Build a **scalable address-matching pipeline** that:

1. **Ingests** both CSVs into a database of your choice (e.g. PostgreSQL, Opensearch, Elasticearch).
2. **Normalizes & parses** the raw text in transactions\_2\_11211.csv into components (street number, street name, unit, etc.)—you may use Python packages (e.g. usaddress), regex, or external APIs.
3. **Joins** each transaction row to its canonical address in 11211 Addresses.csv by:
   1. **Exact matching** on fully normalized fields.
   2. **Fuzzy matching** (e.g. Levenshtein, RapidFuzz) for “Main St” vs. “Main Street,” with a blocking strategy to avoid N² comparisons.
   3. **Other matching** Phonetic / Sound-Based Matching, Advanced Text Tokenization & Embeddings, n-Gram / Trigram Indexing, etc.
4. **Implements a waterfall/fallback** approach—for any record that fails the DB join, retry via:
   1. A secondary matching strategy (e.g. Soundex or Metaphone).
   2. An external address-validation API (e.g. USPS, SmartyStreets) if you choose (mocked or real).
5. **Scales**
   1. Show that your solution can handle **≥ 200 million** transaction rows (you can simulate by sampling/duplicating).
   2. Include performance metrics: total runtime, peak memory, and approximate cost (if using cloud APIs).
6. **Produces**
   1. A final output CSV or table: transaction\_id, matched address\_id, confidence score or match type (exact/fuzzy/API).
   2. A report of unmatched records and reasons (e.g. “failed parse,” “low fuzzy score,” “API rate-limited”).

**Deliverables**

1. **Code repository** (GitHub, GitLab, or zip) containing:
   1. **Ingestion scripts** (Python, SQL, or a mix)
   2. **Matching logic** (modular, e.g. parse.py, match.py, fallback.py)
   3. **Schema DDL** (CREATE TABLE / CREATE INDEX statements)
   4. **Performance harness** (scripts to run at scale & collect timings)
   5. **Configuration** (requirements.txt / environment.yml / Dockerfile)
2. **README** with:
   1. Setup instructions (local or Docker).
   2. How to run end-to-end.
   3. Performance results on your local machine (or cloud).
   4. Design write-up: trade-offs, libraries/APIs used, blocking strategies, fallback waterfall, and any assumptions.
3. **Optional** (extra credit)
   1. A simple REST endpoint (Flask/FastAPI) exposing /match\_address that returns a JSON match for a single input.
   2. A notebook or dashboard summarizing match accuracy (if you have ground-truth data).

Please submit all work to [data@trustscout.com](mailto:data@trustscout.com)