ChatGPT

LM Studio: Running Local LLMs on Windows

LM Studio is a **free desktop application** for running large language models (LLMs) locally on your computer. It provides an easy GUI to download popular models (like LLaMA, Mistral, etc.), load them into memory, and chat with them – all offline on your Windows PC [1] [2]. Below, we cover how to install LM Studio on Windows, use its chat interface, customize settings, supported model formats, optimization tips, common limitations, and community resources.

1. Installation on Windows

System Requirements: LM Studio supports 64-bit Windows (Intel/AMD x64) and Windows on ARM (Snapdragon) PCs [3]. Minimum recommended specs are a CPU with **AVX2 support**, **16 GB or more RAM**, and **at least 4 GB of VRAM** on a GPU for best performance [4]. (It can run without a GPU, but a dedicated NVIDIA or AMD GPU can significantly accelerate model inference [5].) Ensure you have an up-to-date **Windows 10/11** installation with updated graphics drivers.

Download and Setup: Installing LM Studio is straightforward:

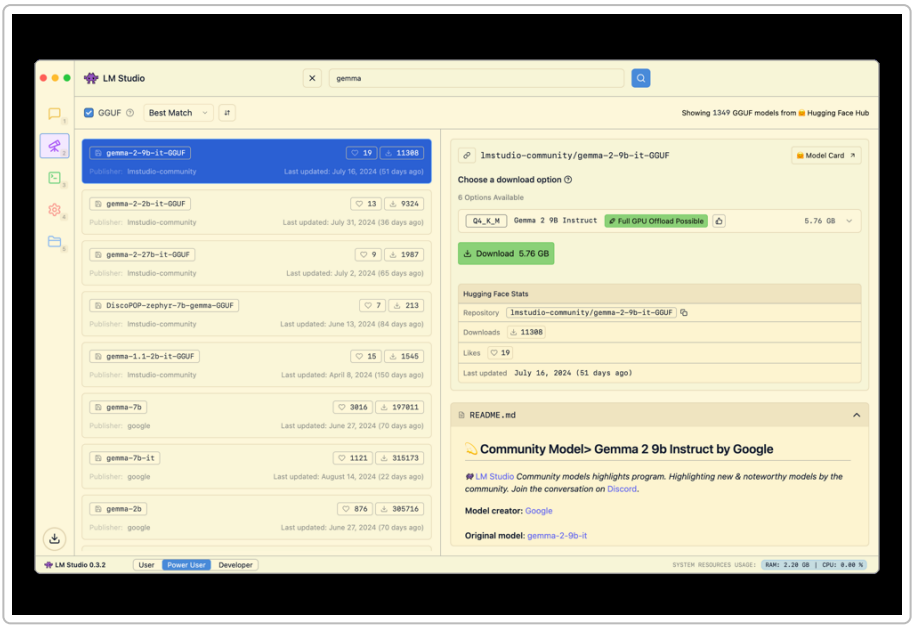
1. **Download the Installer:** Visit the official **LM Studio website** and download the Windows installer (an `.exe` file) [6]. The latest version (as of writing) is 0.3.23 for Windows [7].
2. **Run the Installer:** Run the downloaded installer and follow the prompts. LM Studio includes all necessary dependencies (including its CLI tool and back-end engines), so you typically don't need to install anything else separately.
3. **First Launch:** After installation, launch LM Studio. On first run, it will set up a working directory (usually under your user profile, e.g. `%USERPROFILE%\.lmstudio\` on Windows) for storing models and configuration. No additional configuration is required to start – you should see the LM Studio home screen and be ready to download a model.

Note: LM Studio is cross-platform (Mac and Linux as well), so make sure you grabbed the Windows-specific installer. It's free for both personal and work use [8]. If you encounter SmartScreen or antivirus warnings (since it's a new .exe), you may need to grant permission as it's a trusted app from *Element Labs, Inc.*.

2. Getting Started: Downloading and Loading Models

Once installed, using LM Studio involves **finding a model, downloading it, and loading it into memory** for chat inference:

Model Discovery and Download: LM Studio has a **"Discover" tab** where you can search for openly available models and download them in-app [9]. This interface is integrated with Hugging Face's model hub, so you have access to a wide range of community models. You can either pick from a curated list of popular models or use the search bar to find models by name or keyword (for example, searching "Llama" or a specific Hugging Face repo name) [10].



