Mermaid Hybrid Stack Integration Manual

v2 — Corrected & Complete Edition

Environment: VSCode → WSL 2 (Ubuntu 24.04) → Claude Code Agent Orchestration
Author: Nathan Lim | April 5, 2026
Target: Enterprise-grade diagram-as-code across GitHub markdown, PDF, and HTML
Cost: $0 — all tools MIT/BSD/MPL-2.0 licensed

Thirteen phases, executed sequentially. Each phase ends with a verification test that must pass before advancing. This v2 edition incorporates all corrections, amendments, and troubleshooting discovered during the live implementation walkthrough.

Table of Contents

PART I

Implementation Manual

Every command runs inside your WSL 2 Ubuntu terminal (accessed via VSCode's integrated terminal). When a command must run on the Windows side, it is explicitly marked **[WINDOWS]**.

Phase 0: Docker Desktop WSL 2 Backend Verification

Time: 10 minutes

Why: Docker provides the cleanest Puppeteer/Chromium path for mmdc rendering. Without it, you manually install ~15 system libraries and debug Chromium sandboxing. Docker eliminates that entire class of problems.

0.1 — Check Docker Desktop Installation

[WINDOWS] Open Docker Desktop from your Start menu or taskbar.

If it opens: proceed to 0.2. If not installed: download from https://www.docker.com/products/docker-desktop/ and install with defaults.

0.2 — Enable WSL 2 Backend

[WINDOWS] In Docker Desktop Settings → General → ensure **"Use the WSL 2 based engine"** is checked. Click Apply & restart if changed.

0.3 — Enable WSL Integration for Your Ubuntu Distro

[WINDOWS] Docker Desktop Settings → Resources → WSL integration → toggle ON your Ubuntu distro. Click Apply & restart.

0.4 — Verify Docker Works Inside WSL

```
docker --version
docker run --rm hello-world
docker compose version
```

PASS: All three return valid output → Phase 0 COMPLETE.

Phase 1: Mermaid CLI Installation via Docker

Time: 15 minutes

Why: Installing mmdc natively on WSL 2 requires Puppeteer + Chromium + ~15 system libraries. Docker wraps all of that into a single pre-built image with zero dependency management.

1.1 — Pull the Mermaid CLI Docker Image

```
docker pull minlag/mermaid-cli
```

Downloads ~400MB (Chromium + Node.js + Mermaid bundled). One-time cost.

1.2 — Create the Render Wrapper Script

> **CRITICAL:** The --user flag is required. Without it, Docker runs as an internal user that cannot write to your mounted host directory, causing EACCES permission errors.

```
mkdir -p ~/bin

cat > ~/bin/mermaid-render << 'SCRIPT'
#!/bin/bash
INPUT_FILE="$1"
if [ -z "$INPUT_FILE" ]; then
  echo "Usage: mermaid-render <input.mmd> [output.svg|png|pdf]"
  exit 1
fi

INPUT_DIR=$(cd "$(dirname "$INPUT_FILE")" && pwd)
INPUT_NAME=$(basename "$INPUT_FILE")
OUTPUT_FILE="${2:-${INPUT_FILE%.mmd}.svg}"
OUTPUT_NAME=$(basename "$OUTPUT_FILE")
EXTRA_ARGS="${@:3}"

CONFIG_MOUNT=""
CONFIG_ARG=""
if [ -f "$HOME/.config/mermaid/config.json" ]; then
  CONFIG_MOUNT="-v $HOME/.config/mermaid:/config"
  CONFIG_ARG="-c /config/config.json"
fi

docker run --rm \
  --user "$(id -u):$(id -g)" \
  -v "$INPUT_DIR:/data" \
  $CONFIG_MOUNT \
  minlag/mermaid-cli \
  -i "/data/$INPUT_NAME" \
  -o "/data/$OUTPUT_NAME" \
  $CONFIG_ARG \
```

```
  $EXTRA_ARGS

if [ $? -eq 0 ]; then
  echo "Rendered: $INPUT_DIR/$OUTPUT_NAME"
else
  echo "Render failed for $INPUT_FILE"
  exit 1
fi
SCRIPT

chmod +x ~/bin/mermaid-render
```

