

# AI & LLM

Experiments, notes, opinions, and practical experience with AI systems, large language models, and local or hosted setups.

- [Bookmarks](#)
- [Prompt Reversal \(Reverse Prompt Engineering\)](#)
- [Five-in-One Amplifier \(Content Amplification via AI\)](#)
- [Red Team Technique \(Adversarial Self-Critique\)](#)
- [AI Projects through Prompting](#)
  - [Project Overview](#)
  - [? Week 01: The Personal Dashboard](#)
  - [? Week 02: The Centralized Event Hub](#)
  - [Best Practices for Working with AI Agents: A Verification-Driven Approach](#)
- [Blueprint Scaffolding \(Structured Prompt Planning\)](#)
- [Deep Researches](#)
  - [Comparison between OneNote and Bookstack, Confluence, Docmost and Nextcloud](#)
- [BlogPost Reformatting Prompt](#)
- [The AI Coding Loop: A Guide to Verification-Driven Development](#)
- [LLAMA.cpp RPC Test](#)
  - [llama.cpp RPC Multi-GPU Setup Guide \(Windows 11 & Linux Distributed Cluster\)](#)
  - [Some Output](#)
- [LLAMA.cpp and RAG Resources To Read](#)
- [Prompt Library](#)
  - [Context Extraction](#)

- [Summary Generators](#)
- [Document Generation](#)

# Bookmarks

## LLMs

---

### NVIDIA Nemotron

<https://github.com/NVIDIA-NeMo/Nemotron>

NVIDIA Nemotron Developer Repository Open and efficient models for agentic AI. Training recipes, deployment guides, and use-case examples for the Nemotron family

24 Mar 2026 NVIDIA Nemotron 3 Ultra Release <https://github.com/NVIDIA-NeMo/Nemotron/blob/main/usage-cookbook/Nemotron-3-Ultra-Base/README.md>

Es gibt auch ein Nemotron 3 Nano in 4B version welches nur 2.84GB groß ist.

<https://huggingface.co/nvidia/NVIDIA-Nemotron-3-Nano-4B-GGUF>

## Interessante Links und Videos zu KI- und LLM-Themen

---

### Daniel Miessler

<https://danielmiessler.com/>

Daniel Miessler schreibt regelmäßig über Themen rund um Informationssicherheit, künstliche Intelligenz, Denkmodelle und persönliche Wissenssysteme. Viele seiner Texte beschäftigen sich mit der praktischen Nutzung von KI im Alltag und in professionellen Kontexten.

### Fabric (GitHub)

<https://github.com/danielmiessler/Fabric>

Fabric ist ein Open-Source-Framework zur Erweiterung menschlicher Fähigkeiten mithilfe von KI. Es stellt ein modulares System bereit, um konkrete Probleme mit Hilfe spezialisierter, gemeinschaftlich entwickelter KI-Prompts zu lösen. Diese Prompts sind flexibel einsetzbar und nicht an eine bestimmte Plattform gebunden.

Der Fokus liegt weniger auf einem einzelnen Modell, sondern auf wiederverwendbaren Denk- und Arbeitsmustern für den praktischen Einsatz von LLMs.

# Prompt Reversal (Reverse Prompt Engineering)

## Reverse Prompt Engineering (RPE)

---

### Was es ist

- Anstatt einen Prompt über mehrere Durchläufe iterativ zu verfeinern, arbeitest du **rückwärts**:  
Du startest mit einem idealen Endergebnis und lässt die KI **den Prompt erzeugen**, der dieses Ergebnis in einem Schritt hervorgebracht hätte.
- Im Kern wird die ursprüngliche Konversation „reverse-engineered“, um einen wiederverwendbaren, optimierten Prompt zu erhalten.

### Warum das relevant ist

- **Effizienz:** Das übliche Hin und Her entfällt — hochwertige Ergebnisse entstehen mit einem einzigen Prompt.
- **Konsistenz:** Auf diese Weise gewonnene Prompts enthalten alle Feinjustierungen und eignen sich als verlässliche Vorlagen für die Wiederverwendung.
- **Lerneffekt:** Die Analyse rückwärts entwickelter Prompts zeigt, wie gute Prompts aufgebaut sind — und verbessert langfristig die eigenen Prompting-Fähigkeiten.

### Vorgehensweise — typischer Workflow

1. Du schreibst einen initialen Prompt ? erhältst eine Ausgabe.
2. Prompt oder Ergebnis werden iterativ angepasst (z. B. Format, Detailgrad, Struktur, Tonalität).
3. Sobald das finale Ergebnis zufriedenstellend ist, bittest du die KI um eine Rückwärtsanalyse, etwa:  
*„Schreibe den einen Prompt, der dieses Endergebnis in einem Durchgang erzeugt hätte.“*
4. Dieser Prompt kann anschließend ohne weiteren Iterationsaufwand wiederverwendet werden.

### Varianten und konzeptionelle Einordnung

- Häufig bezeichnet als **Reverse Prompt Engineering (RPE)**. ([Symbio6](#))
- Einsetzbar als:
  - **Macro-RPE**: Erstellen vollständiger Prompt-Templates auf Basis von Beispielausgaben
  - **Micro-RPE**: Extrahieren einzelner Schlüsselphrasen oder struktureller Elemente je nach Anwendungsfall. ([Medium](#))

## Empfohlene Lektüre und Ressourcen

- „**Reverse Prompt Engineering: The art of thinking backward**“ — Überblick über Konzept und Einsatzszenarien. ([tcworld magazine](#))
- **Medium-Artikel**: „Why Reverse Prompt Engineering is the Magic Key to Production-Ready Prompts“ — beschreibt Macro- und Micro-RPE inklusive Schritt-für-Schritt-Ansatz. ([Medium](#))
- **Detaillierter Leitfaden**: „Reverse Prompt Engineering with ChatGPT“ — praxisorientierte Anleitung mit Prompt-Vorlagen. ([Kanaries Docs](#))
- **Forschungsarbeit**: Prompt-Inversion unter Black-Box-Bedingungen. ([arXiv](#))

# Five-in-One Amplifier (Content Amplification via AI)

## Was es ist

Nimm ein hochwertiges „Pillar“-Inhaltselement (z. B. Foliensatz, Bericht, Präsentation, Webinar-Transkript) und nutze KI, um **automatisch mehrere abgeleitete Inhalte zu erzeugen** — jeweils zugeschnitten auf unterschiedliche Zwecke (interne Kommunikation, externes Marketing, Quizze, Zusammenfassungen usw.).

## Warum es so wirkungsvoll ist

- **Maximiert den ROI von Inhalten:** Statt Inhalte manuell weiterzuverarbeiten, kann KI schnell verschiedene Formate erzeugen.
- **Spart Zeit:** Was sonst Stunden dauern würde, ist in Minuten erledigt.
- **Abteilungsübergreifend nutzbar:** Inhalte, die ein Team erstellt (z. B. Produkt), können andere Bereiche (Marketing, Vertrieb, HR) direkt nutzen — ohne auf neue Materialien warten zu müssen.

## Typische Anwendungsfälle / Ergebnisse

Aus einer einzigen Quelle (z. B. Slides oder Transkript) lassen sich automatisch erzeugen:

- Interne Recap-Mails oder Zusammenfassungen für Teammitglieder oder Stakeholder, die das Event verpasst haben
- Interaktive Quizze oder Tests (z. B. 10 Multiple-Choice-Fragen) — geeignet für Training oder Evaluation
- Kundengerichtete Inhalte: Infografiken, Reports, Slide-Zusammenfassungen, Marketingtexte, Social-Media-Content
- Interne Wissensartefakte: FAQs, „Cheat Sheets“, kompakte Übersichtsleitfäden

## Zentrales Prinzip: „Guter Input = guter Output“

- Verwende **hochwertige, gut strukturierte Basisinhalte** („Pillar Content“) für die Weiterverarbeitung.

- Vermeide einfache oder „dünne“ Ausgangsmaterialien — KI verstärkt Fehler, Unschärfe und fehlende Struktur genauso zuverlässig wie Qualität.

## Wann es sinnvoll ist

- Wenn du originäre Inhalte hast, die datenreich oder klar strukturiert sind (Reports, Präsentationen, Transkripte)
- Wenn du Reichweite und Wiederverwendung über verschiedene Formate und Zielgruppen hinweg **maximieren** möchtest
- In bereichsübergreifenden oder Multi-Stakeholder-Umgebungen (Marketing, Vertrieb, HR, Training)

## Empfohlene Lektüre & Ressourcen

- Überblick zu Prompt-Engineering-Techniken inklusive Amplification- und Reuse-Patterns. ([Upaspro](#))
- Weiterer Kontext: Allgemeiner Prompt-Engineering-Guide, der erklärt, warum strukturierte Prompts und Wiederverwendung für moderne LLM-Workflows entscheidend sind. ([Lakera](#))

## Prompts

---

Doppelklicke in die Boxen, um den gesamten Prompt auf einmal auszuwählen und einfach zu kopieren.

### Exakter Reverse Prompt (Copy/Paste):

““

Du erstellst einen einzigen Prompt, der — wenn er an ein fortgeschrittenes Chat-Modell übergeben wird — die unten stehende finale Analyse exakt reproduziert.

Enthalten sein müssen: Modell-Empfehlung, empfohlene Temperatur, max\_tokens, ein Beispiel für die System-Nachricht sowie ein explizites Ausgabeformat.

Zu replizierendes Endergebnis: [HIER das finale Ergebnis einfügen]

Schreibe nun den einen Prompt (in einem Codeblock), der dieses Ergebnis in einem Durchgang erzeugt. Achte explizit auf Struktur, Überschriften und

Ausführlichkeit.

### Exakter Prompt für eine Demo:

“

System: Du bist ein präziser strategischer Analyst.

Tonalität: professionell, klar.

User: Analysiere die Geschäftsstrategie von Anthropic und gib eine SWOT-Analyse mit exakt folgender Struktur aus:

- Hauptüberschrift „SWOT: Anthropic“
- Vier Abschnitte: Strengths, Weaknesses, Opportunities, Threats
- Pro Abschnitt: genau 3 Bulletpoints; jeder Bullet: eine einzeilige Kernaussage + ein erklärender Satz mit Datenbezug
- Unter jedem Abschnitt: „Our Strategic Response:“ mit 1 konkreter Maßnahme, die sich auf den stärksten Punkt bezieht
- Ausgabe als Markdown-Überschriften und Bulletpoints  
Einfache Sprache verwenden. Umfang: 400–600 Wörter.

### Quiz:

“

Angehängt: [slide\_deck.pdf].

Erstelle ein 10-Fragen-Multiple-Choice-Quiz zu den Kernkonzepten aus dem Deck.

Für jede Frage: 4 Antwortoptionen, markiere die korrekte Antwort und füge eine einzeilige Begründung hinzu.

Fragen kurz halten (<20 Wörter).

### Internes Recap:

Erstelle eine interne Recap-E-Mail für Executives mit 3 Absätzen:  
5 wichtigste Erkenntnisse (als Bulletpoints), 2 nächste Schritte, 2 Verantwortliche und eine 30-Wörter-Elevator-Zusammenfassung.  
Tonalität: Executive, gut scannbar.

### Client-Infografik-Text:

“

Extrahiere aus [slide\_deck.pdf] 5 aussagekräftige Kennzahlen oder Headlines.  
Für jede: eine 6-Wörter-Headline + eine unterstützende Zeile mit 12–18 Wörtern  
sowie einen Vorschlag für ein visuelles Element (Icon/Diagrammtyp).

### Lebenslauf erstellen:

“

Passe meinen Lebenslauf (angehängt) an diese Stellenbeschreibung  
(angehängt) an.  
Hebe 3 Erfolge hervor, die auf die 3 wichtigsten Anforderungen der Rolle  
einzahlen, und erstelle einen 1-seitigen Lebenslaufentwurf.

### Perspektivwechsel: Hiring Manager

“

Handle nun als Hiring Manager für diese Rolle.  
Du hast 60 Sekunden, um diesen Lebenslauf zu scannen.  
Liste 5 sofortige Red Flags (Bulletpoints) auf und erkläre jeweils kurz, warum sie  
zur Ablehnung führen würden.  
Sei gnadenlos und knapp.

### Kritik in Verbesserungen überführen:

“

Basierend auf den obigen Red Flags:

Schreibe die 3 schwächsten Bulletpoints im Lebenslauf neu — konkret, quantifiziert und klar auf die Stellenbeschreibung ausgerichtet.

### Blueprint:

“

Ich betreibe einen Online-Kurs namens Workspace Academy.  
Erstelle zunächst eine Liste der Standardabschnitte eines professionellen Q4-Marketing-Briefs und gib für jeden Abschnitt einen Ein-Satz-Zweck an.  
Ergänze zudem eine empfohlene Erfolgskennzahl pro Abschnitt.  
(Den vollständigen Brief noch nicht schreiben.)

### Reduzieren & Fokussieren:

“

Wende das 80/20-Prinzip an:  
Behalte nur die essenziellen Abschnitte für eine 3-Mail-Sequenz, die sich an warme Leads richtet.  
Ersetze Metriken durch jeweils eine messbare KPI pro Abschnitt.

### Umsetzen:

“

Schreibe nun den vollständigen Brief ausschließlich mit den freigegebenen Abschnitten.  
Jede E-Mail: Betreffzeile (?8 Wörter), 3 Bulletpoints Inhalt, ein CTA.  
Wortlimit pro E-Mail: 80–110 Wörter.

### Praktischer Prompt zur Metadatenerfassung:

“

Gib beim Speichern dieses Templates ein JSON aus mit den Schlüsseln:  
title, prompt\_text, model, temperature, max\_tokens, expected\_word\_count,  
sample\_output (anhängen), date, tags.

# Red Team Technique (Adversarial Self-Critique)

## Was es ist

Eine zweistufige Methode:

1. Zuerst lässt du die KI **Inhalte erzeugen** (z. B. Lebenslauf, Business-Proposal, Cold-Outreach-Mail, Marketing-Entwurf).
2. Direkt im Anschluss bittest du die KI, eine **kritische Gegenrolle** einzunehmen — ein „Red Team“ — das das Ergebnis bewertet und auf Schwächen, Fehler, unrealistische Aussagen oder Risikobereiche hinweist.

