µInference: Research platform for a micro inference server for SL5

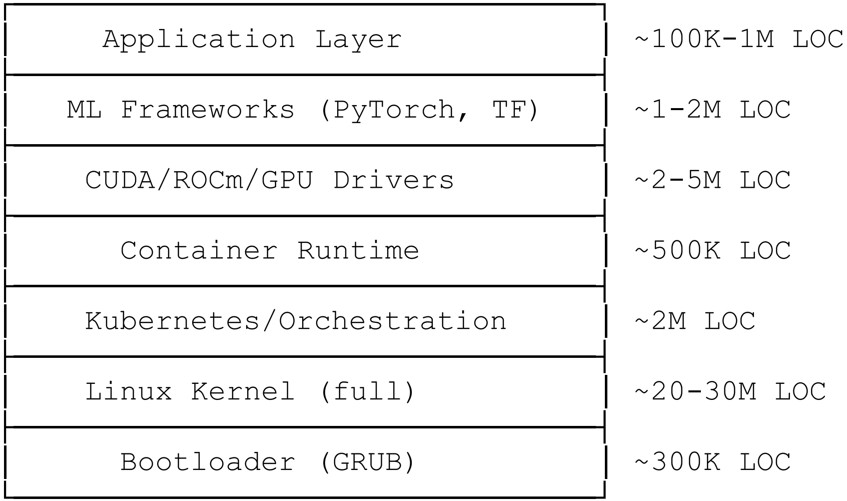
SL5 Security Task Force

What is the smallest inference stack we can build?

From bootloader to inference in minimal lines of code

Current Inference Stack Complexity

Traditional ML Infrastructure Stack

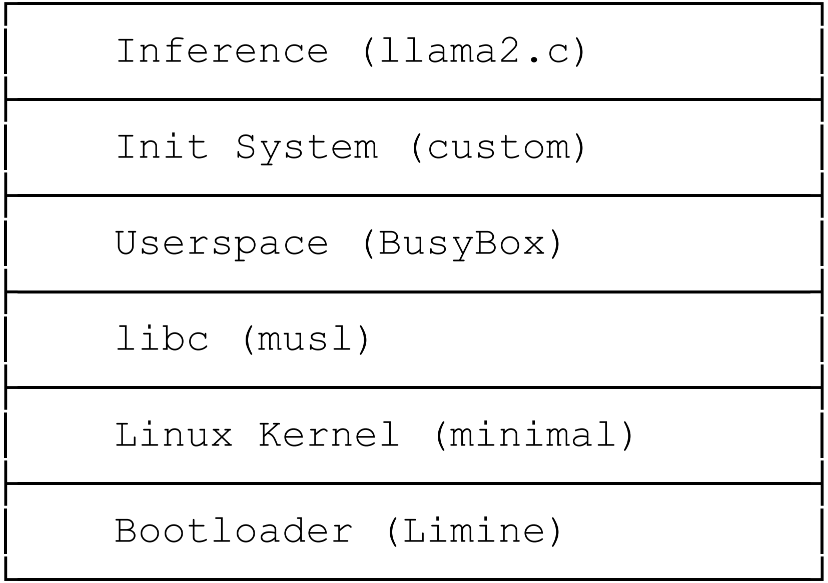


Application Layer	~100K-1M LOC
ML Frameworks (PyTorch, TF)	~1-2M LOC
CUDA/ROCm/GPU Drivers	~2-5M LOC
Container Runtime	~500K LOC
Kubernetes/Orchestration	~2M LOC
Linux Kernel (full)	~20-30M LOC
Bootloader (GRUB)	~300K LOC

Total Estimate: 25-40M LOC + firmware

µInference Stack

Our minimal implementation



Inference (llama2.c)

Init System (custom)

Userspace (BusyBox)

libc (musl)

Linux Kernel (minimal)

Bootloader (Limine)

4,000 LOC

Total: ~1.3M LOC *(3% of traditional stack)*

What we removed from the kernel?

- Network stack
- Filesystems (except tmpfs)
- All drivers except essential for IO

Why Limine?

- Modern UEFI/BIOS support
- Minimal configuration
- Fast boot times
- No unnecessary features

Why musl + BusyBox?

- **musl libc (100K LOC)**
  - 10x smaller than glibc
  - Static linking friendly
  - Clean, modern C implementation
  - No legacy baggage
- **BusyBox (200K LOC)**
  - 300+ Unix utilities in one binary
  - Configurable feature set
  - 1MB static binary
  - Replaces coreutils, util-linux, etc.

Why llama2.c?

- Complete inference in 4,000 lines
- Pure C implementation (and easily portable to Rust with LLMs!)
- Transformer architecture
- BPE tokenizer
- Temperature sampling
- Top-p sampling
- No dependencies
- Runs models efficiently

Achieved Metrics

µInference by the Numbers

Metric	Value
Total LOC	~1.3M
ISO Size	~50MB
Boot Time	<1 seconds
RAM Usage	512MB
Inference Speed	~444 tokens/sec in consumer hardware

Reduction Ratios

- **95%** smaller than typical Linux distro
- **97%** fewer dependencies
- **99%** less attack surface

How Much Smaller Can We Go?

- **Phase 1:**
  - **Kernel Diet (Target: 500K LOC)**
    - Custom minimal kernel config
    - Remove more subsystems
    - Compile-time optimization
- **Phase 2:**
  - **Unikernel Approach (Target: 100K LOC)**
    - Remove userspace
    - Direct hardware access
- **Phase 3:**
  - **Bare Metal (Target: 10K LOC)**
  - No OS, just bootloader + inference
  - Custom memory management
  - Direct hardware control

Scaling Up Capabilities

llama.cpp Integration

- 4-bit quantization support
- GPU acceleration (Vulkan)
- ~50K additional LOC
- 10-100x performance gain
- **GPU Stack**

Security Through Minimalism

Attack Surface Reduction

- 97% less code = 97% fewer bugs
- No network stack = no remote exploits (There is plenty of IO options)
- No filesystem = no persistence
- Read-only system = immutable

Security Features Possible

- Measured boot with TPM
- Memory encryption
- Secure enclave execution
- Formal verification (small enough, see seL4)

Where Minimal Inference Matters

- Air-gapped systems
- High-security facilities
- Compliance-heavy industries
- Research sandboxes

What This Proves

1. **Frontier labs don’t require millions of LOC**
   - Core inference is surprisingly simple
   - Complexity is in the ecosystem
2. **Security through simplicity is achievable**
   - Small enough to audit
   - Small enough to formally verify
3. **Edge ML is practical today**
   - Runs on minimal hardware
   - No cloud dependency

Research Directions

- **Immediate:**
  - Add llama.cpp backend
  - Implement secure boot
- **Medium-term:**
  - Unikernel design
  - Deliberate Hardware stack
  - Formal verification
- **Long-term:**
  - ???

Key Takeaways

✓ **1.3M LOC** total (vs 25-40M traditional)

✓ **50MB** complete system

✓ **Fully functional** LLM inference

✓ **Orders of magnitude** simpler

✓ **Auditable** and potentially **verifiable**