1.3 — Ensure ~/bin Is in PATH

```
echo $PATH | grep -q "$HOME/bin" && echo "Already in PATH" || echo "Need to add"

# If not in PATH:
echo 'export PATH="$HOME/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc

# Verify:
which mermaid-render
```

1.4 — Create a Test Diagram

```
mkdir -p ~/projects/diagram-test

cat > ~/projects/diagram-test/test-flowchart.mmd << 'MMD'
flowchart TD
    A["User Prompt"] --> B{"Complexity<br/>Score"}
    B -->|Score ≤ 4| C["Tier 1<br/>Solo Agent"]
    B -->|Score 5-8| D["Tier 2<br/>Lean Pipeline"]
    B -->|Score ≥ 9| E["Tier 3<br/>Full Orchestra"]
    C --> F["Direct Execution"]
    D --> G["3 Subagents<br/>+ Review"]
    E --> H["OCR + Codex<br/>+ Discourse"]
MMD
```

NOTE: Use
 for line breaks, never \n. The enterprise theme config sets htmlLabels: true, which requires HTML line break syntax.

1.5 — Render and Verify

```
cd ~/projects/diagram-test
mermaid-render test-flowchart.mmd test-flowchart.svg

ls -la test-flowchart.svg
file test-flowchart.svg
```

Open the SVG in VSCode — verify clean flowchart with readable labels, no overlaps.

> **PASS:** SVG renders cleanly → Phase 1 COMPLETE.

Phase 2: Enterprise Theme Configuration

Time: 10 minutes

Why: A shared config file ensures every render produces identical enterprise-quality visual output regardless of who or what triggers the render.

2.1 — Create the Enterprise Theme Config

```
mkdir -p ~/.config/mermaid

cat > ~/.config/mermaid/config.json << 'CONFIG'
{
  "theme": "base",
  "themeVariables": {
    "primaryColor": "#F0F4FC",
    "primaryTextColor": "#1F2329",
    "primaryBorderColor": "#2C5F8A",
    "lineColor": "#555555",
    "secondaryColor": "#E8F5E9",
    "tertiaryColor": "#FFF3E0",
    "fontFamily": "Inter, Segoe UI, Roboto, sans-serif",
    "fontSize": "14px"
  },
  "flowchart": {
    "curve": "basis",
    "htmlLabels": true,
    "rankSpacing": 60,
    "nodeSpacing": 40,
    "padding": 15
  }
}
CONFIG
```

2.2 — Create a Dark Mode Variant

```
cat > ~/.config/mermaid/config-dark.json << 'CONFIG'
{
  "theme": "base",
  "themeVariables": {
    "primaryColor": "#1E293B",
    "primaryTextColor": "#E2E8F0",
    "primaryBorderColor": "#60A5FA",
    "lineColor": "#94A3B8",
    "background": "#0F172A"
  }
}
CONFIG
```

2.3 — Create Dark Render Wrapper

The dark wrapper must also include --user "$(id -u):$(id -g)" to avoid the same Docker permission issue from Phase 1.

```
cat > ~/bin/mermaid-render-dark << 'SCRIPT'
#!/bin/bash
INPUT_FILE="$1"
INPUT_DIR=$(cd "$(dirname "$INPUT_FILE")" && pwd)
INPUT_NAME=$(basename "$INPUT_FILE")
OUTPUT_FILE="${2:-${INPUT_FILE%.mmd}-dark.svg}"
OUTPUT_NAME=$(basename "$OUTPUT_FILE")

docker run --rm \
  --user "$(id -u):$(id -g)" \
  -v "$INPUT_DIR:/data" \
  -v "$HOME/.config/mermaid:/config" \
  minlag/mermaid-cli \
  -i "/data/$INPUT_NAME" \
  -o "/data/$OUTPUT_NAME" \
  -c /config/config-dark.json \
  -b transparent

echo "Dark render: $INPUT_DIR/$OUTPUT_NAME"
SCRIPT

chmod +x ~/bin/mermaid-render-dark
```

2.4 — Test Both Themes

```
cd ~/projects/diagram-test
mermaid-render test-flowchart.mmd test-flowchart-light.svg
mermaid-render-dark test-flowchart.mmd
```

PASS: Both SVGs render with distinct themes → Phase 2 COMPLETE.

