Arch Linux Üzerinde Dağıtık Büyük Dil Modeli (LLM) Altyapısı: Hibrit C++/Rust Mimarisi, Ağ Performans Analizi ve Prima.cpp Entegrasyonu Üzerine Kapsamlı Araştırma Raporu

1. Yönetici Özeti ve Giriş

Yapay Zeka (YZ) araştırmalarındaki son gelişmeler, Büyük Dil Modellerinin (LLM) parametre sayısında ve yeteneklerinde üstel bir artışa tanık olmuştur. Llama-3-70B, Qwen-2.5-72B ve DeepSeek-R1 gibi modeller, akıl yürütme ve kodlama görevlerinde bulut tabanlı tescilli modellerle rekabet eder hale gelmiştir. Ancak, bu modellerin donanım gereksinimleri—özellikle VRAM (Video Random Access Memory) kapasitesi ve bellek bant genişliği—tekil tüketici sınıfı donanımların (örneğin tek bir NVIDIA RTX 4090) kapasitesini aşmaktadır. Bu durum, "Ev Laboratuvarı" (Home Lab) araştırmacılarını ve KOBİ ölçeğindeki geliştiricileri, hesaplama yükünü birden fazla cihaza yayan **Dağıtık Çıkarım (Distributed Inference)** mimarilerine yöneltmiştir.

Bu rapor, Arch Linux işletim sistemi üzerinde yüksek performanslı, dağıtık bir LLM çıkarım kümesinin tasarımını, teorik temellerini ve pratik uygulamasını derinlemesine incelemektedir. Raporun odak noktası, özellikle 1Gbps gibi sınırlı bant genişliğine sahip ağlarda bile verimli çalışabilen **prima.cpp** çerçevesinin analizi ve bu çerçevenin **C++ (Worker/İşçi)** ve **Rust (Server/Sunucu)** dilleri kullanılarak oluşturulan hibrit bir mimari ile nasıl optimize edileceğidir.

Arch Linux, "Rolling Release" (Sürekli Güncel) yapısı, en yeni derleyici setlerine (GCC, Clang) erişim imkanı ve AUR (Arch User Repository) üzerinden sağladığı geniş kütüphane desteği ile bu tür deneysel ve yüksek performanslı hesaplama (HPC) yükleri için ideal bir zemin sunmaktadır.[1] Bu çalışmada, Arch Linux'un çekirdek seviyesindeki ağ optimizasyonlarından, Rust dilinin asenkron çalışma zamanı (runtime) avantajlarına kadar geniş bir spektrumda teknik analizler sunulacaktır.

Analizimiz, 10Gbps ağların ham veri transferinde tartışmasız üstünlüğünü doğrulamakla birlikte, prima.cpp tarafından sunulan **Boru Hattı-Halka Paralelliği (Piped-Ring Parallelism - PRP)** ve **Halda Zamanlayıcısı** algoritmalarının, 1Gbps ağlarda dahi gecikmeyi (latency) hesaplama süresi (compute time) ile örtüştürerek nasıl "gizlediğini" ve bu sayede düşük maliyetli donanımlarla 70B sınıfı modellerin çalıştırılabilmesini mümkün kıldığını ortaya koymaktadır.[2]

2. Dağıtık Çıkarım Mimarilerinin Teorik Temelleri

Bir LLM'i birden fazla cihaza yaymak, basit bir veri kopyalama işlemi değildir. Bu, Transformer mimarisinin matematiksel doğasından kaynaklanan karmaşık senkronizasyon ve iletişim problemlerini beraberinde getirir.

2.1 Bellek Duvarı (The Memory Wall) ve Bant Genişliği Kısıtı

Modern LLM çıkarımındaki temel darboğaz, işlemci hızı (FLOPS) değil, bellek bant genişliğidir. Bir modelin her bir token (kelime parçası) üretebilmesi için, modelin tüm aktif ağırlıklarının (weights) bellekten işlemci çekirdeklerine taşınması gerekir.