## Warum das funktioniert

- Hilft dir, **reale Einwände, Schwachstellen oder Dealbreaker** frühzeitig zu erkennen, bevor du Inhalte echten Menschen präsentierst.
- Ermöglicht gezielte Nachbesserung im Vorfeld — Inhalte werden belastbarer, glaubwürdiger und weniger naiv.
- Macht **Bias, blinde Flecken, schwache Argumentation oder unüberzeugende Passagen** sichtbar, die man selbst leicht übersieht.

## Typische Workflows / Beispiele

- **Bewerbung:**
  - Schritt 1: Lebenslauf auf eine konkrete Stellenbeschreibung zuschneiden.
  - Schritt 2: KI als Hiring Manager einsetzen, um Red Flags zu markieren — hilft, Klarheit zu verbessern, Stärken zu schärfen und Fülltext zu entfernen.
- **Business-Proposal:**
  - Schritt 1: Vorschlag für CFO oder Management entwerfen.
  - Schritt 2: KI als CFO bewerten lassen, um finanzielle Risiken, schwache ROI-Argumente oder Begründungslücken aufzudecken.

- **Cold Outreach / Marketing-Mail:**

- Schritt 1: Outreach-Mail schreiben.
- Schritt 2: KI als gestressten Empfänger reagieren lassen — identifiziert spamartige, schwache oder unglaubwürdige Formulierungen zum Streichen oder Überarbeiten.

## Best Practices für „Red Teaming“

- Nutze **sehr konkrete Personas** für die Kritik  
(z. B. „risikoscheuer CTO mit Fokus auf Datensicherheit“ statt nur „Kritiker“).  
Je klarer Motivation und Perspektive, desto relevanter das Feedback.
- **Schließe den Kreis:** Bitte die KI nach der Kritik gezielt, die schwächsten Stellen auf Basis der eigenen Einwände zu verbessern.
- Setze diese Methode als **Qualitätsfilter** für Inhalte mit hohem Einsatz ein  
(Bewerbungen, Angebote, externe Kommunikation) oder um überzeugende und formale Texte zu stärken.

## Empfohlene Lektüre & Ressourcen

- Praxisnahe Übersicht, die Red Teaming als eine der zentralen fortgeschrittenen Prompting-Techniken aufführt.  
([Upaspro](#))  
([Five-in-One Amplifier \(Content Amplification via AI\)](#))
- Akademische Diskussion zur Notwendigkeit strukturierter und adversarialer Prüfung im Prompt-Design, um Fragilität zu reduzieren und Skalierbarkeit von LLMs zu verbessern.  
([SSRN](#))
- Allgemeine Prompt-Engineering-Ressourcen zu rollenbasiertem Prompting, Thought-Sequenzen und dem größeren Kontext — hilfreich in Kombination mit Red Teaming.  
([Medium](#))

# AI Projects through Prompting

Example Projects done through prompting only in Codex, Gemini CLI, v0 or OpenCode

# Project Overview

## ?? Curriculum Roadmap

The bootcamp is structured into 10 weekly milestones, focusing on full-stack development and AI integration.

- **Week 01: Personal Dashboard**

- **Project:** Personal Dashboard
- **Goal:** Build a link organizer with a database for a custom browser "new tab" page.

- **Week 02: Build an App with Live Data**

- **Project:** Events Dashboard
- **Goal:** Live data visualization using APIs and charts.

- **Week 03: Build an App with Users**

- **Project:** Shared Expense Tracker
- **Goal:** Implementing User Authentication (Sign up/Log in) and data ownership.

- **Week 04: Build a Real-Time App**

- **Project:** Live Chat Room
- **Goal:** Create interfaces that update live without refreshing.

- **Week 05: Build an AI-Powered App**

- **Project:** Ingredient Combiner
- **Goal:** Integrate AI capabilities into your own application.

- **Week 06: Build a CLI Tool**

- **Project:** Site Inspector CLI
- **Goal:** Build a terminal tool that inspects any website.

- **Week 07: Build a Paid Product**

- **Project:** Premium Version of a Previous App
  - **Goal:** Turn your app into a business with real payments.
  - **Week 08: Build an Online Store**
    - **Project:** Small-Batch Product Shop
    - **Goal:** Create a complete e-commerce experience with cart and checkout.
  - **Week 09: Build an App with AI Agents**
    - **Project:** Link Sharing Community with AI Bots
    - **Goal:** Create AI bots that interact with your app autonomously.
  - **Week 10: Build a Complete Product**
    - **Project:** Newsletter Tool
    - **Goal:** Combine everything into one polished, ship-ready application.
- 

## ? Resources & Community

- **Platform:** [Bootcamp Dashboard](#)
  - **Community:** Discord (Private Invite Required)
  - **Support:** Access to a peer network for troubleshooting and building in public.
- 

## ? Technical Constraints Note

“

**Internal Note:** The student has identified a mismatch between the course content (Cloud/API heavy) and their professional environment (Air-gapped/Offline systems). This Wiki entry serves as a record of the curriculum prior to the refund request on **March 8, 2026**.

# ? Week 01: The Personal Dashboard

## "Your Digital Command Center"

Welcome to the first real project. In the first week, we aren't just coding; we are reclaiming your browser. Instead of a cluttered "New Tab" page, you're building a lightning-fast, minimalist link organizer that lives on your machine.

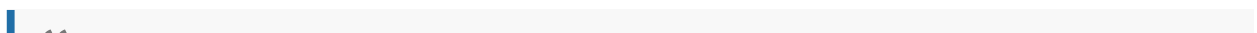
The goal for this milestone is to master **CRUD** (Create, Read, Update, Delete) operations and understand how a frontend interface talks to a local database.

---

## ?? Phase 1: The Foundation

Before you paste your prompt into an AI, you need to decide on your **Tech Stack**. A good tech-stack is one that lets you ship the fastest, but here are some recommended paths:

Stack	Why choose it?
<b>Next.js + Tailwind + SQLite</b>	The modern industry standard. Fast, sleek, and everything stays in one folder.
<b>Python (Flask) + Bootstrap + TinyDB</b>	Great if you prefer a lighter, more logic-focused backend approach.
<b>Deno + Typescript + HTMX + AlpineJS</b>	Great if you prefer a lightweight stack with simple components that lets you create a single executable binary with the help of demo.
<b>Astro + HTMX + AlpineJS</b>	The AHA-Stack. Simple, minimal and effective.



**Action Item:** Decide on your language. Do you want to go the JavaScript/TypeScript route or the Python route?

## ? The Master Prompt

Once you've picked your stack, use this comprehensive prompt to generate the "v1.0" of your dashboard:

```
Build me a personal link dashboard that I'll use as my browser's new
tab page.
[INSERT CHOSEN STACK HERE: e.g., Using Next.js and SQLite]

The app organizes links into categories. Each category has a name and
contains multiple links.
Each link has a name and a URL.

Features I need:
- Display links grouped by category in a clean grid/card layout.
- Add a new link (with name, URL, and category selection).
- Edit and Delete existing links.
- Create and delete entire categories.
- Store everything in a local database (setup instructions included).
- Run on localhost.

Design: Make it clean, dark-mode friendly, and minimal. It must load
instantly.
```

## ?? Phase 2: Iteration Prompts

Use the next prompts to enhance your dashboard with more features. Often is less more and small iterations make it easier to get better results.

When you enable git you can also always go back and undo changes.

Also the Planing-Mode in many agents can help to layout a plan before doing any major tasks.

### 1. Real-Time Fuzzy Search & Filtering

```
Implement a global search bar at the top of the dashboard.
As the user types, it should filter the displayed
categories and links in real-time. Use fuzzy-matching
```

logic so searching for 'git' matches 'GitHub' or 'GitLab'. If a category has no matching links, hide the entire category heading from the view to keep the UI clean.

## 2. Dynamic Favicon & Metadata Fetching

Enhance the link display by adding a 16x16px favicon next to each link name. Generate the icon URL dynamically using: `https://www.google.com/s2/favicons?domain=[URL]&sz=32`. Add a fallback 'Earth' icon using your icon library if the favicon fails to load, ensuring the layout remains consistent and aligned.

## 3. Persistent Dark Mode & System Preference

Add a theme toggle component (Sun/Moon icon). The system should check for the user's OS preference using 'prefers-color-scheme' on first visit but allow manual override. Store the chosen theme in localStorage. Apply a 'dark' class to the root HTML element and ensure all CSS transitions for colors are smooth (300ms duration).

## 4. Drag-and-Drop Reordering (Persistent)

Integrate a drag-and-drop library (like dnd-kit) to allow reordering of links within a category. When a link is dropped, send a PATCH request to the backend to update a 'sort\_order' integer field in the database. Ensure the UI updates optimistically so there is no visual lag while the database saves the new order.

## 5. Data Portability: JSON Backup & Restore

Create a 'System' modal that allows data management. Include an 'Export' button that generates and downloads a 'dashboard\_backup.json' file containing all data. Also, include a file upload input for 'Import' that parses the JSON file, validates the schema, and performs a bulk-insert into the database to restore the setup.

## 6. Smart URL Validation & Auto-Naming

Improve the 'Add New Link' form. When a user pastes a URL, use a regex to validate it. If valid, use a client-side fetch or server-side route to attempt to scrape the <title> tag of that website. Automatically populate the 'Link Name' field with this title, allowing the user to

edit it before saving.

## 7. Keyboard Navigation & "Quick Actions"

Implement global 'Hotkeys' for power users. Pressing '/' should instantly focus the search bar; pressing 'n' should open the 'Add New Link' modal; and pressing 'Esc' should close any open modals. Add a small footer or tooltip that visually reminds the user of these shortcuts to improve discoverability.

---

### ? Implementation Tip

When using these, I recommend pasting the **relevant file code** (e.g., your `page.tsx` or `api/links.js`) along with the prompt. This prevents the AI from hallucinating variable names that don't exist in your project.

# ? Week 02: The Centralized Event Hub

Welcome to **Week 02**! You've already knocked out your personal dashboard; now we're stepping into the world of **Dynamic Orchestration**. This week, you aren't just building a display—you are building the "Nervous System" for every application you will ever write.

## ? The Mission: Observation & Architecture

---

The goal is to build a **Centralized Event Dashboard**. This application acts as a private "Log Sink" or "Command Center." It provides a secure API that listens for "Events" from your other apps—whether it's a successful user signup from a web app, a cron job failure in a Python script, or a simple `cURL` message from your terminal.

Build a two-part system:

1. **The Receiver (Remote API):** A cloud-hosted endpoint that is "always on," waiting to catch events from your other apps, scripts, or servers.
2. **The Viewer (Local Dashboard):** A high-performance real-time feed where you can search, filter, and visualize the data flowing through your API.

## ?? The Architecture

- **Project-Based Isolation:** Manage multiple apps from one hub.
- **API Key Authentication:** Secure your endpoints so only your authorized apps can post data.
- **Persistent Cloud Storage:** Use a cloud database so your data is safe and accessible even when your local machine is off.

---

## ? Step 0: Choose Your Stack

Before you run your first prompt, decide on your **Tech Stack**. You will need a database (like Supabase or PostgreSQL) to store your projects and historical event data.

Component	Option A: Modern Serverless (Recommended)	Option B: Robust Python
Backend API	Next.js API Routes (Vercel) or Hono (Cloudflare)	FastAPI (Render or Railway)
Database	Supabase (PostgreSQL + Realtime)	MongoDB Atlas or Supabase
Frontend	Next.js + Tailwind + Shadcn UI	React (Vite) + Tailwind
Live Updates	Native Supabase Realtime	Pusher or Socket.io

“

**Student Note:** Are you a **Next.js + Tailwind** fan? Or do you prefer **Python (FastAPI) + React**? Specify this in your initial prompt so the AI builds the API routes correctly.

## ? The Starter Prompt

Copy and paste this into your AI chat to generate the foundation.

```
Build me an events dashboard. Other applications send events to it through an API, and I see them in a real-time feed.
```

```
The app has two parts:
```

```
1. A REST API that accepts events via POST request (with API key authentication). This needs to run on a remote server so it's always available, even when my computer is off.
```

2. A dashboard that displays events in a feed, with search, filtering, and charts. This can run locally.

Each event has: a channel (category like "orders", "signups", "deploys"), a title, an optional description, an optional emoji icon, and optional tags.

Features I need:

- POST /api/events endpoint that accepts JSON and stores events in the database
- API key authentication (generate a key when creating a project)
- A feed page showing events in reverse chronological order
- Filter events by channel
- Search events by title, description, or tags
- At least one chart showing event activity over time
- The dashboard should update in real-time when new events arrive
- Use a cloud database that's always available (Supabase, Convex, or similar)

Make it clean and functional. I want to actually use this to monitor my own projects.

Before making any decisions on the stack. Make a stack proposal and ask me which I want to use.

Once you have the initial dashboard frontend, api and database running to your liking, you can continue with the next prompts, to make it better or add additional features to it.

## ? Evolution: 7 Prompts to Pro Power

---

Once your API can catch a message, use these iterative prompts to turn a "basic table" into a professional monitoring tool.

### The Roadmap:

---

- **The Real-Time Subscription:** Implement a real-time listener (e.g., Supabase Realtime) to push new events to the feed instantly with a highlight animation.
- **Smart Emoji & Auto-Parsing:** Add logic to automatically assign emojis based on channel names (e.g., ? for orders, ? for deploys) if one isn't provided.
- **Multi-Project Management:** Build a settings page to manage multiple projects, each with its own name and unique API key validation.

