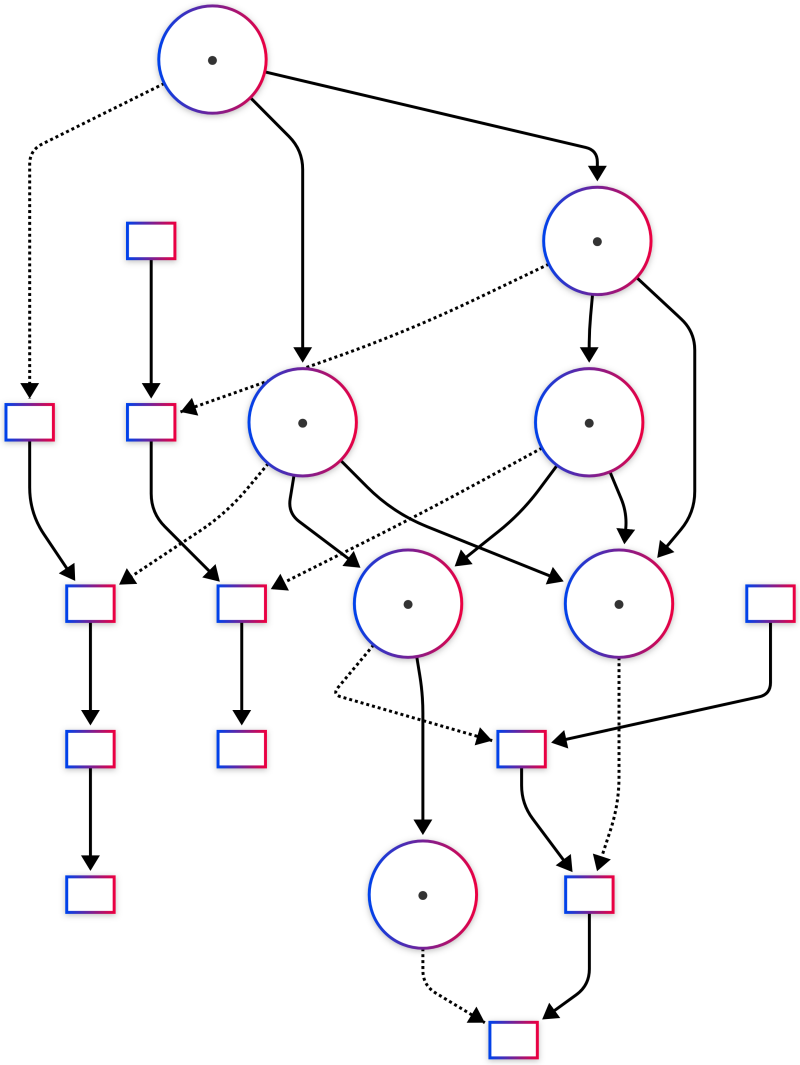
MKCORE: FPGA-Based Domain-Specific Inference Engine

Enrique Flores

August 1, 2025



Introduction

Despite their astounding capabilities, large language models (LLMs) remain shackled to centralized datacenters, requiring high-wattage GPUs, vast cooling systems, and complex runtime stacks to operate. This makes them fundamentally incompatible with edge deployments—especially in tactical or battlefield environments where **efficiency must rise as power availability falls**. In these austere, hostile domains, there is no room for latency, thermal overhead, or software abstraction layers. What's needed is a new paradigm: **MKCORE**, a hardened, power-efficient compute core that executes domain-specific LLM inference with no GPU, no operating system, and no excess. It transforms transformer logic into pure digital circuitry, enabling **LLM-grade cognition in a box the size of a fist, running on watts, not kilowatts**.

Background

Modern large language models (LLMs) typically require general-purpose CPUs and high-end GPUs to run. This imposes constraints on power, cost, and mobility, making such models unsuitable for embedded or tactical systems. Tactical environments require lightweight, deterministic, and tamper-proof compute elements capable of running domain-specific inference models in real-time, even in constrained or adversarial conditions.

