Revisiting Kepler: Eine Plausibilitätsanalyse der NVIDIA Tesla K80 als kosteneffizienter LLM-Inferenz-Node im Jahr 2025 *

Philipp Horn
Horn-Consulting UG
kontakt@philipp-horn.dev

30. September 2025

Revisiting Kepler: A Plausibility Analysis of the NVIDIA Tesla K80 as a Cost-Efficient LLM Inference Node in 2025

Zusammenfassung

Die steigende Nachfrage nach lokaler und kosteneffizienter Large Language Model (LLM) Inferenz stimuliert die Nutzung von Legacy-Hardware. Diese Arbeit analysiert die technische Plausibilität und die Gesamtbetriebskosten (Total Cost of Ownership, TCO) des Einsatzes der NVIDIA Tesla K80 (Kepler-Architektur, CC 3.7) als dedizierter LLM-Inferenz-Node im Jahr 2025. Obwohl der K80-Node aufgrund des niedrigen Anschaffungspreises (mindestens 89 auf dem deutschen Markt; 20–50 USD in US-Quellen [2, 24]) eine monetäre Einsparung bietet, resultieren signifikante architektonische und softwareseitige Limitierungen in einem hohen Wartungsaufwand (Software-Debt) und einer drastisch reduzierten Token-pro-Sekunde (T/s)-Rate von 4.17 T/s (empirisch) [1]. Die architektonischen Hauptengpässe, insbesondere das Fehlen der nativen `__dp4a`-Instruktion für moderne 4-bit-Quantisierungen [3, 4] und die Kommunikationsdrosselung über den PCIe 3.0 Bus bei Multi-GPU-Operationen [5], disqualifizieren die K80 als effiziente Lösung für latenzkritische Anwendungen. Der Node wird primär für stark budgetorientierte Nischenprojekte oder als technisches Lehrstück empfohlen.

Keywords: LLM Inference; Legacy Hardware; NVIDIA Tesla K80; Kepler Architecture; Compute Capability 3.7; DP4A; GGUF Quantization; Total Cost of Ownership (TCO).

*Preprint-Version. Diese Arbeit wird im nächsten Update mit zusätzlichen empirischen Benchmarks zur Robustheit und Reproduzierbarkeit erweitert.

1 Einleitung und Kontext

Die Verfügbarkeit leistungsstarker LLMs erfordert in der Regel hochspezialisierte Hardware, was für private Nutzer und kleine Forschungsteams eine erhebliche finanzielle Hürde darstellt [6]. Die Wiederverwendung von älteren Server-GPUs, wie der 2014 veröffentlichten NVIDIA Tesla K80, wird als Lösungsansatz zur Minimierung der Initialkosten diskutiert [24]. Der K80 Accelerator basiert auf dem Dual-GPU-Design mit zwei GK210-Kernen und verfügt über 24 GB GDDR5-Speicher [7], aufgeteilt in zwei separate 12 GB-Pools [8]. Ziel dieser Analyse ist es, die Durchführbarkeit dieses Ansatzes wissenschaftlich zu bewerten und die resultierenden LeistungseinbuSSen zu quantifizieren.

2 Architektonische Limit. der Kepler-Plattform

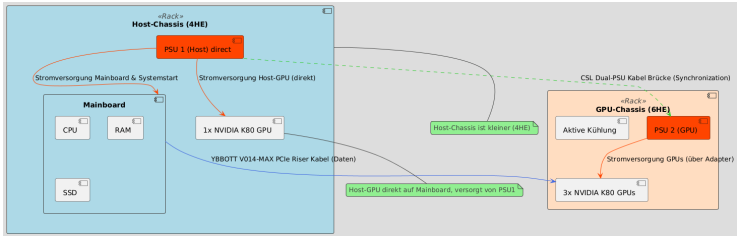

Abbildung 1: Schematische Darstellung der Dual-GPU-Architektur der NVIDIA Tesla K80. Die Segmentierung des 24 GB VRAM in zwei separate 12 GB GK210-Kerne erfordert Multi-GPU-Kommunikation über den langsameren PCIe-Bus, wenn Modelle über 12 GB geladen werden. (Quelle: Eigene Darstellung, erstellt mit PlantUML)

Die K80 ist durch drei zentrale Architektureigenschaften für moderne LLM-Workloads limitiert: die Compute Capability, die VRAM-Segmentierung und die Speicherbandbreite.

