Muhammad Kamran - Python Intern Final Interview Task

Points which I have considered more to focus:

1. 100 different schemas
2. Dynamic Content
3. Antibot Measures
4. Accuracy and noisy html
5. Budget Constraints
6. Data mapping / normialization

Method 1 - Traditional

1. Problem Analysis

You need to identify challenges such as:

- Websites have different structures (no single schema).
- Anti-scraping mechanisms (CAPTCHAs, IP blocking, rate limits).
- Unstructured data (data in tables, JSON, or even images).
- Time and budget constraints (one week, limited resources).
- Accuracy requirement (how to ensure extracted data is correct and clean).

2. Tool / Technology Research

You must **recommend the right stack**:

- **Scraping frameworks**: Scrapy, Playwright, Selenium, Puppeteer.
- **Libraries**: BeautifulSoup, lxml, Requests, HTTPX.
- **AI/NLP for structure**: GPT-4, spaCy, transformers for parsing specs.
- **Data pipeline tools**: Airflow, Prefect (optional if large scale).
- **Proxies / Rotation**: ScraperAPI, BrightData (cheap/free proxy rotation).

3. Resource Optimization

- How to finish in **1 week** with **minimal cost**:
    - Use **AI-assisted schema inference** (LLMs can auto-parse product pages).
    - Build **semi-automated templates** (not manual for 100 sites).
    - Use **cloud free tiers** (AWS Lambda, Colab, or local VMs).
    - Apply **parallel scraping with proxy rotation** to save time.

4. Accuracy Strategy (95%)

- Use **data validation rules** (e.g., product price must be numeric, images must be valid URLs).
- Cross-check with **multiple extraction techniques** (XPath + AI parsing).
- Use **sample human validation** (spot-check 5–10% of the data).
- Apply **deduplication and normalization** (consistent format for specs).

5. Time Management

Day	Task		
1	Research websites, categorize by complexity, set up base scraper template		
2	Build generic scraper with modular config (works for multiple sites)		
3–4	Test on 20–30 websites, refine parsing logic, add AI-assisted schema handling		
5	Scale to 100 websites in parallel, deploy scraping jobs		
6	Data cleaning, normalization, validation		
7	Final QA, manual review, deliver dataset/report		

6. Risk Mitigation

Anticipate risks and propose **backup solutions**:

- **Anti-scraping measures** → rotating proxies, headless browsers.

- **Unexpected schema variations** → AI-based schema inference.
- **Time crunch** → prioritize top 80% of sites first (Pareto principle).
- **Budget constraints** → free tools, open-source frameworks, minimal paid APIs.

Method 2- Paper by Ganghadar and Kulkarni

Redesign: Multi-Site Product Data Scraping with Research Insights

1. Problem Analysis

- **Challenge:** Extract structured product data (titles, details, images, specifications) from 100 e-commerce websites in **1 week**, each with **different HTML schemas**.
- **Core difficulties:**
    - Websites use **different layouts** (tables, bullet lists, free-text specs, mixed content).
    - Many **anti-bot mechanisms** (CAPTCHAs, rate limits).
    - **Data quality requirement:** 95% accuracy.
    - Limited **time & resources** for building site-specific scrapers.

🔍 **Research Integration (from paper):**
Instead of relying only on table/list scraping, treat each page as **blocks of content**. Use **block detection + classification** (hybrid rule-based + ML) to **generalize across schemas**, reducing dependency on manual per-site coding.

2. Tool & Technology Research

- **Scraping Layer:**
    - Selenium / Playwright → handle dynamic content & JS-heavy pages.
    - BeautifulSoup / lxml → parse HTML for structural features.

- **Block Detection & Classification:**
    - Rule-based heuristics (regex for ":" separators, numbers, bullet lists, etc.).
    - ML classifiers (Random Forest, XGBoost, or LightGBM) trained on block-level features:
        - Presence of colon/number.
        - HTML tags (<table>, <ul>,).
        - Font/bold/italic usage.
        - Position in DOM tree.
    - Reuse features from the paper's approach.
- **ETL & Storage:**
    - Pandas for preprocessing.
    - SQLite/Postgres for structured storage.
    - Cloud storage (S3/Google Drive) for images.

