`STAGE - 1`

Phase 0 – Define what you actually want to predict

Step 0.1 – Define target scores (labels)
Decide what numbers your model should output *per session*:

- `confidence` → 0 to 100
- `stress` → 0 to 100
- `clarity` → 0 to 100
- (optional) `hire_label` → 0 (no), 1 (maybe), 2 (yes)

Step 0.2 – Make a clear rubric for each

For example:

- **Confidence**
  - 0–30 → very hesitant, lots of “umm”, broken sentences.
  - 30–60 → mixed; sometimes confident, sometimes unsure.
  - 60–100 → speaks steadily, eye contact, clear answers.

Do the same for **stress** and **clarity**.
This rubric is what you’ll use while labelling.

Phase 1 – Collect raw data (record interviews)

Goal: get **N recorded interviews** with video + audio (start with 30–50; later grow to 200+).

Step 1.1 – Decide interview format

- Duration per session: **5–15 minutes**.
- Question style: basic HR + technical:
  - “Tell me about yourself.”
  - “Describe a challenging problem you solved.”
  - “Why should we hire you?”

You want variety: some good candidates, some bad, some mid.

Step 1.2 – Build a simple Recorder app (or just use tools)

You need **synchronized audio + video**:

- Easiest:
  - Use something like `obs` or `ffmpeg` to record webcam + mic into a single `.mp4` or `.mkv`.
  - Name files like `sess_0001.mp4`, `sess_0002.mp4`, etc.

Step 1.3 – Metadata per session

Create a small JSON/YAML per recording:

```
{
  "session_id": "sess_0001",
  "date": "2025-11-21",
  "interviewer": "you",
  "candidate_id": "anon_01",
  "notes": "first mock"
}
```

Keep these in a folder like `raw_sessions/`.

Step 1.4 – Consent & privacy

Even for mock interviews with friends/classmates, tell them:

- It's for an ML project.
- Data stays local.
- They can ask you to delete their session.

Phase 2 – Label the sessions (ground truth)

Goal: assign **confidence, stress, clarity** scores to each session.

Step 2.1 – Make a simple labeling spreadsheet

Create something like:

session_id	confidence	stress	clarity	hire_label	notes
sess_0001	70	25	80	2	Good overall

session_id	confidence	stress	clarity	hire_label	notes
sess_0002	40	60	50	1	Nervous

Use your rubric every time so scoring is consistent.

Step 2.2 – Optional: 2+ annotators

If possible, ask 1–2 friends to label some sessions too.

- Each person labels independently.
- Final label = average of all raters.

This greatly improves label quality.

Phase 3 – Extract features (turn raw media → numeric sequences)

Now we move from `.mp4` to sequences of feature vectors.
We'll use **fixed windows** and produce sequences of shape `(T, F)` per session:

- `T` = number of time windows
- `F` = feature dimension (we'll fix a design)

Step 3.1 – Choose your windowing scheme

Let's fix this:

- **Window size**: 1.0 second
- **Hop size**: 0.5 seconds (50% overlap)

So a 10-minute (600 s) video gives roughly:

- `T ≈ (600 - 1) / 0.5 ≈ ~1198 windows`

Later we'll cap `T` (e.g. 512) by cutting or down-sampling.

Step 3.2 – Decide exact features per window

We'll build one vector per window like:

> **Feature vector (F ≈ ~120)**

Example structure:

- **Audio statistics** (~35)
  - speaking_flag (0/1)
  - percent_voiced_in_window
  - RMS energy mean, std
  - pitch mean, std, median
  - 13 MFCC means
  - 13 MFCC stds
  - pause_count_in_window
  - avg_pause_duration
- **Text / STT features** (~15)
  - word_count
  - words_per_second
  - average_word_length
  - lexical_diversity (unique/total)
  - sentiment_score
  - filler_word_ratio (“uh”, “um” / total words)
  - STT_confidence_avg (if available)
  - [some syntactic ratio, e.g. pronoun ratio, optional]
- **Visual / face features** (~60)
  - face_present_ratio (0–1)
  - avg_head_pitch, yaw, roll & std
  - avg_eye_openness_L/R & std
  - gaze_off_center_mean
  - mouth_openness_mean & std
  - expression_prob_neutral/happy/sad/angry/surprised (5)
  - expression*prob_std** (5)
  - landmark_velocity_mean
  - landmark_velocity_std
  - blink_rate
  - body_pose_movement (if you do pose)

Total = ~110–130 features. **Fix this list** in a file `feature_schema.json` so you know ordering.

