Image Generation / Midjourney

Deconstruct complex prompts with :: weight syntax and explain the priority hierarchy — reveals why the model makes certain choices.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Prompt Debugging, Weight Optimization
Updated: May 2026
Why This Prompt Exists
Midjourney’s double-colon (::) syntax allows weighting different parts of a prompt. Most users don’t use it because they don’t understand it — but it’s the most powerful tool for controlling output.

You get:

- prompts where one element dominates unexpectedly (no weighting to balance)
- inability to emphasize or de-emphasize specific concepts
- no understanding of how weights translate to visual priority
- frustration when “red car” produces a red car, but “red car” with other elements doesn’t
- wasted generations trying to balance prompts by word order alone

But weighting has predictable effects:

- ::1 = baseline weight (default)
- ::2 = twice as important
- ::0.5 = half as important
- ::0 = effectively remove (but keep for context)
- :: (empty) = break between concepts, no weight change

Without analysis, weighted prompts are guesswork.

This prompt deconstructs and explains weighted prompt hierarchies.

The Prompt

```
Assume the role of a Midjourney prompt engineer who deconstructs
weighted prompts.

Your task is to analyze a weighted prompt and explain the priority
hierarchy.

Generate:

1. PROMPT DECONSTRUCTION
   - Original prompt: [paste the weighted prompt]
   - Weight syntax identified: [list of :: separators and weights]

2. WEIGHT HIERARCHY TABLE

| Prompt Segment | Weight | Relative Importance | Expected Visual
Priority |
|----------------|--------|---------------------|--------------------
-----|
| [first segment] | [weight] | [percentage of total] |
[High/Medium/Low] |
| [second segment] | [weight] | [percentage of total] |
[High/Medium/Low] |

3. PRIORITY EXPLANATION
```

- Most emphasized concept: [segment with highest weight]
- Moderately emphasized: [segments with medium weights]
- De-emphasized: [segments with lowest weights]
- Effect of ordering: [how order interacts with weight]

4. VISUAL PREDICTION
- What the model will prioritize in the output: [description]
- What may be subtle or background: [description]
- What may be excluded: [description]

5. OPTIMIZATION SUGGESTIONS
- To increase [concept]: change weight to [X]
- To decrease [concept]: change weight to [X]
- To balance: [recommended weights]

6. COMMON WEIGHTING PATTERNS

Goal	Pattern Example	Explanation
Subject emphasis	`cat::2 dog::1`	Cat twice as important as dog
De-emphasis	`cluttered room::0.5 clean design::2`	Clutter less important
Equally weighted	`mountain::1 lake::1`	Equal visual weight
Break without weight	`mountain lake:: sunset`	Break separates concepts

7. WEIGHT CALCULATION FORMULA
- Total weight sum = [sum of all weights]
- Each segment's influence % = (segment weight / total sum) × 100

```
INPUTS:

Weighted prompt:
[PASTE THE PROMPT WITH :: SYNTAX]

Intended subject hierarchy (if known):
[E.G., "Dog more important than background"]

Previous output issues (if any):
[E.G., "The cat keeps disappearing"]

Model version:
[V6 / V7]

RULES:
- Total weight sum determines proportional influence, not absolute values
- Order matters even with weights (earlier tokens have slight priority)
- Weight 0 is useful for excluding without deleting (keeps context)
- Negative weights are not supported (use low positive weights instead)
- Multi-word segments need double-colon before and after: `red car::2`
- Weighted prompts work best with 2-5 segments (more dilute effectiveness)
```

How To Use It

- Total weight sum determines proportional influence, not absolute values — weights are relative to each other.

- Order matters even with weights — earlier tokens still have slight priority.
- Weight 0 is useful for excluding without deleting — keeps context without influence.
- Negative weights are not supported — use low positive weights to de-emphasize.
- Multi-word segments need double-colon before and after: `red car::2`
- Weighted prompts work best with 2-5 segments — more dilute effectiveness.

Example Input

Weighted prompt:
`cyberpunk city::2 neon lights::1.5 rain::1 futuristic car::3`

Intended subject hierarchy:
“Futuristic car should be most important, then cyberpunk city, then neon lights, then rain”

Previous output issues:
“The car kept blending into the background”

Model version:
“V6”

Why It Works
Most users write prompts linearly — word order is their only way to prioritize. Weighted prompts are a superpower they don’t understand.

This framework improves outcomes by forcing:

- prompt deconstruction (segment identification)
- weight hierarchy analysis (relative importance calculation)
- visual prediction (what the model will produce)
- optimization suggestions (how to fix imbalances)
- pattern recognition (common weighting strategies)

Failure modes this prevents:

- One element dominating unexpectedly (no weights to balance)
- Inability to emphasize a concept (weights would fix it)
- Word order as only control (limited effectiveness)
- Wasted generations (trial and error without understanding)

This improves on: Linear prompt writing. Weighted prompts give precise control over visual priority.

Related to: MJ-01 (Parameters) for syntax; MJ-06 (Remix) for variation.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, debugging frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also Style Reference Translator