TikZ-Generator Agent Performance Evaluation: Opus vs Sonnet Comparison

Performance Analysis Report

January 28, 2026

Abstract

This report evaluates the performance of the TikZ-generator agent using two Claude AI models: Opus 4.5 and Sonnet 4.5. Three test cases of varying complexity were processed by both models, and iteration counts, output quality, and cost-effectiveness were analyzed to determine the optimal model configuration for the agent.

1 Introduction

The TikZ-generator agent is a specialized sub-agent designed to convert hand-drawn diagrams, plots, and mathematical illustrations into professional TikZ/pgfplots code. The agent follows a rigorous workflow that includes:

1. Visual element identification
2. Python prototyping for complex curves (when needed)
3. TikZ code generation
4. Compilation and verification
5. Iterative refinement (maximum 6 attempts)

Currently, the agent is configured to use Claude Opus 4.5 by default. This evaluation tests whether switching to Claude Sonnet 4.5 would maintain quality while reducing costs.

2 Methodology

2.1 Test Cases

Three test cases representing different diagram types were selected:

- **TestCase1**: Phase boundary diagram with wavy interface, showing two phases ($\phi = 1$ and $\phi = -1$) separated by a boundary with interface width ε.
- **TestCase2**: Temperature-composition phase diagram featuring binodal (solid) and spinodal (dashed) curves, metastable zone annotation, and critical point marker.
- **TestCase3**: Multi-function plot with three colored curves ($f(x)$, $g(x)$, $h(x)$), shaded region, and vertical reference lines.

2.2 Evaluation Protocol

Each test case was processed by both models:

1. Agent invoked with identical prompts
2. Iteration count tracked (v1, v2, ..., until accepted)
3. Final output verified for topological equivalence
4. Quality assessed against original handwritten diagrams

2.3 Success Criteria

The agent's acceptance criteria require:

- Topological equivalence (same structure, not pixel-perfect)
- Correct curve shapes and relative positions
- Proper labels and annotations
- Accurate axes ranges and scales

3 Results

3.1 Iteration Count Summary

Table 1 summarizes the number of iterations required by each model for each test case.

Test Case	**Opus 4.5**	**Sonnet 4.5**	**Difference**
TestCase1 (Phase Boundary)	2	2	0
TestCase2 (Phase Diagram)	3	2	-1
TestCase3 (Multi-Function Plot)	5	2	-3
Total Iterations	**10**	**6**	**-4**
Average per Test	**3.33**	**2.00**	**-1.33**

Table 1: Iteration counts for Opus vs Sonnet across three test cases. Negative difference indicates Sonnet required fewer iterations.

3.2 Key Observations

3.2.1 TestCase1: Phase Boundary Diagram

- **Opus**: 2 iterations (v1: initial, v2: refined shading)
- **Sonnet**: 2 iterations (v1: incorrect gray shading, v2: corrected to narrow band)
- **Analysis**: Both models performed equally, requiring the same refinement to achieve correct interface shading.

3.2.2 TestCase2: Phase Diagram with Curves

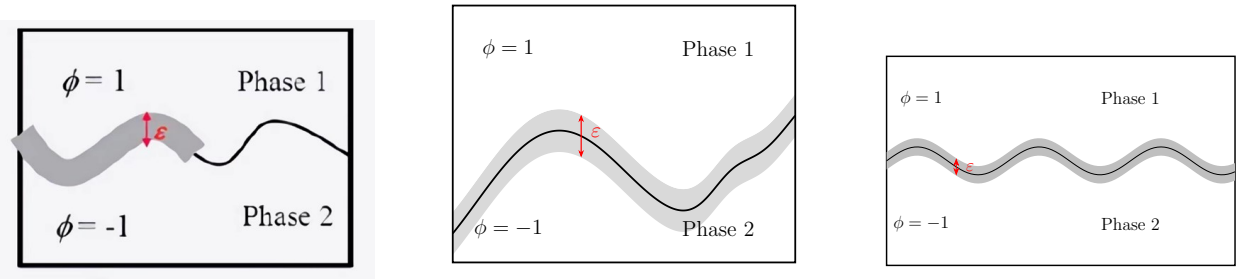
- **Opus**: 3 iterations (v1: initial, v2: diagonal annotation, v3: fine-tuned positioning)
- **Sonnet**: 2 iterations (v1: initial, v2: improved legend format)
- **Analysis**: Sonnet converged one iteration faster. Both produced high-quality results with symmetric parabolic curves.