Step 3.3 – Implement the extractor

For each `sess_XXXX.mp4`:

1. **Read video + audio** (e.g. with `ffmpeg` or `moviepy`).
2. **For audio**:
   - Compute VAD → get speech segments.
   - Split into 1s windows with 0.5s hop.
   - For each window:
     - Extract MFCC, energy, pitch, etc.
     - Use STT (Vosk/Whisper) to get transcript + timestamps.
     - Compute text features in that window (based on words whose timestamps fall in that window).
3. **For video**:
   - Sample frames at e.g. 10 FPS.
   - Feed each frame to MediaPipe FaceMesh.
   - For each window:
     - Aggregate over all frames that fall in that window:
       - compute head pose stats
       - gaze/eye/mouth metrics
       - expression probabilities (averaged)
4. **Combine audio + visual features** per window → single feature vector `f[t]`.
5. Save result as:

```
features: numpy array shape (T, F)  # float32
timestamps: numpy array shape (T,)  # float64 or int64
labels: dict with confidence/stress/clarity/etc
session_id: str
```

Store as:

- File: `data/sessions_npz/sess_0001.npz`
- Contents: `features`, `timestamps`, `labels` (dict serialized via `np.savez_compressed`).

Repeat for all sessions.

Phase 4 – Prepare dataset for training

Goal: train/val/test splits and normalization.

Step 4.1 – Split by session

- List all `.npz` files.
- Sort them and split:
  - **Train**: 70%
  - **Validation**: 15%
  - **Test**: 15%

Crucial: **never** mix windows from the same session across splits.

Step 4.2 – Compute feature normalization (on train only)

- Concatenate features from **all training sessions**.
- For each feature dimension `i`:
  - Compute `mean_i` and `std_i`.
- Save as `feature_norm.json` or `.npz`.

Step 4.3 – Apply normalization

For each session (train/val/test):

- `features_norm[t, i] = (features[t, i] - mean_i) / std_i`
- Use `mean_i` and `std_i` from **training set only**.

This prevents data leakage and stabilizes training.

Step 4.4 – Optional: cap/resize sequence length

Transformers struggle with very long sequences. Pick:

- `max_seq_len = 512` windows.

Strategy:

- If `T > max_seq_len`, either:
  - keep the **center 512** windows, or
  - downsample (e.g. take every 2nd window).

You want each training example to be `(<=512, F)`.

Phase 5 – Build the temporal model (Transformer)

We now treat each session as:

- Input: `X` shape `(T, F)`
- Output: label vector `y` shape `(3,)` = `[confidence, stress, clarity]`

Step 5.1 – Decide model architecture

We stick to one clear design:

- Input projection: `Linear(F → d_model)`
- Positional encoding: standard sinusoidal.
- TransformerEncoder with:
  - `n_layers = 4`
  - `d_model = 256`
  - `n_heads = 8`
  - `ffn_dim = 512`
- Pooling: **mean pooling over time** (but only over real tokens, ignoring padding).
- Output head: `MLP(d_model → d_model/2 → 3)`

Step 5.2 – Attention mask

- For each batch, sequences will be padded to `max_T` (max length in batch or global `max_seq_len`).
- Create `mask` where:
  - `mask[b, t] = 1` if `t` < actual length.
  - `mask[b, t] = 0` if padding.
- Pass `src_key_padding_mask = ~mask` into the Transformer so it ignores padded positions.

