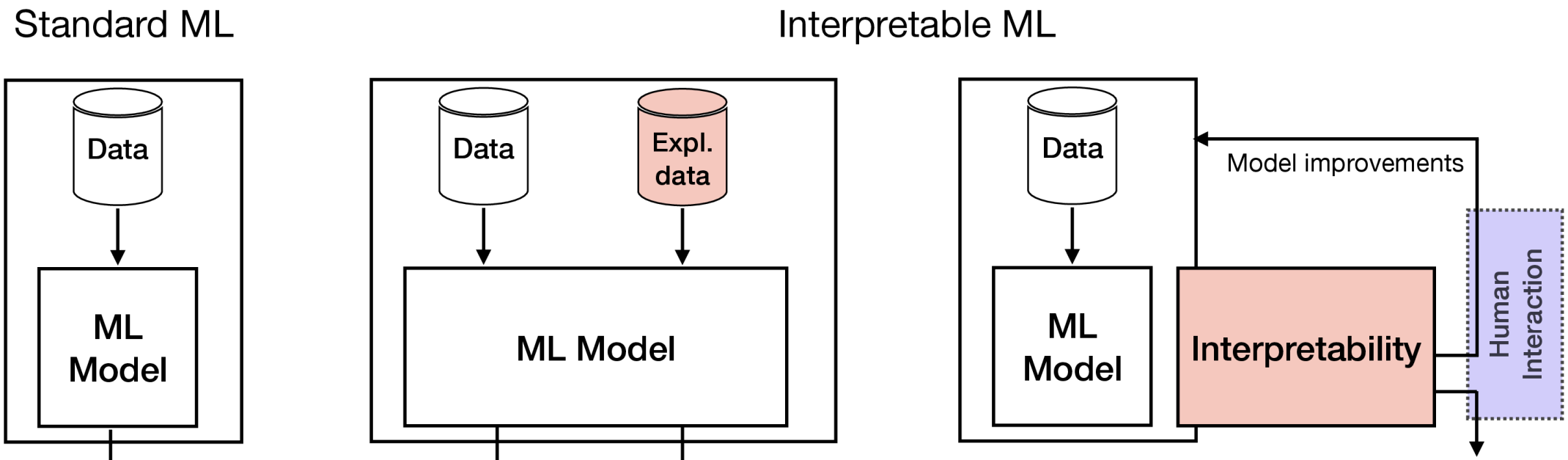
Avishek Anand, Lijun Lyu · TU Delft · Maximilian Idahl, Yumeng Wang, Jonas Wallat, Zijian Zhang · L3S / Leibniz Hannover

arxiv.org/abs/2211.02405



A unified taxonomy of explainability methods for retrieval and ranking systems.
From post-hoc feature attribution to interpretable-by-design architectures.

Ways in which you can use explanations



How do we collect explanation data?

Two regimes — extractive annotations of text vs abstractive natural-language explanations

EXTRACTIVE

Spans, tokens, or rationales selected directly from the document.

Query: romantic weekend near Amsterdam canals

A boutique **canal-side** hotel
offering **romantic suites** with private
balconies, walking distance to Anne
Frank House and the **Jordaan district**.
Couples-only floor available year-round.

canal-side | romantic suites | Jordaan district

Examples

· rationale extraction · token attributions
· feature highlights · supporting sentences
· TREC-DL relevance judgments + spans

+ cheap to collect at scale · grounded in the source

– bounded by what the document literally says

ABSTRACTIVE

Free-form natural-language rationale — now what LLMs produce.

Q: romantic weekend near Amsterdam canals
A: Boutique hotel in the Jordaan, canal-side, couples-only floor.

E: explanation

"This boutique canal-side property fits
because it pairs the *romantic atmosphere*
the query implies (couples-only floor,
private balconies) with a *canal location*
in the Jordaan — the *Amsterdam setting* asked for."

↑ *Q–A–E triple — we call this an* ***exemplar***

Examples

· GPT-4 / Claude relevance rationales
· e-SNLI · CoS-E · ECQA chain-of-thought
· reranker rationales for RAG citations

+ expressive · captures intent · transferable

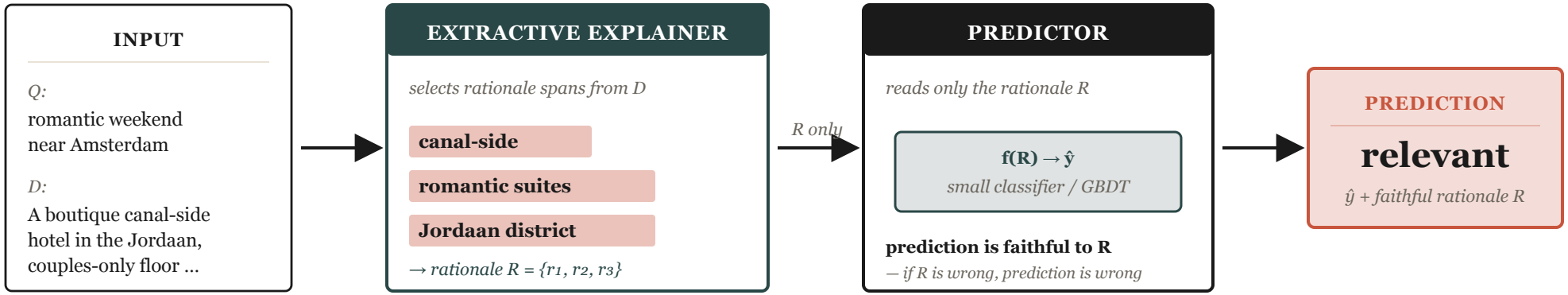
– hallucination risk · harder to verify

Most early IR explanation work is *extractive*. The LLM era moves the field *abstractive*.

Collecting both — and learning to score them — is what makes interpretable retrieval trainable.