Phase 3: Battle Test — Threshold Escalation Flowchart

Time: 20 minutes

Why: Rebuild the threshold escalation flowchart from the Definitive Runbook in Mermaid. This diagram currently has collision and overlap issues in its matplotlib rendering. If this renders cleanly, the approach is validated for all 10 diagrams.

3.1 — Create the Diagrams Directory

```
mkdir -p ~/projects/<project-name>/diagrams
```

3.2 — Build the Threshold Escalation Flowchart

> **NOTE:** All line breaks use
. All edge labels in pipes are unquoted. These two rules prevent the most common Mermaid syntax issues.

```
cat > diagrams/threshold-escalation.mmd << 'MMD'
flowchart TD
    START(["Incoming Prompt"]) --> PARSE["Parse Prompt<br/>Extract Keywords"]
    PARSE --> SCORE{"Complexity<br/>Scoring Engine"}
    SCORE --> S1["File Count<br/>(0-3 pts)"]
    SCORE --> S2["Language Count<br/>(0-3 pts)"]
    SCORE --> S3["Keyword Signals<br/>(0-4 pts)"]
    SCORE --> S4["Cross-Cutting<br/>Concerns (0-2 pts)"]
    S1 --> SUM["Sum Score<br/>(0-12 range)"]
    S2 --> SUM
    S3 --> SUM
    S4 --> SUM
    SUM --> ROUTE{"Route by<br/>Threshold"}
    ROUTE -->|Score ≤ 4| T1["TIER 1<br/>Solo Agent<br/>(Opus 4.6)"]
    ROUTE -->|Score 5-8| T2["TIER 2<br/>Lean Pipeline<br/>(3 Subagents)"]
    ROUTE -->|Score ≥ 9| T3["TIER 3<br/>Full Orchestra<br/>(OCR + Codex)"]
MMD
```

3.3 — Render and Compare

```
mermaid-render threshold-escalation.mmd threshold-escalation.svg
mermaid-render threshold-escalation.mmd threshold-escalation.png -s 3
```

Open the SVG side-by-side with the Definitive Runbook PDF. Verify: no text overlap, no arrow collisions, consistent spacing, distinct tier color coding.

> **PASS:** Visual comparison passes → Phase 3 COMPLETE.

Phase 4: Syntax Validation with @probelabs/maid

Time: 5 minutes

Why: Catches Mermaid syntax errors in 1-2ms without launching Chromium. Error messages are structured for AI self-repair.

4.1 — Install maid

```
sudo npm install -g @probelabs/maid
```

> **NOTE:** Use sudo for all npm install -g commands on WSL 2 where the global node_modules directory is owned by root.

4.2 — Key Syntax Rule: Edge Labels

> **CRITICAL:** maid enforces stricter grammar than the Mermaid renderer. Quoted strings inside pipe-delimited edge labels are rejected even though the renderer tolerates them.

Node labels → ALWAYS quote: `A["Label"]`
Edge labels in pipes → NEVER quote: `-->|Label|`

4.3 — Create the Validation Wrapper

```
cat > ~/bin/mermaid-validate << 'SCRIPT'
#!/bin/bash
INPUT_FILE="$1"
OUTPUT_FILE="${2:-${INPUT_FILE%.mmd}.svg}"

echo "Validating: $INPUT_FILE"
maid "$INPUT_FILE"
if [ $? -ne 0 ]; then
  echo "Validation failed. Fix syntax before rendering."
  exit 1
fi

echo "Rendering: $OUTPUT_FILE"
mermaid-render "$INPUT_FILE" "$OUTPUT_FILE"
SCRIPT

chmod +x ~/bin/mermaid-validate
```

PASS: maid validates clean, broken files produce clear errors → Phase 4 COMPLETE.