LM Studio's Discover interface on macOS (Power User mode) showing a search for a model. The left pane lists matching models on Hugging Face (with publisher name and last updated date). The right pane shows details for a selected model (Gemma 2 9B Instruct by Google), including available quantized files (Q4_K_M, etc.), file sizes, and a green indicator "Full GPU Offload Possible" if your GPU can handle the model [11] [12]. A Download button fetches the model weights directly.

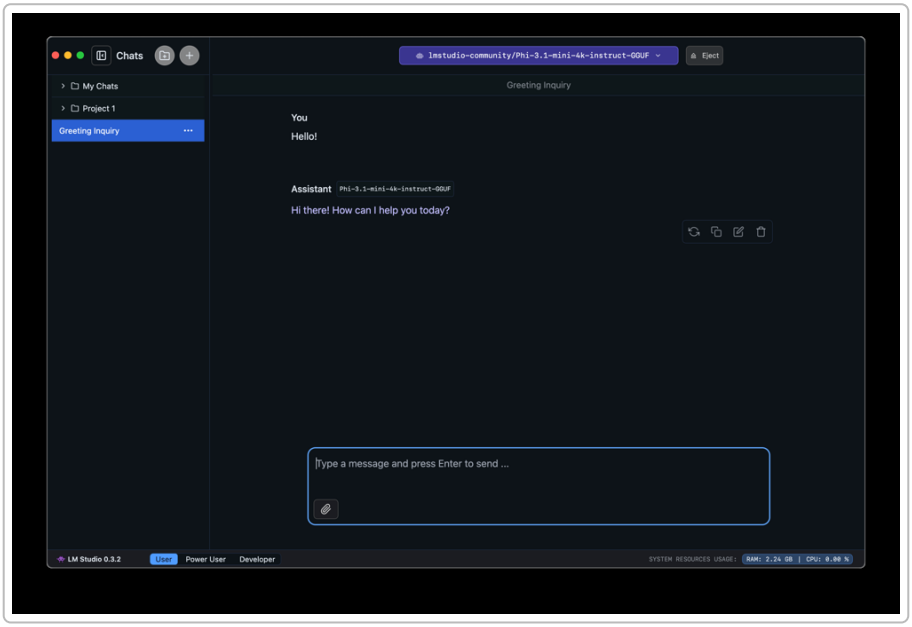
- **Quantization Options:** Many models offer multiple download files with names like `Q3_K_S`, `Q4_K_M`, `Q8_0`, etc. These are *quantized* versions of the model – they trade off some accuracy for significantly smaller size and faster inference. The "Q" number indicates the bit precision (e.g. 4-bit or 8-bit) [13]. In general, **use 4-bit or higher** precision if your hardware can handle it, as it preserves more model quality [12]. Lower-bit quantizations (like 2- or 3-bit) further reduce memory usage but may degrade responses noticeably [11]. The file size is shown for each option, helping you decide which will fit in your RAM/VRAM.
- **Downloading:** Click the **Download** button next to your chosen model version to begin downloading the weights. The model files can be several gigabytes, so this may take time. (You'll see progress in the UI.) By default, models are stored under your `%USERPROFILE%\.lmstudio\models\` directory. You can change the models storage directory in the "My Models" tab of settings if needed [14] – for example, to put large model files on a secondary drive.

Loading a Model into Memory: After downloading, switch to the **Chat** tab to load the model. In the chat interface, click the **model selector** (often a drop-down or a `⊕` icon) to open the **model loader** [15]. You should see the models you downloaded listed. Select the model you want, and then choose any load configuration options if prompted (more on these below). Finally, confirm to **load** the model. Loading may take a few seconds up to a minute depending on model size and your hardware. This process allocates the necessary memory and prepares the model for inference [16].

- **Switching Models:** You can unload or switch models by using the model selector again. LM Studio supports multiple models loaded simultaneously, but be mindful of RAM limits – loading a second large model without unloading the first can exceed memory. Typically, you'll **"Eject"** a model (unload it) before loading another. The interface provides an *Eject* button to unload the current model from memory when needed. You can also use the CLI (`lms unload`) or the system tray if running headless to unload models [17] [18].

