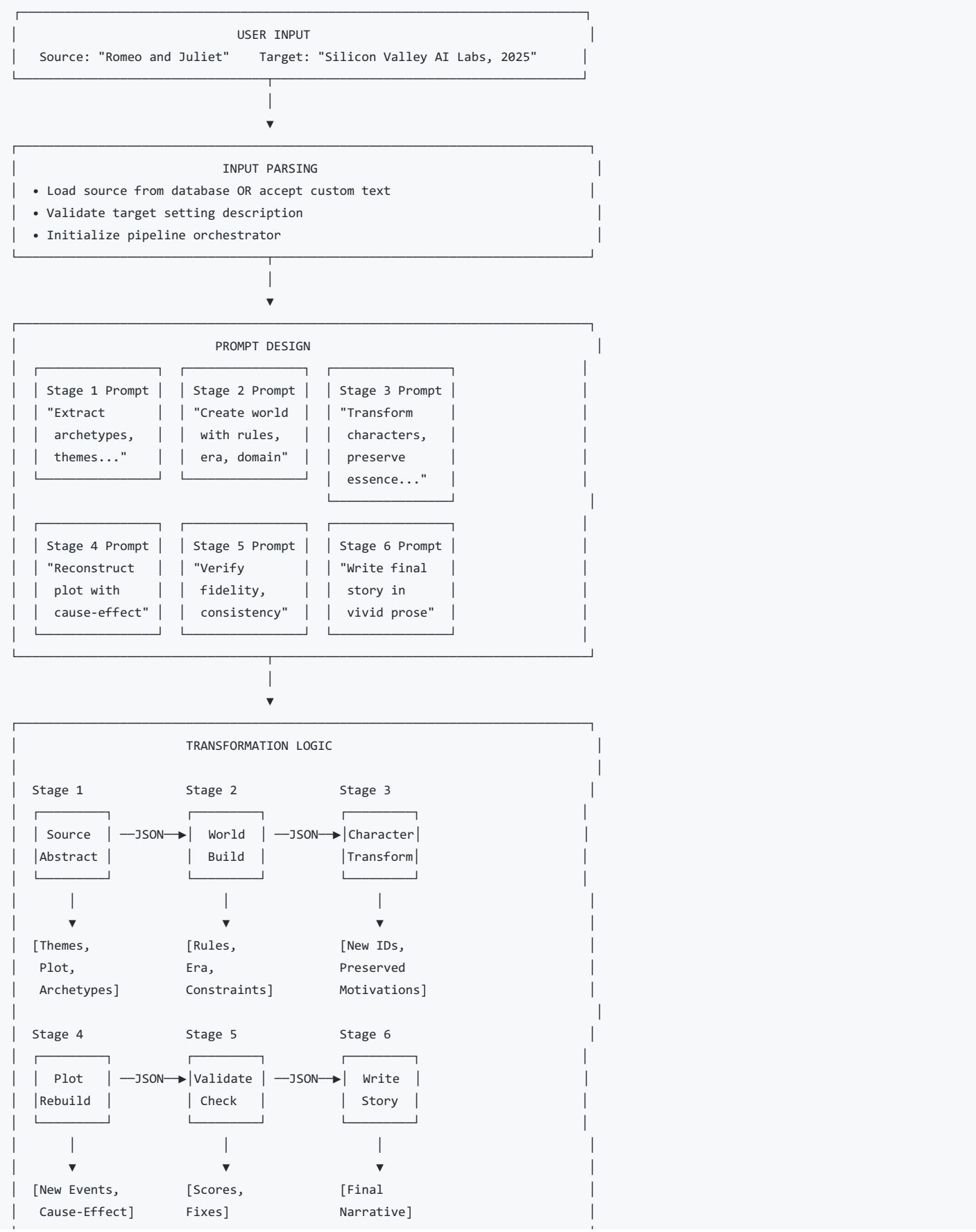
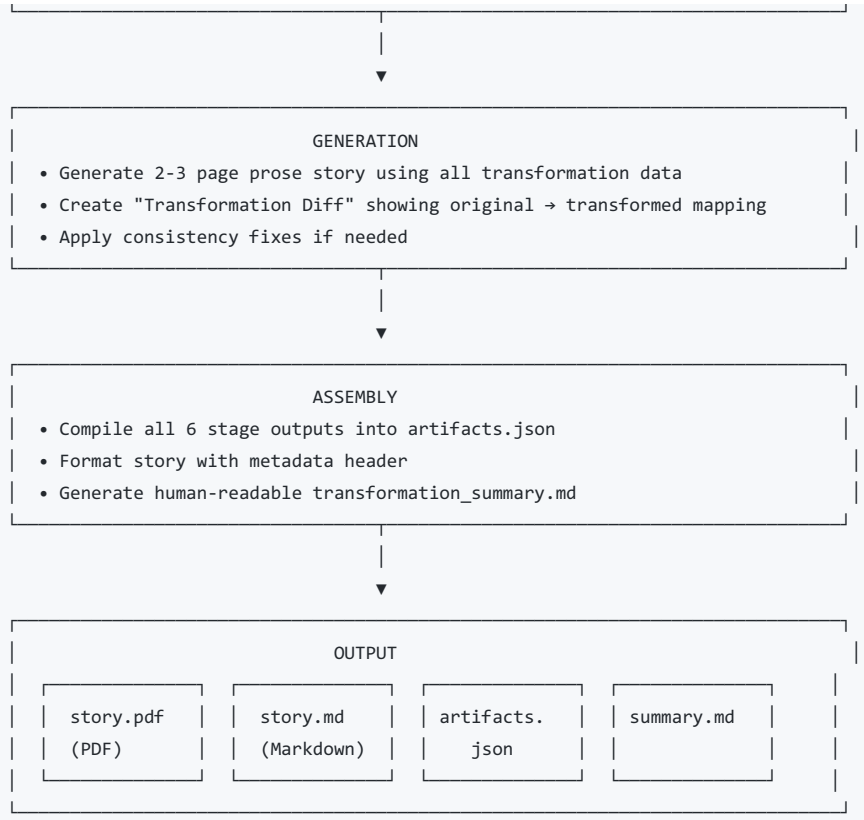
Solution Design Document