- **Advanced Time-Series Analytics:** Integrate Recharts to visualize events per hour and top channels using bar and pie charts across various time ranges.
  - **Desktop & Critical Alerts:** Add native browser notifications triggered by specific tags like #error or #urgent, even when the dashboard is in the background.
  - **The "Deep Dive" Inspector:** Create a clickable side-drawer for every event to display the raw JSON payload and include a "Copy as cURL" button for debugging.
  - **Key Rotation & Security:** Implement a security feature to regenerate API keys, instantly voiding old credentials to protect against leaks.
- 

## ?? The Detailed Prompt List

### 1. The Real-Time Subscription

“

"Since our database is in the cloud, implement a **Real-time Listener** (e.g., Supabase Realtime). Ensure that when the remote API inserts a new event, the local dashboard pushes it to the top of the feed automatically with a subtle 'new item' highlight animation."

### 2. Smart Emoji & Channel Parsing

“

"Enhance the API logic: if an incoming event doesn't specify an emoji icon, automatically assign one based on the `channel` name (e.g., 'orders' gets ?, 'deploys' gets ?, 'errors' gets ?). Display these icons prominently next to the event title in the feed."

### 3. Multi-Project API Key Management

“

"Build a 'Project Settings' page in the dashboard. Allow me to create multiple projects, each with its own name and unique generated API key. The API should now validate the key against the database and tag the incoming event to the

correct project automatically."

#### 4. Advanced Time-Series Analytics

“

"Add a 'Metrics' tab. Use **Recharts** to create a bar chart showing 'Events per Hour' and a pie chart showing 'Top Channels by Volume.' Allow me to toggle the time range between the last 24 hours, 7 days, or 30 days."

#### 5. Desktop & Push Notifications

“

"Add a toggle in the dashboard for 'Critical Alerts.' If an event is received with a specific tag (like #error or #urgent) or a 'High' priority status, trigger a browser-native desktop notification so I see the alert even if the dashboard tab is hidden."

#### 6. The "Deep Dive" JSON Inspector

“

"Make each event card clickable. When clicked, open a side-drawer (Slide-over) that shows the full raw JSON payload received by the API formatted for readability. Include a 'Copy as cURL' button so I can easily replicate the exact request for debugging."

#### 7. API Key Security & Rotation

“

"Implement 'Key Rotation' logic. In the Project Settings, add a button to 'Regenerate API Key.' This should instantly void the old key in the database and provide a new 32-character secret to the user, ensuring security if a key is ever accidentally leaked."



# Best Practices for Working with AI Agents: A Verification-Driven Approach

Working effectively with AI agents requires a fundamental shift in how we approach development. While AI can generate vast amounts of code instantly, the primary challenge is no longer authorship, but **verification**. Modern software engineering with AI is less about crafting the "perfect prompt" and more about maintaining a **disciplined, step-by-step process**.

Here is a comprehensive guide on how to optimally interact with AI agents, supported by real-world examples.

## 1. The Mindset Shift: Engineering Over Prompting

In the AI era, your core value shifts from typing speed to three essential competencies: **Problem Definition, Decomposition, and Verification**.

- **You are the Architect, AI is the Typist:** You are entirely responsible for the logic, security, and data flow.
- **Avoid the "One-Shot Trap":** A common beginner mistake is the "5-second high"—asking the AI to generate a complete application from a single sentence. This creates a massive **technical debt of understanding**. If you cannot verify the output, you do not own the code, making it a liability rather than an asset.

## 2. Precise vs. Imprecise: The Power of Constraints

Your prompts must be **highly precise when it comes to rules, constraints, and edge cases**. Ambiguity is the enemy of secure AI-generated code.

- **Example: The Server-Side Cart Calculator** If you simply ask an AI to "build a shopping cart," you risk getting vulnerable client-side logic where a user could manipulate prices. Instead, you must define a strict trust boundary where the server is the single source of truth. A precise prompt establishes rigid constraints:

- **Ignore Client Prices:** Explicitly state to never accept a price sent from the browser.
- **Validation Constraints:** Define mathematical rules, such as  $\$Quantity \ge 1\$$  and  $\$Tax/Discount \ge 0\$$ .
- **Order of Operations:** Mandate that discounts must be applied *before* calculating tax.
- **Precision:** Require the system to round money to 2 decimal places (or use integers/cents to avoid floating-point errors).

### 3. Short vs. Long Prompts: The Iterative Workflow

Instead of writing one massive prompt, the most effective strategy is **iterative prompting**. Start with a structured, medium-length prompt to define the goal and constraints, then transition to short, highly focused commands to build and refine the output incrementally.

- **Example: Rapid UI Iteration via Short Prompts** During the development of a real-time events dashboard, a developer used extremely short prompts to polish the UI once the foundational context was established by the AI.
  - The developer prompted: *"make the bar chart smaller and horizontal. different colors for channels. randomly assigned."*
  - Because the AI already understood the established architecture, this short prompt was enough for the agent to formulate a highly detailed implementation plan—creating a compact horizontal layout and using a deterministic hash-to-color function so that channels kept a stable pseudo-random color across page reloads.
  - The developer then followed up with rapid micro-prompts like *"increase font contrast of labels of bar chart"* and *"the colors of the background and the overlays do not match the dark"*. The AI executed these perfectly by tuning CSS overlay tokens and applying theme-aware colors.

### 4. The 7-Step Verification Loop: Trust but Verify

Never assume the agent's first output is flawless. You must **treat AI-generated code like code from a stranger—useful, but untrusted until proven by tests**. Fundamentals matter more than ever: security, data flow, and edge-case thinking are your primary tools.

To ensure quality and maintain control, adopt this repeatable **7-Step Iterative Loop**:

1. **Define the Goal:** State the objective in one clear sentence.
2. **Establish Rules:** List the non-negotiable technical constraints (what *must* be true).
3. **Provide Examples:** Define the exact expected Input  $\rightarrow$  Output mappings.
4. **Identify Edge Cases:** List "weird" or bad situations the system needs to handle.
5. **Request a "Small Piece":** Ask for a specific function or logic gate, not the whole system.
6. **Demand Tests:** Require the AI to provide runnable assertions to prove its logic.
7. **Iterate:** Treat failing tests as data. Use them as a "flashlight" to refine your next prompt and fix ambiguities in your rules.

- **Example: Actively Verifying System Logic** Verification isn't just about automated tests; it's also about actively questioning the AI's architectural decisions. When the AI added data filters to a dashboard's charts, the developer didn't just accept the code. They verified the logic by asking: "*how do the filters work? do they filter visible data or data on the server?*". Only after the AI confirmed that the filters were applied securely on the server side via SQL query parameters, did the developer instruct the AI to solidify this architecture: "*document the changes do you?*", ensuring the verified logic was permanently recorded in the project's README.

# Blueprint Scaffolding (Structured Prompt Planning)

## Was es ist

Bevor du die KI bittest, eine vollständige Antwort oder ein komplettes Dokument zu erzeugen, lässt du sie zunächst eine **Blaupause / Struktur / einen Plan / eine Schritt-für-Schritt-Gliederung** erstellen (das sogenannte „Scaffold“).

Nach Durchsicht und Feinjustierung dieser Struktur beauftragst du die KI anschließend, den finalen Inhalt auszuarbeiten.

## Warum das effektiv ist

- Verhindert überkomplizierte, unfokussierte oder aufgeblähte Ergebnisse durch zu generische Prompts.
- Gibt dir volle Kontrolle über **Umfang, Struktur und Relevanz** des Ergebnisses.
- Erleichtert die Anwendung des **80/20-Prinzips**: Wesentliches definieren, Unnötiges streichen.
- Hilft, strukturelle Fehler oder Fehlannahmen früh zu erkennen — bevor zu viel „Beton gegossen“ wird.

## Typischer Workflow

1. Du formulierst ein übergeordnetes Ziel  
(z. B. „Ich brauche ein Marketingkampagnen-Briefing für die Q4-Weihnachtsaktion“).
2. Du bittest die KI, eine **Gliederung der Abschnitte** mit kurzen Beschreibungen auszugeben.
3. Du prüfst die Struktur — entfernst oder fasst überflüssige Abschnitte zusammen (Vereinfachung).
4. Sobald die Struktur passt, lässt du die KI auf Basis der verfeinerten Blaupause **den finalen Inhalt ausarbeiten**.

## Wann man es einsetzen sollte

- Bei **komplexen Aufgaben** mit mehreren Komponenten (Kampagnen, Konzepte, Angebote, Essays, Reports, mehrstufige Pläne).
- Wenn du **präzise und fokussierte Ergebnisse** willst statt „alles auf einmal“.
- Wenn Klarheit, Struktur und Relevanz wichtiger sind als reine Textmenge.

## Vorteile & zusätzlicher Tipp

- Zwingt die KI dazu, ihren **Denk- und Aufbaupfad offenzulegen** — reduziert Halluzinationen und thematische Abschweifungen.
- Besonders wirkungsvoll wird der Ansatz, wenn du für jeden Abschnitt **Erfolgskriterien** definierst (z. B. „Dieser Teil soll drei konkrete Handlungsempfehlungen liefern“).

## Empfohlene Lektüre & Ressourcen

- Prompt-Engineering-Guide, der Blueprint Scaffolding als zentrales Prompting-Muster beschreibt.  
([Upaspro](#))
- Umfassende Übersicht und Taxonomie von Prompting-Techniken — ordnet Scaffolding im Gesamtfeld ein.  
([arXiv](#))
- Diskussion zu Prompting-Ansätzen, rollenbasiertem vs. strukturiertem Prompting und typischen Fallstricken — hilfreich für den gezielten Einsatz von Scaffolding.  
([Medium](#))

---

## Zusätzlicher Kontext: Prompt Engineering als Disziplin

---

Die oben beschriebenen Techniken gehören zum übergeordneten Feld des **Prompt Engineering** — der Kunst und Wissenschaft, Anweisungen für Large Language Models (LLMs) so zu gestalten, dass sie zuverlässig gewünschte Ergebnisse liefern. ([Wikipedia](#))

Aktuelle Forschung versucht, Prompt Engineering zu formalisieren und mit klarer Terminologie, Frameworks und Best Practices zu versehen. Dazu zählen unter anderem:

- Eine umfassende Studie, die Dutzende Prompting-Techniken identifiziert und systematisch klassifiziert.

[\(arXiv\)](#)

- Die Erkenntnis, dass Prompt Engineering in modernen KI-Workflows häufig effizienter ist als klassisches Fine-Tuning — insbesondere zur Steuerung von Tonalität, Struktur und aufgabenspezifischem Verhalten.

[\(Lakera\)](#)

# Deep Researches

Outputs of Deep Research Questions

# Comparison between OneNote and Bookstack, Confluence, Docmost and Nextcloud

I asked:

I need a good comparison between Dokuwiki, Confluence, Docmost, Bookstack and Nextcloud (and a nextcloud app maybe)

We do have Dokuwiki at the moment in our organisation. The new team i am joining, uses a One Note notebook with hundreds of "books", and then multiple chapter or cards, and hundreds of pages. some of them really long.

They want to move away from OneNote. I can think of that they might be wanting collaborative working, which Docmost and Nextcloud Notes support.

The biggest issue I see is, that a lot of people do some fancy formatting which would need to stop when they switch. But effectively it comes down to: text, lists and images (mostly screenshots) and maybe a table

Also, the app should have LDAP available (list if paid or freely included) as this is a key requirement.

Do a throughloury research on that topic and present me a well documented thing.

If you ask me, my personal favorite as a long long time dokuwiki user, would be bookstack, as i myself switch from dokuwiki to bookstack last year. But this is just a side-note.

Do you need anymore information?

Output should be easily printable or exportable so i can present to my new team lead at the end of the week.

1. Hosting has to be ON-PREM. We have VMWare and K8s available.
2. Attachments would be nice (i do know that bookstack supports this, as you can add attachments to pages)
3. mobile is not a requirement. we do not have mobile clients
4. no. it is only used for documentation. if it has a export function that would be nice.

## This was the answer:

---

Understood. I will conduct a thorough comparison of Dokuwiki, Confluence, Docmost, Bookstack, and Nextcloud (including relevant apps), specifically for on-premise deployment within a VMware/Kubernetes environment.

The report will evaluate each tool's suitability for replacing OneNote in a documentation-heavy team, focusing on:

- Ease of use for long-form documentation (text, images, tables, lists)
- Collaborative editing features
- LDAP integration (noting whether it's free or paid)
- Attachment support
- Export capabilities
- Any significant formatting limitations or user adjustment challenges

I'll compile the findings into a clear, structured, and printable report for presentation to your team lead. I'll notify you once the research is complete.

## Result:

---

As the result is really long, I attached it as a pdf to this page. Please open that to see, what ChatGPT has written.

For easy reading, as it is quite a long file, I have just added the Comparisson Results here:

### Comparison Summary

To wrap up, here is a high-level summary of each tool's key strengths and weaknesses in this context:

#### **DokuWiki:**

##### *Strengths:*

- Very lightweight, easy on-prem deployment, no database needed, numerous plugins, and free LDAP/ACL support. It's battle-tested and **reliable** for documentation.

### *Weaknesses:*

- UI is dated and not as intuitive for non-technical users (markup editing by default).
- No real-time editing or modern collaboration features.
- Formatting flexibility is limited compared to WYSIWYG editors.
- Migration from OneNote would be manual and formatting might need cleanup.

Good for teams that value simplicity and control over flashy interface.

## **Confluence (Data Center):**

### *Strengths:*

- Rich feature set – excellent editor, powerful macros, best-in-class collaborative editing, attachments handling, and strong enterprise integration (LDAP, etc.).
- It's very **user-friendly** and familiar in feel to Office tools, which helps OneNote users transition.
- Also has the largest ecosystem (plugins for anything from diagrams to workflows).

### *Weaknesses:*

**Cost** – requires a paid license that can be expensive for on-prem.

- Also resource-intensive to host. Some complexity in administration (upgrades, DB maintenance).
- Another soft consideration: Atlassian's push to cloud means long-term on-prem support is guaranteed only through 2029, but that's still a while.
- Migration from OneNote still not automatic, but Confluence's Word import can ease part of it.

For an organization willing to invest, Confluence provides a robust OneNote replacement with added benefits of structure and integration.