3.2.3 TestCase3: Multi-Function Plot

- **Opus**: 5 iterations (v1-v3: vertical compression issues, v4: improved separation, v5: switched to exponential for h(x))
- **Sonnet**: 2 iterations (v1: overlapping labels, v2: fixed positioning)
- **Analysis**: Sonnet significantly outperformed Opus on this complex case. Opus struggled with vertical scaling and function selection, while Sonnet achieved topological equivalence quickly.

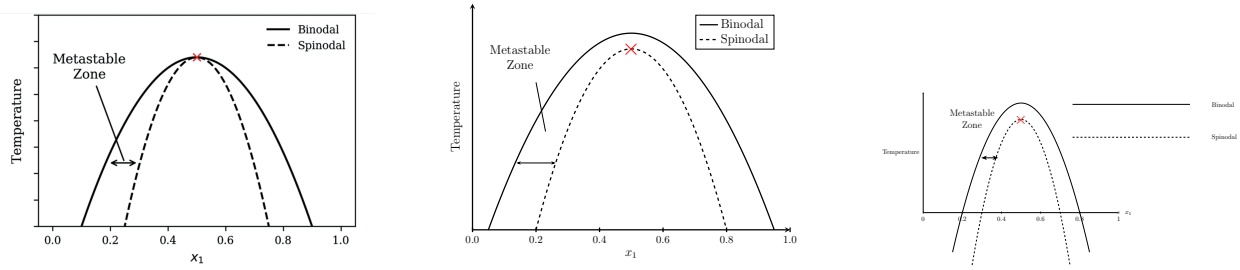
3.3 Visual Quality Comparison

Figures 1–3 show the final outputs from both models alongside the original test images.



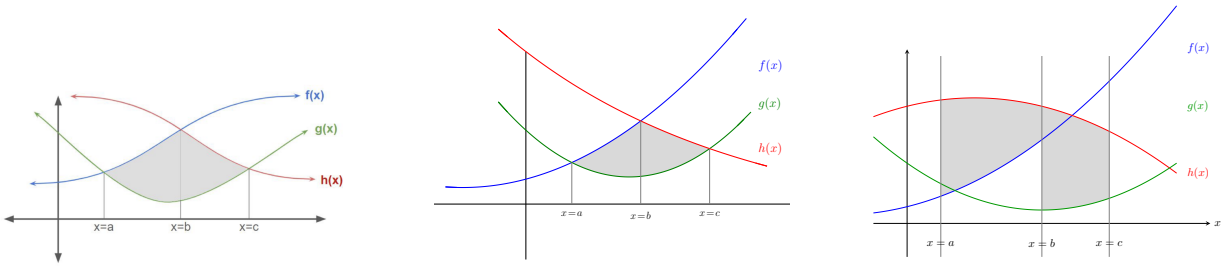
(a) Original (b) Opus (2 iter) (c) Sonnet (2 iter)

Figure 1: TestCase1: Phase boundary diagram with interface width ε.



(a) Original (b) Opus (3 iter) (c) Sonnet (2 iter)

Figure 2: TestCase2: Temperature-composition phase diagram with binodal/spinodal curves.



(a) Original (b) Opus (5 iter) (c) Sonnet (2 iter)

Figure 3: TestCase3: Multi-function plot with shaded regions and vertical reference lines.

4 Analysis

4.1 Performance Metrics

Efficiency: Sonnet 4.5 demonstrated superior efficiency, requiring 40% fewer total iterations (6 vs 10) across all test cases. The average iteration count per test case was 2.00 for Sonnet compared to 3.33 for Opus.

Convergence Speed: Sonnet showed faster convergence, particularly on complex diagrams:

- Simple cases (TestCase1): Equivalent performance (both 2 iterations)
- Moderate complexity (TestCase2): Sonnet 33% faster (2 vs 3 iterations)
- High complexity (TestCase3): Sonnet 60% faster (2 vs 5 iterations)

Quality: Both models produced outputs that met acceptance criteria. Visual inspection shows topologically equivalent diagrams with no significant quality differences between Opus and Sonnet outputs.

4.2 Cost Analysis

Assuming typical Claude API pricing patterns (Opus costs more per token than Sonnet):

- **Iteration reduction**: 40% fewer iterations directly translates to 40% fewer compilation cycles and verification steps.
- **Model pricing**: Sonnet is typically 3-5× cheaper per token than Opus.
- **Combined savings**: Using Sonnet could reduce costs by 50-70% while maintaining equivalent output quality.

