Text Generation WebUI (Oobabooga) Guide

February 3, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** Text Generation WebUI (oobabooga) is the most flexible local AI interface, supporting every model format (GGUF, GPTQ, EXL2, AWQ, Transformers) and offering fine-grained control over inference parameters. Use it when you need features beyond what Ollama or LM Studio offer — character cards, multiple backends, extensions like voice input and RAG, or access to models not in standard libraries. The tradeoff is complexity: installation takes more effort, and the interface has a learning curve. For beginners, start with Ollama or LM Studio. Graduate to text-generation-webui when you hit their limits.

📚 **More on this topic:** Ollama vs LM Studio · llama.cpp vs Ollama vs vLLM · Open WebUI Setup Guide · Model Formats Explained · Planning Tool

If Ollama is the "just works" option and LM Studio is the "pretty GUI" option, text-generation-webui is the "give me all the knobs" option.

Created by oobabooga (the username became synonymous with the project), text-generation-webui aims to be the AUTOMATIC1111 of text generation — a comprehensive interface that supports everything, with an active community adding features constantly. It's not the easiest way to run local models, but it's often the most capable.

This guide covers what it does, when you actually need it, and how to get started without drowning in options.

What Is Text Generation WebUI?

Text-generation-webui is a Gradio-based web interface for running large language models locally. Unlike tools that focus on simplicity, it prioritizes flexibility:

What it supports:

- Every major model format: GGUF (llama.cpp), GPTQ, AWQ, EXL2, Transformers
- Multiple inference backends: llama.cpp, ExLlamaV2, Transformers, AutoGPTQ
- Direct downloads from Hugging Face (any public model)
- Fine-tuning with LoRA adapters
- OpenAI-compatible API server

- Extensions for voice, RAG, translation, and more

The tradeoff: More features means more complexity. The interface can feel overwhelming at first, and installation isn't one-click simple (though it's gotten easier).

When to Use It (vs Alternatives)

Use Case	Best Tool	Why
Quick start, just chatting	Ollama	Simplest setup, works immediately
Nice GUI, model discovery	LM Studio	Polished interface, easy model browsing
Building applications	Ollama	Best CLI/API, Docker support
Maximum control over inference	**Text-generation-webui**	All parameters exposed
Character cards, roleplay	**Text-generation-webui**	Built-in character system
Models not in standard libraries	**Text-generation-webui**	Direct HuggingFace downloads
Fine-tuning/LoRA training	**Text-generation-webui**	Built-in training tab
RAG with your documents	**Text-generation-webui**	SuperboogaV2 extension
Voice input/output	**Text-generation-webui**	Whisper + TTS extensions
Multiple backend formats	**Text-generation-webui**	GPTQ, EXL2, AWQ, etc.

The short version:

- **Start with Ollama** if you want the fastest path to running models
- **Use LM Studio** if you want a clean GUI for testing models
- **Graduate to text-generation-webui** when you hit the limits of those tools

---

System Requirements

Minimum

- **OS:** Windows 10+, Ubuntu 18.04+, macOS (Intel or Apple Silicon)
- **RAM:** 8GB for 7B models, 16GB for 13B models
- **Disk:** ~10GB for base installation + model storage

- **GPU:** Optional but recommended

Recommended for Good Performance

- **GPU:** NVIDIA with 8GB+ VRAM (RTX 3060 or better)
- **RAM:** 32GB+ for larger models or CPU inference
- **Storage:** SSD for faster model loading

VRAM Guidelines

Model Size	Minimum VRAM (Q4)	Recommended
7B	4-6GB	8GB
13B	8-10GB	12GB
30B	16-20GB	24GB
70B	32-40GB	48GB+

See our VRAM requirements guide for detailed breakdowns.

Installation

Option 1: One-Click Installer (Recommended)

The easiest method for most users:

1. **Download the repository:**

```
git clone https://github.com/oobabooga/text-generation-webui
cd text-generation-webui
```

2. **Run the start script for your OS:**
   - Windows: Double-click `start_windows.bat`
   - Linux: `./start_linux.sh`
   - macOS: `./start_macos.sh`

3. **Follow the prompts.** The installer will:
    - Set up a Conda environment (using Miniconda)
    - Install PyTorch and dependencies
    - Download required packages

Installation takes 5-15 minutes depending on your internet speed.

1. **Access the interface** at `http://localhost:7860`

Option 2: Portable/Self-Contained (No Installation)

For GGUF models only, download pre-packaged builds:

1. Go to the Releases page
2. Download the right version:
    - NVIDIA GPU: `cuda12.4` build
    - AMD/Intel GPU: `vulkan` build
    - CPU only: `cpu` build
    - Apple Silicon: `macos-arm64` build
3. Extract and run `start_` script

This method is limited to llama.cpp/GGUF models but requires no setup.