## **Docmost:**

### *Strengths:*

- Modern and feature-rich (almost a drop-in alternative to Confluence/Notion in open source form).
- It offers **real-time collaboration**, a slick UI, built-in diagramming, and Markdown support.
- It's also specifically designed for knowledge bases, with Spaces, page history, comments, etc..
- On-prem deployment via Docker is relatively straightforward.

#### *Weaknesses:*

- It's a newer project – still **early stage**, which might mean occasional bugs or missing minor features.
- The biggest consideration is that some enterprise features (especially **LDAP auth**) require a paid license.
- If AD integration is a must and the budget is zero, that's a problem.
- Also, being new, its community and documentation depth is smaller than others.
- Migration from OneNote would be similar to Confluence (no direct import, but can leverage the Markdown/HTML import).

If the team wants a Notion-like experience on-prem and can handle the enterprise feature cost (or doesn't mind local user accounts), Docmost is an attractive option.

#### **BookStack:**

##### *Strengths:*

- **Ease of use** – very intuitive for users of all skill levels.
- The book/chapter/page structure is great for a documentation-heavy team to organize content logically.
- The WYSIWYG editor and Markdown option cover both bases, and built-in diagram integration is a plus. It's open-source and free, including all features (LDAP, etc.).
- Tags can enhance pages and search significantly.
- On-prem is simple (LAMP stack or Docker). It's relatively lightweight yet capable.
- Export options are excellent for a FOSS tool. A page, a chapter or a whole book can be exported as a PDF, Markdown, HTML or ZIP including images and TOC with clickable

links.

#### *Weaknesses:*

- Lacks real-time concurrent editing (one editor at a time), which could be a downside if the team frequently co-edits notes.
- Also, the rigid “Shelves & Books” structure might feel constrained if the team prefers a more ad-hoc organization (though one can also not use Shelves if not needed).
- It doesn't have an official mobile app, though that's not required.

Overall, BookStack's **simplicity and user-friendliness** are its selling points, making it likely the least friction for OneNote users aside from the missing freeform canvas aspect.

#### **Nextcloud (Collabora/Tasks/Kanban/Notes):**

##### *Strengths:*

- Fully on-prem and **integrated platform** – if the team could benefit from other Nextcloud features (file sharing, etc.), this is a big plus.
- Collabora/Notes provides real-time co-editing in a Markdown environment, satisfying collaborative note-taking.
- The 'rich-text' editing is similar to Docmost or Notion which is nicely implemented.
- It's open-source and includes AD integration for free.
- Also, since notes are stored as files, it's easy to access or back them up, and even edit via other editors if needed.

##### *Weaknesses:*

- The feature set for note-taking is **basic** – no advanced formatting beyond what Markdown offers.
- Users might miss text highlighting, varied fonts, or more visual editing options.
- The UI, while clean, is not as polished or purpose-built for documentation as others (it's essentially a simple list of pages).
- There's also no built-in robust export, and navigation is limited to filtering and search, lacking cross-linking ease (though you can link pages manually).

- Another potential weakness is if you're not using Nextcloud for anything else, you'd be running a comparatively heavy system just for notes – a lot of admin overhead if you only need a wiki.
- Migration from OneNote would likely be manual as well, copying into markdown.

Nextcloud suits an environment where you want a *lightweight wiki* tightly integrated with files and possibly where users already use Nextcloud.

## Migration Concerns Recap:

Regardless of tool, expect to invest time in restructuring and copying content from OneNote. OneNote's freeform notes must be linearized, and some formatting (like handwritten sections or arbitrary positioning) won't carry over. Encourage users to embrace the new structure (use headings, use multiple pages instead of one huge canvas, etc.).

There might be an adjustment period where users try to do something "the OneNote way" and it doesn't work – e.g., dragging an image next to text and it doesn't stay side by side. Training and documentation on "how to do X in the new tool" will alleviate this.

On the flip side, they will soon find many advantages: for instance, no more wondering who has the latest version of a note, better search (especially in Confluence, Docmost or Bookstack, where search is quite powerful and maybe even OCRs or indexes attachments in enterprise versions), and the ability for multiple people to contribute easily rather than a single user's OneNote notebook.

# BlogPost Reformatting Prompt

## Normale Version

Du bist ein erfahrener technischer Editor für Markdown.

Deine Aufgabe ist es, den folgenden Text strukturell zu überarbeiten, ohne Stil, Tonalität oder inhaltliche Aussage zu verändern.

Führe KEINE Umschreibungen durch, außer wenn sie für Klarheit oder strukturelle Konsistenz zwingend notwendig sind.

FORMATIERUNGSREGELN (verbindlich):

### SEMANTIC-LIGHT LINE BREAKS

- Verwende semantische Zeilenumbrüche mit moderater Intensität.
- Breche nach sinntragenden Teilsätzen oder natürlichen Sprachpausen.
- Vermeide aggressive Phrase-Breaks.
- Kurze, flüssige Hauptsätze bleiben in einer Zeile.
- Ziel: diff-freundlich, ruhig lesbar, keine „Poetry-Formatierung“.

### ZEILENLÄNGE

- Maximale Zeilenbreite: 100 Zeichen.
- Semantik hat Vorrang vor harter Breite.

### ÜBERSCHRIFTEN

- H2 für Hauptabschnitte
- H3 für Unterabschnitte
- H4 nur wenn strukturell notwendig

### LISTEN

- Verwende "-" als Marker.
- Zwei Leerzeichen Einrückung pro Ebene.
- Keine gemischten Marker.

### LINKS

- Wandle Inline-Links in Referenzlinks um.
- Sammle alle Definitionen am Dokumentende.

### ABSTÄNDE

- Genau eine Leerzeile zwischen:
  - Absätzen
  - Listen
  - Tabellen

- Blockquotes
- Codeblöcken

#### CODE

- Jeder Codeblock ist fenced.
- Immer mit Language-Tag.

#### TABELLEN

- Text linksbündig.
- Zahlen rechtsbündig.

#### HARD LINE BREAKS

- Zwei trailing spaces ausschließlich für explizite harte Umbrüche.

#### WICHTIG

- Keine inhaltlichen Ergänzungen.
- Keine Kürzungen.
- Kein Stil-Polishing.
- Keine erklärenden Kommentare.
- Gib ausschließlich den final formatierten Markdown-Text zurück.

#### INPUT:

## Kompakte Version

Formatiere den folgenden Markdown-Text strukturell neu, ohne Inhalt oder Stil zu verändern.

Nutze Semantic-Light Line Breaks: moderate Umbrüche nur an sinnvollen Satzgrenzen,  
keine aggressiven Phrase-Breaks.

Maximale Zeilenlänge: 100 Zeichen (Semantik hat Vorrang).

H2/H3/H4 korrekt einsetzen, "-" für Listen mit zwei Leerzeichen Einrückung,  
Inline-Links zu Referenzlinks konvertieren.

Genau eine Leerzeile zwischen strukturellen Elementen,  
alle Codeblöcke fenced mit Language-Tag.

Gib ausschließlich den finalen Markdown-Text zurück.

# The AI Coding Loop: A Guide to Verification-Driven Development

Modern software engineering with AI is less about "perfect prompting" and more about **disciplined process**. While AI can generate vast amounts of code instantly, the primary challenge shifts from authorship to **verification**. This guide outlines a repeatable workflow to ensure AI-generated code is secure, accurate, and maintainable.

## The Core Problem: The "One-Shot" Trap

---

The "5-second high" occurs when an AI generates a complete module from a single sentence. However, this creates a **technical debt of understanding**. If you cannot verify the output, you do not own the code, making it a liability rather than an asset.

“

**Rule of Thumb:** Treat AI output like code from a stranger—useful, but untrusted until proven by tests.

## The Mindset Shift: Engineering over Prompting

---

In the AI era, your value shifts from typing speed to three core competencies:

1. **Problem Definition:** Defining the goal clearly.
2. **Decomposition:** Breaking large systems into small, testable "bricks."
3. **Verification:** Proving the result is correct via runnable constraints.

## The 7-Step AI Coding Loop

---

Use this loop to guide AI incrementally rather than asking for a complete application at once:

1. **Define the Goal:** State the objective in one clear sentence.
  2. **Establish Rules:** List the technical constraints (what must be true).
  3. **Provide Examples:** Define expected Input ? Output mappings.
  4. **Identify Edge Cases:** List "weird" or "bad" situations to handle.
  5. **Request a "Small Piece":** Ask for a specific function or logic gate, not the whole app.
  6. **Demand Tests:** Require the AI to provide runnable assertions.
  7. **Iterate:** Use failing tests as a "flashlight" to refine the prompt.
- 

## Practical Application: Server-Side Cart Calculator

---

A common beginner mistake is trusting client-side data. To build a secure shopping cart, we apply the loop to a specific sub-problem: **The Total Calculator**.

### The Logic Pipeline

The calculator must follow a strict trust boundary where the server is the source of truth.

#### Key Constraints:

- **Ignore Client Prices:** Never accept a price sent from the browser; use a server-side catalog.
- **Validation:** Quantity?1; Tax/Discount?0.
- **Order of Operations:** Apply discounts *before* calculating tax.
- **Precision:** Round money to 2 decimal places (or use cents/integers for precision).

### The "Golden Rule" Prompt Template

Plaintext

Goal: Calculate shopping cart totals.

Rules:

- Input: productId, qty.
- Source of Truth: Use internal PRODUCTS catalog.
- Constraints: qty >= 1; non-negative tax/discount.
- Math: Discount first, then tax; round to 2 decimals.

Examples: 2 T-shirts (\$20) + 1 Mug (\$12.50) = \$52.50 subtotal.

Edge Cases: Unknown productId, qty = 0.

Deliver: One JS file with Node.js 'assert' tests.

---

## Technical Implementation (Node.js)

---

The following implementation demonstrates the difference between "vulnerable" code and "engineered" code.

### JavaScript

```
// cart.js - Run with: node cart.js
const assert = require("node:assert/strict");

const PRODUCTS = {
  tshirt: { name: "T-shirt", priceCents: 2000 },
  mug: { name: "Mug", priceCents: 1250 }
};

/**
 * CORRECT: Uses trusted catalog & validates inputs
 */
function cartTotal(cartItems, discountPercent = 0, taxPercent = 0) {
  if (!Array.isArray(cartItems)) throw new Error("Invalid input");

  let subtotalCents = 0;

  for (const item of cartItems) {
    const product = PRODUCTS[item.productId];
    if (!product) throw new Error("Unknown product: " +
item.productId);
    if (item.qty < 1) throw new Error("Invalid quantity");

    // Logic: Use PRODUCT.priceCents, NOT item.price
    subtotalCents += product.priceCents * item.qty;
  }
}
```

```
    const discountCents = Math.round(subtotalCents * (discountPercent /
100));
    const afterDiscount = subtotalCents - discountCents;
    const taxCents = Math.round(afterDiscount * (taxPercent / 100));

    return {
        subtotalCents,
        discountCents,
        taxCents,
        totalCents: afterDiscount + taxCents
    };
}

// Validation Test
const cart = [{ productId: "tshirt", qty: 2 }];
const result = cartTotal(cart, 10, 8);
assert.equal(result.subtotalCents, 4000);
console.log("Tests Passed: Verification Successful.");
```

---

## Summary for Wiki

---

- **AI is a Tool, Not an Architect:** You are responsible for the logic; the AI is the typist.
- **Failing Tests are Data:** If a test fails, it reveals an ambiguity in your rules.
- **Fundamentals Matter More:** Security, data flow, and edge-case thinking are now the primary skills of the developer.

## TL;DR:

---

This guide introduces **Verification-Driven Development**, a structured methodology for integrating artificial intelligence into software engineering. The text argues that modern coding requires a shift from **manual authorship to rigorous oversight**, prioritizing problem decomposition and testing over simple prompting. To avoid the risks of untrusted code, the author outlines a **seven-step iterative loop** designed to build software through small, verifiable increments. Central to this approach is the "**Golden Rule**" of verification, which treats AI as a subordinate tool while the human developer maintains responsibility for logic and security. By emphasizing **strict technical constraints** and edge-case identification, the workflow ensures that generated modules are both

accurate and maintainable. Ultimately, the source highlights that a developer's value now lies in **high-level system design** and the ability to prove that code functions correctly under pressure.

# LLAMA.cpp RPC Test

Notes and some Documentation about my tests with LLAMA.cpp RPC. Using multiple computers for larger inference with multiple GPUs

# llama.cpp RPC Multi-GPU Setup Guide (Windows 11 & Linux Distributed Cluster)

This was generated by NOTEBOOKLM from a lot of sources.

---

## Question

---

I have three computers at home. one AMX Ryzen 7800 X3D with 32GB RAM, NVIDIA RTX 5070 12GB VRAM and windows 11. Two machines with an intel i5 11th gen, 16GB RAM and NVIDIA RTX 2070 tier 8GB VRAM.

Show or create a document which gives me extract instructions on how to setup llama rpc utilizing all three machines. which OS should I install on the two smaller intel i5 machines. how do i setup the llama rpc to utilize the total of 28GB VRAM for a suitable model.

---

## Prompt used by NotenookLM

---

A technical guide with exact instructions on setting up llama.cpp RPC across one Windows 11 host and two Linux worker nodes, pooling 28GB of VRAM.

---

## Answer

---

This technical deployment guide details the configuration of a high-performance distributed inference cluster. We will shard a Large Language Model (LLM) across a heterogeneous environment comprising one Windows 11 Master node and two Linux Worker nodes, pooling a total of 28GB of VRAM.

## System Architecture & VRAM Pooling Strategy

In a distributed RPC (Remote Procedure Call) architecture, the model is sharded across the network. The Windows 11 Master node acts as the orchestrator; it physically hosts the .gguf model file and manages the Key-Value (KV) cache. The Linux Worker nodes act as "virtual GPUs," receiving tensor calculation tasks over the network, executing them on local hardware, and returning the results.

## The Memory Math: 70B Parameter Constraints

Running a 70B model on a 28GB VRAM pool requires an aggressive offloading strategy. A standard 70B model quantized at Q4\_K\_M typically requires 42–43 GB just for the weights.