AI Narrative Transformation System

---

1. Approach Diagram





2. Solution Design: End-to-End System

2.1 Technology Stack

Component	Technology	Rationale
LLM API	Groq (Llama 3.3 70B)	Free tier, fast inference, high quality
Language	Python 3.12	Rich ecosystem, rapid development
CLI	Rich library	Beautiful terminal UI with progress bars
PDF Export	FPDF2	Lightweight, no external dependencies

2.2 Pipeline Modules

Stage	Module	Input	Output	Key Prompt Strategy
1	`source_abstraction.py`	Story text	JSON: themes, archetypes, plot	Extract without copying
2	`world_definition.py`	Target setting + themes	JSON: world rules, era, constraints	Enforce internal logic
3	`character_transform.py`	Characters + world	JSON: new identities, preserved flaws	Map archetype → role
4	`plot_reconstruction.py`	Plot + characters + rules	JSON: new plot with cause-effect	No deus ex machina
5	`consistency_check.py`	All previous outputs	JSON: scores, issues, fixes	Multi-criteria evaluation
6	`output_generator.py`	World + chars + plot	Story text + transformation diff	Vivid prose generation

2.3 Prompt Engineering Approach

Each prompt template includes:

1. **Role Assignment**: Clear persona (e.g., "You are a narrative analyst")
2. **Structured Input**: Context from previous stages formatted as text
3. **JSON Schema**: Exact output format specification
4. **Guardrails**: Explicit prohibitions (no copying, no deus ex machina)
5. **Quality Criteria**: What makes a good output

Example (Stage 1 - Source Abstraction):

```
You are a narrative analyst. Analyze the following source material...
Do NOT copy any original text verbatim.
Respond in the following JSON format ONLY:
{
    "core_themes": [...],
    "character_archetypes": [...],
    "plot_structure": {...}
}
```

---

3. Alternatives Considered

Approach	Pros	Cons	Decision
Single Mega-Prompt	Simple, one API call, fast	No control over intermediate steps, inconsistent outputs, no artifacts, debugging impossible	❌ Rejected
LangChain Agents	Flexible, auto-routing	Overkill for structured task, less predictable, harder to debug	❌ Rejected
RAG with Vector DB	Could retrieve similar transformations	Unnecessary complexity, no training data available	❌ Rejected
Few-Shot Prompting Only	Less code, faster	Hard to ensure JSON structure, limited context window	❌ Rejected (but used within stages)
Modular Pipeline ✓	Transparent, debuggable, reproducible, intermediate artifacts	More code, multiple API calls	✅ **Selected**

Why Modular Pipeline Won:

1. **Reproducibility**: Same inputs → same outputs (deterministic stages)
2. **Debuggability**: Inspect any intermediate artifact to find issues
3. **Extensibility**: Easy to add/modify individual stages
4. **Transparency**: Shows the transformation process, not just results

---

4. Challenges & Mitigations

Challenge	Impact	Mitigation Strategy
Coherence Across Stages	Characters might contradict world rules	Each stage receives ALL previous outputs as context; structured JSON ensures consistent data flow
Avoiding Copied Content	Legal/originality issues	Explicit prompt instruction: "Do NOT copy original text"; Stage 5 checks for copied elements
Cause-Effect Logic	Plot holes, deus ex machina	Stage 4 requires `cause` field for every event; validation function checks stakes and effects
Cultural Sensitivity	Stereotyping, offensive content	Dedicated scoring in Stage 5; prompt includes sensitivity guidelines
JSON Parsing Failures	Pipeline crashes	Try-catch with regex fallback extraction; graceful error handling

Challenge	Impact	Mitigation Strategy
Theme preservation	Losing original story's essence	Explicit theme mapping in Stage 2; Stage 5 scores thematic fidelity
Reproducibility	Different outputs each run	Fixed temperature (0.7 for creative, 0.3 for analytical); JSON response format enforcement

---

5. Future Improvements

5.1 Short-Term (1-2 weeks)

- **Web Interface**: React/Next.js UI for source/target selection
- **More Source Materials**: Add 20+ public domain works
- **Multiple Output Formats**: EPUB, screenplay, audiobook script

5.2 Medium-Term (1-2 months)

- **REST API**: Flask/FastAPI endpoint for programmatic access
- **Batch Processing**: Transform multiple stories in parallel
- **User Accounts**: Save transformation history
- **Interactive Editing**: Let users modify intermediate artifacts and regenerate

5.3 Long-Term (3-6 months)

- **Fine-Tuned Model**: Train on successful transformations for better quality
- **Multi-Language Support**: Generate stories in different languages
- **Collaborative Mode**: Multiple users work on same transformation
- **Quality Scoring ML**: Train classifier on human ratings
- **Production Deployment**: Kubernetes, rate limiting, monitoring

5.4 Scalability Path

```
Current (Demo)          →   MVP Product          →   Full Platform
──────────────────────────────────────────────────────────────────
CLI only                →   Web UI + API         →   Mobile apps
7 source materials      →   100+ sources         →   User uploads
Single user             →   Multi-tenant         →   Enterprise
Local execution         →   Cloud (Groq/OpenAI)  →   Hybrid edge
```

---

6. Bonus Feature: Transformation Diff

A unique feature that visualizes the systematic transformation process:

Element Type	Original	Transformed	Preservation Note
Character	Romeo Montague	Rohan (lower-caste boy)	Star-crossed lover archetype preserved
Character	Juliet Capulet	Asha (upper-caste girl)	Forbidden love motivation preserved
Theme	Family feud	Caste system rivalry	Social barrier conflict preserved
Setting	Renaissance Verona	Indian village cricket	Competition context preserved
Climax	Double suicide	Match confrontation	Tragic peak moment preserved

Value: Demonstrates the transformation is methodical, not random—key for the assignment's "show process" requirement.

---

7. File Structure

```
narrative-transformer/
├── run.py                      # CLI entry point (interactive + command-line)
├── config.py                   # Groq API configuration
├── requirements.txt            # groq, rich, fpdf2, python-dotenv
├── pipeline/
│   ├── orchestrator.py         # 6-stage coordinator with progress UI
│   ├── source_abstraction.py   # Stage 1: Extract narrative DNA
│   ├── world_definition.py     # Stage 2: Build alternate universe
│   ├── character_transform.py  # Stage 3: Map characters
│   ├── plot_reconstruction.py  # Stage 4: Rebuild plot logic
│   ├── consistency_check.py    # Stage 5: Validate quality
│   └── output_generator.py     # Stage 6: Generate story + PDF
├── prompts/
│   └── templates.py            # All 6 prompt templates with JSON schemas
├── data/
│   └── source_materials.json   # 7 public domain stories
├── docs/
│   └── solution_design.md      # This document
└── output/
    ├── story.pdf               # Final story (PDF)
    ├── story.md                # Final story (Markdown)
    ├── artifacts.json          # All intermediate data
    └── transformation_summary.md
```

Document Version: 1.0 | Last Updated: January 2026