48-HOUR TRIAL TASK — AI CONTENT ENGINEER

Submission — Varsha Tolani

1) Short system explanation (≤300 words)

I built a lightweight, production-minded AI content engine that converts live internet signals into repeatable short-form videos. The pipeline ingests live headlines (RSS/trending), extracts text, runs per-headline emotion classification, and maintains a short-term emotional memory to compute a "dominant mood" snapshot. The UI visualizes each headline with an emoji, color, and opacity tied to emotion, and the entire page reacts (mood pulse, emoji motion) so the output is clearly visible on camera.

Why this matters: instead of a single generated script, this is a system that continuously produces new, post-ready content every time it runs. It's fast to run (seconds), robust (handles TXT/PDF/DOCX inputs, defensive file handling), and repeatable — the same pipeline can be scheduled hourly or daily. For short-form platforms the unit of value is consistent, fast, and understandable: a 30–40s "internet mood" clip that feels live and is trivial to produce at scale.

2) Deliverables (what I'm submitting)

- **Working demo: Streamlit app source (repo/zip included).**
- **Short-form video: 30–40s screen recording of the running app (file / link included).**
- **Code: full app (app.py), helpers (app_funcs.py), `src/` modules (feed_loader, mood_engine, visual_engine).**
- **README + this submission text.**

3) How to run the demo (very short)

1. **Clone repo or unzip.**
2. **Install deps: `python -m pip install -r requirements.txt` (or see setup below).**
3. **Run: `python -m streamlit run app.py`**
4. **Open browser, choose Live World Feed and record, then switch to Plain Text and press Ctrl+Enter to show immediate reaction.**

4) Technical highlights (what I built and hard work I did)

- **Live ingestion: RSS/trending headlines pipeline with rate limits and fallback.**
- **AI layer: sentiment/emotion inference (open model) normalized to friendly labels used by UI.**
- **Emotional memory: rolling window that computes dominant mood across recent inputs.**
- **Visual engine: deterministic mapping emotion→emoji/color/opacity plus mood-reactive UI (pulse / emoji motion).**
- **Document parsing: robust TXT / PDF / DOCX extraction (PyMuPDF + python-docx), with encoding fixes.**
- **Stability & UX: used `st.cache_data` for expensive ops, defensive file handling, safe image loading, `.streamlit/config.toml` to silence dev warnings for crisp demos.**
- **Polish: single-click demo flow, Ctrl+Enter text trigger, visible motion for camera recording, download/balloon UI. I debugged environment issues, migration from old `st.cache`, guarding against gated HF models, and made the UI recording-ready.**

5) TASK 2 — THINK (short answers, bullets)

1. **Why would someone stop scrolling?**
    - **Hook: "What does the internet feel like right now?" is immediate and relatable.**

    - **Live reaction: headlines + changing emoji/color signal real-time behavior — viewers see movement and meaning in <3s.**
2. **What makes this different?**
    - **Data-driven, not prompt-driven: output comes directly from live signals.**
    - **Visible AI feedback: the model's output changes the UI itself (not just overlay text).**
    - **System focus: designed as a repeatable pipeline, not a single generated clip.**
3. **What part could become a product?**
    - **A "Daily Mood" content engine for publishers/brands (scheduled generation, multi-source inputs, brand styling).**
    - **Sentiment dashboards & alerts for PR/marketing teams.**
4. **What to automate next with more time?**
    - **Scheduled capture + automatic upload to social platforms (API posting).**
    - **Multi-source fusion (Twitter trends, Reddit, YouTube titles).**
    - **Smarter smoothing and noise reduction to avoid volatile mood swings.**

6) TASK 3 — SPEED CHECK (README snippet)

Setup (exact commands)

python -m pip install -r requirements.txt

python -m pip install pymupdf python-docx transformers torch streamlit-autorefresh

python -m streamlit run app.py

(If you prefer single install line: `python -m pip install streamlit pymupdf python-docx transformers torch streamlit-autorefresh`)

How long this took

- **Total build + iteration time: ~26–30 hours.**
    - **Core pipeline + model integration: 12–14 hrs**
    - **Document parsing, caching, defensive fixes, env debugging: 6–8 hrs**
    - **UI polish, recording/test, README + submission text: 6–8 hrs**

What I'd improve with +1 day

- **Add scheduled auto-record & uploader.**
- **Add more signal sources (social streams).**
- **Build a small admin to tweak sensitivity and visual templates.**

7) Challenges solved (proof of hard work)

- **Migrated deprecated `st.cache` to `st.cache_data` without breaking behavior.**
- **Avoided gated HF model by switching to an open, reliable model for demo portability.**
- **Fixed Windows Python env / PATH issues so demo runs reproducibly with `python -m streamlit`.**
- **Implemented graceful fallbacks for missing assets and sanitized PDF/docx extraction edge cases.**
  All fixes are committed and documented in the repo.

8) If this concept fails — why and next try

If the concept fails in review, likely reasons: the emotional signal is too noisy or reviewers expected a different signal type (e.g., trend vs. sentiment). Next, I'd pivot to higher-contrast signals (topic spike detection, "top surprise" stories) or multi-model ensemble to improve signal reliability.

Built to ship repeatable AI content, not demos.