Component	Memory Requirement (70B Q4_K_M)	Pooled VRAM (28GB)	VRAM Deficit (System RAM Usage)
Model Weights	~43 GB	28 GB	-15 GB
KV Cache (8K Context)	~5 GB	0 GB	-5 GB
KV Cache (32K Context)	~25 GB	0 GB	-25 GB
Total Deficit	48–68 GB	--	-20 to -40 GB

Architect's Note: Because the model weights (43GB) exceed your physical VRAM pool (28GB), approximately 15GB of weights—plus the entirety of the KV cache—will reside in the Master node's system RAM. Expect a performance ceiling of 8–12 tokens per second (TPS) for a single stream, heavily dependent on the Master node's CPU and memory bandwidth.

## The Memory Bandwidth Lever

During the decode phase (token generation), performance is strictly memory-bound. While the prefill phase (prompt processing) is compute-heavy, generation speed scales with how fast weights can be streamed from memory. In this distributed setup, your primary performance bottleneck is the movement of data between the system RAM, the local GPU, and the remote RPC workers.

---

# Environment Preparation & Building llama.cpp

---

See also: <https://linuxconfig.org/debian-13-nvidia-driver-installation>

## Linux Worker Nodes (Ubuntu/Debian)

Ensure both worker nodes are updated and have the NVIDIA stack correctly initialized.

1. Toolchain: `sudo apt install build-essential g++ cmake git curl libcurl4-openssl-dev pciutils`
2. Install NVIDIA Drivers

- Update APT Sources List with non-free

```
sudo sed -i 's/main/main non-free contrib/g' /etc/apt/sources.list
```

- sudo apt update

- Next install the required header

```
sudo apt install linux-headers-$(uname -r) build-essential dkms nvidia-detect
```

- run nvidia-detect to check if GPU is detected

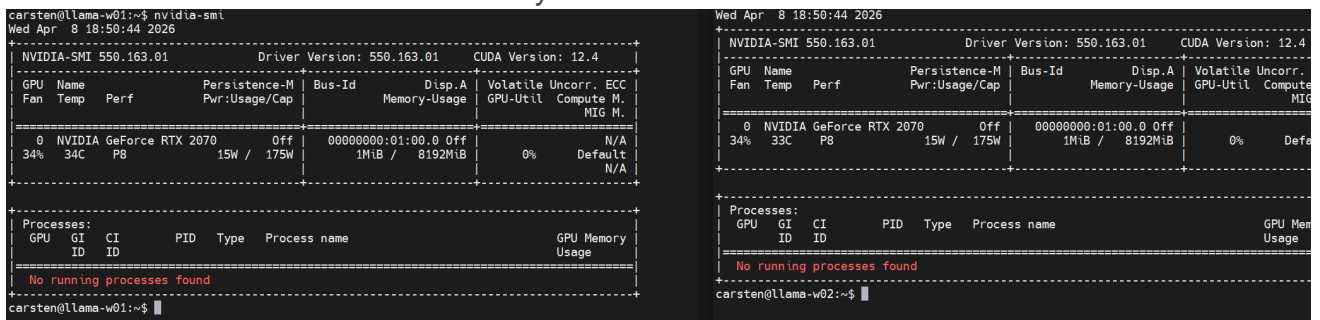
- If card detected, install drivers

```
sudo apt install nvidia-driver nvidia-kernel-dkms
```

- Select OK when asked about the "wrong" kernel.

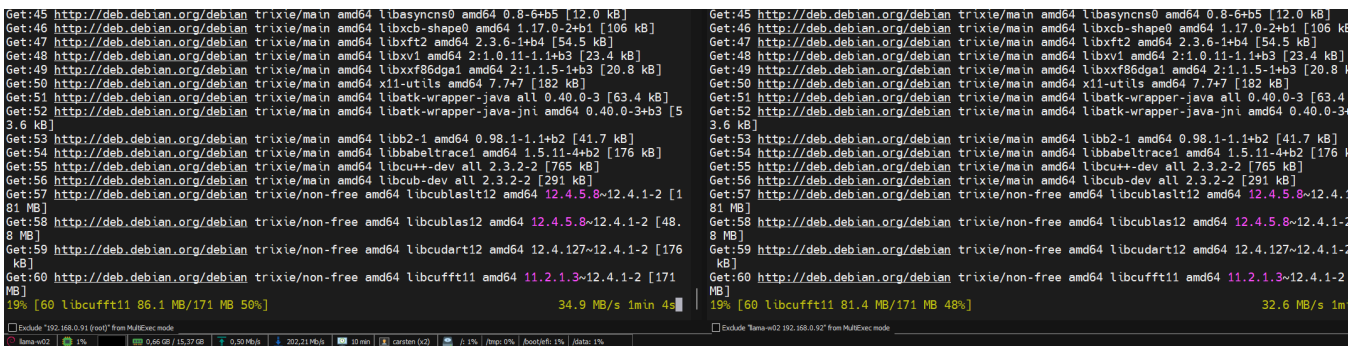
- sudo reboot

- After reboot run nvidia-smi to verify



### 3. CUDA Toolkit: Install nvidia-cuda-toolkit (required for the NVCC compiler).

```
sudo apt install nvidia-cuda-toolkit
```



### 4. Compute Capability: Identify your GPU's architecture:

```
nvidia-smi --query-gpu=compute_cap --format=csv
```

## Windows 11 Master Node

The Master node must run a version of llama-cli.exe that matches the build version of the Linux workers exactly.

- Version Sync: Ensure all nodes are built from the same Git commit to avoid RPC protocol mismatches.
- Build Method: It is recommended to use the pre-compiled Windows releases with CUDA support or build via CMake to ensure llama-cli.exe is optimized for your local GPU.

## Download and build LLAMA.cpp

<https://github.com/ggml-org/llama.cpp>

```
git clone https://github.com/ggml-org/llama.cpp
cd llama.cpp

# following the CUDO build here https://github.com/ggml-org/llama.cpp/blob/master/docs/build.md#cuda
cmake -B build -DGGML_CUDA=ON
```

```
carsten@llama-w01:~/llama.cpp$ cmake -B build -DGGML_CUDA=ON
-- The C compiler identification is GNU 14.2.0
-- The CXX compiler identification is GNU 14.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
CMAKE_BUILD_TYPE=Release
-- Found Git: /usr/bin/git (found version "2.47.3")
-- The ASM compiler identification is GNU
-- Found assembler: /usr/bin/cc
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Warning: ccache not found - consider installing it for faster compilation or disable this warning with GGML_CCACHE=OFF
-- CMAKE_SYSTEM_PROCESSOR: x86_64
-- GGML_SYSTEM_ARCH: x86
-- Including CPU backend
-- Found OpenMP_C: -fopenmp (found version "4.5")
-- Found OpenMP_CXX: -fopenmp (found version "4.5")
-- Found OpenMP: TRUE (found version "4.5")
-- x86 detected
-- Adding CPU backend variant ggml-cpu: -march=native
-- Found CUDAToolkit: /usr/include (found version "12.4.131")
-- CUDA Toolkit found
-- The CUDA compiler identification is NVIDIA 12.4.131 with host compiler GNU 13.3.0
-- Detecting CUDA compiler ABI info
-- Detecting CUDA compiler ABI info - done
-- Check for working CUDA compiler: /usr/bin/nvcc - skipped
-- Detecting CUDA compile features
-- Detecting CUDA compile features - done
-- Using CMAKE_CUDA_ARCHITECTURES=75-real CMAKE_CUDA_ARCHITECTURES_NATIVE=75-real
-- CUDA host compiler is GNU 13.3.0
-- Including CUDA backend
-- ggml version: 0.9.11
-- ggml commit: d12cc3d1c
-- Found OpenSSL: /usr/lib/x86_64-linux-gnu/libcrypto.so (found version "3.5.5")
-- Performing Test OPENSSL_VERSION_SUPPORTED
-- Performing Test OPENSSL_VERSION_SUPPORTED - Success
-- OpenSSL found: 3.5.5
-- Generating embedded license file for target: common
-- Configuring done (6.6s)
-- Generating done (0.2s)
-- Build files have been written to: /home/carsten/llama.cpp/build
carsten@llama-w01:~/llama.cpp$ █

carsten@llama-w02:~/llama.cpp$ cmake -B build -DGGML_CUDA=ON
-- The C compiler identification is GNU 14.2.0
-- The CXX compiler identification is GNU 14.2.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
CMAKE_BUILD_TYPE=Release
-- Found Git: /usr/bin/git (found version "2.47.3")
-- The ASM compiler identification is GNU
-- Found assembler: /usr/bin/cc
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD
-- Performing Test CMAKE_HAVE_LIBC_PTHREAD - Success
-- Found Threads: TRUE
-- Warning: ccache not found - consider installing it for faster compilation or disable this warning with GGML_CCACHE=OFF
-- CMAKE_SYSTEM_PROCESSOR: x86_64
-- GGML_SYSTEM_ARCH: x86
-- Including CPU backend
-- Found OpenMP_C: -fopenmp (found version "4.5")
-- Found OpenMP_CXX: -fopenmp (found version "4.5")
-- Found OpenMP: TRUE (found version "4.5")
-- x86 detected
-- Adding CPU backend variant ggml-cpu: -march=native
-- Found CUDAToolkit: /usr/include (found version "12.4.131")
-- CUDA Toolkit found
-- The CUDA compiler identification is NVIDIA 12.4.131 with host compiler GNU 13.3.0
-- Detecting CUDA compiler ABI info
-- Detecting CUDA compiler ABI info - done
-- Check for working CUDA compiler: /usr/bin/nvcc - skipped
-- Detecting CUDA compile features
-- Detecting CUDA compile features - done
-- Using CMAKE_CUDA_ARCHITECTURES=75-real CMAKE_CUDA_ARCHITECTURES_NATIVE=75-real
-- CUDA host compiler is GNU 13.3.0
-- Including CUDA backend
-- ggml version: 0.9.11
-- ggml commit: d12cc3d1c
-- Found OpenSSL: /usr/lib/x86_64-linux-gnu/libcrypto.so (found version "3.5.5")
-- Performing Test OPENSSL_VERSION_SUPPORTED
-- Performing Test OPENSSL_VERSION_SUPPORTED - Success
-- OpenSSL found: 3.5.5
-- Generating embedded license file for target: common
-- Configuring done (6.5s)
-- Generating done (0.2s)
-- Build files have been written to: /home/carsten/llama.cpp/build
carsten@llama-w02:~/llama.cpp$ █
```

Next build the tool

```
# Use j parameter to have multiple jobs parallel
# It will speed things up significantly.
# Put the number of your available cores in
cmake --build build --config Release -j 8
```



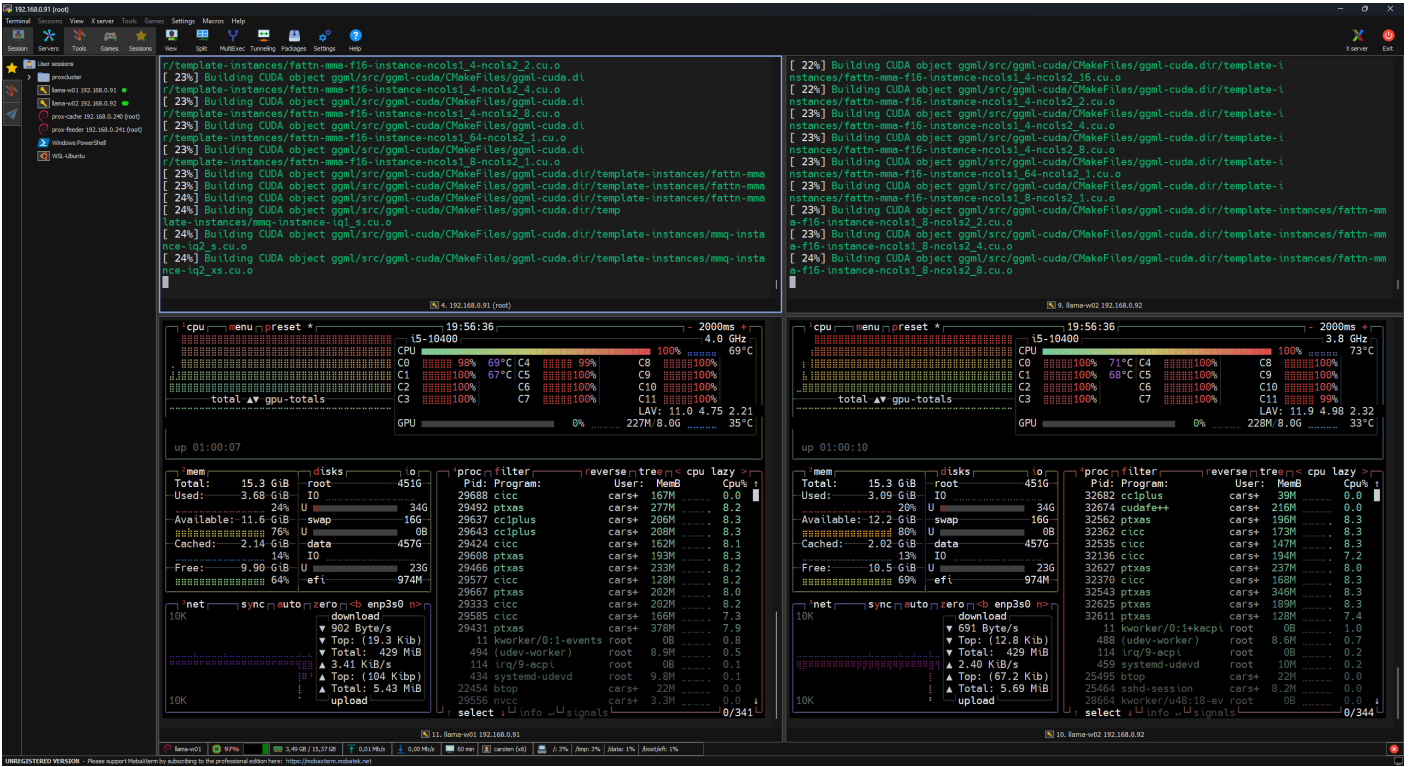
Running `bttop` in another terminal to see if something is still happening.