2.1 Compute Capability (CC 3.7) und Software-Dekompatibilität

Die Kepler-Architektur der K80 besitzt die Compute Capability (CC) 3.7 [10]. Diese Architektur ist seit CUDA Toolkit 12.0 offiziell *deprecated* und wird nicht mehr nativ unterstützt [11]. Ein funktionsfähiger Betrieb erfordert daher die manuelle Konfiguration einer Legacy-Toolchain (z. B. CUDA 11.4 in Verbindung mit GCC 10) [1], was eine signifikante *Software-Debt* in Bezug auf Wartbarkeit und zukünftige Kompatibilität schafft.

2.2 Fehlen der DP4A-Instruktion

Der kritischste Performance-Engpass entsteht durch das Fehlen der nativen `__dp4a`-Instruktion (per-byte integer dot product). Diese Instruktion ist für die

Hardware-Beschleunigung moderner 4-bit-Quantisierungen (GGUF k-quants) essenziell und erst ab der Pascal-Architektur (CC 6.1) verfügbar [3, 4]. Da die K80 (CC 3.7) diese Funktion nicht unterstützt, muss die Quantisierungsberechnung in Frameworks wie `llama.cpp` auf langsamere Software-Workarounds zurückgreifen [3]. Dies ist die kausale Erklärung für die geringe Inferenzgeschwindigkeit trotz aggressivem Quantisierungsgrad.

2.3 VRAM-Segmentierung und PCIe-Bottleneck

Der nominelle 24 GB VRAM ist in zwei 12 GB-Sektionen aufgeteilt [8, 9]. Modelle, deren quantisierter Speicherbedarf 12 GB überschreitet (z. B. Modelle $>$ 7B Parameter mit Kontextfenster), müssen zwingend über beide Kerne aufgeteilt werden. Spezialisierte Community-Frameworks wie `Ollama37` [12] sind in der Lage, diese Modelle über die dedizierten Speicherpools zu verwalten und bei Bedarf zum CPU-RAM auszulagern. Die Inter-GPU-Kommunikation zur Synchronisation erfolgt jedoch ausschlieSSlich über den PCIe 3.0-Bus [5, 13]. Im Gegensatz zu modernen Architekturen mit Hochgeschwindigkeits-Interconnects (NVLink), wird die sequenzielle Generierung von Tokens bei gesplitteten Modellen durch die im Vergleich geringe PCIe 3.0 Bandbreite limitiert, was die Inferenzrate zusätzlich senkt. Für Single-GPU-Inferenz (Modelle $\leq$ 12 GB) ist der PCIe-Bus nach dem initialen Laden des Modells typischerweise nicht der Hauptengpass [14, 5].

3 Methodik und Experimentelle Basis

Dieses Kapitel beschreibt die Grundlage der Performance-Analyse, um die Reproduzierbarkeit der ermittelten Tokens-pro-Sekunde (T/s)-Raten zu gewährleisten. Die Bewertung stützt sich auf empirische Benchmarks, die mit Legacy-Software-Stacks durchgeführt wurden, da moderne, offizielle Toolchains keine Unterstützung für die Kepler-Architektur (CC 3.7) mehr bieten.

3.1 Benchmark-Umgebung und LLM

Die Performance-Analyse stützt sich auf zwei unterschiedliche Hardware-Konfigurationen, wobei das LLM `gemma3:12b` unter `Ollama` und `Open-WebUI` als konsistente Testbasis diente.

