Local AI - Installing Models (Windows, macOS, Linux)

Install the local LLM runtime (Ollama) and pull the recommended Qwen 2.5 instruct models on Windows, macOS, and Linux — fully offline, then point FrameworX at the endpoint.

Overview

FrameworX Local AI talks to an **OpenAI-compatible chat-completions endpoint**. The documented reference runtime is **Ollama** serving a Qwen 2.5 instruct model on a **separate networked machine** (ideally with a GPU) — set the endpoint to `http://192.168.1.50:11434/v1/chat/completions`, replacing `192.168.1.50` with the IP of the machine running your model. **The model must not run on the FrameworX/TServer host.** Everything runs **with no internet connection** after the model is pulled — the runtime, the model weights, and every inference call stay on the model host. Once a model is pulled, the network can be disconnected entirely and the AI features keep working.

The install steps on this page are performed **on the model host** (the separate machine that will serve the LLM), not on the FrameworX/TServer machine. After install, point FrameworX at that host's IP. For cloud and remote GPU options, see Remote and Cloud LLM Models. For first-install orientation — hardware sizing, what to expect, and verifying the connection — see Local AI - First Install Walkthrough; for the full endpoint and tool-bit reference see Local AI Configuration.

Which model to install

FrameworX standardizes on the **Qwen 2.5 instruct** family. The choice between tiers is driven by tool-call reliability (the model must emit a well-formed JSON tool-call envelope for the chat + MCP-tool surface) and by the hardware on hand.

Model	Tier	Disk	Rough VRAM	When to use
`qwen2.5:7b-instruct`	**Recommended default**	~4.7 GB	~6 GB class	The default for most deployments — the recommended model for new solutions, demos, and templates. Best JSON tool-call reliability for the chat + MCP-tool surface; 7B is the floor below which the structured tool-call envelope starts to malform. **Expects a machine with a GPU.**
`qwen2.5:3b-instruct`	Limited-hardware fallback	~2 GB	~3 GB class	Lower speed and quality. **Not recommended for the interactive chat experience**; acceptable for atomic reporting, classification, and summary tasks. CPU-only produces only ~2–4 tokens/sec, too slow even for a demonstration — 3B is at most a last-resort for single-shot atomic tasks, never interactive chat.
`qwen2.5:32b-instruct`	Maximum performance	~20 GB	~20 GB VRAM class	Best reasoning and multi-step tool logic. Requires a strong GPU (roughly the 20 GB VRAM class).

Offline and GPU positioning

The model runs with **no internet connection** after pulling — everything stays on the model host. For the default 7B interactive chat experience a **GPU is expected on the separate model host**; CPU-only produces only ~2–4 tokens/sec, too slow even for a demonstration. The 3B model is at most a last-resort for single-shot atomic tasks, never interactive chat.

Prerequisite: the local runtime

Ollama is the documented reference OpenAI-compatible local server. FrameworX also works with **LM Studio** (in OpenAI mode), **vLLM**, the **llama.cpp** server, or any other endpoint that speaks OpenAI-compatible chat-completions JSON — but Ollama is the runtime these instructions target, and the FrameworX defaults assume it. Whichever runtime you pick, the endpoint must answer at `/v1/chat/completions` for inference and (for Ollama, LM Studio, and vLLM) at `/v1/models` for the reachability listing.

Windows

Install Ollama with `winget`, or download the installer from `https://ollama.com/download` and run it. Ollama installs per-user — no administrator elevation is required.

```
winget install Ollama.Ollama
```

After install, Ollama runs as a background service. Verify it is listening on port **11434**:

```
# Confirm the service is reachable (lists installed models)
curl http://localhost:11434/v1/models

# Or check the listening port directly
netstat -ano | findstr 11434
```

An **NVIDIA CUDA GPU is auto-detected** — Ollama offloads model layers to it transparently, with no configuration needed. This is the expected configuration for the 7B default.