- **What "Loading" Means:** Loading a model essentially means copying the model's weights from disk into RAM (and VRAM, if using GPU acceleration) [16] . Until a model is loaded in memory, you cannot run inference. Once loaded, the model will remain resident in RAM until you unload it or close LM Studio (so ensure you have enough memory for the model's size plus overhead).

Chat Interface and Inference: Once a model is loaded, you can start chatting. The UI is **ChatGPT-like** – there's a text box to type your prompt, and the model will generate a response. LM Studio supports multi-turn conversations, where each prompt and response are appended in a chat thread [19] .



LM Studio's Chat interface (dark theme). The left sidebar shows saved conversations ("Greeting Inquiry" under Project 1 *folder). The current chat with the assistant is in the center. A model (Phi-3.1-mini-4K) is loaded (top bar shows the model name and an* Eject *button). The user's prompt "Hello!" and the assistant's reply are displayed. At the bottom is the message input box (with an option to attach files, for document-based chats). This screenshot is in* User mode *(basic UI)* [19] [20] .

- **Multi-chat Management:** You can have multiple chat sessions. The left panel allows creating new chats (click "+" or use **Ctrl+N** on Windows) and organizing chats into folders [21] . This is useful to keep different conversations (or different roles/projects) separate. Chats are automatically saved in JSON format on disk (under `.lmstudio/conversations`) so you can resume them later [20] [22] . *(Note: The model does not learn or update itself from your chats – each session is purely inference, not fine-tuning* [23] *.)*
- **Chatting and Prompts:** Type your question or instruction in the text box and press Enter. The assistant's reply will appear below, streaming token by token. You can continue the dialogue back-and-forth. LM Studio includes features like stopping generation, copying responses, or editing/regenerating specific messages in Power/Dev modes [24] . It also allows attaching documents in a chat (for retrieval-augmented generation, i.e. question answering over your files) – see the "Chat with Documents" feature for offline Q&A on local PDFs/TXT [25] .

3. Customization and Advanced Settings

LM Studio caters to both casual users and advanced users with a **mode-based UI** and many configurable parameters:

UI Modes (User vs Power User vs Developer): In the app's footer, you can toggle between three modes [26] [27]: - **User Mode:** Simplest interface – just the chat UI with sensible defaults. Great for beginners who want a ChatGPT-like experience without worrying about settings [28]. - **Power User Mode:** Exposes more controls, such as model **load parameters** and **inference settings**, and advanced chat editing features [24]. Use this if you want to tweak things like the context length, number of threads, or sampling parameters (temperature, top-p, etc.) for better results. - **Developer Mode:** Unlocks *all* features and dev tools. In Developer mode, you get additional keyboard shortcuts, a developer console/log, and can adjust experimental settings. There's a **Developer** section in Settings for toggling features. This mode is aimed at those integrating LM Studio with other applications or debugging issues [29]. (For example, Developer mode is useful if you plan to use LM Studio's local API endpoints or need detailed logs.)

You can switch modes at any time; the higher modes include everything in the simpler modes plus more options.

Configuring Model Load Settings: When loading a model, LM Studio allows customization of how the model is loaded. In Power/Dev mode, clicking a gear icon next to a model in the "My Models" tab lets you set **per-model default load parameters** [30]. You can adjust: - **Context Length** – e.g. use 2048 (2K) tokens or 4096 tokens if the model supports longer context [31]. Larger context windows let the model remember more conversation history, at the cost of using more RAM and slightly slower speed. - **GPU Offload** – what fraction of the model to offload to GPU. LM Studio uses the llama.cpp back-end which supports offloading model layers to GPU (NVIDIA or AMD). You can specify, for instance, `--gpu=1.0` for *full* GPU offload (if VRAM permits), or a percentage like 0.5 (half the model on GPU) [32]. There are also options like `max` (use all available VRAM) or `auto` (LM Studio will decide) [32]. Offloading to GPU can dramatically speed up inference if your GPU has sufficient VRAM – the UI will even indicate "Full GPU Offload Possible" for certain model files when your GPU is detected【30†image】. - **Memory and Precision** – whether to use optimizations like **Flash Attention** (speeds up attention mechanism if supported by your hardware) [33], or load the model in 8-bit vs 4-bit if multiple precisions are available. - **Threads/Batch Settings** – In some versions, you can set number of CPU threads for inference and the token batch size. By default, LM Studio uses all logical cores for maximum speed, but you can lower it if you want to multitask on your PC during inference.

