Model Outputs Garbage: Debugging Bad Generations

February 18, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Repetitive loops = set repeat_penalty to 1.1-1.2. Random gibberish = corrupted GGUF file, re-download and verify sha256. Incoherent but grammatical = model too small or wrong chat template. Starts good then degrades = context overflow. Ignores instructions = you're using a base model instead of the instruct variant. Most garbage output is one of these seven causes — find yours below.

📚 **More on this topic:** Quantization Explained · Context Length Explained · Best Local LLMs for Chat · Planning Tool

Your model is running. Tokens are generating. But the output is wrong — repetitive, incoherent, or completely off-topic. The model isn't broken. Something in the pipeline is.

This guide covers seven types of bad output, what causes each, and how to fix it.

Quick Diagnostic

What does your bad output look like?

Symptom	Jump To
Same phrase repeating endlessly	Repetitive Loops
Random characters, unicode soup, gibberish	Random Gibberish
Grammatical sentences that make no sense	Incoherent but Grammatical
Good at first, then falls apart	Starts Strong Then Degrades
Ignores your instructions entirely	Ignores Instructions
Answers confidently with wrong facts	Confidently Wrong
Responds in the wrong language	Wrong Language

---

1. Repetitive Loops

Looks like:

```
The answer is that the answer is that the answer is that the answer is
```

Or a paragraph that circles back to the same point three or four times before stopping.

Cause: The repeat penalty is too low or temperature is too low. Without penalty, the model falls into attractors — high-probability token sequences that reinforce themselves. Low temperature makes the model pick the same high-probability tokens every time.

Fix:

```
# Ollama: set in Modelfile or at runtime
PARAMETER repeat_penalty 1.15
PARAMETER temperature 0.7

# llama.cpp
llama-cli -m model.gguf --repeat-penalty 1.15 --temp 0.7
```

Good defaults:

- **repeat_penalty:** 1.1-1.2 (higher = more variety, but too high makes output erratic)
- **temperature:** 0.7-0.8 for conversation, 0.3-0.5 for factual/coding tasks

If you're using text-generation-webui, check the sampling tab — repeat penalty might be set to 1.0 (disabled).

---

2. Random Gibberish / Special Characters

Looks like:

```
ÐŸÑ€Ð¸ÐtranslationError<|im_end|> tokens ÿÿ \x00\x00
```

Or a mix of special tokens, unicode characters, and fragments of what might be real words.

Cause: Almost always a corrupted download. GGUF files are large (2-50GB) and partial downloads are common. Less commonly: using a quantization format the inference engine doesn't support, or loading a model with the wrong architecture.

Fix:

1. **Re-download the model.** Verify the file size matches what HuggingFace or Ollama lists.
2. **Check the sha256 hash** if one is provided:

```
sha256sum model.gguf
# Compare with the hash on HuggingFace
```

1. **Try a different quantization.** If Q2_K gives gibberish but Q4_K_M works, the Q2 file was either corrupted or the model doesn't quantize well at that level.
2. **Check architecture compatibility.** If you downloaded a GGUF manually, make sure it matches the model architecture your engine expects. A Llama GGUF won't work with a Mamba loader.

In Ollama, `ollama pull model:tag` handles this automatically. If Ollama gives gibberish, run `ollama rm model:tag` and re-pull.

---

3. Incoherent but Grammatical

Looks like: Complete sentences that read like English but don't make sense together. The model seems to understand grammar but not the topic. Responses drift off-topic or string together vaguely related ideas.

Cause: Two common reasons:

1. **Model too small for the task.** A 3B model can form sentences but struggles with complex reasoning, multi-step instructions, or nuanced topics. It's like asking a 6-year-old to explain tax law — the words come out fine, the content doesn't.
2. **Wrong chat template.** Every model family has its own prompt format — how system prompts, user messages, and assistant responses are delimited. If the template is wrong, the model sees your conversation as a jumbled text blob instead of structured dialogue.

Fix:

For model size: try the next tier up. The jump from 3B to 8B is dramatic. From 8B to 14B is noticeable. See best local LLMs for chat for recommendations by VRAM tier.