Summary of Invention

This invention discloses a method and architecture for implementing domain-specific LLM inference using an FPGA-only pipeline, with external RAM and SSD storage for model data. The architecture eliminates the need for a traditional CPU or GPU, enabling real-time, deterministic inference in small, low-power, field-deployable packages.

System Architecture

MKCORE is built around a highly deterministic pipeline that translates high-level model logic into physical circuit behavior. This architecture consists of the following components:

1. **LLM Compilation Pipeline**:

   The LLM compilation flow begins with a domain-specific transformer model trained using conventional frameworks like PyTorch or TensorFlow. The model undergoes the following stages:

   - *Tokenization Freezing*: The tokenizer is fixed at compile time to reduce logic variability. Common domain tokens (e.g., chess FEN strings, tactical commands) are embedded as constants.

- *Quantization*: All weights are quantized to 4-bit fixed-point values using static calibration. Optional sparsity pruning and group-wise quantization are applied to minimize logic footprint.
- *Graph Partitioning*: The transformer graph is split into hardware-mappable subgraphs — e.g., multi-head attention, MLP layers, and normalization units — each mapped to reusable logic blocks.
- *Logic Emission*: Each subgraph is converted into Verilog or VHDL logic using a custom compiler backend. Linear layers are implemented as bit-serial matrix-vector multipliers; attention mechanisms are mux-based comparators.
- *Bitstream Generation*: The synthesized logic is packed into an FPGA bitstream, while weight tensors are flattened and serialized for SSD or flash streaming.

2. **FPGA Inference Engine**:

   The FPGA hosts multiple pipelined processing elements, each dedicated to a component of the transformer:

   - **Embedding Engine**: Converts token indices into fixed vector encodings using ROMs or lookup tables.
   - **Attention Pathways**: Implement key-query dot product using shift-and-add logic. Scores are thresholded and used to select relevant value vectors via hardware muxes.
   - **MLP Layer**: Dense layers are mapped to pipelined MAC arrays operating in fixed-point mode.
   - **Layer Norm and Residuals**: These are approximated with precomputed scaling coefficients and pass-through buffers.
   - **Token Decoder**: Outputs token logits and selects the highest-ranked output directly in hardware.

3. **SSD or Flash Storage**:

   A QSPI or NVMe flash device holds model parameters, attention masks, and decoder lookup tables. The FPGA fetches parameter blocks into RAM as needed using a double-buffered DMA scheme to ensure continuous dataflow without pipeline stalls.

4. **RAM Subsystem**:

   External DRAM (e.g., DDR3 or DDR4) serves as a scratchpad for intermediate activations and history buffers. The system uses a fixed memory map and pipelined access to minimize latency and eliminate dynamic allocation.

5. **Optional Control Core**:

   A lightweight controller (e.g., an ARM Cortex-M or RISC-V MCU) may be used to:

   - Load the FPGA bitstream and model weights during boot.

- Receive user input or telemetry (e.g., via UART, SPI, or CAN bus).
- Trigger inference cycles and collect responses.

Crucially, no operating system or kernel is required, and the inference pipeline runs asynchronously once initialized.

Inference as Physical Logic

Traditional LLM inference on GPUs or CPUs relies heavily on floating-point arithmetic, dynamic memory allocation, and generalized control flow. These models depend on complex software stacks to interpret tensors, schedule compute kernels, and apply nonlinear activations. This process incurs significant power and latency overhead, and limits real-time deployment in edge or tactical environments.

In contrast, MKCORE transforms the model into a deterministic pipeline of logic gates, routing networks, and bit-serial computation units. Every decision the model could make is structurally encoded in silicon — not computed anew on every token, but pre-baked into the FPGA fabric itself. Decision boundaries, activations, and attention patterns are reinterpreted as **truth tables, routing switches, and bit pipelines**, making inference a function of pure signal propagation.