Phase 6 – Training setup

Step 6.1 – DataLoader

- `Dataset.__getitem__` returns:
  - `features_norm` (T, F),
  - `labels` (3,),
  - `length T`
- `collate_fn`:
  - Batch those, pad features to `(B, max_T, F)`
  - Create mask `(B, max_T)`.

Step 6.2 – Loss + optimizer + metrics

- Loss: `MSE` over 3 targets:
  - `loss = mean( (y_pred - y_true)^2 )`
- Optimizer: `AdamW` with:
  - `lr = 3e-4`
  - `weight_decay = 1e-4`
- Metrics (computed on validation):
  - RMSE per target
  - R^2 per target

Step 6.3 – Training loop

For each epoch:

1. Set `model.train()`.
2. Loop over train batches:
   - Move `X`, `Y`, `mask` to GPU.
   - Forward pass → `Y_pred`.
   - Compute loss.
   - Backprop + optimizer.step.
3. After epoch, run validation:
   - `model.eval()`.
   - No gradients.
   - Compute RMSE & R^2.
4. Save **best model** when validation RMSE (mean of 3 targets) improves.
5. Early stopping: if no improvement for `N` epochs (e.g. 8), stop.

Phase 7 – Evaluate & sanity-check

Once you have a trained model:

Step 7.1 – Test set evaluation

- Run predictions on **test sessions**.
- Compute:
  - Test RMSE per target
  - Test R^2 per target

Example expectations (for early version):

- $R^2 > 0.4$ → model captures some pattern.
- $R^2 > 0.6$ → decent.
- As dataset grows and features improve, you can aim higher.

Step 7.2 – Human sanity check

Pick some test sessions, for each:

1. Watch the interview yourself.
2. Guess rough confidence/stress/clarity.
3. Compare your guess vs model output vs original label.

You want the model to be “in the ballpark” most of the time.

Step 7.3 – Error analysis

For sessions with biggest error:

- Check if:
  - Audio quality is bad.
  - Face is off-camera.
  - STT is garbage.
  - Label itself is questionable (maybe your rubric was unclear).

This tells you whether you should fix **data**, **labels**, or **model**.

Phase 8 – Turn it into an actual local profiler

Once training is good enough:

Step 8.1 – Save everything

- `best_model.pt` (state dict or scripted model)
- `feature_norm.npz` (mean/std)
- `feature_schema.json`
- `training_config.json` (hyperparams, date, etc.)

Step 8.2 – Build inference pipeline (offline first)

Given a new `.mp4`:

1. Run the **same extractor** to get features `(T, F)`.
2. Normalize with saved `mean/std`.
3. Cap/pad to `max_seq_len`.
4. Run the Transformer:
   - Get prediction `[confidence, stress, clarity]`.
5. (Optional) compute a timeline by:
   - running the model on sliding segments or adding a per-window head.

Step 8.3 – Real-time mode (later)

- Instead of `.mp4` file:
  - Capture frames + audio live.
  - Build windows on the fly.
  - Maintain a rolling buffer of last N seconds → feed model periodically.

Phase 9 – What to watch out for at each stage

- **Data collection**
  - Audio/video quality.
  - Camera angle (face visible).
  - Enough variation in candidates and behaviors.
- **Labelling**
  - Rubric consistency.
  - Some sessions re-labeled twice to see if you agree with yourself.
- **Feature extraction**
  - Check distributions (mean/std) of features.
  - Look for broken values: NaN, inf, constant zeros.
- **Training**
  - Training loss going down smoothly.
  - Validation loss went down, then maybe plateau.
  - If val loss goes up while train keeps going down → overfitting (try more data, regularization, smaller model).
- **Evaluation**
  - Don’t trust only a single metric; read outputs on real sessions.
  - Look for bias: e.g., candidate seated further away always gets lower clarity, etc.