For chat template: in Ollama, templates are handled automatically for official models. If you imported a custom GGUF, check your Modelfile's TEMPLATE field. Common formats:

```
# ChatML (Qwen, many others)
<|im_start|>system
{system}<|im_end|>
<|im_start|>user
{prompt}<|im_end|>
<|im_start|>assistant

# Llama 3
<|begin_of_text|><|start_header_id|>system<|end_header_id|>
{system}<|eot_id|><|start_header_id|>user<|end_header_id|>
{prompt}<|eot_id|><|start_header_id|>assistant<|end_header_id|>
```

Wrong template = model doesn't know where your question ends and its answer begins.

---

4. Starts Strong Then Degrades

Looks like: First response is great. Second is good. By the fifth message, the model is confused, contradicting itself, or losing the thread of the conversation.

Cause: Context overflow. Every message in your conversation adds to the KV cache. When the total exceeds the model's context window, old messages get truncated. The model literally forgets what you talked about earlier.

Some models also degrade at long contexts even within their official limit — a model with a 32K context window might perform best under 8K.

Fix:

- **Start a new conversation** for a fresh context. The simplest fix.
- **Reduce context length** if you set it higher than the model was trained for.
- **Use a model with longer native context.** Qwen3 models support 32K-128K. Mistral Nemo supports 128K. Llama 3.1 supports 128K.
- **Summarize** long conversations periodically — paste a summary as the first message in a new conversation.

In Ollama, you can check context usage with `ollama ps` — it shows current token count.

5. Ignores Instructions / Wrong Format

Looks like: You ask for JSON and get prose. You give a system prompt and the model ignores it. You ask for a list and get a paragraph.

Cause: You're using a **base model** instead of an **instruct/chat variant.** Base models are trained to predict the next token in general text — they don't follow instructions. Instruct models are fine-tuned to follow prompts.

Fix: Check your model name:

Wrong (base)	Right (instruct)
`llama3:8b`	`llama3:8b-instruct`
`qwen2.5:14b`	`qwen2.5:14b-instruct`
`mistral:7b`	`mistral:7b-instruct`

In Ollama, most default tags point to instruct variants. On HuggingFace, look for `-Instruct`, `-Chat`, or `-it` in the model name. If you downloaded a raw base model, it's not going to follow your system prompt — that's expected behavior.

For structured output (JSON, YAML), also check if your inference engine supports grammar-constrained generation. Ollama's `format: json` parameter forces valid JSON output regardless of prompt.

6. Confidently Wrong Facts

Looks like: The model states things as fact that are completely wrong. Dates, names, statistics, code syntax — delivered with total confidence.

Cause: Normal LLM behavior. Every language model hallucinates. Smaller models hallucinate more. Low-quantization versions hallucinate more than higher quant. This isn't a bug — it's a fundamental limitation of how language models work.

Fix:

- **Use a larger model.** 14B hallucinates less than 7B. 32B less than 14B. The improvement is real and measurable.

- **Use higher quantization.** Q4_K_M preserves more model quality than Q2_K or Q3_K_S. If you have VRAM headroom, Q6_K reduces hallucination further.
- **Lower temperature** for factual tasks. Temperature 0.1-0.3 makes the model more conservative and less likely to generate creative (wrong) answers.
- **Use RAG.** Feed the model relevant documents as context so it can cite sources instead of generating from memory. See our RAG guide.
- **Never trust local LLMs for critical facts** without verification. This applies to cloud models too — but especially to smaller local ones.

---

7. Outputs in Wrong Language

Looks like: You ask in English, the model responds in Chinese, Korean, or another language. Or it mixes languages mid-response.

Cause: Multilingual models (Qwen, Llama, Gemma) are trained on text in many languages. If your prompt is short or ambiguous, the model may default to the language most represented in its training data. Qwen models, for example, have heavy Chinese training data.

Fix:

Add an explicit language instruction to your system prompt:

```
You are a helpful assistant. Always respond in English.
```

In Ollama Modelfile:

```
SYSTEM "You are a helpful assistant. Always respond in English."
```

If the model persistently defaults to another language despite instructions, try a different model. Llama 3 and Mistral are more English-dominant. Qwen models are excellent but may need the language reminder.

Bottom Line

Bad output always has a cause. Repetitive loops are sampling settings. Gibberish is a corrupted file. Incoherence is model size or wrong template. Degradation is context overflow. Instruction-ignoring is a base model. Hallucination is the model being a model.

Most fixes take under a minute. The hardest one to accept: sometimes the model is just too small for what you're asking it to do. When that happens, the answer is a bigger model or more VRAM — not more prompt engineering.

Get notified when we publish new guides.

Subscribe — free, no spam