Phase 5: Claude Code Skill — Mermaid Diagram Generator

Time: 15 minutes

Why: Describes diagrams in natural language to Claude Code; the skill instructs it to generate syntactically valid .mmd, .vl.json, or .d2 files.

5.1 — Create the Skill as a Global Directory

> **CRITICAL:** Claude Code expects skills as **directories** containing a SKILL.md file, NOT standalone .md files. Standalone files will not be detected by /skills.

```
mkdir -p ~/.claude/skills/mermaid-diagrams
```

5.2 — Create SKILL.md

The skill file teaches Claude three routing rules:

Flowcharts, sequences, Gantt → **Mermaid** (.mmd files)
Heatmaps, grouped bars, data charts → **Vega-Lite** (.vl.json files)
Multi-layer architectures → **D2** (.d2 files)

```
cat > ~/.claude/skills/mermaid-diagrams/SKILL.md << 'SKILL'
---
name: mermaid-diagrams
description: Generate production-quality diagrams.
allowed-tools: [Read, Write, Edit, Bash]
---

## Syntax Guardrails — CRITICAL
- Line breaks: use <br/> NEVER \n
- Node labels: ALWAYS quote ["Label"]
- Edge labels in pipes: NEVER quote -->|Label|
- Node IDs: alphanumeric only

## Post-Generation
1. Run maid <file> to validate
2. If fails, read error, fix, re-validate
3. Run mermaid-render <file> to produce SVG
SKILL
```

Full skill content includes enterprise color mappings, diagram type selection matrix, and routing rules for Vega-Lite and D2. See the complete SKILL.md in the repository.

5.3 — Verify Global Scope

```
cd ~/projects/<any-project>
claude
/skills
```

Should show `mermaid-diagrams` with ~36 description tokens regardless of which project directory you are in.

> **PASS:** Skill detected globally, generates valid .mmd files → Phase 5 COMPLETE.

Phase 6: PostToolUse Hook — Auto-Render on Write

Time: 10 minutes

Why: Detects when any .mmd file is written and automatically validates + renders it. Closes the automation loop: you prompt → Claude writes → hook renders → you see SVG.

6.1 — Merge Into Existing Settings (Do NOT Overwrite)

CRITICAL: Your ~/.claude/settings.json already contains PreToolUse security hooks, PostToolUse Prettier/git hooks, and other orchestration config. MERGE the new hook, never overwrite.

6.2 — Add Tool Permissions

Add these to the existing `permissions.allow` array:

```
"Bash(maid*)",
"Bash(mermaid-render*)",
"Bash(mermaid-validate*)",
"Bash(mermaid-render-dark*)",
"Bash(d2*)",
"Bash(d2-render*)",
"Bash(vl2svg*)",
"Bash(vl2png*)",
"Bash(vegalite-render*)",
"Bash(render-all-diagrams*)",
"Bash(docker run*)"
```

6.3 — Add the Mermaid PostToolUse Hook

Add as a new entry in the existing PostToolUse array:

CRITICAL: Three critical requirements: (1) Wrap in explicit `bash -c` — hooks may run in sh which lacks [[ ]]. (2) Export PATH — ~/bin is not in the hook's minimal shell environment. (3) Use POSIX `case/esac` instead of [[== *.mmd]].

```
{
  "matcher": "Write",
  "hooks": [{
    "type": "command",
    "command": "bash -c 'INPUT=$(cat); FILE=$(echo \"$INPUT\" | jq -r
      \".tool_input.file_path // empty\"); case \"$FILE\" in
      *.mmd) export PATH=\"$HOME/bin:$PATH\"; maid \"$FILE\"
      2>/dev/null; if [ $? -eq 0 ]; then mermaid-render \"$FILE\"
      \"${FILE%.mmd}.svg\" 2>/dev/null; mermaid-render \"$FILE\"
```

```
      \"${FILE%.mmd}.png\" -s 3 2>/dev/null; fi ;; esac'",
    "timeout": 60
  }]
}
```

The 60-second timeout accommodates Docker cold-start. The hook auto-generates both SVG and high-res PNG (3x scale) on every .mmd write.

