Circumventing Apple T2 Thunderbolt MMIO Restrictions: Enabling Large-BAR NVIDIA GPUs on Intel Macs via Resizable BAR Windowing

Author: Antonio Telesca — IRST Institute (irst-institute.eu), Italy **ORCID:** 0009-0003-3048-1044 — **Web of Science ResearcherID:** PKR-6753-2026 **Registered Expert, European Commission** (ID: EX2026D1365471) **GitHub:** github.com/TelescaAntonio — **Google Scholar:** gjoWniUAAAAJ — **ResearchGate:** Antonio-Telesca-4 **Date:** August 2026 — **Status:** Preprint v1 — **DOI:** 10.5281/zenodo.22011607

Abstract

Intel-based Apple computers equipped with the Apple T2 security chip have been unable to operate modern discrete NVIDIA GPUs as external GPUs (eGPUs) under Linux, a limitation documented since 2019 and unresolved upstream (t2linux issue #189). We characterize the failure mode experimentally and show that it consists of two distinct, previously conflated mechanisms: (A) an instantaneous, unlogged hard system hang triggered by any MMIO access above the 4 GiB boundary traversing the Thunderbolt tunnel, even when BARs are validly assigned within a firmware-declared 64-bit window; and (B) an identical hang triggered by the boot sequence of the NVIDIA GSP firmware, independent of BAR placement. We then present a software-only mitigation: shrinking the GPU's VRAM aperture (BAR1) to 64 MiB via the PCIe Resizable BAR capability, forcing reallocation of all GPU BARs into the firmware's 32-bit prefetchable window behind the Thunderbolt root port, and loading the proprietary NVIDIA driver with GSP firmware and MSI disabled. We validate the method on a MacBookPro16,1 with an RTX 3090 (24 GiB): the GPU sustains full compute load (352 W, 97% SM utilization) and serves a 27-billion-parameter LLM entirely from VRAM at 36.3 tokens/s. The method requires no firmware or hardware modification and is reproducible with standard Linux tooling.

Keywords: eGPU, Thunderbolt, Apple T2, PCIe, Resizable BAR, MMIO, NVIDIA GSP, Linux, LLM inference

---

1. Introduction

External GPU (eGPU) enclosures connected over Thunderbolt 3 extend laptops with desktop-class accelerators, a configuration of growing relevance for local large-language-model (LLM) inference. On Intel Macs with the Apple T2 security chip (2018–2020), however, attaching a modern NVIDIA GPU under Linux reliably crashes the machine. The t2linux community has tracked this defect for

years without resolution [1]; the accepted explanation attributed the failure to the firmware's 32-bit-only ACPI `_CRS` declarations for the Thunderbolt bridges.

This work makes three contributions:

1. **Failure characterization.** We show experimentally that the crash is *not* a simple address-assignment problem: a BAR validly placed in a firmware-declared 64-bit window (verified at 0x4010000000) still kills the machine on first access. The T2 bridge fabric hard-hangs the platform — no kernel panic, no log, no IOMMU/DMAR fault — on (A) any >4 GiB MMIO transaction through the Thunderbolt tunnel and, independently, (B) the NVIDIA GSP firmware boot sequence.
2. **A software-only mitigation.** All GPU BARs are made to fit inside the ~224 MiB 32-bit prefetchable window that the Apple firmware does grant to the Thunderbolt root port, by shrinking BAR1 (the VRAM aperture) from its native size to 64 MiB using the PCIe Resizable BAR (ReBAR) capability, followed by a PCI subtree remove/rescan. The proprietary NVIDIA driver (branch 580.xx) is then bound with GSP firmware and MSI disabled. Consequence of (B): the open-kernel-module driver (`nvidia-open`), for which GSP is mandatory, can never work on this platform.
3. **Validation under real workload.** On a MacBookPro16,1 + RTX 3090, we demonstrate stable CUDA operation under sustained full-power inference load, serving Qwen3.8-27B (19.1 GiB resident in VRAM) at 36.3 tok/s generation.

2. Background

2.1 PCIe BARs and MMIO windows behind Thunderbolt

A PCIe device exposes its registers and apertures to the host through Base Address Registers (BARs), which the platform firmware or OS must map into non-overlapping regions of the host physical address space. Modern discrete GPUs request three memory BARs; on the GA102-class RTX 3090 these are BAR0 (16 MiB, control registers), BAR1 (the VRAM aperture, natively 256 MiB, 64-bit prefetchable) and BAR3 (32 MiB, 64-bit prefetchable). When a device sits behind a hierarchy of PCI-to-PCI bridges — as is always the case over Thunderbolt, where each hop adds a bridge — every bridge on the path must carry a memory window enclosing the device BARs. The available space is therefore bounded by the *smallest* window granted by firmware to the root of the Thunderbolt subtree.