Explain-then-predict — classical interpretable models

First produce an extractive explanation, then predict from it



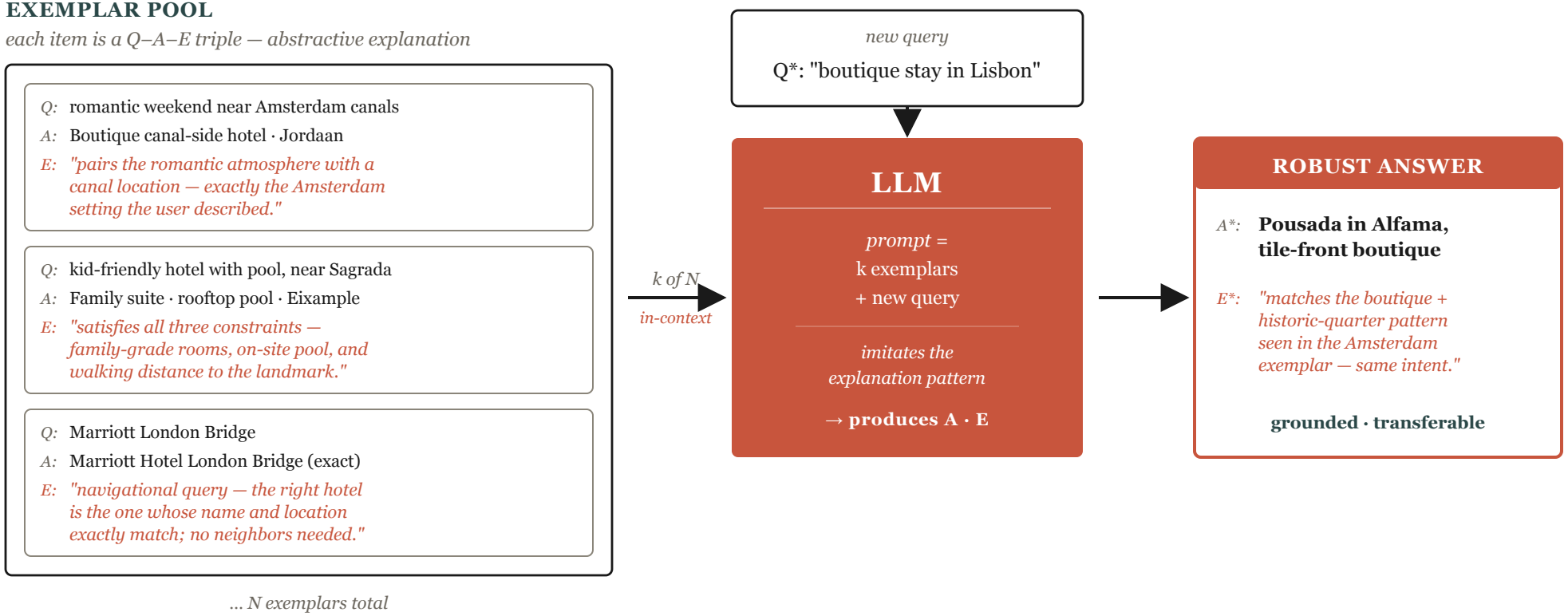
PROPERTIES

· the rationale is the only path from input to prediction — **faithful by construction**
· explanation comes *first*, prediction follows — explain-*then*-predict, not predict-then-rationalize
· representative work: Lei et al. 2016 · Bastings et al. 2019 · Zhang, Rudra, Anand 2021 (KITS)

The classical recipe: pick the spans that justify the answer, predict from those spans alone.

Abstractive explanations as exemplars for the LLM

Today's question — can these Q–A–E exemplars steer LLMs to more robust answers?



THE QUESTION

Can we use abstractive explanations as in-context exemplars to make LLM answers more robust?

Yesterday: explain-then-predict with extractive spans. Today: prompt-then-imitate with abstractive exemplars.

EXPLORA — Choosing Explanations

You have a pool of N explanations. In-context learning needs a *k-subset* — the right k demonstrations decide whether the LLM reasons correctly or not.

The brute-force search: evaluate all $\binom{N}{k}$ subsets with the LLM.
At N = 100, k = 4, that's **3.9 million LLM calls**.

Again **EXPLORA frames this as bandit arm selection.**

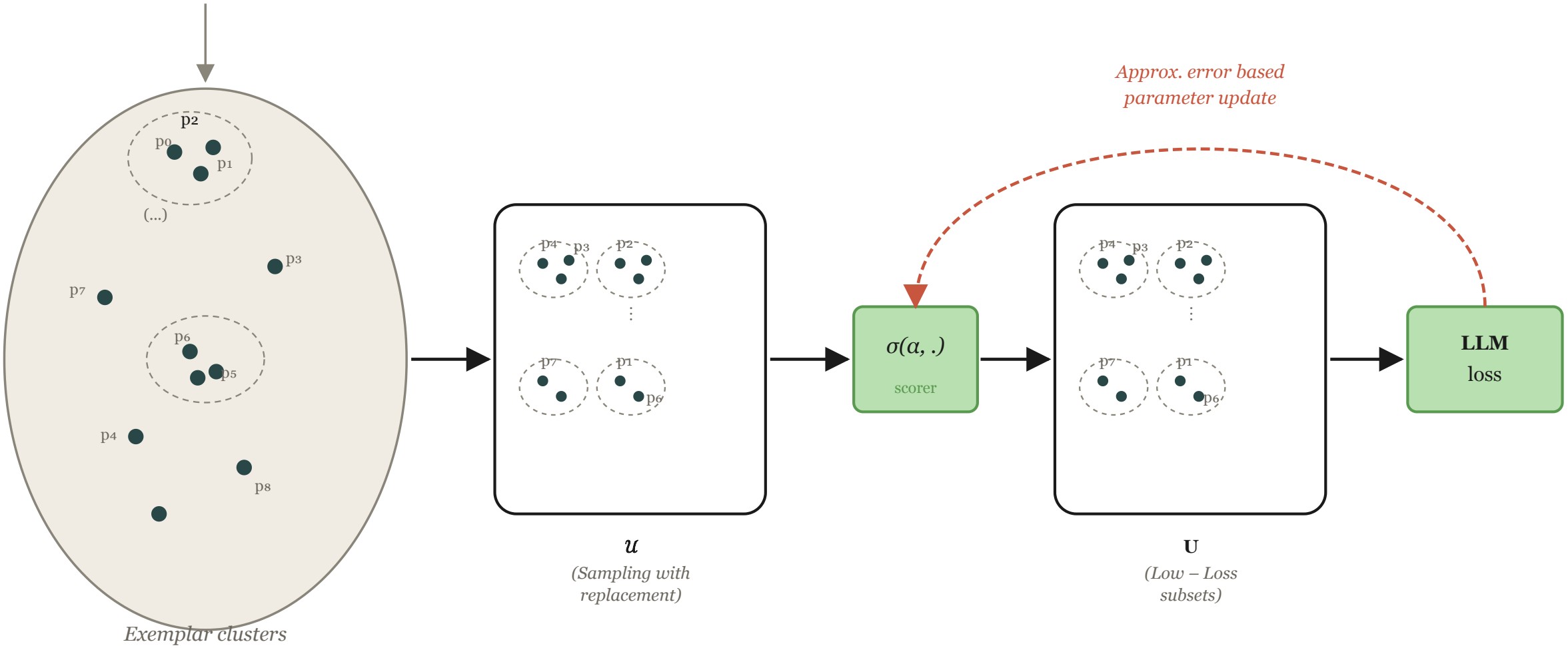
Each k-subset is an arm. An LLM call is a pull. The arms are exponentially many — classical bandit algorithms that compare every pair are infeasible.

[24] Purohit, Venktesh V, Devalla, Yerragorla, Bhattacharya, Anand. EXPLORA: Efficient Exemplar Subset Selection for Complex Reasoning. EMNLP 2024, pp. 5367–5388.

Exemplars

*p1 : While purchasing groceries ram bought **5 apples** ...*

*p2 : Ephraim has **two machines** that make necklaces ...*



CASE — what EXPLORA leaves open

EXPLORA explores efficiently but never answers a fundamental question:

> *When have you found the best subset?*
> *How do you know when to stop?*

Without confidence information, EXPLORA can't eliminate arms — it keeps spending budget on subsets that are clearly suboptimal.

[13] Purohit, Venktesh V, Bhattacharya, Anand. Sample Efficient Demonstration Selection for In-Context Learning. ICML 2025.

Explain-and-predict isn't always perfect

The classical recipe — extract a rationale, then predict from it — is *faithful by construction*. But faithfulness is not free.

Recent work shows that **the joint optimization can succeed without leaking the label or the feature itself** into the rationale — and that the resulting predictors are competitive with their non-interpretable counterparts.