To keep the model resident in memory between requests and avoid paying the cold-load cost on every call, set `OLLAMA_KEEP_ALIVE` as a **system environment variable**:

```
# System environment variable (run as admin), then restart Ollama
setx OLLAMA_KEEP_ALIVE 24h /M
```

The default keep-alive is 5 minutes; idle longer than that and the next call pays the cold-load again.

macOS

Install either the **Ollama macOS app** (download the `.dmg` from `https://ollama.com/download` and drag it to Applications) or via **Homebrew**:

```
brew install ollama
```

Launch Ollama (open the app, or run `ollama serve` if installed via Homebrew). The **Apple-Silicon GPU (Metal) is used automatically** — no configuration needed. Verify the server:

```
curl http://localhost:11434/v1/models
```

Set `OLLAMA_KEEP_ALIVE` so the model stays resident. For a launchd-managed Ollama, set it as a user environment variable with `launchctl`; for a shell-launched `ollama serve`, export it in your profile:

```
# launchd-managed install
launchctl setenv OLLAMA_KEEP_ALIVE 24h

# Or, for a shell-launched server (add to ~/.zshrc)
export OLLAMA_KEEP_ALIVE=24h
```

Linux

Install with the official one-line script from `ollama.com`:

```
curl -fsSL https://ollama.com/install.sh | sh
```

The installer registers Ollama as a **systemd service** and starts it. Verify and check status:

```
systemctl status ollama
curl http://localhost:11434/v1/models
```

If an **NVIDIA GPU** is present with the proper drivers, the install script detects it and Ollama offloads layers automatically — the expected configuration for the 7B default. For real use run `qwen2.5:7b-instruct` on a separate GPU machine — the floor even for demos. CPU-only produces only ~2–4 tokens/sec, too slow even for a demonstration; `qwen2.5:3b-instruct` is at most a last-resort for single-shot atomic tasks, never interactive chat.

Set `OLLAMA_KEEP_ALIVE` in the systemd unit so the model stays resident across requests. Add it under `[Service]` via a drop-in:

```
# Create a drop-in: /etc/systemd/system/ollama.service.d/keepalive.conf
[Service]
Environment="OLLAMA_KEEP_ALIVE=24h"
```

Then reload and restart:

```
sudo systemctl daemon-reload
sudo systemctl restart ollama
```

Pulling the models

Pull the recommended default first. The 3B fallback and the 32B maximum-performance tier are optional — pull only what your hardware supports. The same commands work identically on Windows, macOS, and Linux.

```
# Recommended default (most deployments; expects a GPU)
ollama pull qwen2.5:7b-instruct

# Limited-hardware fallback (no GPU / very low spec)
ollama pull qwen2.5:3b-instruct

# Maximum performance (requires a strong GPU)
ollama pull qwen2.5:32b-instruct
```

Confirm what is installed with `ollama list`:

```
ollama list
```

Multiple tiers can coexist on disk; pulling one does not remove another. Pulling requires internet access (or a pre-seeded model store); once pulled, inference is fully offline. Disk and rough VRAM per tier:

Model	Disk	Rough VRAM
`qwen2.5:3b-instruct`	~2 GB	~3 GB class
`qwen2.5:7b-instruct`	~4.7 GB	~6 GB class
`qwen2.5:32b-instruct`	~20 GB	~20 GB VRAM class

VRAM figures are approximate and depend on quantization and context length. When the model does not fit in VRAM, Ollama runs the remaining layers on CPU — functional, but slower.

Pointing FrameworX at it

Once the runtime is serving and the model is pulled, connect the solution in Designer:

1. Open **Solution Capabilities** (or the Local AI tile under **Unified Namespace Data Servers** — both routes edit the same row).
2. Click **Settings** on the Local AI row and set the `Name` field to the model you pulled — for example `qwen2.5:7b-instruct`. This value goes into the POST body's `"model"` field and must match a model the endpoint can serve.
3. Set the `URL` field to `http://192.168.1.50:11434/v1/chat/completions` — replacing `192.168.1.50` with the IP of the machine running your model (the model must not run on the FrameworX/TServer host). To reach a runtime on a separate GPU host by hostname, use `http://<host-ip>:11434/v1/chat/completions`.
4. Tick **Enabled**. The reachability **status indicator turns green** when the endpoint is reachable.

Full field-by-field reference for the Settings JSON (`URL`, `Name`, `Authorization`, `Headers`, `Info`, `TimeoutSeconds`), the master enable, and the tool-category bits is on Local AI Configuration.

See also

- Local AI - First Install Walkthrough — first-install orientation: hardware sizing, what to expect, verifying the connection, and switching models.
- Local AI Configuration — full endpoint and tool-bit configuration reference.
- Remote and Cloud LLM Models — cloud and remote GPU endpoint options.
- Local AI — the main reference page (parent of this one).

In this section...