Matematiksel olarak, 70 Milyar (70B) parametreli bir modelin FP16 (16-bit kayan nokta) hassasiyetindeki boyutu yaklaşık 140 GB'dır. 4-bit kuantsal (Q4_K_M) sıkıştırma ile bu boyut yaklaşık 40-42 GB seviyesine iner.[4]

- **Tekil Cihaz Sorunu:** Tüketici sınıfı en güçlü kart olan RTX 4090, 24 GB VRAM'e sahiptir. 42 GB'lık bir model bu belleğe sığmaz.
- **Sistem RAM'ine Taşma:** Modelin VRAM'e sığmayan kısmı sistem RAM'ine (DDR4/DDR5) taşındığında, veri yolu hızı 1000 GB/s'den (GDDR6X) 50-100 GB/s (DDR5) seviyesine düşer. Bu, çıkarım hızını 10-20 kat yavaşlatır.

Dağıtık çıkarım, bu sorunu birden fazla GPU'nun VRAM'ini birleştirerek çözer (örneğin 2x RTX 3090/4090 = 48 GB VRAM). Ancak bu durumda darboğaz, cihazlar arası iletişim hattına, yani ethernet ağına kayar.

2.2 Paralellik Stratejileri ve Ağ Etkileşimi

Dağıtık sistemlerde kullanılan paralellik yöntemleri, ağ altypusının performansını doğrudan belirler [6]:

2.2.1 Tensör Paralelliği (Tensor Parallelism - TP)

Bu yöntemde, her bir matris çarpımı işlemi cihazlara bölünür.

- **Mekanizma:** Bir katmandaki W ağırlık matrisi, W_1 ve W_2 olarak ikiye bölünür. Cihaz A W_1 ile, Cihaz B W_2 ile işlem yapar. Sonuçların birleştirilmesi (All-Reduce) için her katmanda birden fazla kez senkronizasyon gerekir.
- **Ağ Gereksinimi:** Son derece düşük gecikme ve devasa bant genişliği (NVLink, 600-900 GB/s).
- **Ethernet Uygunluğu: Yok.** 1Gbps veya 10Gbps ethernet üzerinde TP çalıştırılamaz; iletişim gecikmesi hesaplamadan kat kat uzun sürer.

2.2.2 Boru Hattı Paralelliği (Pipeline Parallelism - PP)

Bu yöntemde model katmanlar bazında bölünür. 80 katmanlı bir Llama-3-70B modeli için; Cihaz A ilk 40 katmanı, Cihaz B son 40 katmanı alır.

- **Mekanizma:** Cihaz A işlemeyi bitirince, ara katman çıktısını (aktivasyon tensörü) Cihaz B'ye gönderir.
- **Ağ Gereksinimi:** Orta seviye. Sadece katman geçişlerinde veri transferi olur.
- **Dezavantaj (The Bubble):** Cihaz B çalışırken Cihaz A boştur. Donanım verimliliği düşüktür.

2.2.3 Prima.cpp'nin Yeniliği: Boru Hattı-Halka Paralelliği (Piped-Ring Parallelism - PRP)

prima.cpp, standart PP'nin verimsizliğini ve ağ gecikmesini aşmak için geliştirilmiştir. Cihazları bir halka (ring) topolojisinde bağlar ve verileri küçük "parçalar" (chunks) halinde işler.[2]

- **Örtüştürme (Overlapping):** Cihaz A, N. token'ı hesaplarken, aynı anda $N-1$. token'ın verisini Cihaz B'ye gönderir ve Cihaz B'den $N-2$. token'ın geri bildirimini alabilir (eğer gerekliyse).
- **Önceden Getirme (Prefetching):** prima.cpp, diskten veya ağdan gelecek veriyi, hesaplama bitmeden talep eder. Bu, özellikle yavaş ağlarda (1Gbps) iletişimin hesaplama süresi içinde "saklanmasını" sağlar.

---

3. Ağ Performans Analizi: 1Gbps vs 10Gbps

Kullanıcı sorgusunun kritik bir bileşeni, ağ altyapısının seçimidir. 1Gbps (standart ev ağı) ile 10Gbps (SFP+ veya 10GBASE-T) arasındaki fark, dağıtık çıkarımda "kullanılabilirlik" ile "yüksek performans" arasındaki farkı belirler.

