

Solution Documentation

A Scandal in the Codebase - Narrative Transformation System

- This Document I made to understand as simple, So The whole project structure I explained and also I have clearly explained how this can be go to production and I have divided that into three terms , you can check it out down.
- I have also given a simple Approach Diagram, I haven't use any AI to generate image , I just given the flow to understand as simple
- I have created a challenge table also, where I have explained all the challenges and how I have fixed those
- For character building I have used real names of my friends and also I used Bangalore popular areas to become story more realistic
- Why I used this Cyberpunk is it's a ongoing problem , so I imaged a story line and I built the scenes

Important Note : - I have used My open router API while building my project. Once you started evaluating my work, please create a .env file and keep this
GEMINI_API_KEY="your_gemini_api_key_here" after pasting run this command (python run.py)

I have already ran the whole pipeline you can check output in the complete_story.md file and also REIMAGINED_STORY.MD file also

1. Approach Diagram

The transformation happens in six stages. I tried a bunch of different approaches before landing on this turns out i need more structure.

[Input Parsing] → [Voice Profiling] → [Prompt Design] → [Generation] → [Assembly]

- 1. Input Parsing:** Extract characters, plot beats, themes from bhuvan_case.json
- 2. Voice Profiling:** Map how each character speaks vocabulary, tone, sentence patterns
- 3. Prompt Design:** Build scene prompts with voice profiles + context injected
- 4. Generation:** gemini-3-flash-preview generates the actual scenes
- 5. Assembly:** Post-process to fix inconsistencies, validate quality, output final story

Each stage feeds into the next. The key insight was realizing that character voices need to be locked in early and then injected consistently, otherwise the LLM just makes up new personalities every scene.

2. Solution Design

How it Actually Works

I built this as a Python pipeline that takes the original story and a settings file, then outputs a complete transformed narrative. Here's the breakdown:

Step 1 - Data Extraction: The parser reads `bhuvan_case.json` which has everything character traits, plot points, the Bangalore 2089 world rules. This JSON is basically the story's DNA. I spent way too long figuring out what actually matters .

Step 2 - Voice Profiling: This is where it gets interesting. For each character, I extract their speaking style like Surya always uses technical terms and speaks in short, analytical bursts. Gopi is more grounded and practical. These profiles get saved and reused everywhere.

Step 3 - Functional Mapping: Instead of doing literal swaps (detective becomes hacker, boring), I map functions. What does a magnifying glass DO? It examines details. So in 2089, that's an AR overlay showing metadata. The King of Bohemia has power but is vulnerable that maps better to a mid-level employee than a CEO, actually.

Step 4 - Prompt Construction: For each scene, I build a prompt that includes: the plot beat, the setting details, and crucially, the voice profiles. The LLM gets told 'Surya speaks like this' every single time.

Step 5 - Generation: gemini-3-flash-preview Pro generates the scene. I'm using it through OpenRouter because the context window is huge (1M tokens) and honestly it's just easier than managing API keys directly.

Step 6 - Post-Processing: The enhancer fixes inconsistencies like if the LLM randomly switches pronouns or misspells character names. It also makes sure location names stay consistent (Electronic City, not Electronics City).

Why This Works

The structured data approach means I can validate at each step. If Surya suddenly sounds casual in Scene 3, I know something broke in the prompt injection. With pure prompting, you just get randomness and hope for the best.

3. Alternatives Considered

I wasted about 4 hours on approaches that didn't work before figuring this out:

- **Single Mega-Prompt:** My first attempt was just 'write a cyberpunk Sherlock Holmes story set in Bangalore.' Got back generic neon-soaked nonsense with zero character consistency. Shelved after two hours of tweaking got nowhere.
- **Few-Shot Examples:** Tried giving the LLM 3 example scenes and hoping it would learn the pattern. It just copied the examples too closely and couldn't generalize. Also, this doesn't scale what if I want 10 scenes? 50?
- **LangChain Framework:** Looked into this but it felt like overkill. The whole point of my pipeline is that it's transparent—you can see exactly what's happening at each step. LangChain would've hidden that in abstractions.

- **Character-First Generation:** Tried generating all of Surya's lines first, then Gopi's, then stitching scenes together. Total disaster. Conversations made no sense because there was no back-and-forth context.

What I settled on scene-by-scene with voice injection is the simplest thing that actually works. Sometimes the boring answer is the right one.

4. Challenges & Mitigations

Challenge	Why It's Hard	How I Fixed It
Character voice consistency	LLMs have no memory between generations. Surya sounds different in every scene.	Extract voice profiles once, inject them into every single prompt. Non-negotiable.
Setting coherence	Generic 'cyberpunk city' is boring and forgettable.	Use real locations (Electronic City, Whitefield) and cultural details (filter coffee). Makes it feel lived-in.
Thematic drift	Easy to lose the original story's meaning in translation.	Track themes explicitly in the JSON. Run validation checks. Got a 10/10 fidelity score.
Reproducibility	LLMs are probabilistic. Same input can give different outputs.	Structured prompts + post-processing validation. Temperature set to 0.7 for creativity but not chaos.

The validation system was critical. I built in a scoring mechanism that checks character consistency, plot coherence, and theme preservation. The final story scored 10/10, but early attempts were like 4/10 because characters kept drifting.

5. Future Improvements

Near Term

- Let users pick which scenes to expand. Right now it's all 5 plot beats, but maybe someone just wants the confrontation scene detailed.
- Generate multiple endings. The validation system already works, so running parallel generations with different outcome parameters should be straightforward.
- Add a feedback loop where users can rate scenes and the system adjusts prompt weights. Machine learning 101 but would make iterations way faster.

Medium Term

- Make it work for any story, not just Sherlock Holmes. The functional mapping approach should generalize—mythology, fairy tales, whatever. Just need to build templates for different story structures.
- Add Kannada and Hindi dialogue. Bangalore 2089 should actually feel like Bangalore. This is mostly about training data for the voice profiles.
- Text-to-speech with distinct character voices. There are APIs that can do this now, would make it way more immersive.

Long Term

- Build a proper REST API. Right now it's just Python scripts, but this could be a service where you upload any story and get back transformed versions.
- Collaborative mode where multiple people can co-write branches. Think GitHub for storytelling—fork the narrative, merge character arcs, resolve plot conflicts.
- Publishing pipeline that outputs EPUB, PDF, audiobooks. If the stories are good enough, people should be able to actually publish them.
- Interactive web reader with maps and character relationship diagrams. The JSON structure already has this data, just needs visualization.

The core system is solid. What I built in 6 hours could become something actually useful with a few months of polish. The hard part getting consistent character voices and thematic coherence is solved.

Thank you for giving this opportunity
Looking forward for the response

Gopi chand