For example, rather than representing a softmax activation as an exponential approximation followed by normalization (requiring costly arithmetic), MKCORE implements thresholded routing: input tokens activate only the top-ranked paths, effectively short-circuiting attention to the most probable outcome. Similarly, quantized matrix multiplies become preconfigured shift-and-add trees with no loops, no scheduling, and no memory indirection.

The result is a model where every inference path is laid out spatially, in gates and wires. Activations become voltages; attention becomes muxed dataflow. This **physicalization** of inference logic collapses multiple layers of software indirection into a single pass of hardware-encoded reasoning — eliminating operating system dependency, floating-point math, memory fragmentation, and dynamic scheduling altogether.

This architectural shift transforms machine learning from a simulation of cognition into a **circuit realization of cognition**, enabling unprecedented performance-per-watt, deterministic timing, and compact form factors for mission-critical inference at the edge.

Chess Inference Pipeline

A tokenized chess FEN string such as `r1bqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1` is mapped to 64 square tokens and auxiliary control tokens. These are streamed into the embedding engine via a FIFO buffer. The FPGA performs the following:

- Embeds the position with a fixed vector encoding
- Passes embeddings through 8 transformer blocks compiled to logic
- Selects the most probable move (e.g., `e2e4`) via the decoder logic

The result is returned within ˜0.5 ms with no reliance on software or floating-point arithmetic.

Use Cases

- Edge inference for military drones or robotic scouts.
- Onboard decision-making for aerospace platforms.
- Private, secure LLM processing in surveillance or communications equipment.

Comparison Tables

Component Cost Comparison

Component	**Description**	**MKCORE (USD)**	**GPU Rig (USD)**
Compute Unit	Mid-range FPGA (e.g., Kintex-7)	\$150	\$3,000 (A100 GPU)
Memory	128GB DDR4 RAM	\$60	\$600
Storage	1TB SSD (M.2 NVMe)	\$40	\$100
Power Supply	12V 5A PSU	\$20	\$150
Microcontroller	ARM Cortex-M or RISC-V SoC	\$15	\$100
Interconnects	Custom backplane, GPIO, QSPI, etc.	\$10	\$50
Total		**\$295**	**\$4,000**

Table 1: Component cost comparison between MKCORE and GPU-based LLM rig.

Power Consumption Comparison

Component	**MKCORE Power (Watts)**	**GPU Rig Power (Watts)**
Compute Unit	15 W	300 W
Memory	10 W	40 W
Storage	5 W	8 W
Controller	3 W	15 W
Misc Overhead	5 W	20 W
Total	**38 W**	**383 W**

Table 2: Typical power consumption during LLM inference.

Feature Comparison

Feature	MKCORE	GPU-based LLM Rig
Size	Handheld / Embedded	Desktop / Rack-mounted
Latency	Sub-ms (fixed datapath)	10–100 ms
Throughput	Moderate	High
Power Efficiency	Extremely high	Moderate
Field Deployable	Yes (air-dropped, UAV, IoT)	No (static infrastructure)
Upgradability	Firmware-based	Software-based
Inference Flexibility	Domain-specific	General-purpose
Training Capability	Limited / External Only	Fully capable

Table 3: Feature comparison between MKCORE and conventional GPU-based LLM systems.

Compiler Toolchain

The MKCORE compiler stack accepts a pre-trained, quantized transformer model in ONNX or PyTorch format. It performs the following transformations:

1. **Graph Lowering**: The high-level model graph is lowered into a minimal intermediate representation (IR) using a static computation DAG with no dynamic branching.
2. **Operation Fusion**: Compatible subgraphs (e.g., MatMul + Add + Activation) are fused into a single block for pipelined logic emission.
3. **Static Scheduling**: All tensor operations are statically scheduled and memory access patterns are flattened for streaming into logic blocks.
4. **HDL Emission**: Verilog or VHDL is generated for each operation using a library of reusable parameterized modules (e.g., shift-add units, quantized matrix multipliers, mux-based attention).
5. **Weight Serialization**: Weights are exported in binary format with a known memory layout to facilitate external RAM loading and prefetch.

