Here’s a practical way to tune the retrieval knobs for **quality vs speed/cost**. Think of it as two stages:

* **Candidate gathering:** (expand\_queries + 1) × per\_query\_k
* **Final context size:** top\_k (what the answer LLM actually sees)

**Rules of thumb**

1. **Start small, scale up only if needed**

* Begin with: --top-k 3, --expand-queries 0–2, --per-query-k 6–8, rerank ON.
* If answers miss key evidence → increase expansions; if you get off-topic snippets → increase rerank or per-query-k a bit.

1. **Control candidates before rerank**

* Candidate count drives rerank latency/cost.  
  Example: --expand-queries 3 and --per-query-k 8 → up to **32 candidates**.
* Keep it ~12–24 for most use cases; go higher only for very broad queries.

1. **Keep top\_k tight (3–5)**

* More than 5 chunks often **hurts** answer quality (dilution + token bloat).
* If each chunk is smaller (e.g., --max-words 200 with overlap), you can push top\_k to 5–6.

1. **Use rerank by default**

* It’s the biggest quality win per token.
* Turn it off (--no-llm-rerank) only for quick, low-stakes runs.

1. **Budget awareness**

* Cost driver hierarchy (roughly):  
  rerank candidates > final top\_k prompt > query expansion (very small).
* If you need to cut cost: lower candidates first (reduce expand-queries or per-query-k) before lowering top\_k.

**Suggested presets**

**Fast (low cost)**

--expand-queries 0

--per-query-k 8

--top-k 3

--no-llm-rerank

Use when the question is narrow and the corpus is small/clean.

**Balanced (recommended default)**

--expand-queries 2

--per-query-k 6

--top-k 4

# (rerank ON by default)

Good recall without too many candidates (≈18). Keeps answers focused.

**Thorough (broad/ambiguous queries)**

--expand-queries 4

--per-query-k 8

--top-k 5

# (rerank ON)

High recall (≈40 candidates). Expect slower runs; best for critical analyses.

**Quick diagnostics**

* **Answer feels off-topic?**  
  Increase --expand-queries (0→2→4) or --per-query-k (6→8→10). Keep rerank ON.
* **Answer rambles or contradicts excerpts?**  
  Lower --top-k (5→4→3). Consider --cite and/or Strict mode (omit -g).
* **You see repeated or near-duplicate chunks in the answer?**  
  Slightly raise per\_query\_k and let rerank sort diversity, or reduce overlap in chunking.
* **Latency too high?**  
  Drop candidate count: reduce expansions or per-query-k. Keep top\_k steady.

**Back-of-the-envelope math (to plan runs)**

* **Candidates** ≈ (N\_expansions + 1) × per\_query\_k
* **Rerank prompt size** ≈ sum of candidate snippet lengths (we pass each chunk’s text).  
  Keep chunks ~200–300 words to bound this.
* **Final prompt size** ≈ top\_k × chunk\_size (+ instructions), so keep top\_k modest.

**Bonus tuning tips**

* If your corpus uses **varied terminology**, expansions help more than per-query-k.
* If your corpus is **uniform but large**, per-query-k helps more than expansions.
* Pair **--cite** with tighter top\_k for crisp, auditable reports.
* For long narrative answers, bump temperature modestly (e.g., 0.5–0.6), but keep citations on to anchor facts.

If you give me a sense of your typical question types and corpus size, I can propose a custom preset tuned to your workload.