> *Interpretable retrieval is no longer paying a quality tax —*
> *but the joint problem is delicate, and most pipelines still get it wrong.*

Oosterhuis, Lyu, Anand. Local Feature Selection without Label or Feature Leakage for Interpretable Machine Learning Predictions. ICML 2024. arXiv:2407.11778

RAG faithfulness isn't perfect either

LLMs can produce a fluent answer, cite the right passage, and still be unfaithful — the citation explains the answer *post hoc* rather than supporting how the answer was actually derived.

We call this **post-rationalization**: the model decides, then dresses the decision in a plausible attribution.

> *Correct ≠ faithful.*
> *A right answer with the wrong reason is still the wrong system.*

Wallat, Heuss, de Rijke, Anand. Correctness is not Faithfulness in Retrieval Augmented Generation Attributions. ICTIR 2025, pp. 22–32. DOI: 10.1145/3731120.3744592

Towards AutoIR

A new subfield: **self-improving retrieval systems.**

A retrieval system that:

- **generates its own data** it needs to improve on
- **evaluates itself** in a way that aligns with deployment
- **optimizes itself**, query by query, over time

A three-step path to AutoIR

Inner loop · bridge · outer loop

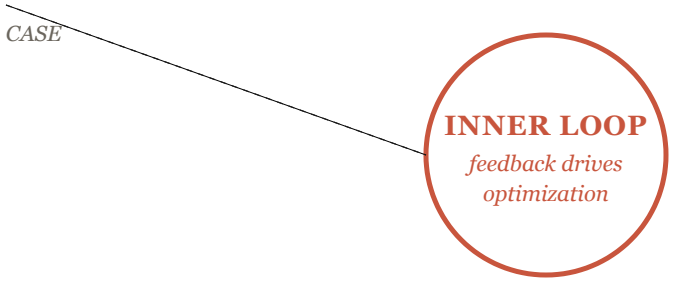
1

Feedback

the inner loop

What feedback, and how to use it to improve and even construct RAG pipelines.

QUAM · SUNAR · ORE · CASE



A three-step path to AutoIR

Inner loop · bridge · outer loop

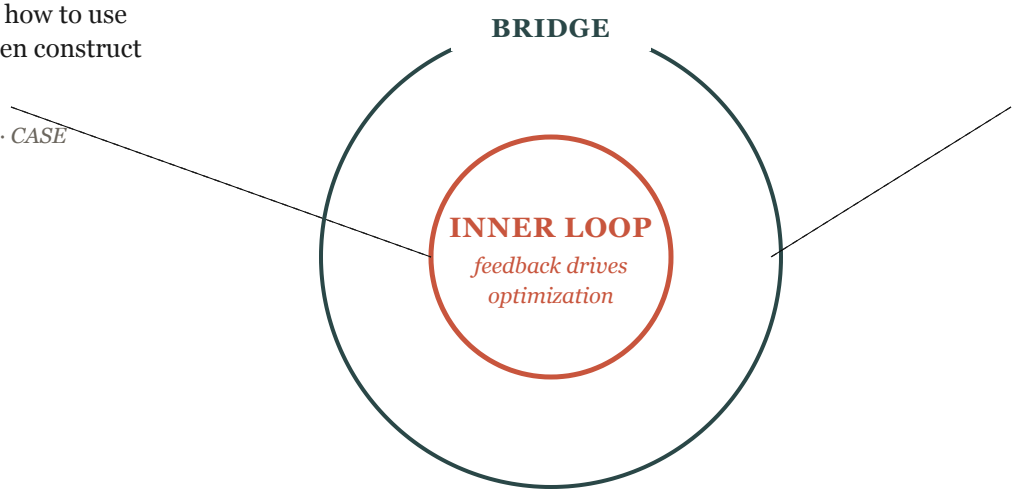
1

Feedback

the inner loop

What feedback, and how to use it to improve and even construct RAG pipelines.

QUAM · SUNAR · ORE · CASE



2

Evaluation alignment

the bridge

Auto-generated offline benchmarks aligned with online behavior.

LLM judges · click models

A three-step path to AutoIR

Inner loop · bridge · outer loop

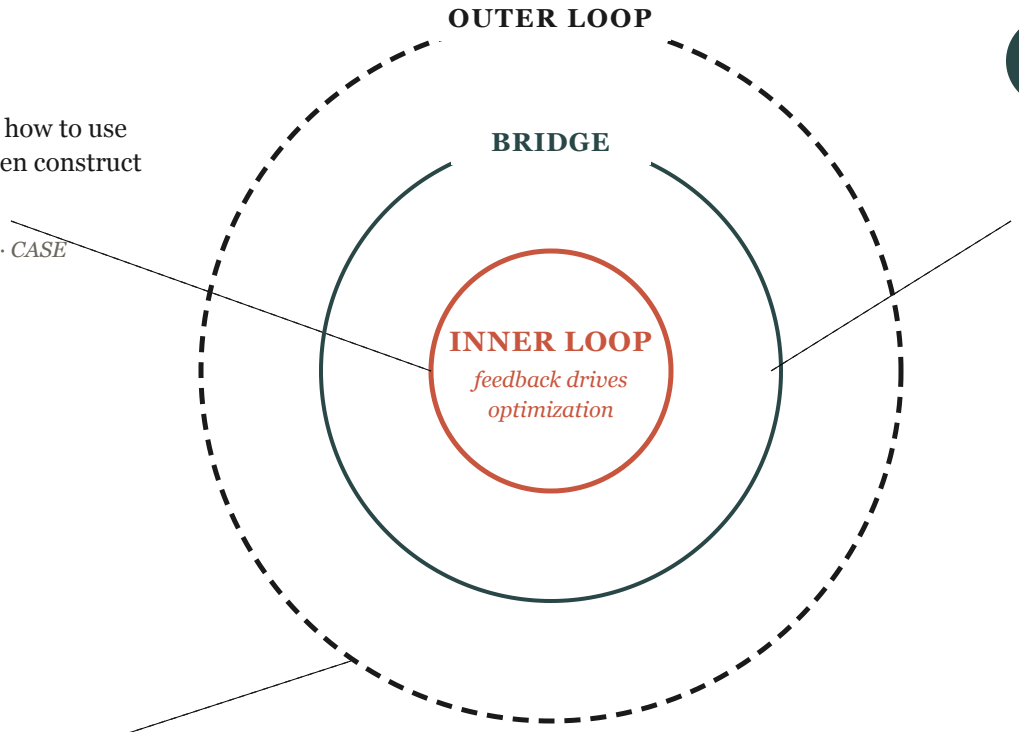
1

Feedback

the inner loop

What feedback, and how to use it to improve and even construct RAG pipelines.

QUAM · SUNAR · ORE · CASE



2

Evaluation alignment

the bridge

Auto-generated offline benchmarks aligned with online behavior.

LLM judges · click models

3

Self-play

the outer loop

The outer loop generates data from (2).
The inner loop optimizes from (1).

Together: a system that learns.

next: the self-play loop in detail

Personas — generating *behavior*, not labels

Most synthetic-data work generates **labels**:

"Is document D relevant to query Q? Yes / No."

That's not how users behave.