You can either set these parameters *before* loading each time, or save them as **default for that model** so next load will reuse them [34] [35]. This is handy if, for example, one model runs best with a 8k context and partial GPU offload, while another smaller model you prefer to run entirely on CPU with 4k context – you can have those preferences remembered per model.

Adding Models Manually: While the built-in downloader covers Hugging Face models, you might have a model file from elsewhere. LM Studio can import such models as long as they are in a supported format (see next section). The simplest way is to place the files into the LM Studio models directory in the correct subfolder. The expected structure is:

```
<models directory>/
└── <publisher>/
    └── <modelName>/
        └── model-file.gguf
```

For example, a model *"infra-ai/ocelot-v1"* would be placed as:

```
.lmstudio/models/infra-ai/ocelot-v1/ocelot-v1-instruct-q4_0.gguf
```

[36] [37].

LM Studio preserves Hugging Face's naming scheme, so using the publisher (user or org) and model name as folders ensures the app will recognize it [38]. After placing files, launch LM Studio and the model should appear in your "My Models" list.

Alternatively, LM Studio provides an **experimental CLI import**: running `lms import <path/to/model.gguf>` will interactively guide you to import a GGUF model into the right folder [39]. This is essentially a helper to move the file and add any metadata.

Backends (Llama.cpp and Beyond): Under the hood, LM Studio uses **llama.cpp** – a popular C++ library for LLM inference – as the engine on Windows and Linux [40]. All models downloaded as GGUF/GGML are run via llama.cpp. On Mac, LM Studio additionally supports Apple's **MLX** engine (Apple's Metal Performance Shaders for ML, available on macOS 14+ for Apple Silicon) [41]. On Windows, you will only be using llama.cpp (with CPU and optional GPU acceleration). The good news is llama.cpp now supports both NVIDIA (via CUDA) *and* AMD GPUs (via Vulkan or DirectML under the hood), which LM Studio takes advantage of [5]. In fact, one advantage noted by users is that **LM Studio supports a wide range of AMD GPUs on Windows** (more so than some other apps) [42]. So whether you have a CUDA-enabled NVIDIA card or an AMD Radeon, LM Studio should leverage it for acceleration – just ensure your GPU drivers are updated to the latest version for compatibility [43].

Tip: You can manage the installed llama.cpp runtime versions by pressing **Ctrl + Shift + R** in LM Studio [44]. This allows advanced users to update or switch out the inference backend if needed (for example, to use a newer llama.cpp build). However, for most users the bundled runtime "just works."

4. Supported Model Formats and Compatibility

LM Studio is compatible with models in several formats common to local LLMs: - **GGUF:** The primary format for Llama-family models now. GGUF is the successor to GGML, introduced in mid-2023 to support new features and tokenizer data. **LM Studio fully supports GGUF** files (this is the default for downloads). If you have older `.ggml` files, note that modern llama.cpp (and thus LM Studio) expects GGUF – you may need to convert those models to GGUF using a conversion script if they won't load [45] [46]. - **GGML:** Older quantized model format. As of August 2023, llama.cpp has deprecated direct GGML support in favor of GGUF [45]. However, many third-party libraries and UIs still handle GGML, and LM Studio's documentation indicates it supports "various model formats (GGUF, GGML, SafeTensors)" [47]. In practice, if you try to load a pure GGML model and it fails, convert it to GGUF. (LM Studio 0.3+ includes a recent llama.cpp build where **GGML models might not load unless using an older backend**.) - **SafeTensors:** This is a format for storing model weights (typically unquantized or in other formats). LM Studio can download models stored as `.safetensors` on Hugging Face, **but it will still require a compatible inference runtime to use them**. Often, safetensors models are full precision (FP16) which are too large to run on CPU-only. In practice, most safetensors would be used with a GPU or need conversion to a quantized GGUF for llama.cpp. The TabbyML documentation explicitly notes LM Studio can handle SafeTensors and provides an OpenAI-like API for them [47], so support is there, possibly via the API mode or future extensions. For now, the easiest path is to stick with quantized GGUF/GGML files, as those run efficiently on CPU/GPU via llama.cpp.