Option 3: Manual Installation

For advanced users who want control over the environment:

```
# Clone repository
git clone https://github.com/oobabooga/text-generation-webui
cd text-generation-webui

# Create virtual environment
python -m venv venv
source venv/bin/activate  # Linux/Mac
# or: venv\Scripts\activate  # Windows

# Install PyTorch (adjust for your CUDA version)
pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121

# Install requirements
pip install -r requirements.txt
```

```
# Start
python server.py
```

Loading Your First Model

Downloading Models

1. Go to the **Model** tab
2. In the "Download" section, enter a Hugging Face model path:
   - For GGUF: `TheBloke/Llama-2-7B-Chat-GGUF`
   - For full precision: `meta-llama/Llama-2-7b-chat-hf`
3. Click **Download**

Models are saved to `text-generation-webui/models/` (or `user_data/models/` in newer versions).

Manual Model Placement

- **GGUF files:** Place the single `.gguf` file directly in the models folder
- **Other formats:** Place the entire model folder (with config.json, tokenizer files, etc.) as a subfolder

Loading a Model

1. Go to **Model** tab
2. Select your model from the dropdown
3. Choose the appropriate **Loader**:
   - `llama.cpp` for GGUF files
   - `ExLlamaV2` for EXL2 files (fastest for NVIDIA)
   - `AutoGPTQ` for GPTQ files
   - `Transformers` for full-precision models
4. Click **Load**

Choosing the Right Loader

Format	Recommended Loader	Notes
GGUF	llama.cpp	Most compatible, CPU+GPU
EXL2	ExLlamaV2	Fastest on NVIDIA GPUs
GPTQ	ExLlamaV2 or AutoGPTQ	Good performance
AWQ	AutoAWQ	Alternative to GPTQ
Safetensors/PyTorch	Transformers	Full precision, most VRAM

See Model Formats Explained for format differences.

Interface Overview

Text-generation-webui has multiple tabs, each for different purposes:

Chat Tab

The main interface for conversation. Features:

- Character selection and creation
- Chat history management
- Instruction templates for different model types
- Regenerate, continue, and impersonate buttons

Notebook Tab

Single-prompt completions without chat formatting. Good for:

- Creative writing
- Completions without instruction-following overhead
- Testing raw model output

Default Tab

Basic text completion interface. Shows the raw output without any formatting.

Parameters Tab

Control everything about generation:

- **Temperature:** Randomness (0.1-2.0, default ~0.7)
- **Top-p/Top-k:** Sampling strategies
- **Repetition penalty:** Avoid loops
- **Max tokens:** Response length limit
- **Context length:** How much history to remember

Model Tab

Model management:

- Download models from Hugging Face
- Load/unload models
- Configure loader-specific settings (GPU layers, cache size, etc.)

Training Tab

Fine-tune models with LoRA:

- Dataset upload
- Training parameters
- LoRA adapter management

Session Tab

- Extension management
- API settings
- Interface preferences

Essential Settings to Understand

GPU Layers (n-gpu-layers)

For GGUF models with llama.cpp, this controls how much runs on GPU vs CPU:

- **0:** Everything on CPU (slow but works with any VRAM)

- **All:** Everything on GPU (fastest, needs enough VRAM)
- **Partial:** Split between GPU and CPU

Start with all layers on GPU. If you get out-of-memory errors, reduce until it fits.

Context Length (n_ctx)

How many tokens the model remembers. Higher = more memory usage.

Context	Approximate VRAM Cost
2048	Baseline
4096	+1-2GB
8192	+2-4GB
16384	+4-8GB

Don't set this higher than you need. See Context Length Explained.

Cache Type

For llama.cpp loader:

- **f16:** Default, good quality
- **q8_0:** 8-bit cache, saves VRAM with minimal quality loss
- **q4_0:** 4-bit cache, more savings, some quality loss

If you're VRAM-constrained, try q8_0 cache.

Instruction Template

Different models expect different prompt formats. Common templates:

- **Llama-2-Chat** for Llama 2 chat models
- **ChatML** for Qwen, many others
- **Alpaca** for Alpaca-style fine-tunes
- **Mistral** for Mistral Instruct models

Using the wrong template = weird outputs. Check the model card on Hugging Face.

Useful Extensions

Extensions add functionality beyond base chat. Enable them in the **Session** tab.

SuperboogaV2 (Built-in RAG)

Add your own documents to conversations:

- Supports PDF, DOCX, TXT, and URLs
- Creates embeddings for semantic search
- Automatically includes relevant context in prompts

Good for: Chatting with your notes, research papers, documentation.