We generate **personas**:

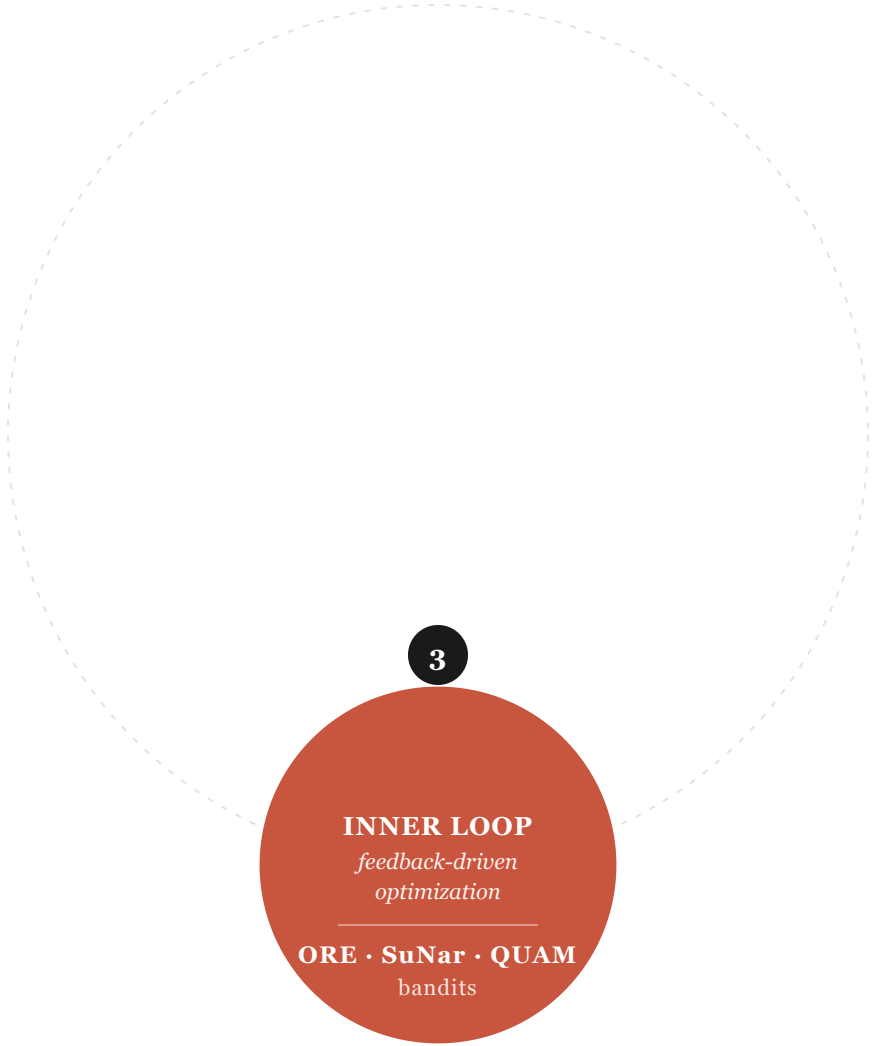
- the budget traveler vs the luxury traveler
- the planner vs the last-minute booker
- the comparison shopper vs the decisive buyer

Each persona issues queries, reformulates, clicks, abandons —
the way users actually do.