3.1 Bant Genişliği Matematiği ve Transfer Boyutları

70B parametreli bir modelde, cihazlar arasında transfer edilen verinin boyutu, modelin "Hidden Size" (Gizli Katman Boyutu) parametresine bağlıdır. Llama-3-70B için bu değer $d_{model} = 8192$'dir. Veriler genellikle FP16 (2 byte) formatında taşınır.[4]

Bir token transferi için veri boyutu:

$$\text{Boyut} = \text{Batch Size} \times d_{model} \times 2 \text{ byte}$$

Senaryo	Batch Size	Veri Boyutu	1Gbps Süresi (Teorik)	10Gbps Süresi (Teorik)

Tekil Token (Chat)	1	~16 KB	~0.12 ms	~0.012 ms
Prompt İşleme (Prefill)	512	~8 MB	~64 ms	~6.4 ms
Prompt İşleme (Prefill)	2048	~32 MB	~256 ms	~25.6 ms

3.2 1Gbps Ağların Darboğaz Analizi

Tablodan görüleceği üzere, tekil token üretiminde (sohbet sırasında cevap yazarken), 1Gbps ağın getirdiği ~0.12 ms gecikme ihmal edilebilir düzeydedir. İnsan algısı bu gecikmeyi fark edemez. Ancak sorunlar şurada başlar:

1. **Prompt İşleme (Prefill Phase):** Kullanıcı uzun bir metin gönderdiğinde (örneğin bir makale özeti istediğinde), sistem yüzlerce token'ı aynı anda işlemelidir. 2048 token'lık bir prompt için 1Gbps ağda her katman geçişinde 256ms gecikme yaşanır. Model 4 cihaza bölünmüşse, bu gecikme kümülatif olarak saniyeler sürebilir.
2. **Ağırlık Yükleme (Weight Loading):** Eğer prima.cpp'nin disk offloading özelliği kullanılıyorsa, VRAM dolduğunda ağırlıklar RAM'den veya Diskten çekilir. 1Gbps ağ, saniyede maksimum 110-120 MB veri taşıyabilir. Bu, modern NVMe disklerin (3000-7000 MB/s) hızının %5'i bile değildir. Bu senaryoda sistem kullanılamaz hale gelir.

3.3 10Gbps Ağların Avantajı ve Maliyet/Performans

10Gbps ağ, bant genişliğini 1.2 GB/s seviyesine çıkarır. Bu, prompt işleme süresini 10 kat hızlandırır ve ağırlık transferinde PCIe 3.0 x1 hızlarına yaklaşır.[9]

Arch Linux Üzerinde Ağ Optimizasyonu:
Hangi ağ kartı kullanılırsa kullanılsın, Linux çekirdeğinin varsayılan TCP parametreleri bu tür yüksek hacimli, düşük gecikmeli trafik için optimize edilmemiştir. Aşağıdaki ayarlar /etc/sysctl.d/99-llm-net.conf dosyasına eklenerek performans artırılmalıdır 10:

Bash

```
# TCP Tampon Belleklerini Artırma (10Gbps için kritik)
net.core.rmem_max = 16777216
```

```
net.core.wmem_max = 16777216
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216

# TCP Pencere Ölçeklendirmesi
net.ipv4.tcp_window_scaling = 1

# BBR Tıkanıklık Kontrolü (Düşük gecikme için)
net.core.default_qdisc = fq
net.ipv4.tcp_congestion_control = bbr
```

3.4 Karşılaştırmalı Karar: Prima.cpp vs Llama.cpp RPC

- **Llama.cpp RPC (Remote Procedure Call):** Senkrondur. Veri gönderilirken işlem durur. 1Gbps ağda, özellikle prompt işleme sırasında ciddi performans kaybı yaşanır. "Bekleme" süreleri toplam sürenin %30-40'ına ulaşabilir.[11]
- **Prima.cpp:** PRP (Piped-Ring) mimarisi sayesinde, veri transferi hesaplama ile örtüştürülür. 1Gbps ağda dahi, eğer hesaplama (GPU/CPU işlemi) veri transferinden uzun sürüyorsa, ağ gecikmesi "sıfıra" indirgenmiş gibi görünür. Bu nedenle **1Gbps ağlarda prima.cpp mutlak en iyi performansı sunan çözümdür**.[2]

---

4. Yazılım Mimarisi: Prima.cpp ve Rakipleri

Dağıtık çıkarım ekosisteminde birkaç ana oyuncu bulunmaktadır. Arch Linux kullanıcısı için en uygun çözümün analizi aşağıdadır.

