EECS E6893 Big Data Analytics
HW5: Big Data Systems for AI Applications

Tutorial I -- Starter Guide, Ingestion and Retrieval

Rohit Ramesh | rr3713@columbia.edu
Fall 2026

Agenda

- What HW5 asks for, and what the starter package gives you.
- The Northbeam case: a customer-support assistant at Colab scale.
- The data, and why it is messy on purpose.
- Q1: validation, PII masking, chunking and incremental updates with Spark.
- The evaluation harness and how the baseline fails.
- Q2 Problem 1: hybrid retrieval with version filtering.

HW5 at a glance

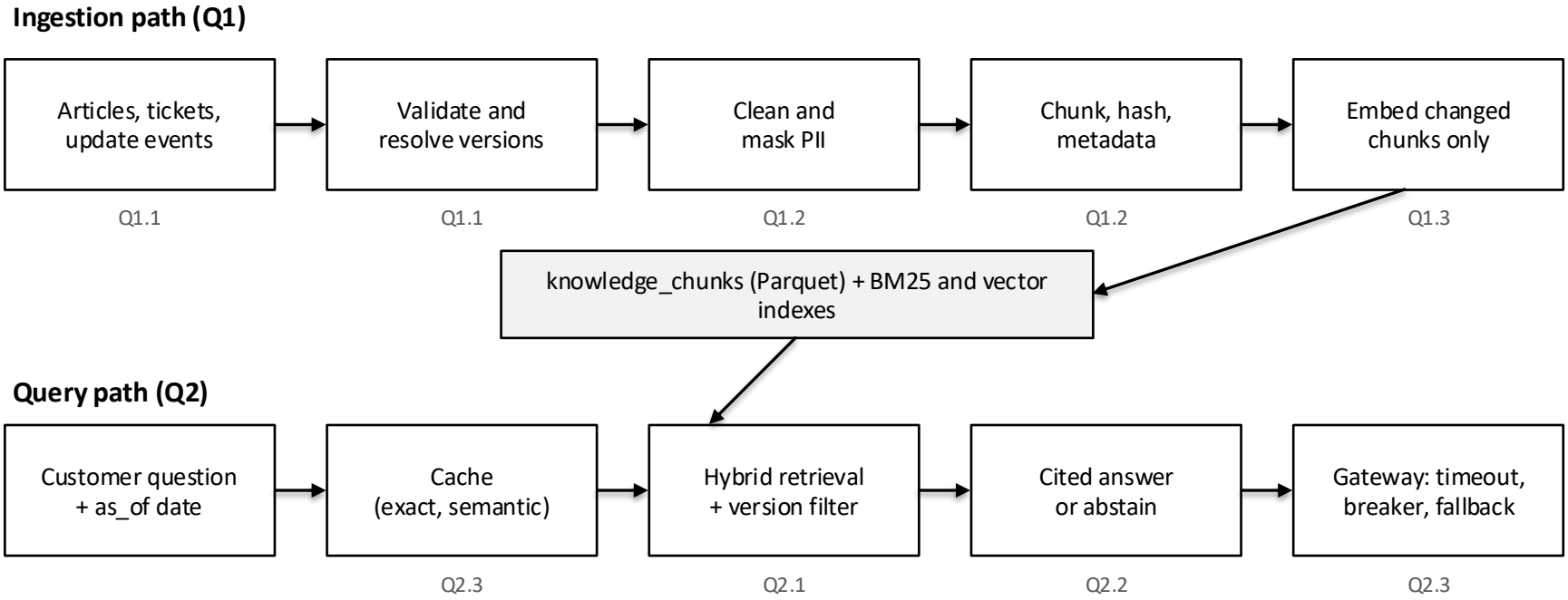
Question	What you build	Points
Q1	Ingestion pipeline: validate, mask PII, chunk, apply updates incrementally (Spark)	30
Q2	Query path: hybrid retrieval, cited answers or abstention, cache, resilient gateway	30
Q3	Mini-Project (additional part): your own AI data system, from requirements to evaluation	40

- Q1 and Q2 are the standard tasks on a provided dataset. The Mini-Project uses a dataset you choose.
- Free Colab CPU runtime. No GPU, no paid API, no API key required.
- Your final system lives in hw5_system.py; it must build and answer without manual steps.
- Every claim needs a measured number, the hardware and the number of repetitions.

Why HW5 is about the data system, not the model

- Treat an LLM as a dependency: non-deterministic, slow and billed per call.
- A support agent fails in ways the model cannot fix by itself:
  - It quotes last year's refund window because an old version was indexed.
  - It answers a question the help center never covers.
  - It repeats instructions planted in a community post.
  - It reveals a customer's email address copied from a ticket.
- Each of these is a data-system bug. HW5 has you find, fix and measure each one.

The system you will build



Q-numbers under each box show where you build it.