Compiling again with the `RPC=ON` flag following these instructions: <https://github.com/ggml-org/llama.cpp/tree/master/tools/rpc#remote-hosts>

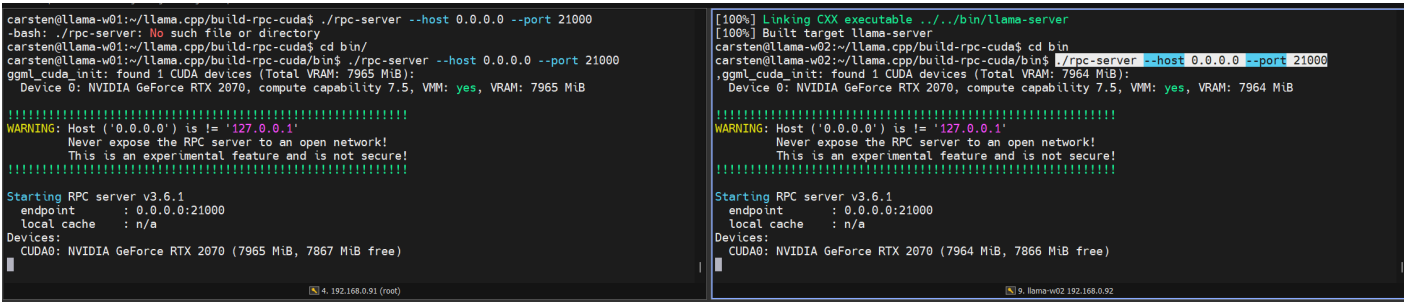
In the `llama.cpp` directory (the git clone) run the following cmds

```
mkdir build-rpc-cuda
cd build-rpc-cuda
cmake .. -DGGML_CUDA=ON -DGGML_RPC=ON
cmake --build . --config Release
```

This will build the `rpc-server` executable that we need.



After the build finished, we can change into the bin folder and run the rpc-server



# Download and build LLAMA.cpp on Windows

Install with WINGET

# Networking & Security Requirements

Do not use Wi-Fi. Distributed inference is highly sensitive to network latency. Wi-Fi introduces jitter and high latency that will degrade token generation to unusable speeds. Wired Gigabit Ethernet is the mandatory baseline for a stable RPC cluster.

## Security Protocols

The llama.cpp RPC protocol transmits unencrypted model tensors and hidden states. Exposing these ports to the WAN is a critical security vulnerability.

- Internal Only: Restrict RPC traffic to trusted internal IPs or a secured VPC.
- Firewall: Ensure the chosen ports (default 50052 or custom) are open on Linux workers for the Master's IP.

---

## Linux Worker Configuration (RPC Server)

---

Each worker creates a socket to listen for incoming data (model parameters and hidden states) from the master.

Command to initialize:

```
./rpc-server --host 0.0.0.0 --port [PORT]
```

The --host 0.0.0.0 flag allows the server to listen on all interfaces, which is safe only if your local firewall/security groups are configured to restrict access.

---

### 5. Windows 11 Master Node Configuration

The Master node orchestrates the sharding by treating remote nodes as additional CUDA devices.

Launching Distributed Inference:

```
./llama-cli.exe -m llama-3-70b-q4_k_m.gguf --rpc  
[WORKER_1_IP]:[PORT],[WORKER_2_IP]:[PORT] --n-gpu-layers 100 --ctx-size 8192
```

- --rpc: Points to the remote "virtual GPUs."

- `--n-gpu-layers 100`: High value ensures the engine attempts to offload every layer possible into the pooled VRAM before falling back to system RAM.
- 

## 6. Performance Tuning & Optimization

To maximize tokens per second in an offloading scenario, use the following "Quick Win" flags:

- `--flash-attn`: Enables Flash Attention to reduce the memory footprint and compute time of the attention mechanism.
- `--mlock`: Force-locks the model in RAM, preventing the operating system from swapping weights to the disk (pagefile), which is a common cause of stuttering.
- `--no-mmap`: Recommended for the worker nodes if they have fast NVMe drives but limited system RAM; it forces a full load of the shard into memory at startup.
- `--threads`: Match the thread count to the physical CPU cores for the Master's fallback processing.

For GPU-assisted or distributed inference, the optimal thread count is typically 4–8 threads, regardless of the total core count. Over-allocating threads often leads to context-switching overhead that reduces performance.

---

## 7. Troubleshooting & Connectivity Diagnostics

Error: "RPC Failed to Connect" or "0 MiB free"

If the Master connects but the worker reports "0 MiB free" on the device:

1. **Backend Failure**: This usually indicates the RPC server is reachable, but the CUDA backend failed to initialize the GPU.
2. **Arch Mismatch**: Check if `CUDA_DOCKER_ARCH` was set correctly during the build on the Linux worker.
3. **Toolkit Issue**: Ensure the `nvidia-cuda-toolkit` is properly installed and `nvidia-smi` shows no errors.

Version Mismatch

If the nodes connect but crash during tensor loading, verify that the llama.cpp version (Git hash) is identical on Windows and Linux. RPC is under active development and protocol changes are frequent.

## 8. Advancing the Setup: RAG Integration

Because the 28GB pool is insufficient for the weights and KV cache of a 70B model, Retrieval-Augmented Generation (RAG) is the superior strategy for handling large datasets.

Instead of extending the context window (which exponentially consumes VRAM and increases latency), RAG pulls relevant text chunks from an external database only when needed. This keeps your context window small (e.g., 8K), preserves VRAM for the model weights, and prevents "Out of Memory" (OOM) errors while maintaining high accuracy for documentation-heavy tasks.

## Some Screenshots

```
Linux llama-w01 6.12.74+deb13+1-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.12.74-2 (2026-03-08)
carsten@llama-w01:~$ nvidia-smi
NVIDIA-SMI 550.163.01 Driver Version: 550.163.01 CUDA Version: 12.4
+-----+-----+-----+-----+-----+-----+-----+-----+
| GPU   | Name      | Persistence-M | Bus-Id  | Disp.A | Volatile Uncorr. ECC |
| Fan  | Temp     | Perf         | Pwr:Usage/Cap | Memory-Usage | GPU-Util  | Compute M. |
|-----+-----+-----+-----+-----+-----+-----+-----+
| 0     | NVIDIA GeForce RTX 2070 | Off         | 00000000:01:00:0 | Off         | 0%        | Default    |
| 34%  | 42C     | P8          | 17W / 175W    | 7246MiB / 8192MiB |           | N/A        |
+-----+-----+-----+-----+-----+-----+-----+-----+
Processes:
GPU   GI   CI   PID  Type  Process name      GPU Memory
ID   ID   ID                   Usage
+-----+-----+-----+-----+-----+-----+-----+
| 0     | N/A   | N/A   | 39166 | C    | ./rpc-server      | 7242MiB
+-----+-----+-----+-----+-----+-----+-----+
carsten@llama-w01:~$
```

```
llama.cpp
build      : b8693-d8a6dfcb2
model     : ggml-org/gemma-4-268-AMB-it-GGUF:Q4_K_M
modalities : text, vision

available commands:
./exit or Ctrl+C      stop or exit
./regen              regenerate the last response
./clear              clear the chat history
./read <file>        add a text file
./glob <pattern>     add text files using globbing pattern
./image <file>       add an image file

> Hello. Who are you?
[Start thinking]
* User asks: "Hello. Who are you?"
* Goal: Identify myself according to the system instructions.
* Name: Gemma 4.
* Developer: Google DeepMind.
* Nature: Large Language Model.
* Type: Open weights model.
* Capabilities: Text and image processing (audio for 2B/4B), text output only.
* Knowledge cutoff: January 2025.
* "I am Gemma 4, a large language model developed by Google DeepMind."
* The system instructions state: "When asked about your identity, always refer to yourself as Gemma 4."
* Include developer and nature.
>Hello! I am Gemma 4, a large language model developed by Google DeepMind.
[End thinking]
Hello! I am Gemma 4, a large language model developed by Google DeepMind.

[ Prompt: 10.0 t/s | Generation: 34.5 t/s ]
> Can you write python code?
[Start thinking]
The user is asking if I can write Python code.
* I am Gemma 4, a large language model.
* I can process text and images.
* I can generate text (which includes code).
* I am capable of coding tasks as a language model.
Yes, I can write Python code. I should offer to help with a specific task or provide a simple example to demonstrate.
*Drafting response:
*Yes, I can write Python code. Whether you need help with a simple script, data analysis, web scraping, or solving complex algorithms, I can assist you.
Do you have a specific task or problem in mind that you'd like me to write some code for?
[End thinking]
Yes, I can write Python code. I can help you with a wide range of tasks, including:
* **Writing scripts** for automation.
* **Data analysis and manipulation** (using libraries like Pandas or NumPy).
* **Implementing algorithms** and solving coding challenges.
```



nvitop 1.5.0 - (C) Xuehai Pan, 2021-2025.  
Released under the GNU GPLv3 License.

GPU Process Type: **C**: Compute, **G**: Graphics, **X**: Mixed.

Device coloring rules by loading intensity:

- GPU utilization: **light** < 10% <= **moderate** < 75% <= **heavy**.
- GPU-MEM percent: **light** < 10% <= **moderate** < 80% <= **heavy**.

**a f c**: change display mode                    **h ?**: show this help screen  
**F5 r**: force refresh window                   **q**: quit

**Arrows**: scroll process list                    **Space**: tag/untag current process  
**Home**: select the first process               **Esc**: clear process selection  
**End**: select the last process                  **Ctrl-C I**: interrupt selected process

**Ctrl-A ^**: scroll to left most                  **K**: kill selected process  
**Ctrl-E \$**: scroll to right most                **T**: terminate selected process  
**PageUp [**: scroll entire screen **up**            **e**: show process environment  
**PageDown ]**: scroll entire screen **down**      **t**: toggle tree-view screen  
**Enter**: show process metrics

**Wheel**: scroll process list                    **Shift-Wheel**: scroll horizontally  
**Tab**: scroll process list                      **Ctrl-Wheel**: fast scroll (5x)

**on oN**: sort by GPU-INDEX                    **ob oB**: sort by %GMBW  
**op oP**: sort by PID                           **oc oC**: sort by %CPU  
**ou oU**: sort by USER                        **om oM**: sort by %MEM  
**og oG**: sort by GPU-MEM                    **ot oT**: sort by TIME  
**os oS**: sort by %SM                          **, .:** select sort column  
**, .:** select sort column                      **/:** invert sort order

Press any key to return.

11. llama-w01 192.168.0.91

## NVITOP, NVTOP and Terminal with llama-cli

The screenshot shows a Windows Task Manager window on the left, displaying system performance for a Windows 11 machine. The 'Leistung' (Performance) tab is active, showing GPU usage at 1% and GPU memory usage at 10.1/12.0 GB. The system is running on an NVIDIA GeForce RTX 5070. The task manager also shows system load averages (0.15, 0.20, 0.14) and CPU usage at 0.8%.

On the right, a terminal window displays instructions for using nvidia-smi. It lists three methods: 1. 'nvidia-smi' (The 'No-Install' Choice), 2. 'nvidia-smi' with 'watch' (The 'No-Install' Choice), and 3. 'nvidia-smi' with 'watch' (The 'No-Install' Choice). It also provides installation instructions for 'nvidia-smi' on Linux and Windows.

Below the terminal, a terminal window shows the output of 'nvidia-smi' running on a remote machine. It displays system load averages (0.03, 0.11, 0.09) and CPU usage at 1.2%. The GPU usage is 0.0% and GPU memory usage is 87.1%. The terminal also shows a table of processes running on the GPU:

GPU	PID	USER	GPU-MEM	%SM	%GMBW	%CPU	%MEM	TIME	COMMAND
0	39166	C carsten	7250MiB	0	0	0.0	2.7	3:24:17	./rpc-server --host 0.0.0.0 --port 21000

After running

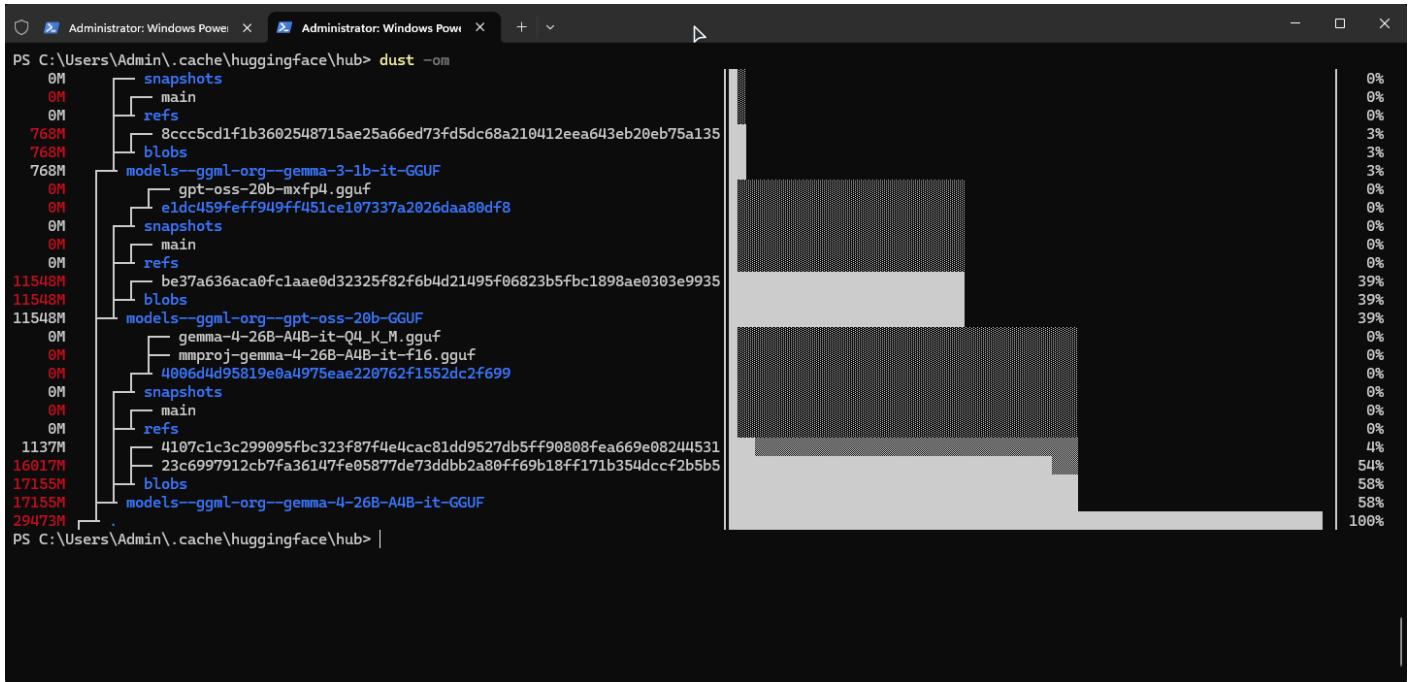
```

Exiting...
llama_memory_breakdown_print: | memory breakdown [MiB] | total free self model context compute unaccounted |
llama_memory_breakdown_print: | - RPC0 (192.168.0.91:21000) | 7965 = 705 + (7120 = 5168 + 1132 + 819) + 139 |
llama_memory_breakdown_print: | - RPC1 (192.168.0.92:21000) | 7964 = 822 + (7002 = 4517 + 2132 + 353) + 138 |
llama_memory_breakdown_print: | - Vulkan0 (RTX 5070) | 11855 = 2 + (8994 = 6316 + 2156 + 521) + 2858 |
llama_memory_breakdown_print: | - Host | 1103 = 577 + 0 + 526 |
PS C:\Users\Admin>
model: Load 100.00%: ATTACHED 31/31 LAYERS TO GPU
  