6.4 — Validate and Test

```
jq . ~/.claude/settings.json > /dev/null && echo "Valid JSON"

# In Claude Code, write any .mmd file.
# Terminal should show "3 PostToolUse hooks ran"
# and the .svg file should appear automatically.
```

PASS: Hook fires and SVG auto-generates → Phase 6 COMPLETE.

Phase 7: GitHub Native Rendering Setup

Time: 10 minutes

Why: GitHub renders Mermaid natively in markdown. Diagrams are version-controlled as plain text. Changes show in PR diffs as readable text.

7.1 — Create Docs with Embedded Mermaid

````
mkdir -p docs

# In any .md file, use:
```mermaid
flowchart TD
    A["Start"] --> B["End"]
```
````

7.2 — GitHub Rendering Limitations

No ELK layout (dagre only), no Font Awesome icons, no hyperlinks, no custom themes, ~50KB max per code block, block-beta may not be supported.

For diagrams needing custom styling, commit pre-rendered SVGs and reference with `![Alt text](./assets/diagram.svg)`.

7.3 — Global Gitignore for All Repos

```
cat > ~/.gitignore_global << 'GLOBAL'
diagrams/*.svg
diagrams/*.png
diagrams/*.pdf
GLOBAL

git config --global core.excludesFile ~/.gitignore_global
```

This applies to every repo automatically — no per-project .gitignore needed.

7.4 — Bootstrap Script Update for New Projects

Append to ~/bin/new-project so every new repo gets diagrams/ and docs/ directories:

```
mkdir -p diagrams docs
# + gitignore rules for !diagrams/*.mmd, !diagrams/*.vl.json, !diagrams/*.d2
```

> **PASS:** Mermaid renders on GitHub, global gitignore active → Phase 7 COMPLETE.

Phase 8: PDF Integration Pipeline

Time: 20 minutes

Why: Replace matplotlib chart images in existing ReportLab pipelines with Mermaid-rendered PNGs at 3x scale (300+ DPI) for significantly sharper output.

8.1 — Install Pandoc

```
sudo apt-get install -y pandoc
```

> **NOTE:** Pandoc's PDF engine requires LaTeX (~2GB). Not worth installing — the mermaid-pdf script includes an HTML fallback. Primary PDF pipeline uses ReportLab.

8.2 — Batch Render at Print Resolution

```
cd ~/projects/<project-name>/diagrams

for f in *.mmd; do
  mermaid-render "$f" "${f%.mmd}.svg"
  mermaid-render "$f" "${f%.mmd}.png" -s 3
done
```

8.3 — ReportLab Integration Point

```
# BEFORE (matplotlib):
# img = Image("charts/threshold_escalation.png", width=6.5*inch)

# AFTER (Mermaid at 3x scale):
img = Image("diagrams/threshold-escalation.png", width=6.5*inch)
```

8.4 — Markdown-to-PDF Script

```
cat > ~/bin/mermaid-pdf << 'SCRIPT'
#!/bin/bash
INPUT="$1"
OUTPUT="${2:-${1%.md}.pdf}"
pandoc "$INPUT" -o "$OUTPUT" -V geometry:margin=1in --toc 2>/dev/null
if [ $? -ne 0 ]; then
  pandoc "$INPUT" -o "${OUTPUT%.pdf}.html" --standalone --toc
fi
SCRIPT
chmod +x ~/bin/mermaid-pdf
```

PASS: Print-quality PNGs generated (100KB+) → Phase 8 COMPLETE.

Phase 9: HTML Portfolio Embedding

Time: 10 minutes

Why: Pre-rendered SVGs in Astro's public/ directory provide instant loading, SEO-friendly rendering, and infinite resolution scaling.

9.1 — Copy SVGs to Public Assets

```
cp diagrams/*.svg public/assets/
cp diagrams/*.png public/assets/
```

9.2 — Reference in Astro Components

```
<img
  src="/assets/threshold-escalation.svg"
  alt="Threshold Escalation Flowchart"
  width="800"
  loading="lazy"
/>
```

9.3 — Verify Build

```
npm run build
```

> **PASS:** Astro build succeeds, SVGs in dist/ → Phase 9 COMPLETE.