> *We don't simulate labels.*
> *We simulate users.*

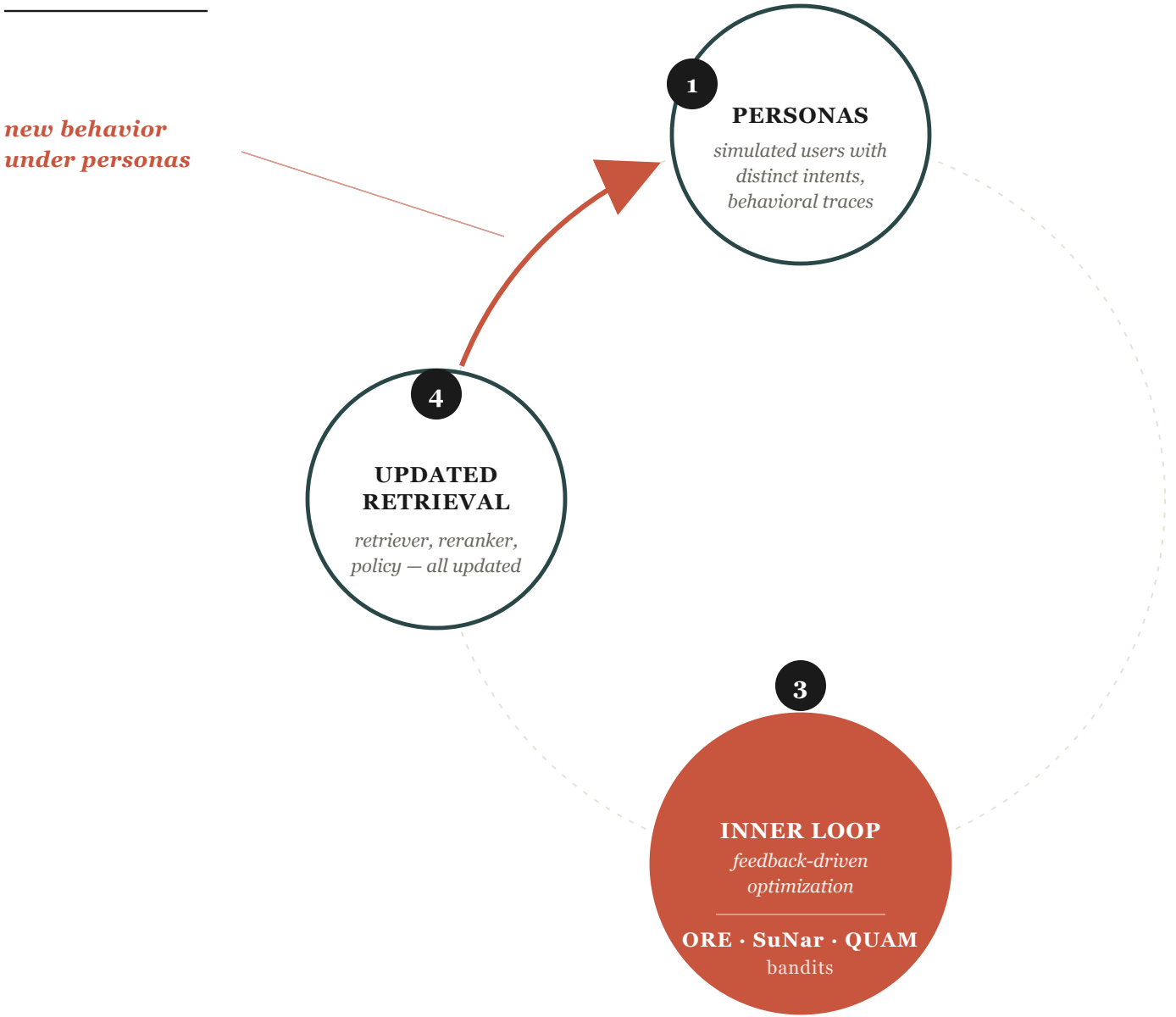
The self-play loop

AutoIR · the system creates the data it needs to improve



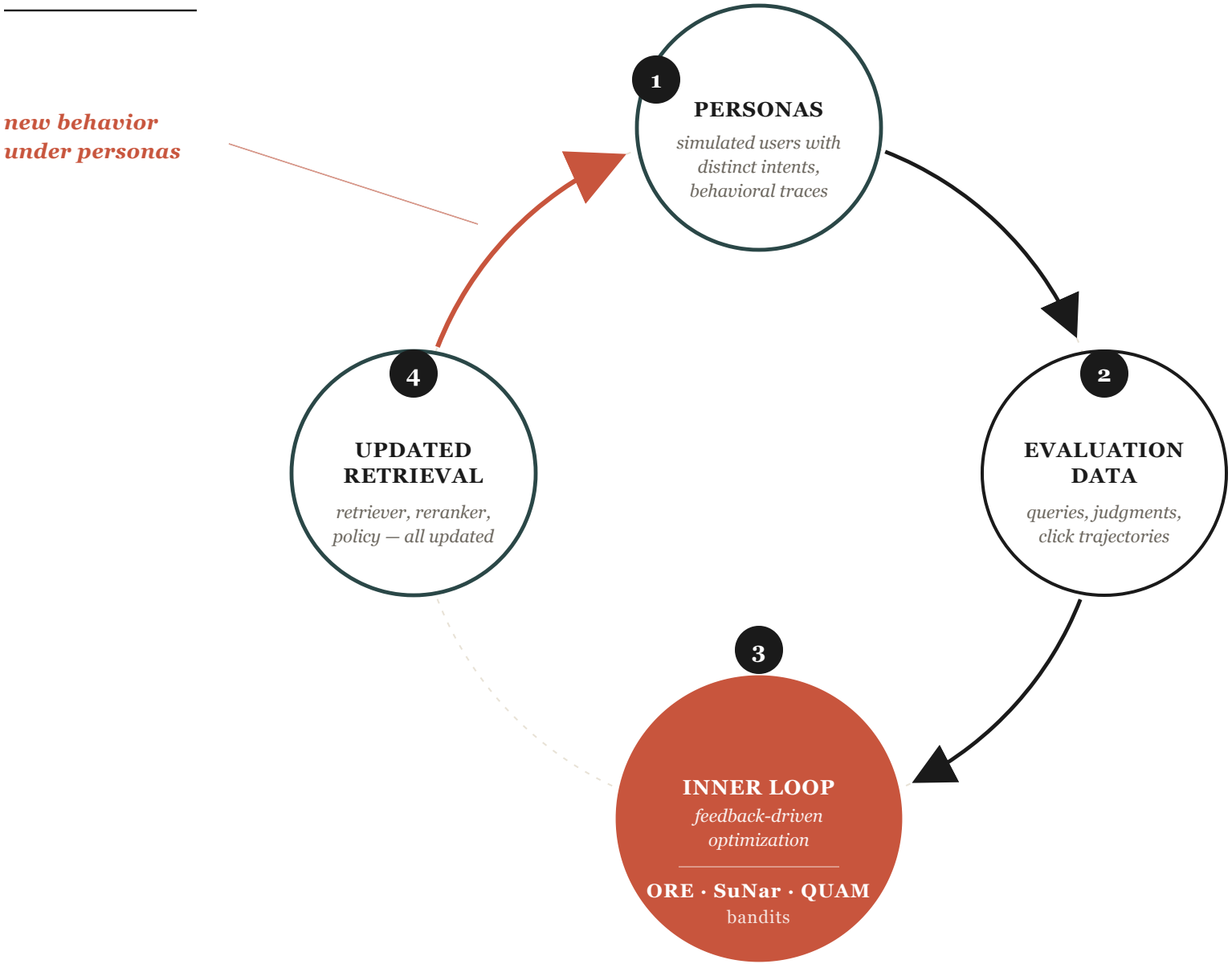
The self-play loop

AutoIR · the system creates the data it needs to improve



The self-play loop

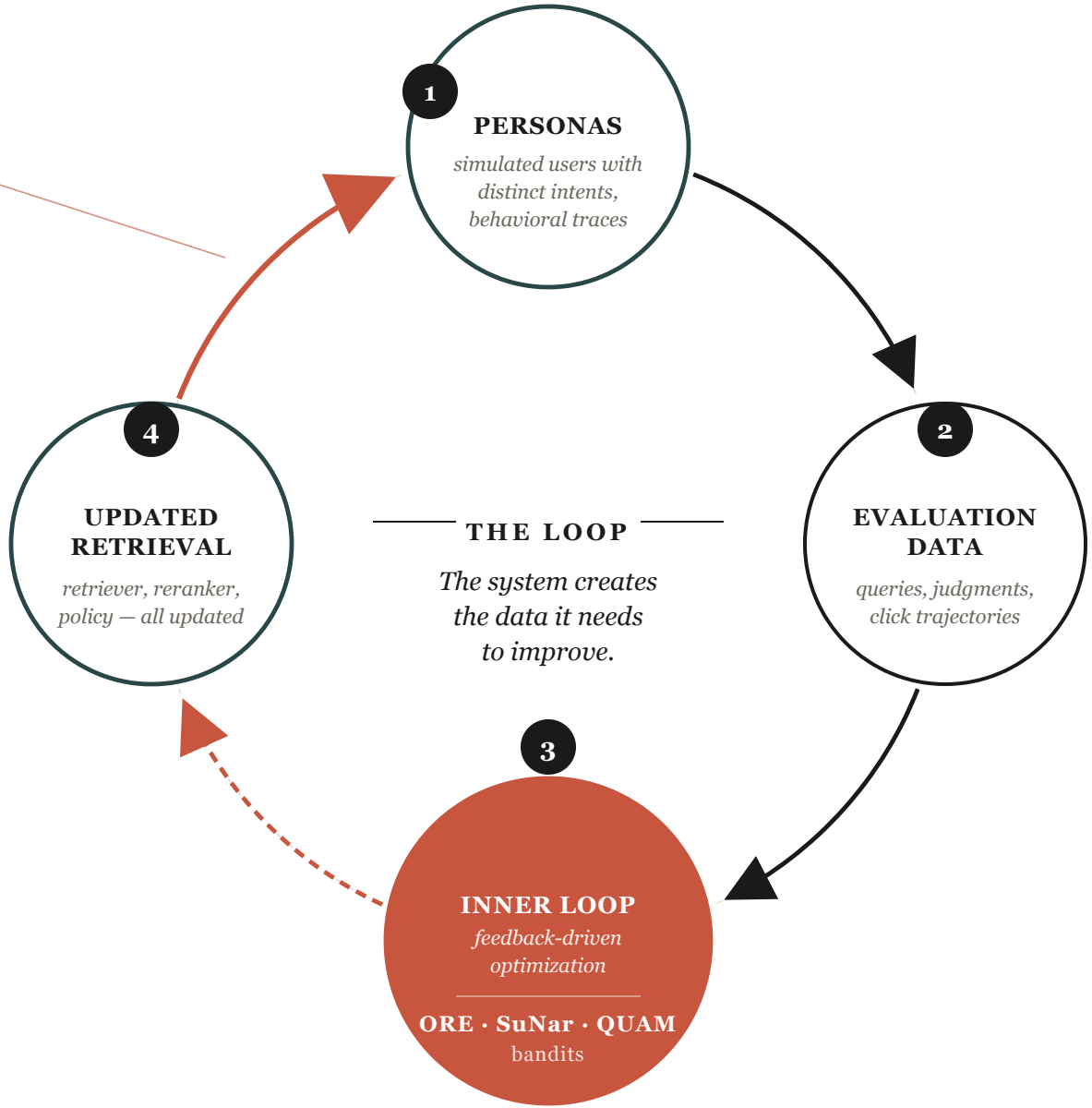
AutoIR · the system creates the data it needs to improve



The self-play loop

AutoIR · the system creates the data it needs to improve

new behavior under personas



FROM

Static benchmarks

fixed at release

TO

Continuous self-evaluation

deployed system re-evaluates itself

The system creates the data it needs to improve.