This compiler transforms neural network structure into fully routable logic circuits, bypassing the software runtime entirely.

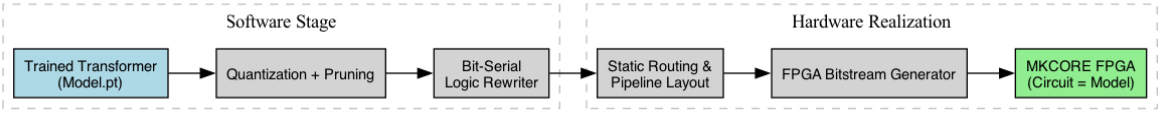


Figure 1: MKCORE Compilation Pipeline: From Transformer Model to FPGA Circuit

The MKCORE compilation pipeline converts a trained transformer model into a deterministic hardware circuit suitable for low-power deployment. It begins with a domain-specific model trained in PyTorch or TensorFlow. This model is passed through a quantization and pruning stage that compresses its parameters into 4-bit fixed-point representations. Next, the model graph is rewritten into bit-serial arithmetic blocks using a static, pipelined architecture. The compiler then schedules routing paths, flattens control flow, and emits logic in Verilog or VHDL. A bitstream generator produces the final FPGA configuration. Once flashed, the FPGA becomes a fixed-function inference device—executing LLM behavior with no software stack, no operating system, and no runtime interpreter.

Tamper Resistance and Secure Boot

MKCORE is designed to operate in hostile and adversarial environments. All logic is encrypted at rest and decrypted into SRAM upon boot via a secure key stored in OTP (one-time programmable) memory. Boot process can be initiated via physical jumper or secure channel. Tamper sensors (voltage, temperature, timing) trigger immediate shutdown.

The lack of dynamic code execution further hardens the system: no stack, heap, or syscall surface exists to exploit. The result is a trusted inference module capable of operating with integrity in environments where conventional systems fail.

Threat Model and Resilience

MKCORE is designed for operation in contested environments where adversaries may attempt to degrade, disable, or compromise electronic systems through physical, electrical, or cyber means. Unlike conventional compute systems reliant on general-purpose operating systems and dynamic execution models, MKCORE employs a fixed-function inference datapath encoded directly in FPGA logic, making it resilient to many forms of attack.

Denial-of-Service Resistance

Because MKCORE contains no OS kernel, no syscall interface, and no dynamic memory allocator, the attack surface for software-based denial-of-service is virtually eliminated. There is no scheduler to overwhelm, no process table to overflow, and no instruction stream to corrupt. The inference process is driven by a pure signal path from input to output with no conditional branches or timing variability.

Tamper and Intrusion Detection

Optional voltage, temperature, and timing sensors detect physical tampering or abnormal environmental conditions. In such cases, the system can shut down instantly or destroy sensitive keys in OTP (One-Time Programmable) memory. Because inference logic is burned into a volatile SRAM-based FPGA, any loss of power instantly clears the configuration—leaving no extractable model state.

Electromagnetic Hardening (EMP)

MKCORE's minimal component count and low-power, spatially sparse architecture makes it an ideal candidate for electromagnetic pulse (EMP) hardening. By avoiding high-density silicon like GPUs or multicore CPUs, the system exhibits lower cross-sectional vulnerability to EMP transients. Further, since inference requires no external DRAM refresh or active software process, the system can be hardened with metal shielding and passive filters to withstand electromagnetic disruption.

Supply Chain Security

The entire inference pipeline is pre-compiled and encrypted at rest. No third-party drivers, operating systems, or cloud APIs are required. The hardware logic is verifiable, reproducible, and bootstraps securely from a single trust root. This makes MKCORE suitable for deployment in sensitive, classified, or hostile regions where reliance on foreign cloud services or opaque binaries is unacceptable.