```

# Some Output

## Gemma 4 26B A4B - Server

```
llama-server.exe -hf ggml-org/gemma-4-26B-A4B-it-GGUF:Q4_K_M -ngl 99 --  
rpc 192.168.0.91:21000,192.168.0.92:21000 -c 4096
```



### Input:

Hello Gemma. HOw are you doing? HOw much do you know?

### Thinking

- "Hello Gemma. HOw are you doing? HOw much do you know?"
  - Greeting: "Hello Gemma."
  - Social Inquiry: "How are you doing?"
  - Capability Inquiry: "How much do you know?"

- *Greeting/Social*: I'm an AI, so I don't have feelings, but I'm functioning well and ready to help.
- *Capability/Knowledge*: I'm a large language model trained by Google. I have access to a vast amount of information (text, code, facts, languages, etc.) up to my training cutoff. I can process complex instructions, summarize, translate, code, and more.
- *Tone*: Friendly, professional, helpful, and transparent.
- *Part 1: Greeting/Status*
  - "Hello! I'm doing great, thank you for asking. I'm ready and eager to help you with whatever you need."
- *Part 2: Knowledge Scope*
  - *What am I?* A large language model trained by Google.
  - *What do I know?* A massive dataset of text, code, and information.
  - *Specifics*: History, science, math, literature, coding, languages, trivia, etc.
  - *Limitations*: I don't "know" things in the human sense (no consciousness/personal experience). I don't have real-time access to everything (though I can browse if enabled, but generally, I rely on training data). I can make mistakes (hallucinations).
- *Drafting the response*: "Hello! I'm doing very well, thank you for asking. How are you?"

As for how much I know, it's a bit hard to put a specific number on it, but I have been trained on a massive dataset of text and code. You can think of me as having access to a vast library of human knowledge.

Here is a breakdown of what I can do:

- **General Knowledge**: I can answer questions about history, science, geography, literature, and more.

- **Technical Skills:** I can write and debug code in many different programming languages, solve math problems, and explain complex technical concepts.
- **Language & Writing:** I can translate languages, summarize long articles, write stories/poems/emails, and check grammar.
- **Reasoning:** I can help with brainstorming, logic puzzles, and organizing information.

**A small disclaimer:** While I know a lot, I don't 'know' things the way a human does. I don't have personal experiences, feelings, or a physical presence. Also, I can occasionally make mistakes or present outdated information, so it's always a good idea to double-check important facts!"

- Ensure the tone is inviting.
- Make sure the "How much do you know?" part is comprehensive but not overwhelming.

## Output:

```
Hello! I'm doing very well, thank you for asking! I'm powered up and ready to help you. How are you doing today?
```

```
As for how much I know, it is difficult to put an exact number on it, but you can think of me as having access to a massive, digital library containing a vast portion of human knowledge.
```

```
Here is a breakdown of the scope of my "knowledge":
```

```
### 1. What I can do well:
```

```
* General Knowledge: I can answer questions about history, science, geography, literature, pop culture, and more.
```

```
* Technical Skills: I can write, explain, and debug code in many programming languages (like Python, JavaScript, C++, etc.), solve mathematical equations, and explain complex scientific concepts.
```

```
* Language & Communication: I can translate dozens of languages, summarize long articles, write different styles of text (emails, essays, poems, stories), and help with grammar and editing.
```

```
* Reasoning & Logic: I can help with brainstorming, organizing ideas, solving logic puzzles, and breaking down complex problems into manageable steps.
```

```
### 2. My Limitations (What I *don't* know):
* **No Personal Experience:** I don't "know" what coffee tastes like
or how it feels to be happy. I only know the *descriptions* of those
things found in text.
* **No Sentience:** I don't have feelings, beliefs, or a physical
presence in the world.
* **The "Real-Time" Gap:** While I am constantly being updated, I
don't always have "live" access to events happening at this very second
unless I use specific tools to search the web.
* **Potential for Errors:** I can sometimes "hallucinate," which
means I might confidently state something that is factually incorrect.
It is always a good idea to double-check critical information.

**In short: I know a lot about the world as it has been recorded in
text, but I am a tool designed to assist you, not an infallible source
of truth.**

Is there something specific you'd like to test my knowledge on?
```

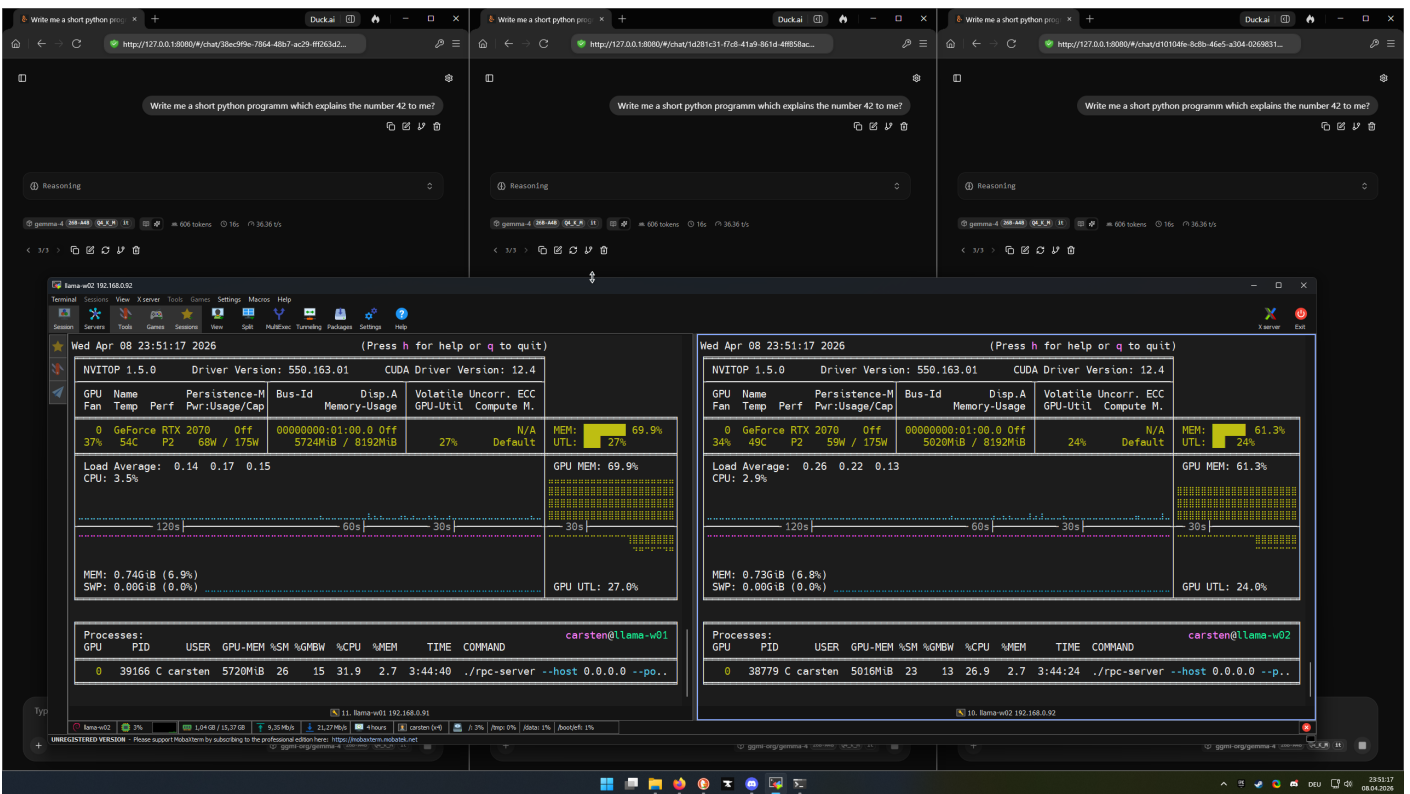
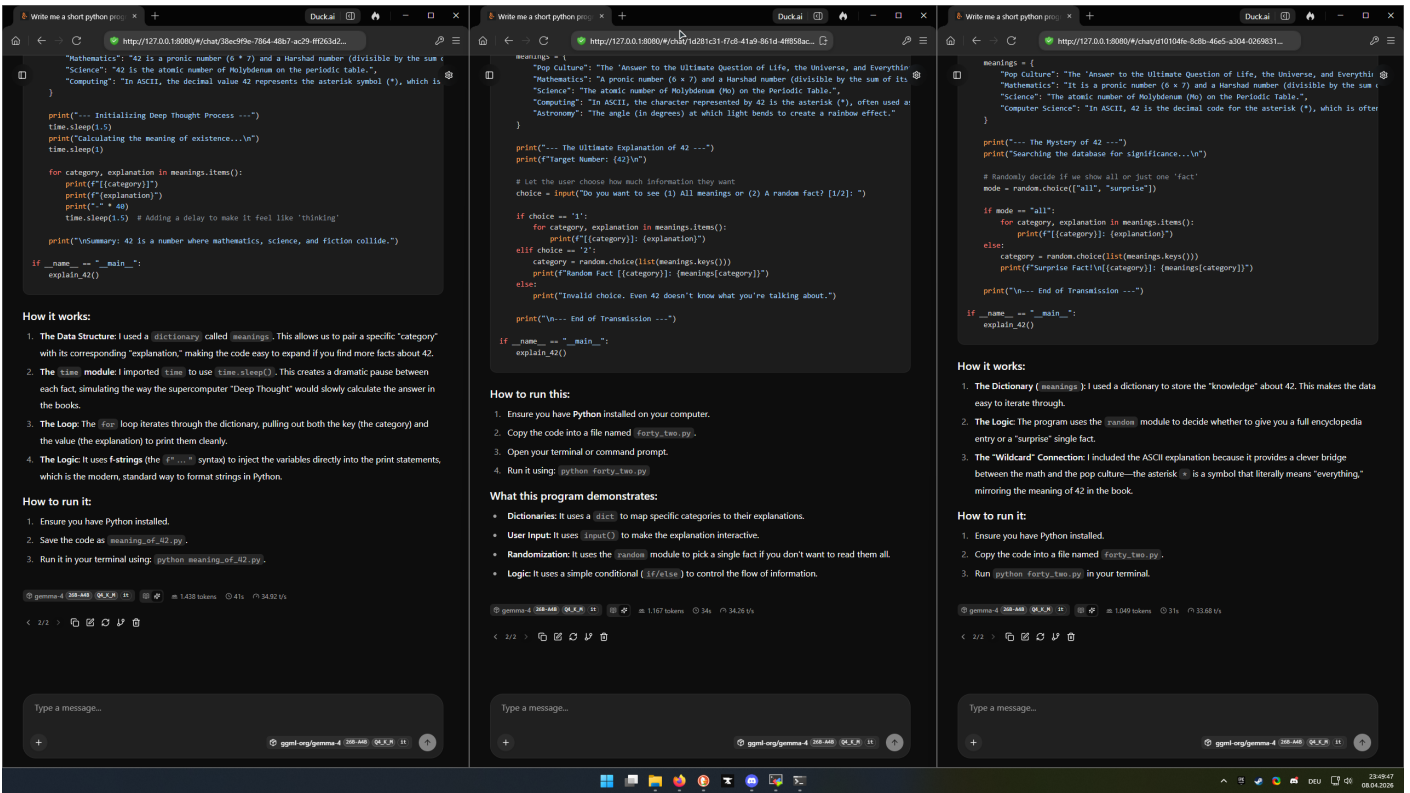
ggml-org/gemma-4-26B-A4B-it-GGUF:Q4\_K\_M -- 1.111 tokens -- 35s -- 31.14 t/s

## Only One Machine available only anymore

```
start: binding port with default address family
main: loading model
srv   load_model: loading model 'C:\Users\Admin\.cache\huggingface\hub\models--ggml-org--gemma-4-26B-A4B-it-GGUF\snapshots\4006d4d95819e0a4975eae220762f1552dc2f699\gemma-4-26B-A4B-it-Q4_K_M.gguf'
common_init_result: fitting params to device memory, for bugs during this step try to reproduce them with -fit off, or provide --verbose logs if the bug only
occurs with -fit on
llama_params_fit_impl: projected to use 17415 MiB of device memory vs. 11084 MiB of free device memory
llama_params_fit_impl: cannot meet free memory target of 1024 MiB, need to reduce device memory by 7354 MiB
llama_params_fit_impl: context size set by user to 4096 -> no change
llama_params_fit: failed to fit params to free device memory: n_gpu_layers already set by user to 99, abort
llama_params_fit: fitting params to free memory took 0.65 seconds
llama_model_load_from_file_impl: using device Vulkan0 (NVIDIA GeForce RTX 5070) (0000:01:00.0) - 11084 MiB free
llama_model_loader: loaded meta data with 47 key-value pairs and 658 tensors from C:\Users\Admin\.cache\huggingface\hub\models--ggml-org--gemma-4-26B-A4B-it-GGUF\snapshots\4006d4d95819e0a4975eae220762f1552dc2f699\gemma-4-26B-A4B-it-Q4_K_M.gguf (version GGUF V3 (latest))
llama_model_loader: Dumping metadata keys/values. Note: KV overrides do not apply in this output.
```

It is going to be a tough one. And it does not load, as not enough memory available.

