# Ollama with Intel GPU acceleration on Proxmox

Run Ollama with Intel GPU acceleration on Proxmox by exposing the iGPU to a Linux container/VM and enabling Intel's oneAPI/SYCL or OpenVINO backends for model offloading to the Xe iGPU. Below is a clean, step-by-step path using a privileged Ubuntu LXC with GPU device mapping, plus an optional OpenVINO backend route if higher throughput is desired on Intel hardware.

## What you'll build

- A Proxmox host with IOMMU and i915 GuC enabled for Alder Lake Xe graphics, exposing /dev/dri to guests for GPU compute access.
- A privileged Ubuntu LXC that gets direct access to the iGPU render node, where Intel oneAPI and Ollama run with GPU offload via SYCL (IPEX-LLM) or OpenVINO.
- Optional: a full VM with PCI passthrough if containers are not preferred, though Intel iGPU passthrough is fussier on Alder Lake, especially on Windows.

### Step 1: Prep Proxmox host (IOMMU + i915 GuC)

- In BIOS/UEFI, enable VT-d (Intel IOMMU) for device assignment; this is required for clean device mapping/passthrough.
- Enable IOMMU on Proxmox and update boot config:
  - Edit GRUB:

```
sudo nano /etc/default/grub
# Ensure this includes:
GRUB_CMDLINE_LINUX_DEFAULT="quiet intel_iommu=on iommu=pt"
```

Then apply and reboot:

```
sudo update-grub
sudo reboot
```

Validate after reboot:

```
dmesg | grep -e DMAR -e IOMMU
```

(You should see IOMMU/DMAR enabled messages.)[^1][^4]

- Enable GuC for Intel i915 (better scheduling/perf on Xe iGPU):

```
echo "options i915 enable_guc=3" | sudo tee /etc/modprobe.d/i915.conf
sudo update-initramfs -u -k all
sudo reboot
```

After reboot, confirm the render node exists:

```
ls -l /dev/dri
# Expect renderD128 at minimum
```

## Step 2: Create a privileged Ubuntu LXC and map the GPU

- Create a privileged Ubuntu 22.04/24.04 LXC in Proxmox (privileged avoids extra uid/gid mapping hurdles with GPU devices).
- Add these lines to the container's config at /etc/pve/lxc/.conf to pass the render node and allow access:

```
lxc.cgroup2.devices.allow: c 226:0 rwm
lxc.cgroup2.devices.allow: c 226:128 rwm
lxc.mount.entry: /dev/dri/renderD128 dev/dri/renderD128 none bind,optional,create=file
```

Start the container and verify /dev/dri/renderD128 exists inside it.

- Note: This also works for unprivileged containers with the correct device rules, but privileged is simpler; if staying unprivileged, consult a known working pattern for iGPU into unprivileged LXC.
- Alternative (optional): Use a full VM with PCI passthrough of the iGPU; this can work well on Linux guests, but Windows guests on Alder Lake can throw Code 43 errors without careful tuning.

## Step 3: Install Intel runtimes and Ollama (inside the LXC)

- Install Intel oneAPI Base runtime and prerequisites, then use Intel's IPEX-LLM integration to prime Ollama for Intel GPU offload via SYCL/Level Zero.
- In Ubuntu LXC:

```
sudo apt update
sudo apt install -y python3.11-venv
python3.11 -m venv ~/llm_env
source ~/llm_env/bin/activate
pip install --pre --upgrade ipex-llm[cpp]
mkdir -p ~/llama-cpp && cd ~/llama-cpp
# Initialize Ollama with Intel GPU support via IPEX-LLM helper
init-ollama
```

Intel’s guide uses a Python venv, installs ipex-llm[cpp], then initializes an Ollama build configured for Intel GPUs.

- Optional sanity check: ensure the container can see the Intel GPU compute stack by confirming OpenCL exposure if installed; many users validate with clinfo after installing Intel’s OpenCL runtime.

