Drawing Figures with AI/LLM

Apaper – Translating Structured Text to Visual Diagrams

isomo[1]

2025-12-15

[1]github/jiahaoxiang2000

The Problem

From Concept to Figure

Traditional Approach

- Manual drawing tools
- Pixel-based images
- Hard to version control
- Difficult to modify
- Time-consuming iteration

AI-Assisted Approach

- Generate library code
- Metadata-rich formats
- Git-friendly text files
- Easy modification
- Rapid iteration

Core Concept

Library Code Generation

Instead of generating images directly, generate code for figure libraries:

Library Code Generation

Instead of generating images directly, generate code for figure libraries:

Ecosystem	Library	Use Case
LaTeX	TikZ	Academic papers, precise diagrams
Markdown	Mermaid	Documentation, flowcharts

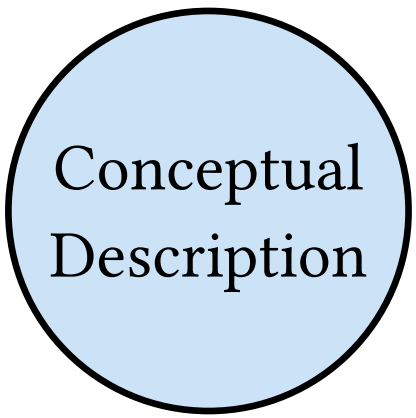
Library Code Generation

Instead of generating images directly, generate code for figure libraries:

Ecosystem	Library	Use Case
LaTeX	TikZ	Academic papers, precise diagrams
Markdown	Mermaid	Documentation, flowcharts

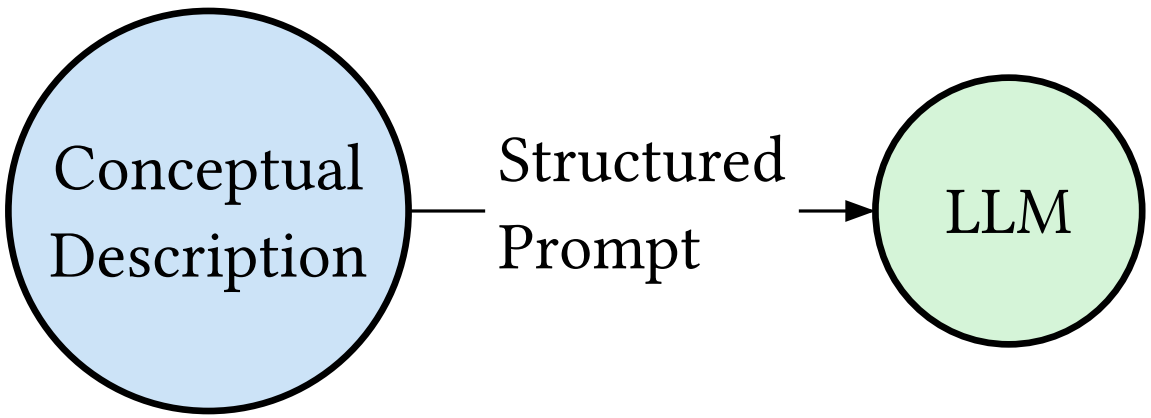
Key Insight: Libraries encode structure and semantics, not just pixels.

The Approach



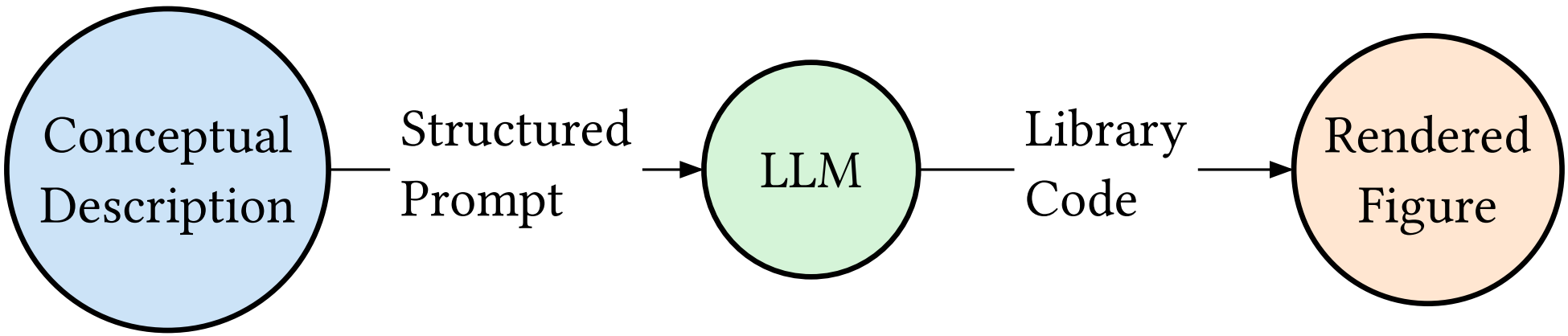
The prompt contains constraints and best practices specific to the target library.

The Approach



The prompt contains constraints and best practices specific to the target library.

The Approach



The prompt contains constraints and best practices specific to the target library.