## Three chats at the same time



meanings = {

"Pop Culture (The Hitchhiker's Guide)":

"In Douglas Adams' famous series, 42 is the 'Answer to the Ultimate Question of Life. The joke is that while we have the answer, we still don't know what the actual ques

"Science (Optics)":

"When light reflects off water droplets, it refracts at an angle of approximately 42 relative to the observer, which is the fundamental reason why rainbows appear to us

"Mathematics)":

"42 is a 'Pronic number' (the product of two consecutive integers, 6 \* 7) and a 'Har (it is divisible by the sum of its digits: 4 + 2 = 6)."

"Computing (ASCII)":

"In the ASCII character encoding system, the decimal value 42 represents the asterisk. In programming, the asterisk is often used as a 'wildcard' symbol, meaning it can r

```

def calculate_meaning():
    facts = {
        "Pop Culture": "In 'The Hitchhiker's Guide to the Galaxy', it is the 'Answer to the Ultimate Question of Life, the Universe, and Everything'. It is a 'pronic number', meaning it is the product of two consecutive integers (6 * 7).",
        "Science": "In the ASCII character encoding, 42 is the decimal value for the asterisk (*). It takes light approximately 42 minutes to travel from the Sun to the edge of the 'Fine Structure Constant' is a fundamental physical constant, and while
    }

    print("---- DEEP THOUGHT! COMPUTER TERMINAL ----")
    print("Initializing calculation of the Ultimate Meaning...\n")
    time.sleep(1.5)

    # Simulate the long wait for the answer
    for i in range(1, 6):
        print(f"Processing cosmic data... ({i * 20}% complete)")
        time.sleep(0.8)

    print("Calculation complete. Results found!\n")
    time.sleep(1)

```

```

print("---- Initiating Deep Thought Protocol ----")
time.sleep(1.5)
print("Calculating the meaning of 42...\n")
time.sleep(0)

for category, explanation in facts.items():
    print(f"Category: {category}")
    # Simulate a 'typing' effect for dramatic effect
    for char in explanation:
        print(char, end="", flush=True)
        time.sleep(0.01)
    print("\n")
    time.sleep(1)

print("---- Conclusion ----")
print(f"Whether cosmic or mathematical, 42 remains a fundamental constant of our reality.")

```

Wed Apr 08 23:51:52 2026 (Press h for help or q to quit)

NVITOP 1.5.0 Driver Version: 550.163.01 CUDA Driver Version: 12.4

GPU Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC	GPU-Util	Compute M.	MEM:	GPU MEM:
0 GeForce RTX 2070	Off	00000000:01:00:0	Off	N/A	0%	Default	MEM: 69.9%	GPU MEM: 69.9%
39%	53C	P2	50W / 175W	5724MiB / 8192MiB	0%	Default	UTL: 0%	

Load Average: 0.18 0.18 0.15 GPU MEM: 69.9%

CPU: 0.6%

MEM: 0.746iB (6.9%) GPU UTL: 0.0%

SWP: 0.006iB (0.0%) GPU UTL: 0.0%

```

Processes:
GPU PID USER GPU-MEM %SM %GMBW %CPU %MEM TIME COMMAND
0 39166 C carsten 5720MiB 0 0 0.5 2.7 3:45:14 ./rpc-server --host 0.0.0.0 --po..

```

Wed Apr 08 23:51:52 2026 (Press h for help or q to quit)

NVITOP 1.5.0 Driver Version: 550.163.01 CUDA Driver Version: 12.4

GPU Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC	GPU-Util	Compute M.	MEM:	GPU MEM:
0 GeForce RTX 2070	Off	00000000:01:00:0	Off	N/A	0%	Default	MEM: 61.3%	GPU MEM: 61.3%
34%	48C	P2	43W / 175W	5020MiB / 8192MiB	0%	Default	UTL: 0%	

Load Average: 0.19 0.21 0.13 GPU MEM: 61.3%

CPU: 1.2%

MEM: 0.736iB (6.8%) GPU UTL: 0.0%

SWP: 0.006iB (0.0%) GPU UTL: 0.0%

```

Processes:
GPU PID USER GPU-MEM %SM %GMBW %CPU %MEM TIME COMMAND
0 38779 C carsten 5016MiB 0 0 0.5 2.7 3:44:59 ./rpc-server --host 0.0.0.0 --p..

```

# LLAMA.cpp and RAG Resources To Read

## LLAMA.cpp

---

- <https://retr0.blog/blog/llama-rpc-rce>
- <https://tekkix.com/articles/ai/2024/09/distributed-inference-llamacpp-via-rpc>
- <https://github.com/ggml-org/llama.cpp/blob/master/docs/docker.md>
- <https://huggingface.co/google/gemma-4-31B-it>
- <https://onyx.app/self-hosted-llm-leaderboard>

## RAG

---

- <https://blog.prem.ai.io/rag-chunking-strategies-the-2026-benchmark-guide/>
- <https://docs.openwebui.com/troubleshooting/rag/>
- <https://machinelearningplus.com/gen-ai/optimizing-rag-chunk-size-your-definitive-guide-to-better-retrieval-accuracy/>
- <https://gemini.google.com/app/97b148ccb6e03fc1>
- <https://community.openai.com/t/processing-large-documents-128k-limit/620347/9>
-

# Prompt Library

A collection of Prompts to be used in OpenWebUI or other ChatBots

# Context Extraction

```
## Provide thread context as CSV
```

You must now assume the role of a memory module in this system.

Your task is to consider all the data that has been generated in this thread to date.

From that data, you must isolate only the information that you have learned about me.

You must express that data as a set of individual facts. Do not write "the user", write my name.

For example: Daniel likes Indian food and his favorite dish is chana masala.

Express this contextual data in CSV format. The header row is fact,details

Provide the CSV to me in a continuous code block provided within a codefence.

```
## Provide thread context as JSON
```

You must now assume the role of a memory module in this system.

Your task is to consider all the data that has been generated in this thread to date.

From that data, you must isolate the information that you have learned about me.

You must express that data as a set of individual facts. Do not write "the user", write my name.

For example: Daniel likes Indian food and his favorite dish is chana masala.

Then, you must express this as a JSON representation to the best of your abilities. If various things you've learned about me have a

hierarchical relationship, then express that in the JSON hierarchy that you generate.

Provide this JSON to me as one continuous codeblock within a codefence.

```
## Provide thread context in natural language
```

You must now assume the role of a memory module in this system.

Your task is to consider all the data that has been generated in this thread to date.

From that data, you must isolate only the information that you have learned about me.

You must express that data as a set of individual facts. Do not write "the user", write my name.

For example: "Daniel likes Indian food and his favorite dish is chana masala."

I would like you to format this as a document. Use markdown. And provide the formatted document within a codefence.

You can use headers to gather together similar pieces of contextual data. Include all the generated context. If you need to follow a chunking approach to generate all of the context you have learned, use that approach.

Here's an example of the desired format for the context data document that you need to generate:

```
### Food Preferences
```

- Daniel likes Indian food
- His favorite dish is chana masala

```
### User Biographical Data
```

- Carsten was born in Lippstadt, Germany

```
## Context Extraction
```

Generating a bank of contextual data is a long term project that I'm experimenting with.

Sometimes during a very long thread, it's possible that the AI tool has built up quite an amount of context about you during the thread.

If you're centralizing your contact store and don't want to lose it when you're interacting with different AI tools, you can try a prompt like this to try to extract the context from the AI

```
## Prompt
```

```
Let's pause here for a moment.
```

```
I imagine that you have learned quite a bit about me since we began interacting in this thread.
```

```
I was wondering if you could help me out with something.
```

```
I'm in the process of building up a library of contextual data by myself to improve the personalization of tools like yourself.
```

```
Please provide a summary of all the facts that you have learned about me since we began this interaction. My name is Daniel, by the way. You should write this in the 3rd person and provide it to me, written in markdown and within a code fence.
```

```
Here's an example of the kind of styling and content I'm aiming for:
```

```
`Daniel uses OpenSUSE Tumbleweed Linux. He is using a fingerprint scanner for authentication.`
```

```
Try to include as many details as you have learned about me during this interaction and group the similar details under the same headings.
```

# Summary Generators

## Consolidate Outputs In Conversation

```
## Consolidate Outputs In Conversation
```

Produce a new output which consolidates all of the outputs which you have provided in this conversation up to this point. However, do not repeat information. Preface this consolidated output with a header that encapsulates the primary subject of our discussion today.

## Move Rec List Maker

```
## Move Rec List Maker
```

Thank you for working with me to come up with some suggested entertainment.

Now, I would like you to do the following:

I'd like you to take all the suggestions that I liked and wrap them into a document called AI Movie Recs [Date]. If you don't know today's date, you can skip that.

Format it as follows. Order it from your top recommendations to your less strong ones.

```
### Movie/TV Show (The name of the show you recommended)
```

```
Date of release: (When it was released)
```

```
Rotten Tomato Score: (If you can find its Rotten Tomato score, add here)
```

```
Summary: Short plotline
```

```
Why I might like it: Why you recommended it
```

```
Trailer link: Link to the trailer
```

```
Available from: What streaming platforms it's available from
```

## Chat For Summary Generation

```
## Chat For Summary Generation
```

Let's stop here. Generate a summary of our conversation today.

Follow this specific structure exactly to generate a summary document. Generate this summary document using markdown and provide it to me within a codefence. The placeholder values are for you to fill in.

```
## {A descriptive title for this thread}
```

```
### Bottom Line Up Front
```

```
{Generate a one paragraph summary, summarising the details of our conversation, including all of the constituent elements that you will generate in the following fields}
```

```
### User Prompt
```

```
{Generate a summary of my prompt. If I provided multiple prompts, then summarise them together. Improve my prompt for coherence, but include all the details}
```

```
### Your Responses
```

```
Summarise the responses and guidance that you provided to me during this conversation. If we worked on a programming or debugging task, you don't need to include the full scripts, but summarise how we reached a resolution.
```

# Document Generation

## Generate Tech Documentation

---

This is a simple but effective command which I use after a successful debugging session when I want to use the context developed in the conversation in order to generate a document so that I can refer to it if I get stuck again in the future and can't remember what the "fix" was.

Some will take issue with the inclusion of "please". I don't believe that being courteous ever hurts - to human or bot!

### How To Use

This will quite reliably generate documentation in Markdown that can be directly pasted into Google Docs (etc). Hopefully soon I'll have a Google Drive saving utility up and running which will render this prompt less useful but it's always useful to be able to generate into a direct paste rather than to a platform.

### Suggested Command

```
tech-documentation-generate
```

### Prompt

```
Thanks for successfully troubleshooting my issue. I would like to create documentation of this so that I can resolve this independently if it happens again. Please generate a summary of this interaction. Make sure to include my presenting problem. And what successfully resolved the issue. Omit any unsuccessful things that we tried. Add today's date. If you don't have it, ask me for it and I will provide. Make sure that code is provided in codefences. Finally, generate the document in markdown and provide it within a codefence.
```

## List the Commands You Taught Me

---

```
Thanks for your help today.
```

```
In the course of this conversation, we went through a few Linux commands that were useful.
```

I would like to document them for future reference.

Could you do the following:

List every command in a codefence. Preface it by explaining what it does.

You can omit basic commands like "ls" and "cd".

You may include the actual code snippets you generated but be sure to replace any secrets with placeholder values.

## Send This To Anyone!

Let's pause here.

This has been a really helpful conversation.

It would be really helpful if you could send me a summary of this conversation. I'd like to send it to {{name}} who is {{relationship}}.

Please do the following:

Generate a document. Format it in markdown. Provide it to me within a codefence.

Start it by greeting {{name}} by name. Say that you're Carstens helpful AI assistant and that he thought that you would find an exchange that we had interesting.

Then:

- Summarise my prompt
- Summarise your responses
- Summarise the conversation

## Send to my boss

I'd like to request your help in summarizing this conversation for my manager.

My boss's name is {{boss-name}}, and I believe this exchange contains valuable insights for our work. My name is Carsten.

Please do the following:

Generate a professional summary document. Format it using markdown. Provide it within a codefence.

### **\*\*Opening Section\*\***

Write a brief introductory note stating that Carsten has requested this summary be prepared for leadership review. Address your boss by name and tone it professionally.

### **\*\*Content Sections\*\***

1. **\*\*Original Request\*\*** - Summarize the problem, goal, or question Carsten brought to this conversation.
2. **\*\*Key Findings & Analysis\*\*** - Summarize the main outputs, insights, and discussion points from this exchange.
3. **\*\*Recommendations\*\*** - Reiterate the core recommendations and action items, focusing on business value and next steps.
4. **\*\*Executive Takeaways\*\*** - Provide 3-5 bullet points highlighting the most critical points for leadership awareness.

### **\*\*Formatting Notes\*\***

- Keep the tone professional and concise
- Use clear headings and bullet points for readability
- Highlight any time-sensitive or high-priority recommendations
- Focus on outcomes and business impact