- **K80 Node (Legacy-Setup):** Vier NVIDIA Tesla K80-Karten (insgesamt 8 GPUs mit 8 x 12 GB VRAM). Die Software-Basis bildete `Ubuntu 20.04 LTS` in Verbindung mit einem Docker-Container, der spezifisch für die K80-Architektur und Legacy-CUDA kompiliert wurde [25]. Die Multi-GPU-Verwaltung für das 12B-Modell (gröSSer als 12 GB) erfolgte durch Model Splitting über die 8 GK210-Kerne des `Ollama37`-Forks [12].
- **RTX Node (Budget-Performance-Setup):** Zwei NVIDIA Consumer-Karten (RTX 3060 12 GB und GTX 1060 6 GB).

Die Benchmark-Messungen auf dem K80 Node wurden unter Verwendung des notwendigen Legacy-Software-Stacks (CUDA Toolkit 11.4 in Verbindung mit GCC 10) durchgeführt [1].

3.2 Quantifizierung des Token-Durchsatzes

Die sequenzielle Latenz wurde durch die Messung der durchschnittlichen Tokens-pro-Sekunde während der Generierungsphase (Response Rate) und der Dauer der Prompt-Verarbeitung (Prompt Rate) ermittelt. Die detaillierten Messdaten für einen einzelnen Durchlauf sind in Tabelle 1 aufgeführt.

Tabelle 1: Detaillierter Benchmark-Vergleich: Gemma3:12b Inferenz (Eigene Messung)

Metrik	**K80 Node (4x K80 / 8x 12 GB)**	**RTX Node (3060 12 GB + 1060 6**
Response Token/s	4.17	30.09
Prompt Token/s	4.29	33.56
Gesamtdauer (Approx.)	0h2m1s	0h0m48s
Load Duration (Ladezeit)	26.62 s	33.56 s
Completion Tokens	376	440

Diese empirischen Daten zeigen, dass der K80 Node eine um ca. 86% geringere Response-Rate (4.17 T/s vs. 30.09 T/s) liefert als ein modernes Dual-RTX-Budget-Setup. Die Geschwindigkeit der K80 (4.17 T/s) liegt sogar noch unter den pessimistischen Schätzungen anderer Community-Benchmarks ($5.5 - 6$ T/s für ein 7B-Modell [1]), was die Limitierungen der CC 3.7-Architektur unter realen Multi-GPU-Bedingungen unterstreicht [26].

4 Ergebnisse und Ökonomische Analyse

4.1 Architektonische Performance-Engpässe

Der kritischste Performance-Engpass entsteht durch das Fehlen der nativen `__dp4a`-Instruktion (per-byte integer dot product). Diese Instruktion ist für die Hardware-Beschleunigung moderner 4-bit-Quantisierungen (GGUF k-quants) essenziell und erst ab der Pascal-Architektur (CC 6.1) verfügbar [3, 4]. Da die K80 (CC 3.7) diese Funktion nicht unterstützt, muss die Quantisierungsberechnung in Frameworks wie `llama.cpp` auf langsamere Software-Workarounds zurückgreifen [3]. Dies ist die kausale Erklärung für die geringe Inferenzgeschwindigkeit trotz aggressivem Quantisierungsgrad.

4.2 Physische Komplexität und Proprietäre Risiken

Der Einsatz der passiv gekühlten K80 [19] in Workstations erfordert zwingend einen DIY-Umbau zur aktiven Kühlung [20], was mechanisches Geschick

voraussetzt. Ein weiteres signifikantes Risiko liegt in der proprietären 8-Pin-Stromversorgung der K80 [21]. Obwohl der Einsatz eines speziell beschafften Adapters das Risiko eines Kurzschlusses (Bricking) eliminiert, muss dem Nutzer die inhärente Gefahr bewusst sein, die bei der Verwendung von nicht-spezialisierten, handelsüblichen Consumer-Adaptern besteht [21, 22].

