Predictive Prefetching for Retrieval-Augmented Generation

Hiding retrieval latency by predicting information needs

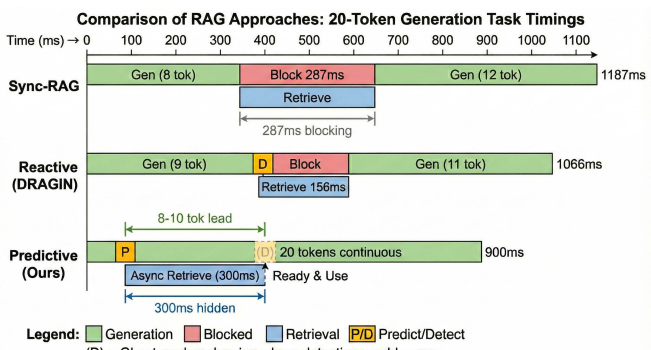
Wuyang Zhang **Shichao Pei**

University of Massachusetts Boston

ICML 2026

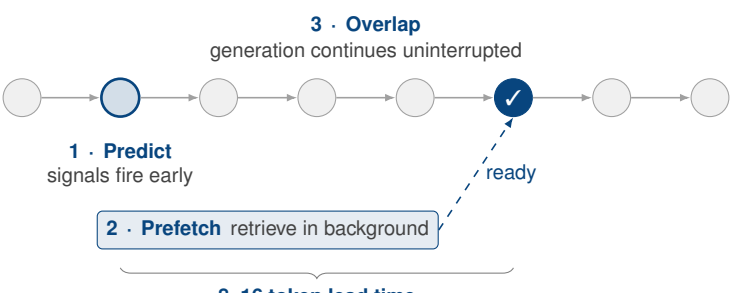
THE LATENCY BOTTLENECK IN RAG

- Retrieval-Augmented Generation grounds LLM outputs in external knowledge — but each external retrieval costs 100–500 ms, and complex queries trigger many.
- **Synchronous RAG** (top): generation blocks during every retrieval — 287 ms idle per call. The result is a forced trade-off between answer quality and latency.
- **Parallel methods** hide the time but issue stale, heuristic queries — prefetching content misaligned with the actual need.



KEY INSIGHT: INFORMATION NEEDS ARE FORESHADOWED

- Before uncertainty becomes critical, the model's internal state has already shifted: **entropy trajectories**, **attention allocation**, and **value-vector dynamics**.
- These precursors appear 8–16 tokens early — enough lead time to retrieve in the background before generation reaches the gap.
- The same signals also indicate what will be needed, so the query can be built before the gap appears.

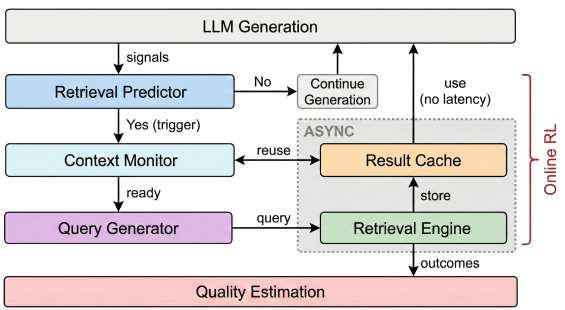


Approach: decouple retrieval from generation

Three learned components answer three questions:

- **Retrieval Predictor** — When to trigger, from internal signals within a lookahead horizon.
- **Context Monitor** — Whether context is sufficient, and how long to wait before querying.
- **Query Generator** — What to retrieve, as a context-aware query (T5-small).

Retrieval runs asynchronously; results sit in a cache and are used with no added latency. A synchronous fallback guarantees correctness.



LEARNING: PRETRAIN ONCE, THEN ADAPT ONLINE

- **Multi-task pretraining** on generation traces with automatic oracle labels — no manual annotation.
- **Online adaptation** via policy gradient with action-specific feedback (a contextual-bandit structure — dense, per-decision reward).
- A gated cascade of four actions — **Generate** / **Reuse** / **Accumulate** / **Fetch** — where each action's reward updates only the component that made it.
- Total cost: +62M parameters over an 8B model and +2.7 ms per token.

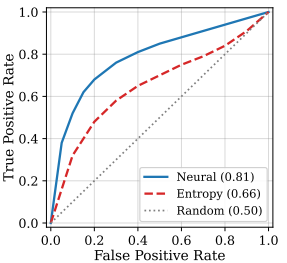
Results: latency down, quality held

Method	EM	TTFT	E2E	Ret/1K
No retrieval	32.8	42 ms	2.3 s	0
Sync-RAG	69.2	287 ms	9.2 s	86
PipeRAG (async)	66.8	118 ms	5.6 s	67
Ours (predictive)	**68.7**	**108 ms**	**5.2 s**	**59**

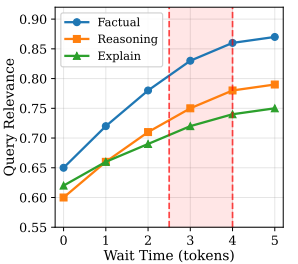
Llama-3.1-8B, four QA benchmarks (HotpotQA shown). TTFT: time-to-first-token; E2E: end-to-end latency; Ret/1K: retrievals per 1,000 tokens.

- −62.4% time-to-first-token, −43.5% end-to-end latency, 31% fewer retrievals.
- Answer quality within 1% of synchronous RAG — and better on *both* quality and efficiency than prior async (PipeRAG).

Why it works: accurate, well-timed prediction



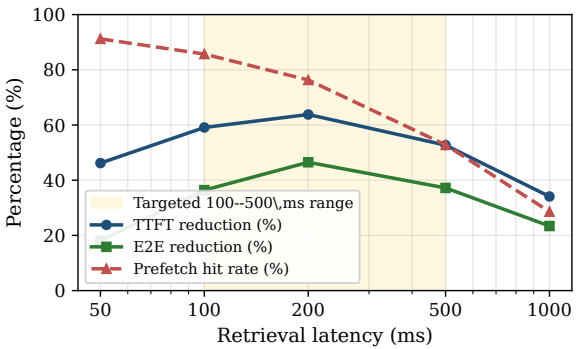
Prediction accuracy



Query relevance vs. wait time

- Predictor reaches AUROC 0.81 vs. 0.66 for entropy alone — a 23% gain in identifying upcoming needs.
- High-confidence predictions give 8.7 tokens of lead — over 400 ms for retrieval to complete.
- Waiting just 3–4 tokens raises query relevance by up to 23%.

It holds up across settings



Gains vs. retrieval latency

- Gains peak near 200 ms and degrade gracefully; the synchronous fallback always floors at the baseline.
- Consistent across six models (Llama, GPT-OSS, Qwen): TTFT -61.5% to -63.4%.
- Also improves **code completion** and **summarization**.
- Tail latency stays bounded: miss-query P99 (14.0 s) $<$ Sync-RAG P95 (15.2 s).

TAKEAWAY

- Retrieval needs are predictable from generation dynamics — several tokens in advance.
- Deciding when, whether, and what to retrieve hides latency without sacrificing relevance.
- -43.5% end-to-end latency, -62.4% time-to-first-token, quality within 1% of synchronous RAG.

Wuyang Zhang **Shichao Pei**
University of Massachusetts Boston
shichao.pei@umb.edu

Thank you.