Phase 10: Vega-Lite — Heatmaps and Data Charts

Time: 15 minutes

Why: Mermaid cannot produce heatmaps or grouped bar charts. Vega-Lite is the only free, declarative, text-based tool for these at enterprise quality.

10.1 — Install Vega-Lite CLI Tools

```
sudo npm install -g vega-lite vega-cli vega
sudo npm install -g canvas
```

> **NOTE:** The canvas package is required for PNG rendering. SVG works without it.

10.2 — Create and Render

Decision matrix heatmap → `diagrams/decision-matrix.vl.json`
Token economics grouped bars → `diagrams/token-economics.vl.json`

```
vl2svg decision-matrix.vl.json decision-matrix.svg
vl2png decision-matrix.vl.json decision-matrix.png -s 3
```

10.3 — Create Wrapper

```
cat > ~/bin/vegalite-render << 'SCRIPT'
#!/bin/bash
INPUT="$1"
BASENAME="${INPUT%.vl.json}"
vl2svg "$INPUT" "${BASENAME}.svg"
vl2png "$INPUT" "${BASENAME}.png" -s 3
SCRIPT
chmod +x ~/bin/vegalite-render
```

> **PASS:** Heatmap and grouped bar chart render at print quality → Phase 10 COMPLETE.

Phase 11: D2 — Complex Architecture Diagrams

Time: 15 minutes

Why: D2's nested containers produce the cleanest multi-layer architecture diagrams. Your 4-layer system diagram benefits from strict spatial positioning.

11.1 — Install D2

```
curl -fsSL https://d2lang.com/install.sh | sh -s --
```

11.2 — Install Playwright Dependency

> **NOTE:** D2 SVG works immediately. PNG requires Playwright + Chromium which needs libasound2t64 on WSL 2 Ubuntu.

```
sudo apt-get install -y libasound2t64
```

11.3 — Render and Create Wrapper

```
d2 --layout=elk system-architecture.d2 system-architecture.svg
d2 --layout=elk system-architecture.d2 system-architecture.png --scale 3

cat > ~/bin/d2-render << 'SCRIPT'
#!/bin/bash
INPUT="$1"
BASENAME="${INPUT%.d2}"
d2 --layout=elk "$INPUT" "${BASENAME}.svg"
d2 --layout=elk "$INPUT" "${BASENAME}.png" --scale 3
SCRIPT
chmod +x ~/bin/d2-render
```

> **CRITICAL:** D2's TALA layout engine is paid (~$240/yr). Always use --layout=elk (free).

> **PASS:** 4-layer architecture renders with clean containers → Phase 11 COMPLETE.

Phase 12: MCP Server Evaluation (Rejected)

Time: 5 minutes

Outcome: The mermaid-mcp package referenced in pre-implementation research does not exist as a published npm executable. `npx -y mermaid-mcp` returns "could not determine executable to run."

The skill + hook pipeline (Phases 5-6) provides equivalent diagram generation, validation, and rendering capabilities without MCP overhead. No MCP server was added to .mcp.json.

> If a viable Mermaid MCP server becomes available as a published npm package in the future, re-evaluate. The integration point is the mcpServers object in .mcp.json.

Phase 13: Full Validation and Cleanup

Time: 30 minutes

13.1 — Batch Render Script

```
cat > ~/bin/render-all-diagrams << 'SCRIPT'
#!/bin/bash
DIR="${1:-$(pwd)/diagrams}"
echo "Rendering all diagrams in: $DIR"

for f in "$DIR"/*.mmd; do
  [ -f "$f" ] || continue
  echo "  [Mermaid] $f"
  maid "$f" 2>/dev/null && mermaid-render "$f" "${f%.mmd}.svg" \
    && mermaid-render "$f" "${f%.mmd}.png" -s 3
done

for f in "$DIR"/*.vl.json; do
  [ -f "$f" ] || continue
  echo "  [Vega-Lite] $f"
  vegalite-render "$f"
done

for f in "$DIR"/*.d2; do
  [ -f "$f" ] || continue
  echo "  [D2] $f"
  d2-render "$f"
done
SCRIPT
chmod +x ~/bin/render-all-diagrams
```