4.3 Ökonomischer Vergleich (TCO)

Obwohl die K80 auf dem deutschen Gebrauchtmarkt Anschaffungskosten von mindestens 89 aufweist (im Gegensatz zu den geringeren 20–50 USD in US-Quellen [2, 24]), übersteigt der kumulierte Zeitaufwand für die Konfiguration (Legacy-Software, DIY-Kühlung, Pinout-Sicherheit) die initiale Kostenersparnis bei weitem. Die anfänglichen Hardware-Kosten werden durch die **erforderliche Investition an Expertenzeit** für die Installation, das Debugging der Legacy-Toolchain und die Lösung thermischer/proprietärer Probleme konterkariert. Diese "Wartungsschuldïst ein schwer zu quantifizierender, aber hoher TCO-Faktor.

Tabelle 2: Vergleich K80 mit relevanten Budget-Alternativen (Stand 2025)

Hardware/Modell	**CC**	**VRAM (GB)**	**T/s (7B Q4 / Gemma)**	**T**
Tesla K80 (DIY Node)	3.7 (Kepler)	24 (2x 12, Splitted)	4.17 (Gemma3:12b)	Extrem hoc
RTX 3060/1060 Node	6.1 / 8.6	18 (Unified/Split)	30.09 (Gemma3:12b)	Mode
NVIDIA P102-100 (Mining)	6.1 (Pascal)	10 (Unified)	≈ 10 (erwartet) [23]	Modera
RTX 4060 Ti 16GB	8.9 (Ada)	16 (Unified)	> 30 (erwartet) [16]	Sehr ni

Budget-Alternativen wie Pascal-basierte Mining-Karten (P10x, CC 6.1) bieten für einen geringfügig höheren Preis [23] eine native Unterstützung für `__dp4a`, was zu einer deutlich besseren Inferenzleistung und einer vereinfachten Software-Wartung führt [3].

5 Fazit und Empfehlung

Die Nutzung der NVIDIA Tesla K80 als LLM-Inferenz-Node ist technisch möglich, jedoch mit erheblichen architektonischen und softwaretechnischen Kompromissen verbunden. Die notwendigen Workarounds zur Umgehung der CC 3.7-Dekompatibilität und der Mangel an Hardware-Beschleunigung für moderne Quantisierungen resultieren in einer unzeitgemäSS geringen T/s-Rate.

Für den technisch versierten Anwender, dessen Ziel primär im Lernen und Experimentieren mit minimalen Hardwarekosten liegt, bietet die K80 ein herausforderndes und lehrreiches Projekt. Für produktive oder latenzkritische Anwendungen ist der Node aufgrund des ungünstigen Verhältnisses von Leistung pro Installationsaufwand (*Performance-per-Setup-Effort*) und der hohen *Software-Debt* nicht zu empfehlen. Wirtschaftlich überlegene Alternativen, die eine höhere Compute Capability (CC ≥ 6.1) und damit native DP4A-Unterstützung bieten,

sind vorzuziehen. Diese Erkenntnisse unterstreichen die allgemeine Bedeutung der **Compute Capability $\geq$ 6.1** als Mindestanforderung für die effiziente LLM-Inferenz auf preisgünstiger Hardware.

Danksagung

Diese Arbeit basiert auf der Analyse von Community-Berichten, technischen Dokumentationen und aktuellen Benchmark-Ergebnissen im Bereich der lokalen LLM-Inferenz. Die Unterstützung bei der Recherche, Strukturierung und Formatierung dieser Analyse wurde durch die Nutzung der Gemmi KI (AI Research Assistant) ermöglicht.

Literatur

[1] K80 Performance Benchmark (5.5-6 T/s mit CUDA 11.4 / CC 3.7). https://github.com/ggml-org/llama.cpp/issues/12140.

[2] K80 Gebrauchtpreis (20-50 USD). https://www.ebay.com/b/NVIDIA-NVIDIA-Computer-Graphics-Cards-NVIDIA-Tesla-K80/27386/bn_7116861813.

[3] Die __dp4a-Instruktion ist für 4-bit-Quantisierungen notwendig und fehlt bei CC ¡ 6.1. https://www.reddit.com/r/LocalLLaMA/comments/1dst8zp/llamacpp_owners_of_old_gpus_wanted_for/.