`STAGE - 2`

Steps for the summarization for the Preprocessed data from the stage 1

1. What you already have (after temporal model inference)

For each session, your Transformer gives you something like:

```
{
  "confidence": 72.4,
  "stress": 21.0,
  "clarity": 81.3,
  "timeline": [
    {"t": 0, "events": ["low_eye_contact"]},
    {"t": 5, "events": []},
    {"t": 10, "events": ["high_pause_ratio"]},
    ...
  ]
}
```

You may also have:

- Per-feature importance (SHAP or attribution)
- Key quantified behaviors (e.g., average eye-contact %, pauses, gaze deviation, speaking rate, facial emotion stats)

---

2. You will NOT use a huge LLM for summarization

(Unless you want to—local LLM works too.)

But to keep your entire system **local, offline, deterministic**, and **lightweight**, the best approach is:

Rule-based + Template-based NLG
→ Very reliable
→ Fast
→ No hallucinations
→ Works perfectly for structured profile summaries

Later you can use a local LLM like **Llama 3.1 8B** (fully local) if you want natural flow — but rule-based is enough.

---

3. The Rule-Based Summary Generator (Core idea)

You convert *ranges of scores* + *detected patterns* into human-language statements.

Example:

- If **confidence > 70** → “The candidate appeared confident and composed.”
- If **stress < 30** → “There were minimal signs of stress.”
- If **clarity between 60–80** → “Their communication was generally clear, with slight irregularities.”

Combine these into a structured paragraph.

I will now give you the full logic.

---

4. Template Logic for Paragraph Generation

A. Confidence

```
if confidence > 80:
    "The candidate demonstrated excellent confidence, maintaining stable posture, steady eye contact, and assertive responses."
elif confidence > 60:
    "The candidate showed good confidence overall with minor moments of hesitation."
elif confidence > 40:
    "The candidate showed moderate confidence with noticeable hesitation and inconsistent delivery."
else:
    "The candidate appeared visibly hesitant, with frequent pauses and limited assertiveness."
```

B. Stress

```
if stress < 20:
    "Signs of stress were minimal, and the candidate remained calm throughout."
```

```
elif stress < 40:
    "Mild signs of stress were present, but the candidate largely maintained
composure."
elif stress < 60:
    "There were moderate stress indicators such as vocal tension or gaze
instability."
else:
    "The candidate displayed high stress levels, including inconsistent eye
contact and tense facial expressions."
```

C. Clarity

```
if clarity > 80:
    "Responses were highly clear, well-structured, and easy to understand."
elif clarity > 60:
    "Communication was generally clear, though a few explanations lacked
precision."
elif clarity > 40:
    "The candidate's clarity was inconsistent, with several unclear or
poorly structured responses."
else:
    "Communication was frequently unclear, making important points difficult
to follow."
```

---

5. Behavioral Events → Extra Sentences (Timeline → Text)

You automatically inspect timeline events and add micro-observations:

Example rules:

- If **eye_contact < 50% overall** →
  "The candidate struggled to maintain consistent eye contact."
- If **average gaze deviation > threshold** →
  "There were frequent moments of gaze diversion, indicating possible discomfort."
- If **high pause ratio (> 0.25)** →
  "The candidate paused often between sentences, which may reflect hesitation."
- If **emotion: stress/anger spikes** →
  "Occasional spikes in facial tension were observed."

- If **speaking_rate > 180 wpm** →
  "The speaking pace was unusually fast at times."

You can easily add more.

---

6. Combine everything into ONE paragraph

Example final output:

Final Profile Summary (Auto-Generated)

The candidate demonstrated good confidence overall, maintaining stable delivery with only minor moments of hesitation. Signs of stress were minimal, and they generally appeared calm throughout the interview. Communication clarity was high, with structured and understandable responses.

However, the system detected occasional extended pauses and a few instances of gaze diversion, which may indicate brief discomfort or uncertainty. Facial expressions were mostly neutral, with no significant signs of negative emotion. Overall, the candidate delivered a well-balanced performance with strong communication and emotional stability.

---

7. This can be produced with 80–150 lines of Python

You simply:

1. Load model outputs
2. Apply rule-based templates
3. Join them using natural connectors

This is a **post-processing layer**, separate from the ML model.