Because 64-bit prefetchable BARs are conventionally placed above the 4 GiB boundary, a device whose BARs cannot be placed there must fit entirely within the bridge's 32-bit window — typically a few hundred MiB at best. A native 256 MiB BAR1 plus 32 + 16 MiB companions often exceeds what remains once other hotplugged functions are accounted for. The PCIe Resizable BAR (ReBAR) extended capability [3] makes the aperture size negotiable: on GA102 the supported BAR1

sizes range from 64 MiB to 32 GiB. Critically for this work, ReBAR is usually exploited to *enlarge* BAR1; we use it in the opposite direction, shrinking BAR1 to its 64 MiB minimum so that the full BAR set (16 + 64 + 32 MiB) fits comfortably inside a small 32-bit window. The aperture size is irrelevant to CUDA compute workloads, which move data via DMA rather than through the CPU-visible VRAM aperture.

2.2 The Apple T2 as PCIe/Thunderbolt intermediary

On 2018–2020 Intel Macs the Apple T2 security chip is interposed in the platform's I/O fabric and mediates, among other functions, the Thunderbolt path. On the test machine (MacBookPro16,1) the firmware grants the Thunderbolt root port (00:01.1) a 32-bit prefetchable window of 224 MiB (0xc1700000–0xcf6fffff). A 64-bit window is *declared* as well, and Linux will happily assign large BARs inside it; the central empirical finding of this work (§3.1) is that the declaration is not honoured by the fabric: any CPU-initiated MMIO transaction targeting an address above 4 GiB on the Thunderbolt path causes the T2 to hard-hang the platform instantly, with no panic, no logged fault and no IOMMU/DMAR event. The prior community diagnosis ("32-bit-only `_CRS` declarations") described the symptom but understated the severity: the problem is not that the kernel *may not* place BARs above 4 GiB, but that it *must not* — even when firmware tables permit it.

2.3 NVIDIA GSP firmware

Since the Turing generation, NVIDIA GPUs embed a GPU System Processor (GSP). The open kernel modules require GSP firmware offload; the proprietary driver retains a host-driven path selectable with `NVreg_EnableGpuFirmware=0`. Driver branches that mandate GSP are therefore unusable on this platform (§3.2).

3. Failure analysis

Test platform: MacBookPro16,1 (Intel i9, T2), CachyOS Linux, Thunderbolt enclosure (JHL9480 controller) hosting an RTX 3090 (GA102, 24 GiB).

3.1 Mechanism A: >4 GiB MMIO through the Thunderbolt tunnel is fatal

With `pci=realloc`-style approaches or manual reassignment, BAR1 can be placed at e.g. 0x4010000000 inside the firmware-declared 64-bit window, and `lspci` reports a fully valid configuration. The first MMIO access to that region hard-hangs the machine instantly: no kernel panic, no serial/pstore output, no DMAR fault (ruling out a DMA/IOMMU cause — the transaction is a CPU-initiated MMIO read/write). The high half of the address space is unreachable through the T2's Thunderbolt path, regardless of what the firmware advertises.

Approaches verified NOT to work: `pci=realloc[,force]`; rewriting 64-bit BARs below 4 GiB via `setpci` without ReBAR shrink (insufficient window space); simple rescan without BAR resize.

3.2 Mechanism B: GSP firmware boot kills the T2 fabric

With BARs correctly placed below 4 GiB (§4), loading a GSP-enabled driver still hangs the machine during GPU initialization. The Thunderbolt traffic generated by the GSP boot/handshake triggers the same silent kill. Zero DMAR faults are observed, again indicating the kill is not IOMMU-mediated. The proprietary driver with `NVreg_EnableGpuFirmware=0` initializes cleanly. MSI delivery proved unstable on this path; `NVreg_EnableMSI=0` (INTx) is required for reliability.

4. Method

The mitigation, implemented as an idempotent boot-time script (`egpu-attach.sh`, systemd oneshot, guards at every step), executes:

1. **Guard checks** — firmware window identity, GPU present, correct driver branch, no driver bound.
2. **Autoprobe off; ReBAR shrink** — locate the Resizable BAR capability, program BAR1 to 64 MiB (`setpci`), with the device's COMMAND register cleared.
3. **Subtree remove + rescan** — remove the upstream bridge, rescan; the kernel now allocates BAR0 (16 MiB), BAR1 (64 MiB), BAR3 (32 MiB) inside the 32-bit window (observed: 0x82000000 / 0xc4000000 / 0xc2000000).
4. **Post-rescan guards** — verify no BAR above 4 GiB, none unassigned, no driver bound; abort otherwise (loading the driver with a stray >4G BAR would hang the machine at first access).
5. **Controlled bind** — `driver_override` + explicit `modprobe nvidia` with `NVreg_EnableGpuFirmware=0 NVreg_EnableMSI=0`; the GPU audio function is deliberately left unbound; GSP state is verified disabled before first GPU touch; `nvidia-uvm` loaded for CUDA.

