First-Principles AI Study Plan — Rust Edition (20–40h)

Style
Specs & papers only · slow thinking · implementation-first · no frameworks

Goal
Rebuild the primitive stack behind modern AI in Rust the same way you rebuilt ALUs, hashes, and VMs.

Focus on **execution, memory, invariants** — not performance, hype, or tooling.

Scope

You will rebuild, from scratch:

- Math primitives → gradients → backpropagation
- Attention → multi-head attention
- One Transformer block (forward pass)

Optional extensions add efficiency and control layers.

Final Outputs (non-negotiable)

By the end you must have:

1. A pure-Rust MLP trained via **manual backprop** (XOR + small classification)
2. A pure-Rust attention implementation (Q/K/V + masking + stable softmax)
3. A Transformer block **forward pass** (attention + FFN + residuals + layer norm)
4. Gradient-check tests (finite differences) for MLP and attention

If it's not testable, it's not understood.

Tooling Rules

- Language: **Rust only**
- Data structures: `Vec<f32>` / `Vec<f64>`
- Optional helper crate: `ndarray` (arrays only, **no autograd**)
- No ML frameworks
- No hidden gradients
- No GPU

Phase 0 — Math Foundations (4h)

Concept
Linear algebra is executable structure.

Read (slowly)

- Vectors, matrices, dot products
- Gradients, Jacobians, chain rule
- Log-likelihood, cross-entropy intuition

Supplement (read only if stuck)

- **Book**
  Deep Learning — Ian Goodfellow
- Chapter 2: Linear Algebra
- Chapter 6.2–6.5: Gradients, Jacobians, chain rule
- **Blog (text)**
  Andrej Karpathy — *Micrograd: The spelled-out intro to neural nets*
  (Focus on math + computational graph explanation, ignore Python details)

Build (Rust)

- Matrix multiplication (nested loops)
- Stable softmax
- Cross-entropy loss
- Finite-difference gradient checker

Deliverable

```
math_primitives/
 ├── matmul.rs
 ├── softmax.rs
 ├── loss.rs
 ├── gradcheck.rs
 └── tests/
```

Phase 1 — Backpropagation (6h)

Concept
Backprop is the learning algorithm. Everything stacks on it.

Read

- *Learning Representations by Back-Propagating Errors*
  Rumelhart et al., 1986
  (Focus on algorithm flow, not history)

Supplement

- **Book**
  Neural Networks and Deep Learning — Michael Nielsen
- Chapter 2: How the backpropagation algorithm works
- **Blog (text)**
  Andrej Karpathy — *Yes, you should understand backprop*

Build (Rust)

- 2–3 layer MLP
- Activations: sigmoid + ReLU
- Loss: MSE → cross-entropy
- Manual backward pass using chain rule
- Gradient checking on small networks

Deliverable

```
mlp_from_scratch/
 ├── forward.rs
 ├── backward.rs
 ├── train.rs
 ├── gradcheck.rs
 └── tests/
```

Phase 2 — Attention (6h)

Concept
Attention replaces recurrence with pure linear algebra.

Read

- Attention chapter from a deep learning textbook
  (Focus on Q/K/V, scaling, masking, multi-head logic)

Supplement

- **Book**
  Dive Into Deep Learning
- Chapter 10.1–10.3: Attention mechanisms
- **Blog (text)**
  Jay Alammar — *The Illustrated Transformer*

Build (Rust)

- Scaled dot-product attention
- Causal masking
- Stable softmax

- Multi-head attention
  (reshape → attention → concat → projection)

Deliverable

```
attention_from_scratch/
 ├── attention.rs
 ├── multi_head.rs
 ├── mask.rs
 ├── gradcheck.rs
 └── tests/
```

Phase 3 — Transformer Block (4h)

Concept
Attention + MLP + normalization + residuals

Read

- *Attention Is All You Need*
  (Architecture sections only)

Supplement

- **Book**
  Deep Learning — Ian Goodfellow
- Chapter 9.3: Normalization
- **Blog (text)**
  Sebastian Raschka — *Layer Normalization Explained*

Build (Rust — forward pass only)

- Token embeddings
- Positional encoding (sinusoidal)
- Multi-head attention
- Residual connection + layer norm
- Feed-forward network
- Residual connection + layer norm

Deliverable

```
transformer_block/
 ├── embeddings.rs
 ├── positional_encoding.rs
 ├── transformer_block.rs
 └── tests/
```

Extended Track (Optional — up to 40h total)

Only start after completing core phases.

Extension A — Train a Tiny Transformer (8–10h)

Tasks

- Toy tasks (copy, next-token prediction)
- Rust only
- Expect pain

Supplement

- **Book**
  Deep Learning — Ian Goodfellow
- Chapter 8: Optimization for Training Deep Models
- **Blog (text)**
  Andrej Karpathy — *A Recipe for Training Neural Networks*

Extension B — Distillation (4–6h)

Tasks

- Teacher → student setup
- Soft targets with temperature
- KL loss + supervised loss

Supplement

- **Blog (text)**
  Hugging Face — *DistilBERT Explained*
- **Book**
  Deep Learning — Goodfellow
- Chapter 7.3: Regularization as information control

Extension C — Control / Alignment (4–6h)

Tasks

- Reward model (pairwise ranking)
- Simple policy improvement loop

Supplement

- **Blog (text)**
  Lilian Weng — *RLHF Explained*
- **Blog (text)**
  OpenAI — *InstructGPT* (methodology sections only)

Core Papers & How to Use Them

Rule
- If a paper defines a primitive → **build it**
- If it defines a policy or control layer → **understand it and move on**

Required

- *Learning Representations by Back-Propagating Errors* (1986)
  → Derive equations, then code backprop

- *Attention Is All You Need* (2017)
  → Treat as a CPU architecture spec

Optional

- *DistilBERT* (2019) — optimization layer
- *RLHF* (2022) — context only, read-only

Success Criterion

You can **re-derive modern AI from math and Rust code**, not belief, tooling, or APIs.

If you can build attention with explicit loops, **you understand the system**.