Other formats: LM Studio (through llama.cpp) does *not* natively run **GPTQ** or other transformer formats (like HF `.bin` or `.pt` models) directly. You would need to convert those to GGUF (for example, many models on Hugging Face have community-provided GGUF quantizations – look for filenames ending in `.gguf`). The built-in search primarily surfaces GGUF quantized models from the community (often maintained by users like TheBloke, or LMStudio's own community account). In summary, to ensure compatibility, prefer models labeled as **GGUF**. If you only find a model in safetensors or GPTQ, consider converting or searching if someone has shared a GGUF version.

Tokenizer and Architecture Compatibility: Generally, LM Studio/llama.cpp supports models based on the LLaMA architecture and its derivatives (Llama-2, etc.), as well as newer architectures that the community extends support for (e.g. Mistral 7B, Qwen, Falcon, etc., have GGUF versions). If a model introduces a radically new architecture not supported by llama.cpp, it won't run in LM Studio unless support is added. Keep LM Studio updated for the latest compatibility – e.g., support for Mistral and other 2024 models was added in newer releases. If a GGUF model fails to load, it might be due to using an older LM Studio; check their releases for notes and update to the latest stable or beta which might include the necessary backend improvements.

5. Optimization Tips for Performance

Running large models locally can be demanding. Here are some tips to optimize performance and memory usage in LM Studio:

- **Choose the Right Quantization:** As mentioned, quantized models significantly reduce RAM requirements. A 7B model in 4-bit might use ~4GB RAM, whereas 16-bit would use 16GB. If you have 16GB system RAM, a 13B model in 4-bit (which might be ~10GB) is feasible, but the 30B or larger models likely require 8-bit quantization on a GPU or won't fit at all. Use *Q4_K_M* or *Q5* quantizations for a good quality-speed tradeoff if you can; use *Q8* only if you have abundant memory and need maximum fidelity. Lower than 4-bit (Q3 or Q2) should be last resort if you're extremely RAM-limited, as quality may drop noticeably [11] [12]. LM Studio's interface shows how large each model file is – ensure it's comfortably within your RAM.
- **Utilize GPU Offloading:** If you have a supported GPU, make use of it. In the load configuration, set GPU offload to "Auto" or a high fraction. Offloading 100% (if VRAM allows) will give the best speed. For example, a 7B model fully on an RTX 3060 (12GB) can generate text much faster than on CPU. If VRAM is smaller, offload partially – e.g. offload 20–40 layers of a 70B model to a 6GB GPU, which still yields a boost. Keep an eye on **System Resources** at the bottom of the app: it shows RAM and CPU usage. If you see RAM is maxed out but GPU memory is free, consider offloading more to GPU. Conversely, if you over-offload and your GPU runs out of memory, the model may fail to load or performance could degrade (due to GPU RAM swapping). A good rule is to leave ~0.5–1GB headroom in your GPU VRAM to avoid out-of-memory errors.
- **Reduce Context Size if Needed:** Larger context = more memory and compute per token. If you don't need a 4096-token context for a given chat, using a 2048 or 1024 context length can save resources (and potentially improve speed). You can adjust this in the model load settings before loading [32]. Some 13B+ models with long context (8k or 16k) might be too slow on your hardware; try shorter context to see the effect.
- **Use Appropriate Threads:** By default LM Studio will use all your CPU cores, which is usually optimal for throughput. But if you find the UI becoming unresponsive or your system lagging during generation, you can limit the number of threads. For example, set it to use 6 threads on an 8-core CPU to leave some headroom. This can be done in Developer mode via a config file or CLI (the GUI may not expose it directly yet, depending on version). Also, if running on battery (like a laptop), fewer threads can reduce power draw and heat.