Jitter-Free Determinism

MKCORE offers exact, cycle-deterministic inference. For time-critical operations—such as signal decryption, drone control, or real-time threat classification—this means responses are predictable down to the clock. Unlike GPU-accelerated systems which suffer from unpredictable kernel latencies, MKCORE's token-to-token latency remains constant under all loads.

Future Work

The MKCORE architecture represents a foundational shift in how language models and neural inference systems can be deployed—moving from dynamic, software-defined pipelines to fully static, logic-encoded silicon flows. Several promising avenues remain for further research and development:

1. Support for Non-Transformer Architectures

While the current pipeline targets transformer-based LLMs, the same principles can be extended to:

- **Recurrent Neural Networks (RNNs)**: Implementing LSTM or GRU logic using looped state-holding modules and pipelined gating units.
- **Convolutional Neural Networks (CNNs)**: Convolution kernels can be emitted as shift-register windows and fused MAC trees, enabling embedded image classification, object detection, or OCR at the edge.

- **Graph Neural Networks (GNNs)**: Routing logic for neighbor aggregation could be statically scheduled in hardware, supporting embedded reasoning over sparse relational data.

2. Dynamic Partial Reconfiguration

Future versions of MKCORE may support partial reconfiguration, allowing sections of the FPGA fabric to be dynamically reprogrammed while the core inference pipeline remains active. This would enable:

- On-demand swapping of model heads, vocabularies, or domain-specific adapters.
- Field updates without full system reboot.
- Resource sharing among multiple mission-specific tasks.

3. MKCORE Logic Libraries for Ada and VHDL

MKCORE will provide a suite of reusable, parameterized libraries written in **standard VHDL** and optionally **Ada** for logic-side coordination, targeting existing toolchains like GHDL, ModelSim, and vendor synthesis suites (Xilinx Vivado, Intel Quartus).

This library suite will include:

- **Fixed-Point Matrix Units**: Bit-serial or parallel multiply-accumulate blocks with configurable precision and quantization scaling.
- **Attention Selector Units**: Pre-compiled key-query comparators, priority encoders, and muxed value selectors tailored for sparse or windowed attention.
- **Token Embedding ROMs**: Pre-generated lookup tables compiled from frozen tokenizer vocabularies.
- **LayerNorm Approximators**: Lookup-based or pipelined fixed-scaling modules for normalization and residual fusion.
- **Inference Scheduler**: An Ada state machine that coordinates token advancement, memory streaming, and result collection across hardware modules.

This approach ensures compatibility with existing aerospace and defense tooling environments, which often mandate the use of Ada or VHDL for mission-critical systems. It also allows for easier certification, test coverage, and documentation under standard DO-254 and related safety assurance frameworks.

Future releases of the MKCORE toolkit will include:

- A set of `gnatmake`-compatible Ada packages for compile-time configuration of inference graphs.

- VHDL component templates with well-defined generics for instantiating reusable neural building blocks.
- Optional bindings to support co-simulation or formal verification workflows using tools like OSVVM and UVVM.

Example MKCORE Library Components

1. VHDL: Bit-Serial Matrix-Vector Multiplier (4-bit Quantized)

This component implements a 4-bit signed dot product using bit-serial MACs. The entire layer can be pipelined for throughput, or serialized to reduce area.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity MatrixVecMul is
  generic (
    N_ROWS : integer := 8;
    N_COLS : integer := 8
  );
  port (
    clk     : in  std_logic;
    start   : in  std_logic;
    vec_in  : in  std_logic_vector(N_COLS*4-1 downto 0);  -- 4-bit input vector
    mat_in  : in  std_logic_vector(N_ROWS*N_COLS*4-1 downto 0); -- 4-bit weights
    result  : out std_logic_vector(N_ROWS*8-1 downto 0);   -- 8-bit accum results
    done    : out std_logic
  );
end MatrixVecMul;

architecture Behavioral of MatrixVecMul is
  signal acc   : integer_vector(0 to N_ROWS-1);
  signal count : integer range 0 to N_COLS := 0;
begin
  process(clk)
  begin
    if rising_edge(clk) then
      if start = '1' then
        acc <= (others => 0);
        count <= 0;
      elsif count < N_COLS then
        for r in 0 to N_ROWS-1 loop
          -- Extract 4-bit weight and input
```