Static benchmarks → continuous self-evaluation.

Where this leaves us

- Pipelines → systems
- Retrieval → decisions under uncertainty
- Static benchmarks → self-improving loops

> *RAG today is a pipeline.*
>
> *RAG tomorrow is a system that learns.*

I work with a small number of companies a year on exactly this.
If any of is interesting for you — let's talk after.

Avishek Anand · TU Delft
[avishek.anand@tudelft.nl] · [avishekanand@gmail.com] . www.avishekanand.com

References

[1] Rathee, Venktesh V, MacAvaney, Anand. Reproducing Adaptive Reranking for Reasoning-Intensive IR. *SIGIR 2026* (to appear).

[2] Rathee, Venktesh V, MacAvaney, Anand. Test-time Corpus Feedback: From Retrieval to RAG. *Findings of ACL: EACL 2026*, pp. 5637–5656.

[3] Venktesh V, Rathee, Anand. When More Reformulations Hurt: Avoiding Drift using Ranker Feedback. *SIGIR 2026* (to appear).

[4] Purohit, Venktesh V, Bhattacharya, Anand. Sample Efficient Demonstration Selection for In-Context Learning. *ICML 2025*.

[5] Rathee, MacAvaney, Anand. Guiding Retrieval Using LLM-Based Listwise Rankers. *ECIR 2025*, pp. 230–246. DOI: 10.1007/978-3-031-88708-6_15.

[6] Purohit, Venktesh V, Devalla, Yerragorla, Bhattacharya, Anand. EXPLORA: Efficient Exemplar Subset Selection for Complex Reasoning. *EMNLP 2024*, Miami, FL, pp. 5367–5388. DOI: 10.18653/V1/2024.emnlp-main.310.

[7] Rathee, MacAvaney, Anand. Quam: Adaptive Retrieval through Query Affinity Modelling. *WSDM 2025*, pp. 954–962. DOI: 10.1145/3701551.3703584.

[8] Rathee, Venktesh V, MacAvaney, Anand. Breaking the Lens of the Telescope: Online Relevance Estimation over Large Retrieval Sets. *SIGIR 2025*, pp. 2287–2297. DOI: 10.1145/3726302.3729910.

References (cont.)

[9] Yoon, Kim, Kwon, Anand, Hwang. On Listwise Reranking for Corpus Feedback. *WSDM 2026*, pp. 1273–1277. DOI: 10.1145/3773966.3779404.

[10] Anand, Saha, Venktesh V. Explainable Information Retrieval. *ECIR 2025*, pp. 254–261. DOI: 10.1007/978-3-031-88720-8_40.

[11] Chungkham, Venktesh V, Setty, Anand. Think Right, Not More: Test-Time Scaling for Numerical Claim Verification. *Findings of ACL: EMNLP 2025*, pp. 24345–24363.

[12] Heuss, de Rijke, Anand. RankingSHAP — Faithful Listwise Feature Attribution Explanations for Ranking Models. *SIGIR 2025*, pp. 381–391. DOI: 10.1145/3726302.3729971.

[13] Nanhekhan, Venktesh V, Martin, Vatndal, Setty, Anand. FlashCheck: Exploration of Efficient Evidence Retrieval for Fast Fact-Checking. *ECIR 2025*, pp. 385–399. DOI: 10.1007/978-3-031-88717-8_28.

[14] Saha, Agarwal, Venktesh V, Anand et al. ir_explain: A Python Library of Explainable IR Methods. *SIGIR 2025*, pp. 3563–3572. DOI: 10.1145/3726302.3730343.

[15] Venktesh V, Rathee, Anand. SUNAR: Semantic Uncertainty based Neighborhood Aware Retrieval for Complex QA. *NAACL 2025*, pp. 5818–5835. DOI: 10.18653/V1/2025.NAACL-LONG.300.

[16] Wallat, Heuss, de Rijke, Anand. Correctness is not Faithfulness in Retrieval Augmented Generation Attributions. *ICTIR 2025*, pp. 22–32. DOI: 10.1145/3731120.3744592.

Thats it !!!

First-generation RAG — where it breaks

These systems work. But they break in systematic, reproducible ways.

I've seen the same three failures across search engines, fact-checkers, and recommendation systems.

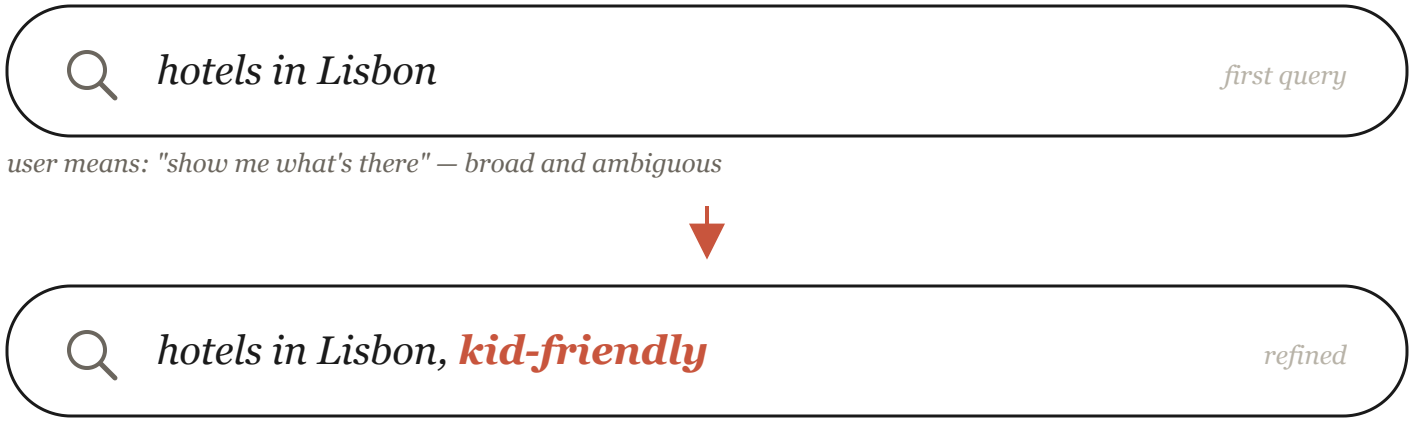
A query you've seen a thousand times

Retrieval here isn't a step. It's a process.

user means: "show me what's there" — broad and ambiguous

A query you've seen a thousand times

Retrieval here isn't a step. It's a process.



A query you've seen a thousand times

Retrieval here isn't a step. It's a process.

Three queries · one user · one session. *Each one reshapes the retrieval problem.*

The system that treats these as independent shots will lose information at every step.