Prompt Engineering

Key Components of a Good Prompt

Example from `figure-tikz.md`[1]:

[1] https://github.com/jiahaoxiang2000/deeper-paper

Key Components of a Good Prompt

Example from `figure-tikz.md`[1]:

```
1. **Pure relative positioning only**
2. **NO anchor-based offsets** — avoid manual coordinates
3. **Simple paths**
4. **Row-based layout** — structure as logical rows
- `minimum width`: 1.4–2.2cm
- `node distance`: 0.5–0.7cm vertical
```

[1] https://github.com/jiahaoxiang2000/deeper-paper

Key Components of a Good Prompt

Example from `figure-tikz.md`[1]:

```
1. **Pure relative positioning only**
2. **NO anchor-based offsets** — avoid manual coordinates
3. **Simple paths**
4. **Row-based layout** — structure as logical rows
- `minimum width`: 1.4–2.2cm
- `node distance`: 0.5–0.7cm vertical
```

Prompt = Expert knowledge + Constraints + Anti-patterns

[1] https://github.com/jiahaoxiang2000/deeper-paper

Prompt Structure

```
You are a TikZ/LaTeX Expert.

## Design Principles
[Best practices and constraints]

## Required Libraries
[Dependencies to include]

## Code Structure Template
[Skeleton with style definitions]
```

```
## Robust Patterns
[Recommended approaches]

## Anti-Patterns to Avoid
[Common mistakes]

## Task
Create a TikZ figure for: $ARGUMENTS
```

This structure works for any figure library (TikZ, Mermaid, CeTZ, etc.)

Iterative Refinement

Step 1: Generate Code

Use your structured prompt with the LLM to generate initial figure code.

Step 1: Generate Code

Use your structured prompt with the LLM to generate initial figure code.

Example: “Create a TikZ diagram showing data flow between three components”

Step 1: Generate Code

Use your structured prompt with the LLM to generate initial figure code.

Example: “Create a TikZ diagram showing data flow between three components”

→ LLM outputs TikZ code following your prompt’s constraints

Step 2: Evaluate Output

Compile and render the generated code to see the actual figure.

Step 2: Evaluate Output

Compile and render the generated code to see the actual figure.

Check for:

- Does it match the conceptual design?
- Are sizes/spacing appropriate?
- Is the style consistent?
- Does it compile without errors?

Step 2: Evaluate Output

Compile and render the generated code to see the actual figure.

Check for:

- Does it match the conceptual design?
- Are sizes/spacing appropriate?
- Is the style consistent?
- Does it compile without errors?

If YES → You're done! ✓

If NO → Continue to next step...

Step 3: Find Issues

Identify specific problems with the output:

Step 3: Find Issues

Identify specific problems with the output:

Issue Type	Example
Layout	Nodes overlap, poor alignment
Sizing	Too wide for column, text truncated
Style	Used absolute positioning, wrong colors
Compilation	Missing library, syntax error

Step 3: Find Issues

Identify specific problems with the output:

Issue Type	Example
Layout	Nodes overlap, poor alignment
Sizing	Too wide for column, text truncated
Style	Used absolute positioning, wrong colors
Compilation	Missing library, syntax error

Each issue reveals a missing constraint in your prompt.

Step 4: Update Prompt

Refine the prompt based on issues found:

Add constraints:

```
- Use relative positioning only (no xshift/yshift)
- Maximum width: 8.5cm for single column
- Node spacing: 0.6cm vertical, 1.8cm horizontal
```

Step 4: Update Prompt

Refine the prompt based on issues found:

Add constraints:

```
- Use relative positioning only (no xshift/yshift)
- Maximum width: 8.5cm for single column
- Node spacing: 0.6cm vertical, 1.8cm horizontal
```

Add anti-patterns:

❌ Avoid: `above right=-0.1cm of node.south`

✅ Use: `below=of node` with separate row

Step 5: Iterate

Generate → Evaluate → Find Issues → Update Prompt

Each issue discovered → prompt improvement → better future outputs

Result: A refined prompt that generates good figures consistently.

Broader Context

This Pattern Everywhere

The same prompt engineering approach powers:

This Pattern Everywhere

The same prompt engineering approach powers:

Tool/System	What It Does
Claude Code SKILLS	Reusable expert prompts for coding tasks
MCP Servers	Structured interfaces for specific domains
Custom Commands	Domain-specific prompt templates

This Pattern Everywhere

The same prompt engineering approach powers:

Tool/System	What It Does
Claude Code SKILLS	Reusable expert prompts for coding tasks
MCP Servers	Structured interfaces for specific domains
Custom Commands	Domain-specific prompt templates

Common thread: Encode expert knowledge in structured prompts

Why It Works

Without Prompt Engineering

- Generic LLM output
- Inconsistent style
- May violate constraints
- Requires manual fixes
- Not reusable

With Prompt Engineering

- Expert-level output
- Consistent patterns
- Respects constraints
- Ready to use
- Reusable template

Summary

Key Takeaways

- Generate code, not pixels for figures
- Use structured prompts with constraints and examples
- Iterate to refine both output and prompt
- Apply to any figure library (TikZ, Mermaid, CeTZ, Fletcher)
- Same technique as SKILLS and MCP servers

Key Takeaways

- Generate code, not pixels for figures
- Use structured prompts with constraints and examples
- Iterate to refine both output and prompt
- Apply to any figure library (TikZ, Mermaid, CeTZ, Fletcher)
- Same technique as SKILLS and MCP servers

Core insight: Prompt engineering = Encoding expert knowledge

Key Takeaways

- Generate code, not pixels for figures
- Use structured prompts with constraints and examples
- Iterate to refine both output and prompt
- Apply to any figure library (TikZ, Mermaid, CeTZ, Fletcher)
- Same technique as SKILLS and MCP servers

Core insight: Prompt engineering = Encoding expert knowledge

Quality of output $\propto$ Quality of prompt

Resources

- Example TikZ prompt:
  `github.com/jiahaoxiang2000/deeper-paper`
  `.opencode/command/figure-tikz.md`
- Typst libraries:
  - CeTZ: `typst.app/universe/package/cetz`
  - Fletcher: `typst.app/universe/package/fletcher`
- LaTeX TikZ: `tikz.dev`
- Mermaid: `mermaid.js.org`