```
          variable w : signed(3 downto 0);
          variable x : signed(3 downto 0);
          w := signed(mat_in((r*N_COLS+count)*4+3 downto (r*N_COLS+count)*4));
          x := signed(vec_in(count*4+3 downto count*4));
          acc(r) <= acc(r) + to_integer(w * x);
        end loop;
        count <= count + 1;
        if count = N_COLS-1 then
          done <= '1';
        end if;
      end if;
    end if;
  end process;

  result <= std_logic_vector(concat_all(acc)); -- Flattened 8-bit outputs
end Behavioral;
```

2. Ada: Inference Controller Package (Token Scheduler)

This Ada package manages the step-by-step operation of the inference pipeline by scheduling token input, launching matrix multiply units, and reading output logits.

```
package MKCORE_Controller is
  type Inference_State is (Idle, LoadInput, RunAttention, RunMLP, Decode, Done);

  procedure Init;
  procedure Tick; -- Called on each FPGA system clock
  function Get_Output return String;
end MKCORE_Controller;

package body MKCORE_Controller is
  Current_State : Inference_State := Idle;
  Token_Buffer  : array (0 .. 63) of Character;
  Output_Token  : String(1 .. 4);

  procedure Init is
  begin
    Current_State := LoadInput;
    -- Zero memory, prefetch weights, etc.
  end Init;

  procedure Tick is
  begin
    case Current_State is
      when LoadInput =>
```

```
        -- DMA input tokens into embedding block
        Current_State := RunAttention;

      when RunAttention =>
        -- Trigger MatrixVecMul for QK^T and V
        -- Wait for done signal
        Current_State := RunMLP;

      when RunMLP =>
        -- Execute MLP logic
        Current_State := Decode;

      when Decode =>
        -- Choose top logit
        Output_Token := "e2e4"; -- Example output
        Current_State := Done;

      when others =>
        null;
    end case;
  end Tick;

  function Get_Output return String is
  begin
    return Output_Token;
  end Get_Output;