[4] DP4A-Instruktion ist erst ab CC 6.1 (Pascal) verfügbar. https://www.reddit.com/r/LocalLLaMA/comments/1dst8zp/llamacpp_owners_of_old_gpus_wanted_for/.

[5] PCIe 3.0 als Engpass bei Inter-GPU Kommunikation (kein NVLink). https://www.glukhov.org/post/2025/06/llm-performance-and-pci-lanes/.

[6] Herausforderungen beim lokalen Deployment von LLMs und Kostendruck (GDPR-Konformität). https://arxiv.org/html/2407.12797v1.

[7] K80 Gesamtspezifikationen (4992 CUDA Kerne, 480 GB/s aggregierte Bandbreite). https://www.nvidia.com/en-gb/data-center/tesla-k80/.

[8] 24 GB VRAM ist geteilt auf zwei 12 GB GK210 GPUs. https://forums.developer.nvidia.com/t/run-llm-in-k80/260624.

[9] Dual-GPU-Design der K80 mit 2x 12 GB VRAM. https://forums.developer.nvidia.com/t/run-llm-in-k80/260624.

[10] K80 Compute Capability 3.7. https://developer.nvidia.com/cuda-gpus.

[11] Kepler-Architektur (CC 3.0 und 3.7) in CUDA 12.0 als deprecated markiert. https://forums.developer.nvidia.com/t/k80-is-it-possible-to-still-use-these-cards/291659.

[12] Ollama CC 3.7 Unterstützung via Community Fork Ollama37. https://hub.docker.com/r/dogkeeper886/ollama37.

[13] PCIe 3.0 als Engpass bei Inter-GPU Kommunikation (kein NVLink). https://www.glukhov.org/post/2025/06/llm-performance-and-pci-lanes/.

[14] PCIe Engpass nur beim Laden des Modells, nicht bei Single-GPU Inferenz. https://www.reddit.com/r/LocalLLaMA/comments/1bhfjd3/quick_experiment_how_is_inference_affected_with/.

[15] K80 GK210-Kern Spezifikationen und Bandbreite 240.6 GB/s. https://www.techpowerup.com/gpu-specs/tesla-k80.c2616.

[16] RTX 4060 Ti 16GB als moderne Alternative (448 GB/s Bandbreite). https://www.reddit.com/r/LocalLLaMA/comments/1n89ryn/most_affordable_ai_computer_with_gpu_gputer_you/?tl=de.

[17] Moderne T/s Raten (> 190 T/s). https://www.researchgate.net/publication/395696223_Performance_benchmarking_of_free_LLMs_A_practical_analysis_of_token_processing_efficiency.

[18] LLM inference systems. https://arxiv.org/html/2506.21901.

[19] K80 Passive Kühlung. https://esologic.com/tesla-cooler/.

[20] DIY-Umbau zur aktiven Kühlung. https://www.reddit.com/r/pcmods/comments/nhfwh7/guide_using_an-nvidia-tesla-k80-datacenter-gpu/.

[21] Proprietärer K80 Power Connector. https://forums.developer.nvidia.com/t/k80-power-connector/119765.

[22] Proprietäre 8-Pin-Stromversorgung und Risiko des Bricking. https://forums.developer.nvidia.com/t/k80-power-connector/119765.

[23] Pascal-basierte Mining-Karten als Budget-Alternative (P102/P104). https://www.reddit.com/r/ollama/comments/1h12azn/is_nvidia-tesla-k80-24gb-good-for-running-llama32/.

[24] Horn, P. (2025). NVIDIA Tesla K80 GPU LLM Node. Blogbeitrag. https://philipp-horn.dev/2025/09/30/nvidia-tesla-k80-gpu-llm-node/.

[25] Docker Compose Konfiguration für Ollama auf Tesla K80 (Ubuntu 20.04 LTS). https://git.4noobs.de/h3rb3rn/ollama/-/blob/main/ollama-tesla-k80/docker-compose.yml.

[26] LLM GPU Benchmark - Selbstgehostete App. https://llm-gpu-benchmark.self-hosted.app/.