- **Flash Attention and Optimizations:** If the option for **Flash Attention** is available (likely only on certain GPU backends or newer CPU instructions), enabling it can increase token generation speed by optimizing attention calculations [31] [48]. Similarly, ensure **BLAS** or acceleration libraries are enabled in llama.cpp (LM Studio's llama.cpp builds typically have BLAS and FMA enabled by default for x64).
- **Speculative Decoding (Advanced):** In recent versions (v0.3.10+), LM Studio introduced *speculative decoding* [49] [50]. This technique uses a smaller "draft" model to generate tokens ahead of the large model, and if the large model agrees, it accepts them – resulting in faster generation with no quality loss in ideal cases. Using this requires loading two models (a fast small model and your main model) and enabling speculative decoding in the sidebar. If you have the RAM/VRAM to spare for an extra model, this can give 1.5× to 2× speedups in token throughput [51] [52]. The UI will show a "Draft model" selector when a main model is loaded [53]. Make sure the draft model is **compatible** (e.g., from the same family/tokenizer – LM Studio will indicate compatibility) [54]. This feature is more experimental but worth trying for power users who want faster responses.
- **Avoid Background Load:** For best performance, close other heavy applications to free up CPU and RAM. LM Studio will benefit from having as much of the CPU and memory bandwidth as possible.
- **Running Headless:** If you only need the API or CLI (no GUI overhead), you can run LM Studio in **headless mode** as a background service [55] [56]. This can slightly reduce resource use and is useful if you call the local API from other programs. Headless mode can be enabled in settings ("Run server on login/background") [57].

6. Common Issues and Limitations (Windows Version)

While LM Studio is a powerful tool, you might encounter a few issues or limitations on Windows:

- **Initial Launch Problems:** If LM Studio fails to start (e.g. hangs on the loading screen or crashes on startup), it could be due to an incomplete model download or out-of-date drivers. One known issue is that if a model download was interrupted and left a corrupt file, the app might crash when opening (as it tries to index the broken model). The workaround is to delete the incomplete model files (check your `.lmstudio\models` directory) or remove and re-download them [58] [59]. Also ensure you have enough disk space on the drive where models are stored – a full disk can prevent LM Studio from launching properly until space is freed [60].
- **Antivirus/Firewall Interference:** In some cases, security software might sandbox LM Studio (since it runs a local server for the API). If the app isn't responding, check if your antivirus is blocking it. Mark the app as trusted if needed. Also, Windows may ask for network permission for LM Studio's local API (listening on localhost); you can allow that safely – it's contained to your machine.
- **GPU Not Being Used / Errors:** If you have a supported GPU but LM Studio still runs on CPU, or you see errors like `vk::Device::createShaderModule: ErrorInitializationFailed` [61] [62], it may indicate a GPU issue. This particular error was reported with an AMD Radeon integrated GPU (Vega graphics) and suggests the **Vulkan** backend couldn't initialize (possibly unsupported GPU or driver) [63] [64]. The solution is to update your GPU drivers (use the latest Adrenalin drivers for AMD, or NVIDIA drivers as applicable) [43]. If that doesn't work or your GPU is too old, you might disable GPU offloading (load on CPU only) to avoid the error. In general, **NVIDIA GPUs (with CUDA 12+)** and **AMD GPUs GCN 5 or later** should work; older GPUs might not support the needed compute operations.
- **High VRAM but Slow Speed:** If you offload a model to a GPU but performance is unexpectedly slow, check if the GPU is actually being utilized. Sometimes laptops with dual graphics need you to force LM Studio to use the discrete GPU (via NVIDIA Control Panel or Windows Graphics

Settings). Also ensure **CUDA** is enabled if you have NVIDIA – the llama.cpp backend in LM Studio v0.3.5+ uses CUDA 12.8 for acceleration on Nvidia cards [65] [52].

