

Project Brief – Intelligent Customer Message Routing with Guardrails

Senior Data & AI Scientist - Take-home Assessment (2 – 3 hours)

1) Business context (banking)

Our retail bank receives thousands of secure messages daily via in-app and web banking. These messages range from suspected fraud on account, card disputes, balance/limits queries, credit-limit requests, and complaints. Slow or incorrect routing leads to poor CX, higher handling times, and critically missed economic-crime signals.

Your task is to propose and implement a minimal, production-minded approach to triage and respond to customer messages that:

- Classifies intent and routes to the correct queue (e.g., Fraud/Economic Crime Prevention, Disputes/Chargebacks, General Banking, Credit/Risk).
- Detects and redacts PII (e.g., account numbers, addresses) prior to storage/processing.
- Generates a first-draft response grounded in bank policy for certain intents (e.g., ‘card lost’ or ‘suspected fraud’), with clear guardrails and fallbacks.
- Justifies when to use LLMs vs. traditional ML/NN, discussing cost, latency, privacy, reliability, and evaluation.

2) What we provide (synthetic)

You will receive the following artefacts in this pack:

- `messages.csv` - ~300 anonymised secure messages with columns: `message_id`, `text`, `label`, `sensitive`, `suggested_queue`.
- `kb/` - 6–8 short policy snippets (Markdown) for: suspected fraud, card lost/stolen, dispute timelines, credit-limit policy, general servicing, and how to authenticate safely. You must ground draft responses on these snippets.
- `pii_patterns.yaml` — starter regex/patterns for common PII (sort code/account, phone/email, UK postcode). You may augment this.

If you prefer to bring your own models and not use the supplied labels, explain how you would label and evaluate quickly in a time-boxed setting.

3) Your assignment

A) Problem framing & approach (write-up + slides)

- Describe the triage problem, business KPIs (e.g., precision for fraud routing, latency SLA, redaction recall), and constraints (privacy, cost, explainability).
- Provide a decision rationale for LLM vs. traditional ML at each step (classification, redaction, response drafting, guardrails).

B) Solution design (slides + README)

- Present a high-level architecture: data ingress → PII redaction → intent classification → knowledge-grounded response drafting → observability/feedback.
- Call out your tech stack (libraries/models, vector store or not, model hosting choice, infra assumptions).
- Show guardrails (prompt constraints, allowed tools, regex/grammar checks, policy-grounding, unsafe-content filters, escalation paths).
- Discuss evaluation & monitoring (offline metrics and how you'd test drafts safely pre-production).

C) Build a thin vertical slice (code)

1. PII redaction: Redact sensitive tokens before any call to a non-local model/service. Show a simple test proving redaction works.
2. Intent classifier: Option 1 (Traditional ML) or Option 2 (LLM few-shot). Explain when you'd prefer each; consider latency and per-message cost.
3. Draft response (for 2 intents minimum): Use policy snippets from kb/ for grounding, include citations; provide no-LLM fallback and a confidence-based escalation.
4. Evaluation: Basic classification metrics; redaction unit tests; at least one automated check on draft responses (masking present, policy citation present, basic safety rule).

D) Explainability & risk

Surface features (traditional ML) or rationales/quotes (LLM). State residual risks: hallucination, prompt-leak, data exfiltration, bias, operational abuse.

4) Constraints & expectations

- Timebox: 2–3 hours build + 10–15 minutes slides. Prioritise simplest thing that could work; stub anything you can't complete but explain trade-offs.
- Languages/stack: Your choice. If using LLMs, prefer local/open models for the exercise, or mock calls; be explicit about cost & latency for a managed API.
- Use of AI assistants: Allowed. Note which parts were AI-assisted and how you validated outputs.
- Data privacy: Treat all inputs as confidential; do not send raw text to third-party APIs unless redacted first.

- Reproducibility: Provide a requirements file and one-command run entry-point.

5) Deliverables

- Code: runnable app or notebooks; README with setup/run, what's implemented vs. stubbed, assumptions, and cost/latency estimates.
- Slides (≤ 6 slides): problem framing & KPIs; architecture & tech stack; guardrails & risk; evaluation & early results; trade-offs & next steps.

6) Non-goals (to keep scope tight)

- No need for high accuracy on all intents—show a baseline and how you'd improve.
- No need for a web UI; CLI or notebook is fine.
- No need for advanced retrieval infrastructure; a simple in-memory store is acceptable.

7) Panel interview (potential follow-up)

Present solution to the wider team and discuss design decisions, tradeoffs and approach.