3. Resource Optimization

- Instead of **100 custom scrapers**, build **1 generic pipeline**:
    - Crawl → Block Detection → Classification → Structured Output.
- Reduce manual work:
    - Use **active learning**: manually annotate only a small set of product pages, retrain classifier iteratively.
    - Use AI tools (ChatGPT/Claude) to validate extracted blocks quickly.

4. Accuracy Strategy (95% Goal)

- **Hybrid pipeline:**
    - **Rules**: High-precision capture of obvious cases (tables/lists).
    - **ML classifier**: Catch ambiguous/free-text specs.
- **Validation layer:**

    - Cross-check extracted specs against **known vocabularies** (brand names, unit dictionaries like “cm”, “kg”, “GB”).
    - Use **consistency rules** (e.g., numeric values must follow spec labels).
- **Quality Monitoring:**
    - Random sampling (5–10%) of outputs manually checked.
    - Active retraining if accuracy <95%.

5. Timeline (One Week Plan)

- **Day 1–2:**
    - Collect sample pages (5–10 per site).
    - Annotate **specification blocks** (training dataset).
    - Build feature extraction pipeline.
- **Day 3–4:**
    - Train/test classifier (Random Forest/XGBoost).
    - Integrate rule-based + ML detection.
    - Build ETL pipeline (CSV + DB).
- **Day 5–6:**
    - Run pipeline on all 100 sites.
    - Monitor scraping, fix failed cases.
    - Apply rate-limiting & retries.
- **Day 7:**
    - Validate accuracy (manual spot-check).
    - Generate final structured dataset.
    - Deliver reports (CSV/DB + summary).

6. Risk Mitigation

Risk	Mitigation Strategy
Website blocking / anti-bot	Use headless browsers + random delays + rotating proxies + polite scraping (robots.txt).
Schema diversity	Use block detection + classification (not site-specific scraping).
Accuracy below 95%	Active learning: correct errors on-the-fly, retrain model, reprocess affected sites.
Time constraints	Parallel scraping (multi-threading / async). Prioritize top 50 sites first, then expand.
Data inconsistency (duplicate/missing specs)	Apply cleaning rules, unit normalization, deduplication in ETL.

Limitation

- Going pure ML-heavy (deep learning) is overkill.
- A lightweight ML + rules hybrid is feasible:
    - Rules handle 80% obvious cases (tables, bullet lists, colon-separated specs).
    - ML classifier handles ambiguous cases (e.g., blocks with mixed text).
    - Training data needed: few hundred annotated blocks (manageable).
    - Cost: CPU-only training (cheap cloud VM or local machine).

Conclusion

I prefer to use Method 2 and here is why:

1. 100 different schemas (Block detection + ML classification)

2. Dynamic Content (Selenium / Playwright)

3. Antibot Measures (Headless browser + **random delays + polite scraping**)

Rotating proxies and retry logic.

4. Accuracy and noisy html

Hybrid pipeline:

- Rules (regex for tables, colon-separated specs, numeric values).
- ML classifier for ambiguous free-text blocks.

Validation layer:

- Check extracted values against vocabularies (brands, units like “kg”, “GB”, “cm”).
- Consistency checks (price must be numeric, image must be URL).

5. Budget Constraints

- Lightweight ML (RandomForest/XGBoost) → runs on CPU, no GPU/cloud costs.
- One pipeline works for all sites → **no per-site engineering cost**.
- Free/open-source stack (Python, Pandas, SQLite, Playwright).

6. Data mapping / normialization

- **ETL phase with Pandas** → normalization of units, deduplication, mapping synonyms (e.g., “RAM” vs “Memory”).
- Apply dictionaries for units, categories, and brand names.
- Store in **structured DB (SQLite/Postgres)** for consistency.