- **Memory Constraints:** Running out of RAM will cause model loading to fail or swap heavily (which is effectively a freeze). If a model fails to load with no clear error, it might simply be too large – watch the RAM usage during load (displayed at bottom). If it hits 100% and crashes, consider a smaller model or lower context. As a 32-bit limitation note, very large contexts (e.g. 32k tokens) on huge models might exceed memory even if quantized – not all combos are practical on current PCs.
- **No Training/Fine-tuning:** LM Studio is designed for inference (chatting with models) only. It does not have built-in tools for fine-tuning or training LLMs on Windows. If you need to fine-tune a model, you'd have to use external tools and then import the resulting model into LM Studio.
- **Interface Quirks:** Some UI issues noted by users: e.g., on certain high-DPI displays the text might appear small/blurry (check settings for a scale or use Windows scaling). If the UI ever becomes unresponsive, you can try using the CLI `lms unload` or restarting the app. The team is actively fixing bugs – for example, updates in v0.3.18 addressed stability and an issue with model context protocol [66].
- **Windows on ARM:** LM Studio does have an ARM64 build for Windows (Snapdragon X Elite, etc.) [67], but note that performance on current ARM Windows laptops might be much lower than on x64 due to the state of emulation and optimization. As ARM PCs become more powerful (and if llama.cpp optimizes for them), this may improve.
- **Missing Features vs Mac:** The Mac version's MLX backend can utilize the Apple Neural Engine and some models (like multi-modal Gemma) very efficiently. On Windows, you are limited to llama.cpp's capabilities. This means, for instance, some multi-modal models that rely on image input might not function fully on Windows yet. Also, the latest Apple chips might outperform typical Windows PCs in efficiency for LLMs – but that's a hardware difference rather than LM Studio itself. Windows users can still run cutting-edge text models with excellent performance, especially with a good GPU.

In summary, most issues can be resolved by updating to the latest LM Studio version, keeping drivers updated, and checking the official **LM Studio Bug Tracker** on GitHub for patches or workarounds. The Discord community (see below) is also very helpful if you run into a strange issue.

7. Community Resources and Model Hubs

Official Documentation: LM Studio's docs are available on their website (under the Docs section) covering all features in detail – from basic usage to advanced API use [68] [69]. If you're looking to integrate LM Studio with your own scripts or applications, check out the sections on the **OpenAI-compatible REST API** [70] and the **LM Studio CLI (`lms`)** [71]. These allow you to use local models in place of OpenAI services for development purposes. For example, the Tabby project has a guide on configuring its editor assistant to use LM Studio's API [72] [73].

Discord Community: Join the **LM Studio Discord** for real-time help, tips from other users, and announcements [74]. The developers are active there, and it's a great place to ask questions or see how others are optimizing models on similar hardware.

Hugging Face Model Hub: Since LM Studio leverages Hugging Face, the **Hugging Face Hub** is effectively your model store. Many contributors upload GGUF versions of popular models – notable ones include user *TheBloke* (who provides a lot of quantized models in various sizes/quantizations) and the **lmstudio-community** account (which highlights recommended models, sometimes under *Community Models* in the app). You can go to Hugging Face's website to read model cards and reviews; then simply use the same model name in LM Studio's search. Keep an eye on the **Likes/download**

counts in the Discover tab – they can guide you to well-regarded models. For instance, "Llama 3.1 70B" or "Mistral 7B Instruct" with high likes might be good choices to try.

Tutorials and Guides: There are community-made tutorials on YouTube – e.g., *"LM Studio Tutorial: Run Large Language Models on your laptop"*. These can be helpful if you prefer a visual walkthrough of the installation and setup (search for LM Studio on YouTube). Some tech blogs and Medium articles (like the one by Saeed Zarinfam comparing LM Studio and Ollama on AMD GPUs [75] [76]) provide deeper dives into specific use-cases. These resources can offer insight into performance tuning on different hardware.

Alternate Tools: While LM Studio is one of the top tools for local LLMs, other popular ones include **Ollama** (Mac-focused, now with Windows support in beta), text-generation-webui (web-based UI), and TavernAI/Kobold for storytelling. If LM Studio lacks a feature you need, exploring those could help, but note that LM Studio aims to be a one-stop solution (with cross-platform support, chat UI, API, etc. in one package). The *GetStream* blog even ranked LM Studio #1 among local LLM tools [77], citing its balance of user-friendliness and power features.