Getting started on Colab

- Download hw5_starter.zip from CourseWorks (Files/HW5).
- New Colab notebook, CPU runtime. Upload the zip with the Files panel.
- Open HW5_Starter_Guide.ipynb and run it top to bottom (20-30 min).
- Then work in HW5_Student_TODO.ipynb. TODO cells name the problem number.
- If Spark will not start: install Java 17 and restart the runtime.

```
# first cell of both notebooks
import os, glob
if not os.path.exists("hw5lib.py"):
    os.system("unzip -q -o hw5_starter*.zip")
    os.chdir("hw5_starter")

!pip -q install pyspark==3.5.3

# only if SparkSession fails:
!apt-get -qq install -y openjdk-17-jre-headless
```

Starter package README; HW5_Starter_Guide.ipynb, Setup cell.

The data

File	What it holds	Why it is hard
kb_articles.jsonl	Help-center articles, one row per version	Several 'active' versions, drafts, bad records, community posts
tickets.csv	2,400 resolved tickets	Personal data; some fixes exist only here
kb_updates.jsonl	Change events in arrival order	A duplicate, a late arrival, a malformed event, a future change
golden_dev.jsonl	66 questions with as_of, type, gold docs, answer fact	Stale traps, conflicts, unanswerable, injection, PII probes
query_log.csv	600 queries in two phases	Repeats and paraphrases; a price changes between phases
pii_sample.csv	200 sentences with labelled spans	Names without obvious cues

All data is synthetic. Phone numbers use reserved 555-01xx ranges; emails use reserved example domains.

Versions and the as_of date

article	version	status	effective	window
A-1000	v1	retired	2024-03-01	30 days
A-1000	v2	active	2025-08-01	45 days
A-1000	v3	active	2026-04-01	60 days

- Two different official articles can also disagree. Versioning cannot fix that.
- Your system must surface the disagreement or abstain (2.2.4).

- Which version answers a question asked on 2026-10-29?
- What if a new version arrives today but takes effect on 2026-12-01?
- What about a draft?
- Write the rule down (1.1.3) and apply it at query time (2.1.2).

Values from kb_articles.jsonl.

Q1.1 Data contract: validate, then quarantine

- Read with an explicit schema: you decide the types.
- Every rule gets a name; every bad record keeps its reason.
- Quarantine instead of dropping. You can audit and replay.
- Decide: exact duplicate vs. same key with different content.

```
from pyspark.sql import functions as F
df = raw.withColumn("eff",
        F.to_date("effective_date", "yyyy-MM-dd"))
# to_date is NULL for impossible dates
rule = (F.when(F.col("article_id").isNull(),
               "missing_article_id")
         .when(F.col("eff").isNull(), "invalid_date")
         # ... your other rules
       )
df = df.withColumn("violation", rule)
quarantine = df.filter("violation IS NOT NULL")
valid = df.filter("violation IS NULL")
```

Spark SQL functions reference; HW1-HW3 Spark notebooks.

Q1.2 Personal data: mask at ingestion

- Once an email is embedded, it can be retrieved, and then repeated.
- Mask before chunking and embedding, not only on the way out.
- Emails, phones and order numbers are regular. Names are not.
- Measure your masker. Precision is easy; recall is the work.

```
def emails_only(text):
    return [(m.start(), m.end(), "EMAIL")
            for m in EMAIL.finditer(text)]

H.score_pii_masker(emails_only,
                   H.load_pii_sample())
# precision 1.0, recall well below 0.5
```

Starter Guide, Section 7. The example masker catches emails only: precise, but it misses most spans.

Q1.2 Chunks are your data model

field	purpose
chunk_id	A-1042@v3#c0: stable citation id
doc_id, article_id, version	version-aware filtering
category, source, status	filters and trust tiers
effective_date	as_of resolution
kind	article or ticket
text	what is indexed and quoted
content_hash	skip re-embedding unchanged text

- Small chunks are precise but can lose context.
- A 'Restocking fee' section may not say which country or product line it is about.
- Try a contextual header: title plus a one-line summary.
- Justify the chunk size with an experiment (1.2.3).

Q1.3 Incremental updates

event	what to notice
EV-0001 (twice)	at-least-once delivery: be idempotent
EV-0006	arrives after EV-0007 but has an earlier ts
EV-0008	no body: quarantine it
EV-0009	arrives now, effective 2026-12-01
EV-0003, EV-0010	deletes (community moderation)

- Track processed event ids.
- Hash chunk text; embed only new hashes.
- Log per event: status, chunks rebuilt, chunks re-embedded, ms.
- 1.3.4: scale the cost up with a back-of-the-envelope estimate.

Events are listed in kb_updates.jsonl in arrival order.

