Option B: Adaptive Learning Engine

> **Note:** You will be provided the dataset; build the pipeline to work with the provided JSON once shared.

What you get

- One JSON file with **19 persuasion simulations** (scenario: *Persuading a Friend to Play an FPS Game*) (single user, ignore IDs). Each session has a transcript and rubric scores:
- **Overall (0–100)** and four criteria: **Clarity & Enthusiasm**, **Active Listening & Objection Handling**, **Effective Call to Action**, **Friendliness & Respectful Tone**.

Goal

Build a small pipeline that:

1. extracts **LLM-based features** from every session transcript,
2. implements an **adaptive policy** to choose next-step focus, and
3. produces **adaptive coaching** for the *next* session (focus skill + coaching card + scenario stub).

Then show that LLM features give **better signals** than numbers alone.

Must-haves

A) LLM-derived features (context for decisions)

From each session's transcript, compute at least **6** features in **[0,1]** that are plausibly tied to persuasion quality. Examples (pick ≥ 6, define each clearly):

- **Objection mirroring** (reflecting concerns before rebutting)
- **Question ratio** (questions / total user turns)
- **CTA explicitness** (clear ask + timeframe)
- **Empathy markers** (acknowledgement, labeling feelings)
- **Hedging intensity** ("maybe, might, kind of")
- **You/We orientation** (customer-centric wording)
- **Structure cues** (numbered steps, summaries)
- **Pushiness vs. collaborative tone**

Implement as one of:

- **Zero-/few-shot classification** prompts that return a 0–1 score, **or**
- An **embedding → simple classifier** (thresholds/rules), **or**
- A hybrid (LLM labels + lightweight rules).

Cache model calls; keep runs deterministic (seed, fixed prompts).

B) Adaptive policy (focus selection)

Implement **one** of the following policies using context features:

- **LinUCB (contextual bandit)** with context `x_t = [current rubric vector, improvement_rate, talk_ratio, latency, politeness, LLM features...]`
- **Thompson Sampling (Bayesian linear)** per action (same context)

Action space: {clarity, active_listening, call_to_action, friendliness}.
Reward (offline, proxy): `0.6 * Δ(skill_focus) + 0.4 * Δoverall`.
Add a simple safety: don't pick the same focus >3 times in a row.

C) Adaptive coaching generation (next-step content)

For each step *t* → *t+1*:

1. Use your policy to pick a **focus skill**.
2. **Generate a Coaching Card** (120–180 words max):
3. **Why this focus (2–3 lines)** tied to transcript signals.
4. **3 micro-exercises** tailored to the focus.
5. **Tone**: supportive and specific.
6. **Scenario stub** for next run (difficulty 0.45):
7. Persona one-liner, opening objection, and two follow-ups stressing that skill.
8. Include 2 toggles for difficulty 0.55 (e.g., shorter time, stronger pushback).

D) Lightweight evaluation

- **Feature usefulness:** Predict **Δoverall (t→t+1)** or **positive Δoverall (yes/no)** using:
- numeric baseline (rubric scores + talk ratio + latency), vs.
- baseline + your **LLM features**.
- Use leave-one-step-out (LOSO). Report **R^2** or **AUC/accuracy**.
- Show an ablation table (with/without LLM features).
- **Policy performance:** Report average reward and % positive Δoverall across the sequence for your chosen policy vs. a baseline (e.g., weakest-skill-first).
- **Alignment sanity:** For 2–3 steps, show that the chosen focus matches either the lowest rubric area or an LLM-flagged weakness.

Constraints

- **Language:** Python preferred (Pandas + your LLM client).
- **No UI required.**
- **Models:** Hosted API or local model; keep cost trivial.
- **Privacy:** Redact any real names if present.
- **Determinism:** Fix seeds, cache outputs; reruns should match.

Deliverables

- **Repo** with:
- `README.md` (how to run, model used, cost notes, design choices).
- `env.example` (expected env vars).
- `requirements.txt` or `pyproject.toml`.
- `main.py` (or notebook) end-to-end:
    1. loads JSON →
    2. computes LLM features →
    3. runs policy to pick focus + generates Coaching Card + scenario stub →
    4. runs evaluation and prints a report.
- **Artifacts**:
- `features.csv` (per session, incl. 6+ LLM features).
- `coaching_next.json` (per step: focus, card, scenario stub).
- `report.md` or console output with metrics and tables.

Reporting template (example)

```
Sessions: 19 | Steps: 18
Features: baseline (scores+talk_ratio+latency) vs. +LLM(8)

Policy = LinUCB (α=0.4)
Reward function = 0.6*Δskill + 0.4*Δoverall

Results:
- Baseline weakest-skill: mean reward=+0.021 | +Δ%overall=56%
- LinUCB (with LLM feats): mean reward=+0.037 | +Δ%overall=67%

Feature ablation:
- Baseline only:   AUC=0.58
- + LLM features: AUC=0.69

Example foci:
t=7: call_to_action | weak CTA score (0.42), LLM cta=0.23
→ Coaching Card + stub generated
t=12: active_listening | high interruption, mirroring=0.18

Validator avg: specificity=0.74, actionability=0.71
```

Starter prompt sketches

Feature extraction:

You are an expert rater of sales/persuasion dialogs.
Rate **0–1** the following: Objection mirroring, CTA explicitness, Empathy markers, Hedging intensity, You/We orientation, Structure cues.
Return JSON with 6 keys.
Transcript: <<>>

Coaching card generation:

You are a concise sales coach.
Given rubric scores + LLM features, pick {clarity, active_listening, call_to_action, friendliness} and produce a **coaching card** ≤180 words: why, 3 exercises, scenario stub\@0.45, upgrades for 0.55.
Return JSON with fields: focus, why, exercises[3], scenario_stub{persona,opening,followups[2]}, difficulty_upgrades[2].