13.2 — Final Validation Counts

```
echo "Mermaid files: $(ls diagrams/*.mmd | wc -l)"
echo "Vega-Lite files: $(ls diagrams/*.vl.json | wc -l)"
echo "D2 files: $(ls diagrams/*.d2 | wc -l)"
echo "SVG outputs: $(ls diagrams/*.svg | wc -l)"
echo "PNG outputs: $(ls diagrams/*.png | wc -l)"
```

13.3 — Post-Validation Cleanup

Remove test artifacts from the repo. Infrastructure (scripts in ~/bin, global skill, settings.json hooks, themes) lives outside the repo and persists.

```
git reset HEAD .
rm -rf diagrams/
echo ".claude/settings.local.json" >> .gitignore
git add .gitignore
git commit -m "chore: add diagram-as-code gitignore rules"
```

PASS: All diagram types render, cleanup complete → Phase 13 COMPLETE.

PART II

Troubleshooting & Debugging Log

Every error documented below was encountered during the live implementation walkthrough. No hypotheticals — each entry includes the exact symptom, root cause, and fix.

Error 1: Docker EACCES Permission Denied on Write

Phase: 1 | **Step:** 1.5

Symptom	[Error: EACCES: permission denied, open '/data/output.svg']
Root Cause	Docker container runs as internal user without write permission to volume-mounted host directory.
Fix	Add --user "$(id -u):$(id -g)" to docker run command. Maps container process to host WSL user.

Error 2: Literal \n in Rendered Diagram Labels

Phase: 3 | **Step:** 3.4

Symptom	Node text displays "Complexity\nScoring Engine" as single line with visible \n characters.
Root Cause	Enterprise theme config sets htmlLabels: true. Mermaid expects HTML markup ( ) for line breaks, not e
Fix	Replace all \n with in .mmd files: sed -i 's/\\n/<br\/>/g' filename.mmd

Error 3: npm EACCES on Global Package Install

Phase: 4, 10 | **Step:** 4.1, 10.1

Symptom	npm error code EACCES — permission denied, mkdir '/usr/local/lib/node_modules/...'
Root Cause	Global node_modules directory owned by root. Standard user cannot write without elevation.
Fix	Prefix with sudo: sudo npm install -g <package>

Error 4: maid Rejects Quoted Strings in Pipe Edge Labels

Phase: 4 | **Step:** 4.2

Symptom	error[FL-EDGE-LABEL-QUOTE-IN-PIPES]: Quotes not supported inside pipe-delimited edge labels.
Root Cause	maid enforces stricter grammar than the renderer. Pipe edge labels do not support quoted strings.
Fix	Remove quotes: BEFORE: -->|"Score ≤ 4"| AFTER: -->|Score ≤ 4|

Error 5: Claude Code Skill Not Detected

Phase: 5 | **Step:** 5.3

Symptom	/skills does not list mermaid-diagrams despite the file existing.
Root Cause	Claude Code expects skills as directories containing SKILL.md, not standalone .md files.
Fix	mkdir -p ~/.claude/skills/mermaid-diagrams; move file to SKILL.md inside that directory.

Error 6: PostToolUse Hook Fires but SVG Not Generated (PATH)

Phase: 6 | **Step:** 6.4

Symptom	"3 PostToolUse hooks ran" reported but no .svg file created. No error visible.
Root Cause	Hooks run in minimal shell where ~/bin is not in PATH. maid and mermaid-render silently fail.
Fix	Add export PATH="$HOME/bin:$PATH" inside the hook command.

Error 7: Hook Still Fails After PATH Fix (Shell Compatibility)

Phase: 6 | **Step:** 6.4

Symptom	Same as Error 6 — hook fires, no SVG, no error.
Root Cause	[[ ]] is bash-specific. Hooks may execute in sh/restricted shell that does not support [[ ]].
Fix	Wrap in bash -c '...'; replace [[ ]] with POSIX case/esac; increase timeout to 60s.