4.1 Prima.cpp: Heterojen Kümeler İçin Kral

prima.cpp, llama.cpp'nin bir çatalı (fork) olarak başlamış ancak dağıtık sistemler için çekirdeği yeniden yazılmıştır. En önemli iki özelliği:

1. **Halda Zamanlayıcısı (Halda Scheduler):** Heterojen cihazları (örneğin güçlü bir Masaüstü + zayıf bir Laptop) yönetmek için geliştirilmiş bir algoritmadır. Sistemdeki her cihazın hesaplama gücünü (TFLOPS), bellek bant genişliğini ve ağ hızını ölçer. Ardından, en yavaş cihazın sistemi tıkamasını önlemek için katmanları optimize edilmiş şekilde dağıtır.[2]
    - Matematiksel olarak, bir Lineer Programlama (LP) problemi çözer. Bu nedenle HiGHS kütüphanesine bağımlıdır.
2. **Disk Offloading:** VRAM ve RAM yetersiz kaldığında, model ağırlıklarını diskten (mmap ile) anlık olarak okuyarak çalışmaya devam eder. Bu, performans düşürür ancak "Out of Memory" (OOM) hatası almadan devasa modelleri çalıştırmayı sağlar.

4.2 Llama.cpp RPC

Standart llama.cpp içindeki RPC backend'i, kurulumu en kolay olanıdır ancak zekadan yoksundur. Katmanları manuel olarak -ngl (number of gpu layers) parametresiyle bölmeniz gerekir. Hangi cihazın ne kadar yük alacağını elle hesaplamanız gerekir. Heterojen kümelerde (hızlı ve yavaş cihaz karışımı) en yavaş cihaza endekslenir.[11]

4.3 Diğerleri (Exo, vLLM)

- **Exo:** Python tabanlıdır, kurulumu kolaydır ancak C++ tabanlı prima.cpp kadar düşük seviye bellek yönetimi sunmaz.[2]
- **vLLM:** Veri merkezi odaklıdır. Tüketici donanımlarında kurulumu zordur ve genellikle dağıtık yapı için Ray kümesi gerektirir, bu da ev kullanıcısı için aşırı karmaşıktır.[7]

---

5. Hibrit Mimari Tasarımı: C++ Worker & Rust Server

Kullanıcı talebi doğrultusunda, sistemin hesaplama gücünü C++ ile, yönetim ve orkestrasyon katmanını Rust ile tasarlıyoruz. Bu, modern yüksek frekanslı ticaret sistemlerinde kullanılan "Hız için C++, Güvenlik için Rust" paradigmasının bir yansımasıdır.

5.1 Neden Hibrit?

- **C++ (Worker):** Tensör işlemleri (GEMM), CUDA çekirdekleri ve bellek manipülasyonu (pointer aritmetiği) için C++ vazgeçilmezdir. ggml kütüphanesi saf C/C++ ile yazılmıştır ve donanımla en düşük seviyede konuşur.
- **Rust (Server):** Dağıtık bir sistemi yönetmek (hataları yakalamak, süreçleri yeniden başlatmak, HTTP API sunmak) yüksek güvenilirlik gerektirir. Rust'ın "Memory Safety" (Bellek Güvenliği) ve Tokio kütüphanesinin sunduğu asenkron G/Ç (Async I/O) yetenekleri, binlerce isteği aynı anda yönetirken sunucunun çökmemesini garanti eder.[13]

5.2 Mimari Diyagramı (Kavramsal)

| (HTTP / JSON)
v
|-- 1. İstek Doğrulama & Kuyruklama (Axum/Tower)
|-- 2. Yük Dengeleme (Load Balancing)
|-- 3. Süreç Yönetimi (Tokio Command / SSH2)
|
| (TCP Kontrol Sinyalleri / ZeroMQ)
v
|-- Node A (Master) <===> Node B (Worker) <===> Node C (Worker)
(Piped-Ring Paralellik Veri Hattı)

5.3 Rust Sunucu İmplementasyon Detayları

Rust sunucusu, prima.cpp süreçlerini başlatmaktan ve izlemekten sorumlu olacaktır. prima.cpp

kendi içinde bir HTTP sunucusu (llama-server) barındırsa da, Rust katmanı şunları ekler:

1. **Güvenlik:** API anahtarı yönetimi, Rate Limiting (Hız sınırlama).
2. **Otomatik İyileştirme:** Bir worker çökerse (Segfault), Rust sunucusu bunu algılayıp SSH üzerinden yeniden başlatabilir.
3. **Çoklu Model Yönetimi:** İsteye göre farklı modelleri (örneğin Kodlama için Qwen, Sohbet için Llama-3) dinamik olarak yükleyip boşaltabilir.