> *Retrieval here isn't a step. It's a process.*

CASE — what EXPLORA leaves open

EXPLORA explores efficiently but never answers a fundamental question:

> *When have you found the best subset?*
> *How do you know when to stop?*

Without confidence information, EXPLORA can't eliminate arms — it keeps spending budget on subsets that are clearly suboptimal.

CASE (ICML 2025) adds gap-index theory:

- Tracks a **shortlist of challenger arms** — subsets still statistically competitive with the current best
- Arms with a large performance *gap* below the leader are eliminated early
- The gap-index **bounds the expected number of pulls needed** to certify the best arm

EXPLORA — learning to score subsets

A scoring function **σ(α, S)** predicts the quality of subset S.
α is unknown — the parameters we learn from LLM feedback.

The loop:

1. Sample a batch of candidate subsets from the pool
2. Query the LLM on a handful — observe accuracy as reward
3. Fit α by minimizing prediction loss over observed rewards
4. Use σ(α, ·) to guide which subsets to evaluate next
5. Repeat until budget exhausted

No confidence bounds. No elimination. Explores by loss minimization alone.

Results on AquaRat · FinQA · GSM8K · TabMwp:
Uses ~11% of the LLM calls required by prior SOTA · +12.24% accuracy

[24] Purohit et al. EXPLORA: EMNLP 2024

Algorithm 1 — QUAM

Failure 1: the recall ceiling — the relevant doc never made it into the top-1000.

QUAM asks the reranker to *teach* the retriever:

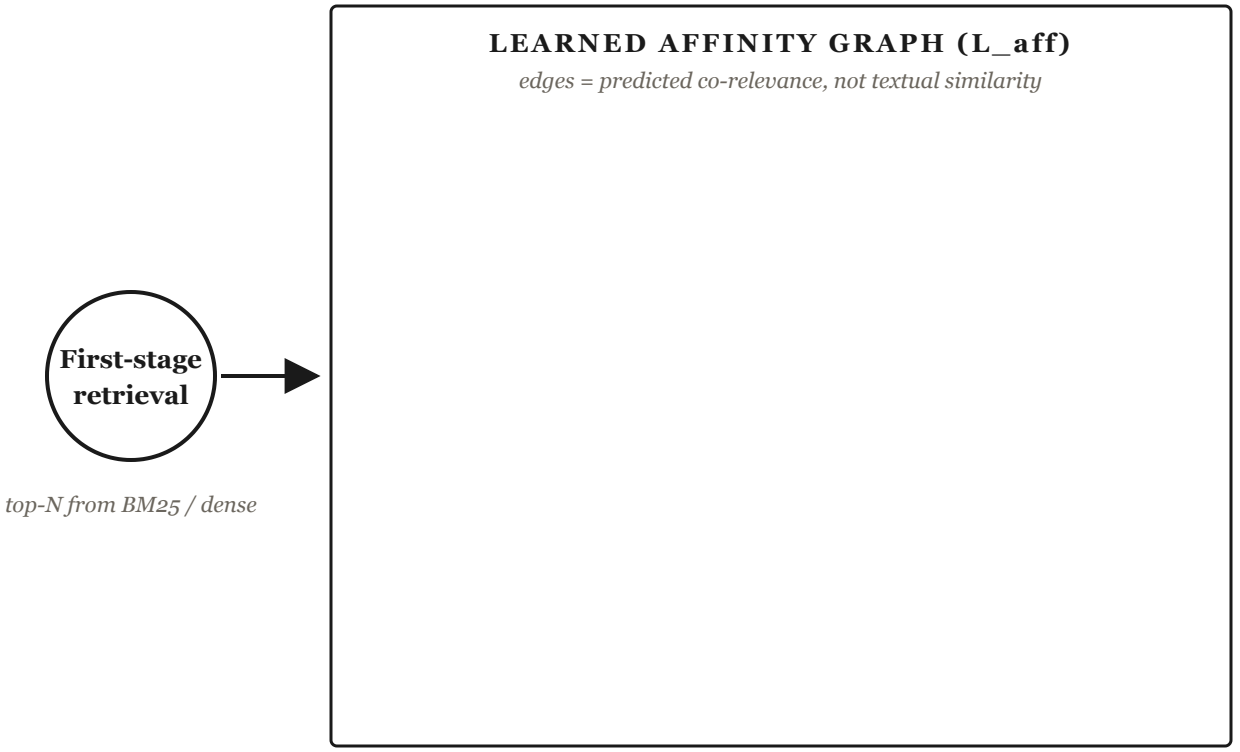
1. Rerank a small batch
2. For high-scoring docs, look up their **learned-affinity neighbors**
3. Score neighbors by **set affinity** — proximity to the *set* of already-relevant docs
4. Send the top neighbors to the reranker next
5. Repeat until budget exhausted

The learned affinity graph encodes **co-relevance**, not just textual similarity.

[15] Rathee, MacAvaney, Anand. Quam: Adaptive Retrieval through Query Affinity Modelling. WSDM 2025.