## Step 4: Run Ollama accelerated on the Intel iGPU

- Set recommended environment variables for full layer offload and Level Zero behavior, then start the service:

```
# From inside the LXC (activate venv if needed)
export OLLAMA_NUM_GPU=999
export no_proxy=localhost,127.0.0.1
export ZES_ENABLE_SYSMAN=1
export SYCL_CACHE_PERSISTENT=1
# If oneAPI setvars is installed/system-wide, source it:
source /opt/intel/oneapi/setvars.sh || true
# For certain kernels/GPUs this can help:
export SYCL_PI_LEVEL_ZERO_USE_IMMEDIATE_COMMANDLISTS=1
# Optional: listen on all interfaces if exposing to LAN
# export OLLAMA_HOST=0.0.0.0
ollama serve
```

**OLLAMA_NUM_GPU=999** forces all layers that can run on the GPU to offload, while the Level Zero variables improve device telemetry and command submission for Intel GPUs.

- In a second shell, pull and run a model, for example:

```
ollama run llama3.1:8b
```

The IPEX-LLM integration steers the underlying llama.cpp execution toward Intel’s SYCL/Level Zero path when possible.

## Option B: OpenVINO backend for Ollama (higher throughput path)

- Intel’s OpenVINO integration can accelerate inference on Intel CPU/iGPU/NPU and offers a dedicated backend for Ollama via OpenVINO GenAI, which can outperform generic SYCL paths in many cases.
- High-level flow in the LXC or VM:

1) Download and initialize the OpenVINO GenAI runtime; set GODEBUG=cgocheck=0 for the Ollama executable using this backend. 2) Obtain an OpenVINO IR model (e.g., quantized DeepSeek-R1-Distill-Qwen-7B int4), then package it. 3) Write a Modelfile declaring ModelType

“OpenVINO” and InferDevice “GPU,” and create the Ollama model image.

- Practical commands (example flow shown by the OpenVINO team and contributors):

```
# 1) Prepare OpenVINO GenAI runtime (env example)
export GODEBUG=cgocheck=0
# Source the OpenVINO/GenAI setup if provided by your runtime package
# source setupvars.sh

# 2) Download an OpenVINO IR model (example uses ModelScope tools)
pip install modelscope
modelscope download --model zhaohb/DeepSeek-R1-Distill-Qwen-7B-int4-ov --local_dir ./DeepSeek-R1-Distill-Qwen-7B-int4-ov
tar -zcvf DeepSeek-R1-Distill-Qwen-7B-int4-ov.tar.gz DeepSeek-R1-Distill-Qwen-7B-int4-ov

# 3) Modelfile (OpenVINO backend)
cat > Modelfile << 'EOF'
FROM DeepSeek-R1-Distill-Qwen-7B-int4-ov.tar.gz
ModelType "OpenVINO"
InferDevice "GPU"
PARAMETER stop ""
PARAMETER stop "```
PARAMETER stop "</User|>"
PARAMETER stop "<|end_of_sentence|>"
PARAMETER stop "</□"
PARAMETER max_new_token 4096
PARAMETER stop_id 151643
PARAMETER stop_id 151647
PARAMETER repeat_penalty 1.5
PARAMETER top_p 0.95
PARAMETER top_k 50
PARAMETER temperature 0.8
EOF

# Create and run the model with the OpenVINO backend
ollama create DeepSeek-R1-Distill-Qwen-7B-int4-ov:v1 -f Modelfile
ollama run DeepSeek-R1-Distill-Qwen-7B-int4-ov:v1
```

## Troubleshooting and tips

- If /dev/dri/renderD128 is missing inside the container, recheck the LXC config lines and that i915 has created the render node on the host after enabling GuC.
- If choosing a full VM instead of LXC, Linux guests are typically smoother than Windows on Alder Lake; Windows guests often hit Code 43 unless carefully configured.
- Expect meaningful gains from OpenVINO on Intel iGPU versus generic SYCL; community benchmarks have reported several-fold speedups on some 7B models, though results vary by model and kernel.
- Vulkan on Intel iGPU often underperforms versus CPU or Intel’s native stacks; prefer oneAPI/SYCL or OpenVINO backends on Intel hardware.

## Why this layout

- LXC with /dev/dri mapping is the simplest, most stable way to put the iGPU to work without full PCI passthrough, and is widely used for GPU-accelerated services on Proxmox.
- Intel’s documented IPEX-LLM environment for Ollama is straightforward and stays close to upstream Ollama usage, while OpenVINO offers an alternative backend with strong hardware-specific optimizations on Intel platforms.

**Ins0mniA**

---

Revision #1
Created 2025-08-30 15:02:57 EEST by Green
Updated 2025-09-04 02:07:44 EEST by Green