Error 8: Vega-Lite PNG Render Fails — Missing Canvas

Phase: 10 | **Step:** 10.3

Symptom	Error: CanvasRenderer is missing a valid canvas or context
Root Cause	vl2png requires native canvas npm package for server-side PNG rasterization.
Fix	sudo npm install -g canvas. SVG rendering works without it.

Error 9: D2 PNG Render Fails — Playwright Missing Dependencies

Phase: 11 | **Step:** 11.3

Symptom	Host system is missing dependencies to run browsers.
Root Cause	D2 uses Playwright + Chromium for PNG. WSL 2 Ubuntu lacks libasound2t64.

Fix	sudo apt-get install -y libasound2t64. D2 SVG works without Playwright.

Error 10: MCP Server mermaid-mcp Fails to Connect

Phase: 12 | **Step:** 12.3

Symptom	npx -y mermaid-mcp returns: could not determine executable to run.
Root Cause	Package is a GitHub repo, not a published npm package with bin entry.
Fix	Removed from .mcp.json. Skill + hook pipeline provides equivalent capabilities.

Error 11: Batch Render Stalls on Sequential Docker Runs

Phase: 8 | **Step:** 8.2

Symptom	Script hangs mid-execution during third diagram's PNG render.
Root Cause	Rapid sequential Docker container spin-ups exhaust resources. Each spawns full Chromium.
Fix	Wait 60-90s before Ctrl+C. Re-run stalled file individually. Increase Docker resource allocation.

PART III

Reference

Installed Tools Summary

Tool	Install	Purpose
Docker (mermaid-cli)	docker pull minlag/mermaid-cli	Mermaid → SVG/PNG
@probelabs/maid	sudo npm install -g @probelabs/maid	Syntax validation (1-2ms)
Vega-Lite + vega-cli	sudo npm install -g vega-lite vega-cli vega	Heatmaps, bar charts
canvas	sudo npm install -g canvas	Vega-Lite PNG rendering
D2	curl -fsSL https://d2lang.com/install.sh | sh	Architecture diagrams
libasound2t64	sudo apt-get install -y libasound2t64	D2 PNG (Playwright dep)
Pandoc	sudo apt-get install -y pandoc	Markdown → PDF/HTML
jq	sudo apt-get install -y jq	JSON parsing in hooks

Custom Scripts Summary

Script	Location	Purpose
mermaid-render	~/bin/	Docker wrapper for mmdc
mermaid-render-dark	~/bin/	Dark theme variant
mermaid-validate	~/bin/	Validate then render
vegalite-render	~/bin/	Vega-Lite JSON → SVG/PNG
d2-render	~/bin/	D2 → SVG/PNG with ELK layout
render-all-diagrams	~/bin/	Batch render entire diagrams/
mermaid-pdf	~/bin/	Markdown+Mermaid → PDF/HTML

File Naming Conventions

Extension	Tool	Tracked in Git?
.mmd	Mermaid source	Yes — diffable text
.vl.json	Vega-Lite spec	Yes — diffable JSON

.d2	D2 source	Yes — diffable text
.svg	Rendered output	No — generated artifact
.png	Rendered output	No — generated artifact

Diagram-to-Tool Assignment Matrix

#	Diagram	Tool	Source File
1	System Architecture	D2	system-architecture.d2
2	Threshold Escalation	Mermaid	threshold-escalation.mmd
3	Token Economics	Vega-Lite	token-economics.vl.json
4	Before/After Workflow	Mermaid	before-after-workflow.mmd
5	Config File Hierarchy	Mermaid	config-hierarchy.mmd
6	Phase Dependency Graph	Mermaid	phase-dependencies.mmd
7	Decision Matrix Heatmap	Vega-Lite	decision-matrix.vl.json
8	Gantt Timeline	Mermaid	gantt-timeline.mmd
9	Error Resolution	Mermaid	error-resolution.mmd
10	Component Inventory	Markdown	inventory-table.md