The driver is blacklisted from normal udev autoloading so that binding happens exclusively after BAR remediation. Full script and configuration files are provided in the companion repository [2].

5. Validation

All measurements on MacBookPro16,1, CachyOS (kernel 7.1.8-1-cachyos), driver 580.173.02, CUDA 13.0.

Static verification. - `nvidia-smi -q`: RTX 3090 recognized; GSP Firmware Version: N/A (disabled); BAR1 Memory Usage: Total 64 MiB. - `lspci -vv`: Region 0 @ 0x82000000 (16 MiB), Region 1 @ 0xc4000000 (64 MiB, 64-bit prefetchable), Region 3 @ 0xc2000000 (32 MiB) — all below 4 GiB; ReBAR capability confirms BAR1 current size 64 MB (supported 64 MB–32 GB). - `dmesg`: initial enumeration fails to assign the native 256 MiB BAR1 ("can't assign; no space"); after ReBAR shrink and rescan, all BARs assigned within the bridge window 0xc2000000–0xc7ffffff.

Dynamic verification (LLM inference). llama.cpp server with Qwen3.8-27B (dense, GGUF), 19.1 GiB resident in VRAM, zero CPU offload:

Metric	Value
Prompt processing	101.7 tok/s (72 tokens)
Generation	**36.26 tok/s** (800 tokens, 22.0 s)
Sustained power	352–355 W (cap 370 W)
SM utilization	97–98%
Peak temperature	47 °C
BAR1 used	4 MiB of 64 MiB

The GPU sustained full compute power without instability, thermal events, PCIe replays (Replays Since Reset: 0), or errors in the kernel log.

6. Limitations

- **Link width/speed.** The upstream path negotiated 2.5 GT/s ×4 (~8 Gb/s usable) on the test setup. This lengthens model load times and bounds host↔device transfer-heavy workloads; generation-phase LLM inference (VRAM-resident, compute/memory-bound on-device) is essentially unaffected. [Investigate whether the ×4 @ 2.5 GT/s is enclosure- or T2-path-specific.]
- **64 MiB VRAM aperture.** Irrelevant for CUDA compute (DMA does not traverse BAR1), but display/graphics workloads and technologies assuming large ReBAR would suffer; this method targets headless compute.
- **Driver constraint.** Bound to proprietary branches with GSP disable support (validated: 580.173.02). GSP-only branches and `nvidia-open` are permanently incompatible with this platform (§3.2).
- **Address specificity.** Bus/device addresses and window bases are machine- and topology-specific; the published script encodes guards that abort rather than proceed on mismatch.
- **Single-platform validation.** Results are from one MacBookPro16,1 + one GA102 board. [Call for replication on other T2 models: MacBookPro15,x, Mac mini 2018, iMac Pro.]

7. Reproducibility

Companion repository [2]: annotated scripts (`egpu-attach.sh`, modprobe and systemd units), full `nvidia-smi -q`, `lspci -vv`, dmesg logs, and telemetry captures. Hardware/software manifest: MacBookPro16,1 (T2), RTX 3090 (GA102, VBIOS 94.02.59.00.5C), Thunderbolt enclosure w/ JHL9480, CachyOS kernel 7.1.8, bolt 0.9.11, nvidia-580xx-dkms 580.173.02.

Safety note. Executing BAR manipulation in the wrong order (driver bound before remediation, or GSP enabled) hard-hangs the machine with no log. The published scripts fail closed at every guard.

8. Conclusion

The long-standing inability of T2-equipped Intel Macs to host modern NVIDIA eGPUs under Linux stems from two separable T2 behaviours — fatal >4 GiB Thunderbolt MMIO and fatal GSP boot traffic — not from BAR assignment failure alone. Both are avoidable entirely in software by aperture shrinking via Resizable BAR, 32-bit window reallocation, and GSP/MSI-disabled proprietary driver binding. This restores full CUDA capability to a class of hardware previously considered incompatible, with immediate practical value for local LLM inference.

References

[1] t2linux, "RTX 3060 eGPU via Thunderbolt 3 (Alpine Ridge): BAR1 not assigned — prefetchable memory window disabled on bridge," GitHub, t2linux/T2-Debian-and-Ubuntu-Kernel, issue #189 (open). https://github.com/t2linux/T2-Debian-and-Ubuntu-Kernel/issues/189 [2] A. Telesca, "t2-egpu-nvidia: NVIDIA eGPU enablement on Apple T2 Macs," GitHub (github.com/TelescaAntonio) / Zenodo, 2026. https://github.com/TelescaAntonio/t2-egpu-nvidia [3] PCI-SIG, *PCI Express Base Specification*, Resizable BAR Extended Capability. [4] NVIDIA, "Open GPU kernel modules — GSP firmware requirements," documentation. [5] llama.cpp, ggml-org, 2026.

Preprint v1 — published on Zenodo, DOI 10.5281/zenodo.22011607. Companion code: github.com/TelescaAntonio/t2-egpu-nvidia