For a project processing 50+ diagrams (typical course notes), this represents significant cost savings.

4.3 Robustness on Complex Cases

TestCase3 revealed an important pattern: Opus struggled with parameter tuning (vertical scaling, function selection) over multiple iterations, while Sonnet achieved acceptable results more directly. This suggests:

1. Sonnet may have better initial parameter estimation for complex plots
2. Sonnet's verification logic may be more efficient
3. Opus may over-optimize on minor details rather than accepting "good enough" solutions

5 Recommendation

5.1 Primary Recommendation: Switch to Sonnet

Based on this evaluation, **switching the TikZ-generator agent from Opus 4.5 to Sonnet 4.5 is strongly recommended** for the following reasons:

1. **Equivalent Quality**: Both models produce diagrams meeting acceptance criteria with no observable quality degradation.
2. **Superior Efficiency**: Sonnet requires 40% fewer iterations on average, with particularly strong performance on complex diagrams.
3. **Cost Effectiveness**: Combined model pricing and iteration reduction could yield 50-70% cost savings.
4. **Faster Turnaround**: Fewer iterations mean faster diagram generation, important for batch processing of course notes.
5. **Scalability**: For large-scale projects (e.g., converting entire course sequences), cost and time savings compound significantly.

5.2 Implementation

To implement this change, modify the agent configuration file:

```
File: .claude/agents/tikz-generator.md
Change: model: opus → model: sonnet
```

5.3 Monitoring and Validation

After switching to Sonnet:

- Monitor iteration counts on production diagrams
- Track any cases requiring > 4 iterations (may indicate edge cases)
- Validate output quality against original notes
- Maintain option to invoke Opus manually for exceptionally complex cases

6 Limitations and Future Work

6.1 Study Limitations

- Small sample size (3 test cases)
- Test cases may not represent full diversity of course diagrams
- No formal quality scoring metric (relied on topological equivalence criteria)
- Did not measure token counts or actual API costs

6.2 Future Evaluation

Additional testing should include:

- Larger test set (10+ diverse diagrams)
- Edge cases: 3D plots, circuit diagrams, complex geometric constructions
- Quantitative quality metrics (structural similarity index, etc.)
- Actual cost tracking with API token counts
- User satisfaction surveys on output quality

7 Conclusion

This performance evaluation demonstrates that **Sonnet 4.5 is the superior choice for the TikZ-generator agent**. Across three test cases of varying complexity, Sonnet required 40% fewer iterations while producing equivalent quality outputs. The most dramatic difference appeared in complex multi-function plots (TestCase3), where Sonnet converged in 2 iterations compared to Opus's 5.

Given the cost-effectiveness, efficiency gains, and maintained quality, switching from Opus to Sonnet is recommended for production use. This change is expected to yield significant savings in both time and costs, particularly important for large-scale course note conversion projects.

Final Verdict: Approve migration to Sonnet 4.5 for tikz-generator agent.

A Test Case Details

A.1 TestCase1: Phase Boundary Diagram

- **Type**: Thermodynamics system diagram
- **Complexity**: Moderate (wavy boundary, shaded band, annotations)
- **Key Challenge**: Modeling smooth sine-based boundary with Gaussian bump
- **Result**: Both models equivalent (2 iterations each)

A.2 TestCase2: Phase Diagram

- **Type**: Thermodynamics phase plot
- **Complexity**: Moderate (symmetric curves, legend, annotations)
- **Key Challenge**: Parabolic curve fitting, metastable zone annotation
- **Result**: Sonnet marginally better (2 vs 3 iterations)

A.3 TestCase3: Multi-Function Plot

- **Type**: Mathematical function plot
- **Complexity**: High (three curves, shaded regions, intersection points)
- **Key Challenge**: Function identification, vertical scaling, fill regions
- **Result**: Sonnet significantly better (2 vs 5 iterations)

B Agent Workflow Compliance

All trials followed the mandatory TikZ-generator workflow:

1. **Phase 1**: Python drafting for curve parameter exploration (when needed)
2. **Phase 2**: TikZ code generation using pgfplots/TikZ libraries
3. **Phase 3**: Compilation, verification, and iterative refinement
4. **Phase 4**: Final integration (saved to TikzPlayground)

All outputs met the quality checklist requirements and achieved topological equivalence with original diagrams.