---

6. Kurulum Rehberi: Arch Linux Üzerinde Adım Adım

Bu bölüm, Arch Linux üzerinde prima.cpp kümesinin ve Rust sunucusunun kurulumunu içerir.

6.1 Ön Hazırlıklar ve Bağımlılıklar (Tüm Cihazlar)

Arch Linux'un güncel yapısı avantajdır. Tüm düğümlerde (Nodes) aşağıdaki adımları uygulayın.

1. Sistem Güncelleme ve Temel Araçlar:

Bash

```
sudo pacman -Syu
sudo pacman -S base-devel git cmake gcc gdb neofetch htop
```

2. Sürücü Kurulumu (Örnek: NVIDIA):
CUDA desteği için NVIDIA sürücüleri ve CUDA toolkit şarttır.

Bash

```
sudo pacman -S nvidia nvidia-utils cuda
```

Not: AMD GPU kullanıyorsanız rocm-hip-sdk paketini kurmalısınız.

3. ZeroMQ Kurulumu:
prima.cpp iletişim için ZeroMQ kullanır.

Bash

```
sudo pacman -S zeromq
```

Kurulumu doğrulamak için ls /usr/include/zmq.h komutunu kullanın.[15]

4. FIO ve HiGHS Kurulumu:
Halda zamanlayıcısı için gereklidir. HiGHS paketi AUR üzerindedir, bu yüzden bir AUR yardımcısı (yay veya paru) kullanacağız.16

Bash

```
# AUR yardımcısı yoksa manuel kurulum
git clone https://aur.archlinux.org/yay.git
cd yay
makepkg -si

# HiGHS ve FIO kurulumu
yay -S highs fio
```

6.2 Prima.cpp Derleme (Gitee Üzerinden)

Orijinal GitHub deposu erişilemez durumda olduğu için [18], resmi yedek olan Gitee deposunu kullanacağız.

Bash

```
# Depoyu klonla
git clone https://gitee.com/zonghang-li/prima.cpp.git
cd prima.cpp

# Derleme
# USE_HIGHS=1 parametresi Halda zamanlayıcısını aktif eder.
# NVIDIA GPU için GGML_CUDA=1 eklenmelidir.
make USE_HIGHS=1 GGML_CUDA=1 -j$(nproc)
```

Derleme sonucunda llama-cli ve llama-server dosyaları oluşacaktır.

6.3 Rust Sunucusunun Geliştirilmesi ve Kurulumu

Bu adım, hibrit yapının "Server" tarafını oluşturur. Rust ve Cargo'nun kurulu olduğunu varsayıyoruz (sudo pacman -S rustup && rustup default stable).

Proje Oluşturma:

Bash

```
cargo new distributed_llm_server
cd distributed_llm_server
cargo add axum tokio serde serde_json reqwest ssh2 anyhow
```

Örnek Rust Kodu (src/main.rs):
Bu kod, basit bir HTTP sunucusu başlatır ve gelen istekleri yerel veya uzak prima.cpp kümesine yönlendirir. Aynı zamanda ssh2 kütüphanesi ile uzak sunucularda komut çalıştırma yeteneğine sahiptir.19

Rust

```
use axum::{routing::post, Router, Json};
use serde::{Deserialize, Serialize};
use std::process::Stdio;
use tokio::process::Command;

#
struct ChatReq {
    prompt: String,
}

#
struct ChatResp {
    response: String,
}

#[tokio::main]
async fn main() {
    println!("🚀 Rust Orchestrator Başlatılıyor...");
```

```
    // 1. Prima.cpp Master Node'unu Başlat (Arka planda)
    // Gerçek senaryoda bu süreç yönetimi daha karmaşık olmalıdır.
    // SSH üzerinden uzak node'ları başlatma mantığı buraya eklenebilir.

    // 2. HTTP Sunucusunu Başlat
    let app = Router::new().route("/chat", post(chat_handler));
    let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
    println!("📡 API Dinleniyor: 0.0.0.0:3000");
    axum::serve(listener, app).await.unwrap();
}

async fn chat_handler(Json(payload): Json<ChatReq>) -> Json<ChatResp> {
    // Burada Prima.cpp'nin kendi HTTP arayüzüne (llama-server) istek atılır
    // Veya ZeroMQ ile doğrudan konuşulabilir (İleri Seviye)

    let client = reqwest::Client::new();
    // Varsayım: Prima Master Node 8080 portunda çalışıyor
    let res = client.post("http://127.0.0.1:8080/completion")
        .json(&serde_json::json!({
            "prompt": payload.prompt,
            "n_predict": 128
        }))
        .send()
        .await;

    match res {
        Ok(r) => {
            let body: serde_json::Value = r.json().await.unwrap();
            Json(ChatResp { response: body["content"].to_string() })
        },
        Err(_) => Json(ChatResp { response: "Hata: LLM Kümesine Erişilemedi".to_string() })
    }
}
```

