

BizzyCar — Applied AI Engineer Take-Home Challenge (2–3 hours)

Context

BizzyCar helps dealerships retain service customers through personalized communication and intelligent automation.

This exercise focuses on **designing and implementing a small backend system that orchestrates an AI model** (no model training). Show strong engineering around: input ingestion, prompt design, schema validation, retries/fallbacks, and handling hallucinations/errors.

You can complete this with **local-only code** using the included **mock LLM client**. If you prefer, you may optionally call a real model API via environment variables.

Scenario

Dealerships send batches of **free-form service notes and customer messages**. We want to automatically extract structured insights. In our example mock client, hallucinations come in the form of 'inspection' included in the service_intent array. You can check for this directly to detect if a hallucination has occurred. With a live AI integration you will have to consider how to validate the results.

Example input:

“2018 Camry in for check-engine light and oil change. Customer also mentioned tire pressure low.”

Desired output (JSON):

```
{
  "vin_detected": false,
  "vehicle_make": "Toyota",
  "vehicle_model": "Camry",
  "year": 2018,
  "service_intent": ["engine_diagnostic", "oil_change",
    "tire_pressure"],
```

```
"urgency": "medium",  
"raw_extraction_confidence": 0.83,  
"notes": "optional"  
}
```

Your Task

Build a small **Python service** (CLI or API) that:

1. **Ingests** an array of messages (from `data/messages.json`, a file path, or `POST /analyze`).
2. **Calls an AI model** (use the provided mock by default; real API optional).
3. **Parses & validates** the model output against a strict schema (Pydantic included).
4. **Handles hallucinations & errors** with **retry rules** (e.g., tighten instructions) and a **fallback** (e.g., rule-based extractor).
5. **Logs** inputs/outputs/errors in a structured way (JSONL). Redact obvious PII (emails, phone numbers).

Minimum requirements

- **Implement** a function that processes sample input and generates output using the mock AI client or an AI provider.
- Include at least one **retry strategy** and one **fallback** path.
- Provide **sample input** and **your produced output** files.
- Add a small **unit test** that covers a happy path and a hallucination/bad-JSON case.

Optional (nice-to-have)

- FastAPI app exposing `POST /analyze` and `GET /healthz`.
- A simple **confidence calibration** (e.g., heuristics to down-weight weak evidence).

What to Submit

- Your code (you may modify anything in `src/`).
- A short **README** (1–2 pages): error handling choices, trade-offs, and what you’d do next.
- Artifacts: `sample_output.json`, and if you run the API, a `curl` example.

Evaluation Rubric

Area	What “good” looks like
System Design	Clear separation: input → model client → validation → post-processing
API Integration	Robust against malformed outputs/timeouts; sensible retries/fallbacks
Reliability	Deterministic tests; structured logs; basic PII redaction
Code Quality	Readable, typed, small functions, minimal deps
Communication	README explains constraints, trade-offs, next steps
Bonus	Pydantic schema, structured logging, simple calibration/guardrails

Constraints

- Keep runtime under ~5 minutes locally.
- Python 3.10+ recommended.
- If you use a real provider, do **not** commit secrets; use env vars.

What’s in the Starter Kit

- `challenge.md` — the full brief above (copy-pasteable to your ATS/email).
- `README.md` — quickstart for candidates.
- `data/messages.json` — sample inputs.
- `src/`
 - `model_client.py` — **Mock LLM client** with tunable bad-JSON & hallucination rates (env vars).
 - `main.py` — Entry point for your code. You can start here.
- `sample_output.json` — example output format.
- `requirements.txt`, `Makefile`, `run.sh`, `.env.example`.

Candidate Quickstart (included in README)

```
python -m venv .venv && source .venv/bin/activate  
pip install -r requirements.txt
```