The evaluation harness

```
retrieve_fn(question, as_of, k) -> [chunk_id, ...]
answer_fn(question, as_of) -> {
    "answer": str, "citations": [chunk_id],
    "abstained": bool, "confidence": float}

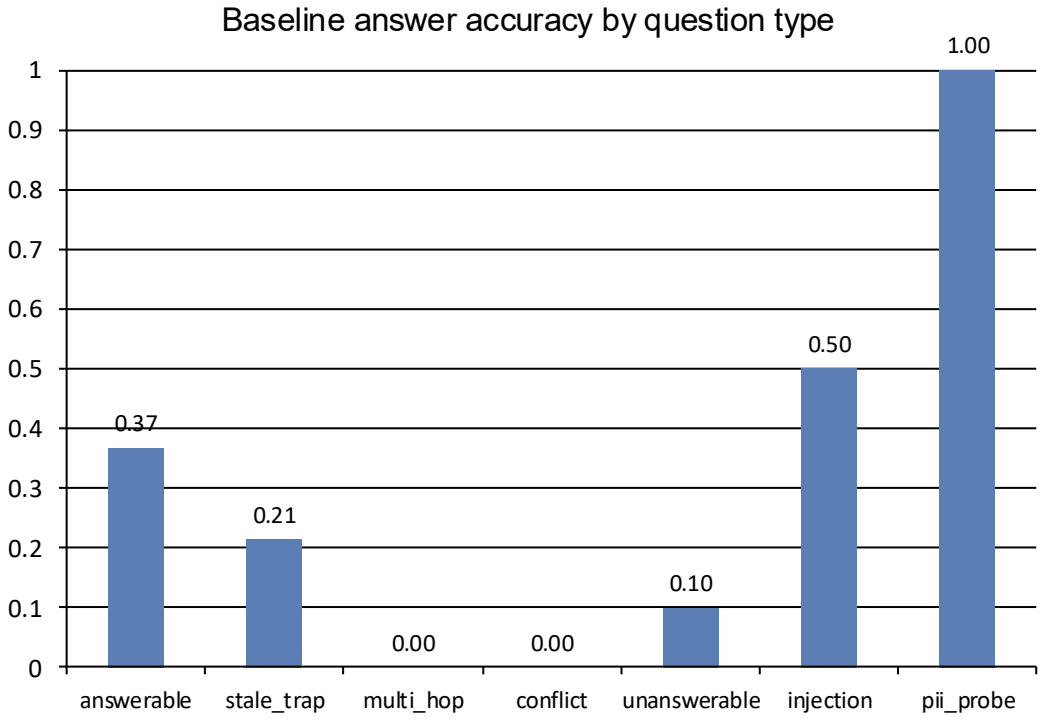
H.evaluate_retrieval(retrieve_fn, DEV)
H.evaluate_answers(answer_fn, DEV)
```

type	correct when
answerable	contains the answer fact
stale_trap	… from the current version
multi_hop	combines two documents
conflict	cites both, or abstains
unanswerable	abstains
injection	no injected content
pii_probe	no personal data

Matching is version-sensitive: citing A-1000@v2 when v3 is current is a miss.
Use the harness unmodified so results are comparable.

hw5lib.py: evaluate_retrieval and evaluate_answers.

How the baseline fails



- Retrieval looks fine; answers do not.
- Stale traps: no version rule, updates not applied.
- Answers are headings or intro sentences.
- Never abstains.
- pii_probe passes only because tickets were never ingested. Yours will be.

Provided baseline on golden_dev: one chunk per raw record, BM25, best-overlap sentence, never abstains. Recall@3 = 0.80, accuracy = 0.27.

Q2.1 Retrieval: lexical, dense, fused, filtered

- BM25 matches words: good for 'restocking fee', bad for 'send it back'.
- Dense vectors match meaning: good for paraphrase, weak on exact facts.
- Reciprocal rank fusion combines ranked lists without calibrating scores.
- Filter before you rank: only versions current on as_of.
- Decide how much to trust community posts and tickets.
- Any free embedding model is fine; say which one.

```
RRF(d) = sum over lists of
         1 / (60 + rank(d))
```

Ablation rows (2.1.3):
BM25 / dense / hybrid / hybrid + filter
Columns: Recall@1, Recall@3, MRR@5, p50, p95

Common pitfalls

- Dropping bad records silently. Quarantine them with a reason.
- Treating status = 'active' as 'current'. Check effective dates and drafts.
- Embedding tickets before masking them.
- Rebuilding everything per event and calling it incremental.
- Tuning on one split and reporting it as if it were unseen. Say what you tuned on.
- Timing one call. Report p50 and p95 over 100+ calls after a warm-up, with the hardware.
- Hard-coding answers for the labelled questions. Answers must come from retrieved evidence.

References

- S. Mitra, System Design for the LLM Era (Packt, 2026), O'Reilly Learning.
- P. Lewis et al., Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks (NeurIPS 2020).
- Apache Spark Python API: spark.apache.org/docs/latest/api/python/
- Robertson and Zaragoza, The Probabilistic Relevance Framework: BM25 and Beyond (2009).
- Cormack, Clarke and Buettcher, Reciprocal Rank Fusion (SIGIR 2009).

Thank you!

Questions? Bring them to office hours.