6.4 Kümenin Başlatılması (Orkestrasyon)

İki cihazlı bir küme varsayalım:

- **Node A (Master - 192.168.1.10):** Rust Server + Prima Rank 0
- **Node B (Worker - 192.168.1.11):** Prima Rank 1

Adım 1: Worker'ı Başlat (Node B)
Worker, Master'dan emir bekler. --next parametresi halkayı tamamlamak için Master'ın IP'sini

gösterir.21

Bash

```
./llama-cli \
  -m /modeller/Llama-3-70B-Q4.gguf \
  --world 2 \
  --rank 1 \
  --master 192.168.1.10 \
  --next 192.168.1.10 \
  --prefetch \
  --gpu-mem 24
```

Adım 2: Master'ı Başlat (Node A)
Master başlatıldığında, önce Halda algoritması ile ağı ve diskleri test eder, ardından iş bölümünü yapar.

Bash

```
./llama-cli \
  -m /modeller/Llama-3-70B-Q4.gguf \
  --world 2 \
  --rank 0 \
  --master 192.168.1.10 \
  --next 192.168.1.11 \
  --prefetch \
  -p "Arch Linux dağıtık sistemler için neden iyidir?" \
  --force # Prefetching'i zorla (1Gbps ağlar için kritik)
```

---

7. Sonuç ve Gelecek Öngörüleri

Arch Linux üzerinde kurulan bu hibrit sistem, modern yazılım mühendisliğinin uç noktalarını temsil etmektedir. C++'ın donanım üzerindeki mutlak hakimiyeti ile Rust'ın güvenli eşzamanlılık yeteneklerinin birleşimi, ev ve laboratuvar ortamlarında "Süper Bilgisayar" simülasyonunu mümkün kılmaktadır.

Kritik Bulgular:

1. **Ağ:** 1Gbps ağlarda prima.cpp kullanımı zorunludur. Piped-Ring mimarisi olmadan standart RPC yöntemleri kullanılamaz derecede yavaştır. 10Gbps ağ imkanı varsa, kurulum basitleşir ancak prima.cpp hala verimlilik avantajı sağlar.
2. **Yazılım:** GitHub yerine Gitee kullanımı ve AUR üzerinden bağımlılık yönetimi, Arch Linux ekosisteminin esnekliğini kanıtlamaktadır.
3. **Performans:** Test verilerine göre, 4 adet tüketici cihazı ile 70B bir model, saniyede 2-3 token hızında (okuma hızına yakın) çalıştırılabilir. Speculative Decoding (Tahmini Çözümleme) aktif edildiğinde bu hız 1.5-2 katına çıkabilir.[3]

Bu rapor, kullanıcıya sadece bir kurulum rehberi değil, aynı zamanda dağıtık yapay zeka sistemlerinin altında yatan mühendislik prensiplerini de sunmaktadır. Gelecekte, 100Gbps+ ThunderBolt bağlantıları ve özel NPU (Neural Processing Unit) entegrasyonları ile bu sistemlerin performansı katlanarak artacaktır.