Updates and Releases: The LM Studio team regularly posts release notes on their Blog (e.g., v0.3.10 added speculative decoding [78], v0.3.18 fixed bugs, etc.). Following their blog or Twitter/X account can keep you informed of new features. Because the field of LLMs evolves quickly, staying updated will ensure you get improvements like new model support or speed optimizations.

Summary: LM Studio makes running LLMs like LLaMA, Vicuna, Mistral, or Qwen on Windows accessible – even if you're not comfortable with CLI tools. With easy installation, an intuitive chat interface, and plenty of tuning options for power users, it's a great way to experiment with AI models locally. Just remember to choose appropriate model sizes for your hardware, leverage the community for recommendations, and have fun exploring what these models can do – all privately on your own PC [1] [79]!

Sources:

- LM Studio Official Docs – *About, Getting Started, Downloading Models, Chat Interface, Importing Models, Advanced Settings* [1] [80] [11] [33] [36]
- LM Studio Official Site – *Download page and feature highlights* [81] [7]
- LM Studio System Requirements [67]
- Hugging Face (TheBloke) – *GPU acceleration on Windows (Nvidia & AMD)* [5]
- TabbyML Docs – *Supported formats and API* [47]
- Medium (Zarinfam) – *AMD GPU support on Windows* [42]
- Reddit (r/LocalLLaMA) – *User experiences on Windows (crash due to disk space)* [58] [60]
- Stack Overflow – *Example error loading model on AMD GPU* [61] [43]

---

[1] [2] [6] [25] [40] [41] [44] [68] [70] [74] About LM Studio | LM Studio Docs
https://lmstudio.ai/docs/app

[3] [4] [67] System Requirements | LM Studio Docs
https://lmstudio.ai/docs/app/system-requirements

[5] [45] [46] TheBloke/EverythingLM-13b-V2-16K-GGML · Hugging Face
https://huggingface.co/TheBloke/EverythingLM-13b-V2-16K-GGML

7 8 81 LM Studio - Download and run LLMs on your computer
https://lmstudio.ai/

9 15 16 79 80 Get started with LM Studio | LM Studio Docs
https://lmstudio.ai/docs/app/basics

10 11 12 13 14 Download an LLM | LM Studio Docs
https://lmstudio.ai/docs/app/basics/download-model

17 18 32 71 lms — LM Studio's CLI | LM Studio Docs
https://lmstudio.ai/docs/cli

19 20 21 22 23 Manage chats | LM Studio Docs
https://lmstudio.ai/docs/app/basics/chat

24 26 27 28 29 User, Power User, or Developer | LM Studio Docs
https://lmstudio.ai/docs/app/user-interface/modes

30 31 33 34 35 48 Per-model Defaults | LM Studio Docs
https://lmstudio.ai/docs/app/advanced/per-model

36 37 38 39 Import Models | LM Studio Docs
https://lmstudio.ai/docs/app/basics/import-model

42 75 76 How to use Models run by LM Studio from a Spring AI application | by Saeed Zarinfam | AI-Driven | Medium
https://medium.com/vibecodingpub/how-to-use-models-run-by-lm-studio-from-a-spring-ai-application-cbbb32b031ab

43 61 62 63 64 Failed loading model in LM Studio - Stack Overflow
https://stackoverflow.com/questions/78851901/failed-loading-model-in-lm-studio

47 72 73 LM Studio | Tabby
https://tabby.tabbyml.com/docs/references/models-http-api/lm-studio/

49 50 51 52 53 54 78 LM Studio 0.3.10: Speculative Decoding | LM Studio Blog
https://lmstudio.ai/blog/lmstudio-v0.3.10

55 56 57 69 Run LM Studio as a service (headless) | LM Studio Docs
https://lmstudio.ai/docs/app/api/headless

58 59 60 LM Studio crashes : r/LocalLLaMA
https://www.reddit.com/r/LocalLLaMA/comments/1ltqf9a/lm_studio_crashes/

65 ggml - GitHub
https://github.com/ggml-org

66 LM Studio 0.3.18
https://lmstudio.ai/blog/lmstudio-v0.3.18

77 The 6 Best LLM Tools To Run Models Locally - Stream
https://getstream.io/blog/best-local-llm-tools/