Setup:

1. Enable `superboogav2` in Session tab
2. Scroll down in Chat to find the interface
3. Upload files or paste text
4. Chat normally — relevant context is auto-injected

Whisper STT (Voice Input)

Speech-to-text using OpenAI's Whisper:

1. Enable `whisper_stt` extension
2. Select model size (tiny → large)
3. Use the microphone button to dictate

Smaller models (tiny, base) are faster but less accurate. Medium or large recommended for serious use.

Coqui TTS (Voice Output)

Text-to-speech for model responses:

1. Enable `coqui_tts` extension
2. Select voice model
3. Responses are read aloud

AllTalk TTS

Alternative TTS with more voice options and better quality. Requires separate installation.

Character Cards

Not an extension, but a core feature. Create persistent personas:

- Go to **Parameters** → **Character**
- Create new or download from character hubs
- Characters remember their personality across sessions

Popular for roleplay, but also useful for creating specialized assistants.

Performance Optimization

For NVIDIA GPUs

1. **Use ExLlamaV2 loader for EXL2/GPTQ models** — It's typically 20-50% faster than alternatives
2. **Flash Attention:** Enable if supported (reduces memory, improves speed)
3. **Set all GPU layers:** Maximize GPU usage
4. **Use CUDA 12.x:** Latest CUDA typically has better performance

For CPU Inference

1. **Use llama.cpp loader** — Most optimized for CPU
2. **Enable GPU offload if possible:** Even partial GPU helps
3. **Use quantized models:** Q4_K_M or Q5_K_M for balance of speed and quality
4. **Match threads to physical cores:** Don't exceed physical core count

Memory Optimization

If you're hitting VRAM limits:

1. Reduce context length
2. Use smaller quantization (Q4 instead of Q8)
3. Enable q8_0 or q4_0 cache
4. Offload some layers to CPU

Common Issues and Fixes

“Failed to load model”

Causes:

- Wrong loader selected for model format
- Insufficient VRAM
- Corrupted download

Fixes:

- Match loader to format (GGUF → llama.cpp, EXL2 → ExLlamaV2)
- Reduce GPU layers or context length
- Re-download the model

“CUDA out of memory”

Fixes:

- Reduce `n-gpu-layers`
- Lower context length
- Use smaller quantization
- Close other GPU applications

Model outputs garbage or wrong format

Cause: Wrong instruction template

Fix: Check model’s Hugging Face page for correct template, set it in Parameters tab

“Connection errored out” in browser

Causes:

- Extension conflicts
- Pydantic version issues

Fixes:

- Disable recently added extensions

- Run the update script: `update_wizard` option
- Fresh install if persistent

Character "None" error after update

Fix: Create a file called `none.yaml` in the `characters/` folder with basic content, or select a different default character in settings.

Slow on first load

Normal behavior — the model needs to load into memory. Subsequent generations are faster. Using an SSD significantly improves load times.

Text-generation-webui vs Alternatives

Feature	Text-gen-webui	Ollama	LM Studio
Ease of setup	Medium	Easy	Easy
Model formats	All	GGUF only	GGUF, some GGML
GUI quality	Functional	None (CLI)	Polished
Character cards	Yes	No	No
Extensions	Many	Limited	Few
Fine-tuning	Built-in	No	No
API server	Yes	Yes	Yes
HuggingFace direct download	Yes	No	Via search
Performance	Excellent (with right backend)	Good	Good
Community	Large, active	Large	Moderate

Bottom line:

- Ollama wins on simplicity and developer experience
- LM Studio wins on polish and beginner-friendliness
- Text-generation-webui wins on features and flexibility

Bottom Line

Text-generation-webui is the power user's choice for local AI. It's not the easiest tool to learn, but it's the most capable:

Use it when:

- You need model formats beyond GGUF
- You want character cards or roleplay features
- You need extensions (RAG, voice, translation)
- You want to fine-tune models locally
- Standard tools don't have the model you want

Skip it when:

- You just want to chat with models quickly (use Ollama)
- You prefer a polished, simple interface (use LM Studio)
- You're building applications (Ollama's API is cleaner)

The learning curve is real, but once you understand the interface, you have access to capabilities that simpler tools can't match. Start with Ollama or LM Studio, and graduate to text-generation-webui when you hit their limits.

```
# Get started
git clone https://github.com/oobabooga/text-generation-webui
cd text-generation-webui
./start_linux.sh  # or start_windows.bat / start_macos.sh
```

The interface lives at `http://localhost:7860` — bookmark it, you'll be back.

Source: https://insiderllm.com/guides/text-generation-webui-oobabooga-guide/

Free guides for running AI locally