Alıntılanan çalışmalar

1. AUR llama.cpp and paru question / AUR Issues, Discussion & PKGBUILD Requests / Arch Linux Forums, erişim tarihi Ocak 10, 2026, https://bbs.archlinux.org/viewtopic.php?id=311367
2. PRIMA.CPP: Speeding Up 70B-Scale LLM Inference on Low-Resource Everyday Home Clusters - arXiv, erişim tarihi Ocak 10, 2026, https://arxiv.org/html/2504.08791v1
3. Prima.cpp: Fast 30-70B LLM Inference on Heterogeneous and Low-Resource Home Clusters - arXiv, erişim tarihi Ocak 10, 2026, https://arxiv.org/html/2504.08791v2
4. Calculating GPU Requirements for Efficient LLAMA 3.1 70B Deployment on AWS Sagemaker - IBM TechXchange Community, erişim tarihi Ocak 10, 2026, https://community.ibm.com/community/user/blogs/arindam-dasgupta/2024/09/18/calculating-gpu-requirements-for-efficient-llama-3
5. Self-Hosting LLaMA 3.1 70B (or any ~70B LLM) Affordably | by Abhinand | Medium, erişim tarihi Ocak 10, 2026, https://abhinand05.medium.com/self-hosting-llama-3-1-70b-or-any-70b-llm-affordably-2bd323d72f8d
6. Distributed Llama - Distributed Inference of Large Language Models with Slow Synchronization over Ethernet - GitHub, erişim tarihi Ocak 10, 2026, https://raw.githubusercontent.com/b4rtaz/distributed-llama/main/report/report.pdf
7. Introduction to distributed inference with llm-d - Red Hat Developer, erişim tarihi Ocak 10, 2026, https://developers.redhat.com/articles/2025/11/21/introduction-distributed-inference-llm-d
8. Using llamacpp and RCP, managed to improve promt processing by 4x times (160 t/s to 680 t/s) and text generation by 2x times (12.67 t/s to 22.52 t/s) by changing

the device order including RPC. GLM 4.6 IQ4_XS multiGPU + RPC. : r/LocalLLaMA - Reddit, erişim tarihi Ocak 10, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1o96mwq/using_llamacpp_and_rcp_managed_to_improve_promt/
9. GPU Bandwidth for LLMs (text-generation-webui) and Utilizing Multiple Computers Over LAN : r/LocalLLaMA - Reddit, erişim tarihi Ocak 10, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1hvdicy/gpu_bandwidth_for_llms_textgenerationwebui_and/
10. What is the actual maximum throughput on Gigabit Ethernet? - CableFree, erişim tarihi Ocak 10, 2026, https://www.cablefree.net/maximum-throughput-gigabit-ethernet/
11. Llama.cpp now supports distributed inference across multiple machines. - Reddit, erişim tarihi Ocak 10, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1cyzi9e/llamacpp_now_supports_distributed_inference/
12. RPC-server llama.cpp benchmarks : r/LocalLLaMA - Reddit, erişim tarihi Ocak 10, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1pxai05/rpcserver_llamacpp_benchmarks/
13. So you think Rust be spared from the LLM takeover of programming? - Reddit, erişim tarihi Ocak 10, 2026, https://www.reddit.com/r/rust/comments/1nevvyn/so_you_think_rust_be_spared_from_the_llm_takeover/
14. I just rewrote llama.cpp server in Rust (most of it at least), and made it scalable - Reddit, erişim tarihi Ocak 10, 2026, https://www.reddit.com/r/rust/comments/1ml5ogd/i_just_rewrote_llamacpp_server_in_rust_most_of_it/
15. IPv6 Provider - How to Install ZeroMQ on Arch Linux - Self Host with IPv6rs, erişim tarihi Ocak 10, 2026, https://www.ipv6.rs/tutorial/Arch_Linux/ZeroMQ/
16. Running LLM with Distributed Inference using prima.cpp on K1 Cluster - BIT-BRICK, erişim tarihi Ocak 10, 2026, https://www.bit-brick.com/2025/05/08/running-llm-with-distributed-inference-using-prima-cpp-on-k1-cluster/
17. ERGO-Code/HiGHS: Linear optimization software - GitHub, erişim tarihi Ocak 10, 2026, https://github.com/ERGO-Code/HiGHS
18. This is the official account of prima.cpp - GitHub, erişim tarihi Ocak 10, 2026, https://github.com/fengwenjiao/Prima.cpp
19. ssh2 - Rust - Docs.rs, erişim tarihi Ocak 10, 2026, https://docs.rs/ssh2
20. How to build an SSH client using Rust - _CLOUD, erişim tarihi Ocak 10, 2026, https://blog.ediri.io/how-to-build-an-ssh-client-using-rust
21. eopsu/prima.cpp - Gitee, erişim tarihi Ocak 10, 2026, https://gitee.com/eopsu/prima.cpp
22. magic/prima.cpp - Gitee, erişim tarihi Ocak 10, 2026, https://gitee.com/magicor/prima.cpp