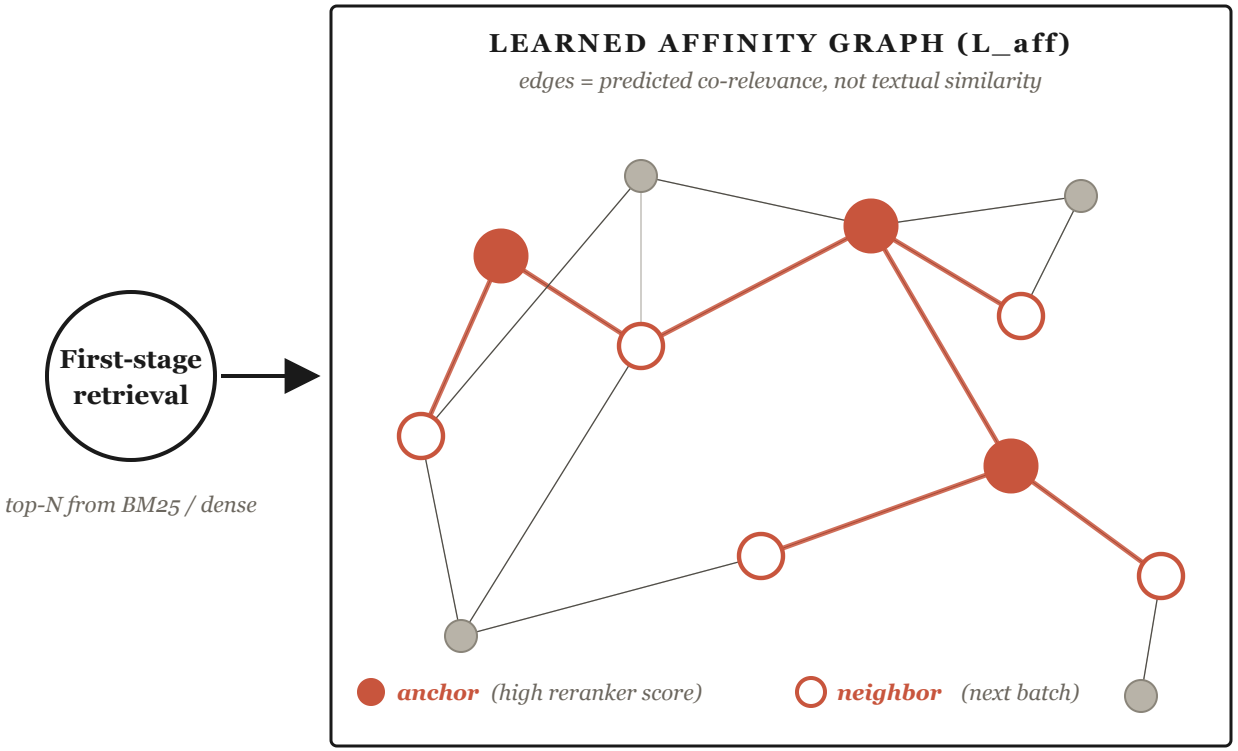
QUAM — Query Affinity Modelling

A learned co-relevance graph guides the next batch to rerank



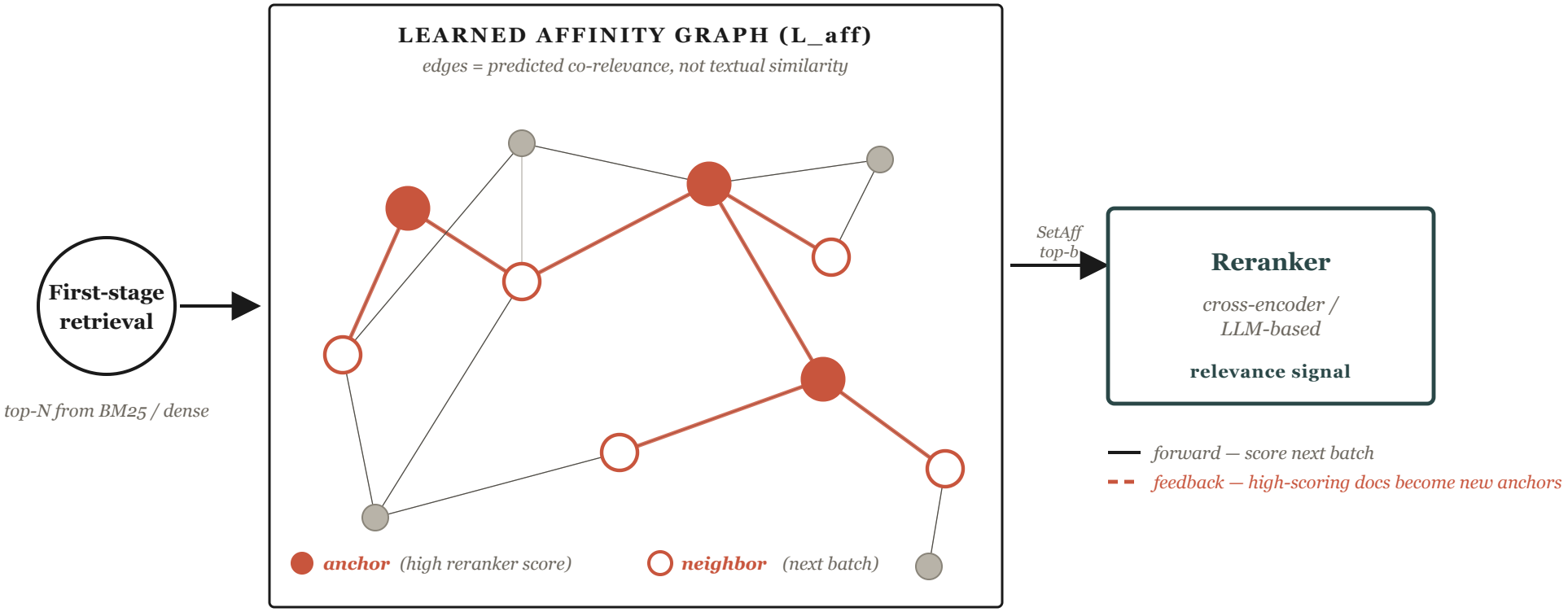
QUAM — Query Affinity Modelling

A learned co-relevance graph guides the next batch to rerank



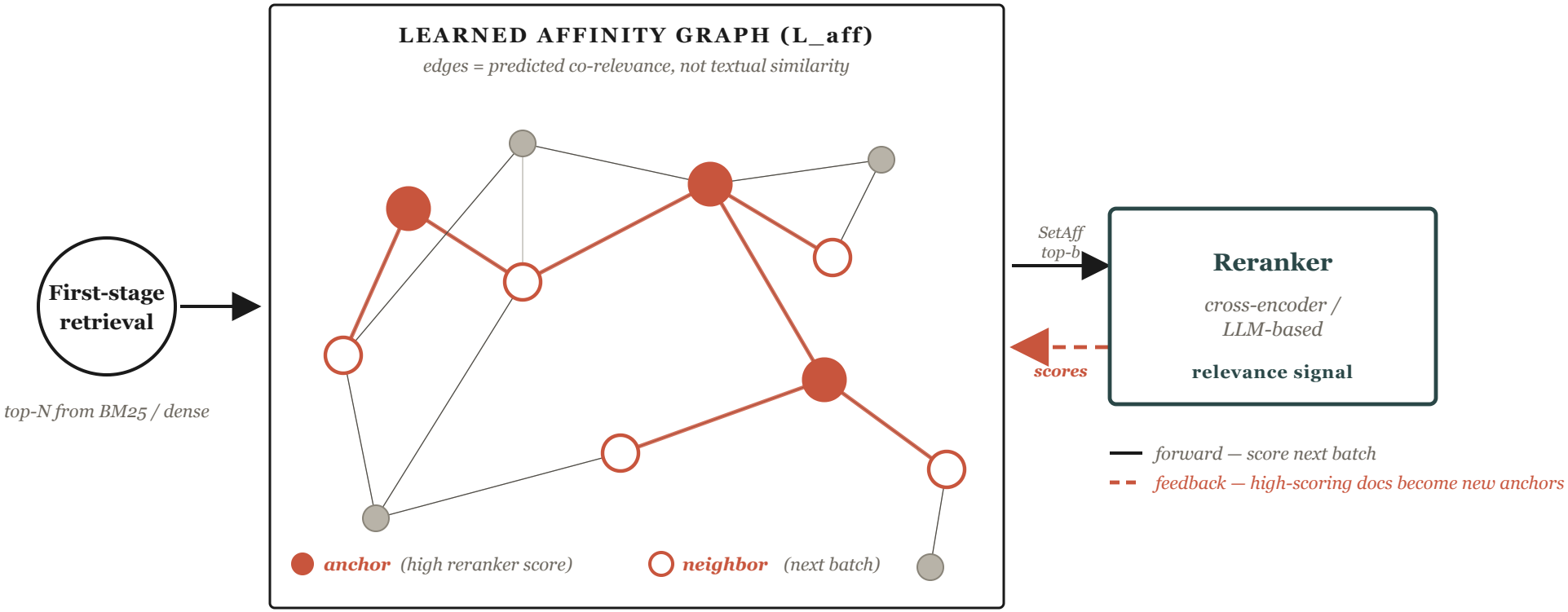
QUAM — Query Affinity Modelling

A learned co-relevance graph guides the next batch to rerank



QUAM — Query Affinity Modelling

A learned co-relevance graph guides the next batch to rerank



The graph is **learned, not crawled** — edges encode whether two documents satisfy the same need.

Up to +26% Recall@1000 over the standard cascade · ~3% latency overhead.