end MKCORE_Controller;
```

3. Toolchain Integration

The above pseudocode libraries are designed to work with standard tools:

- **GHDL / ModelSim**: Simulation and waveform debug.
- **GNAT**: Compiling Ada packages for embedded control cores.
- **Vivado / Quartus**: Synthesis and bitstream generation.
- **OpenOCD / JTAG**: Secure firmware deployment and debug hooks.

Result: All model logic is compiled into deterministically verifiable hardware + Ada coordination routines — deployable on air-gapped systems, drones, or tamper-resistant modules.

4. Federated MKCORE Networks

Multiple MKCORE units could form distributed inference networks, each handling part of a larger model or dataset. A mesh or bus-based protocol could allow FPGAs to exchange intermediate activations or voting decisions—enabling swarm-level cognition or federated intelligence in sensor nets, autonomous vehicles, or mobile UAV fleets.

5. Radiation-Hardened MKCORE Variants

For aerospace or high-radiation environments, future versions of MKCORE could be built using radiation-tolerant FPGAs (e.g., Microsemi RTG4, Xilinx XQR series). Logic modules could be triplicated using TMR (triple modular redundancy), and ECC memory controllers added to DRAM buffers, making MKCORE deployable on satellites, missiles, or planetary rovers.

6. Hardware-Embedded Training and Adaptation

Though MKCORE is inference-only by design, lightweight forms of adaptation could be supported:

- **Contextual Fine-Tuning**: Using on-chip RAM to cache token statistics, adapt embedding scales, or perform LoRA-style residual tweaks without retraining core weights.
- **FPGA-Based Gradient Accumulation**: Prototyping low-rate, quantized training modules that could run overnight or in-the-field under supervision.

7. Open-Source Reference Designs and Education Kits

To support adoption, stripped-down MKCORE boards and toolchains can be published under permissive licenses. These educational kits would allow students, hobbyists, and defense contractors to explore FPGA-native LLM inference, helping demystify hardware compilation and bringing deterministic AI into the mainstream.

Civilian and Humanitarian Applications

- **Precision Agriculture and Crop Health Monitoring**
  Deployed across large farmlands, MKCORE-powered sensor nodes can process multispectral drone imagery locally to:
  - Detect crop stress, disease, or pest patterns in real-time
  - Adjust irrigation or pesticide deployment autonomously
  - Reduce bandwidth by transmitting only critical updates

  This enables low-cost AI for farmers in regions without reliable internet or power.

- **Air Quality and Environmental Surveillance**
  Urban and rural areas alike can benefit from MKCORE-enabled air sensors that:
  - Detect particulates, NOx, CO_2, or VOC anomalies
  - Perform AI-based pattern recognition to correlate spikes with human activity or weather patterns
  - Alert authorities only when thresholds or new patterns emerge

  With inference done at the sensor, there's no need for constant cloud connectivity.

- **Disaster Response and Damage Assessment**
  After hurricanes, earthquakes, or floods, deploying drone swarms or fixed-position MKCORE units enables:
  - Local image and sound analysis to find survivors
  - Prioritization of areas for rescue without overwhelming central servers
  - Resilient coordination even when comms are degraded

  This creates an "AI first responder net" that doesn't rely on infrastructure.

- **Urban Safety and Crime Detection**
  MKCORE modules embedded in light poles or transit stations can:
  - Detect fights, gunshots, or crowding
  - Run anonymized behavior models to avoid privacy violations
  - Alert local law enforcement or EMS only on verified anomalies

  Unlike CCTV + cloud AI, this respects latency and privacy while enhancing urban resilience.

- **Poaching Prevention and Wildlife Monitoring**
  In conservation zones, solar-powered MKCORE nodes can:
  - Identify gunshots or motorbike sounds in protected areas
  - Distinguish between ranger vs. poacher activity
  - Trigger silent alerts to mobile anti-poaching teams

  All without revealing their location or depending on satellite links.

Strategic Deployment Scenarios

- **Orbital Intelligence Filtering (LEO AI Satellites)** MKCORE enables onboard transformer-class inference in microsatellites, allowing real-time analysis of ISR data without needing round-trip communications to Earth. This dramatically reduces downlink congestion, enabling:

  - Onboard filtering of hyperspectral or SIGINT feeds
  - Local decision-making for orbital swarm coordination
  - Triggering of burn commands or camera focus only on anomalous events

  By collapsing inference into physical circuits, MKCORE empowers satellites to act as autonomous agents, not passive sensors.

- **Autonomous Naval Reconnaissance (Submerged Drones)** In maritime theaters, MKCORE cores embedded in small, expendable underwater drones or sensor pods allow:

  - Pattern recognition of surface vessels, sonar pings, or communications chatter
  - Self-directed mission logic without uplink
  - Stealth recon in GPS-denied, EM-hostile waters

  Submarines may deploy clouds of AI-guided probes without signature, each with a specific mission—surveillance, jamming, or even deception.

- **Distributed Psychological Warfare (Smart Mines)** MKCORE enables the creation of **cognitive mines** — passive munitions that:

  - Detect enemy radio chatter, uniforms, or vehicle engine sounds
  - Respond using locally generated voice prompts, mimicry, or confusion tactics
  - Delay detonation until optimal psychological effect or force-multiplier opportunity

  These mines don't just explode — they talk first. Using embedded NLP, they can impersonate friendly units, generate misleading instructions, or feign surrender to create hesitation or emotional response. This is the first step toward **psychological shaping munitions**.

- **Air-Dropped Inference Mesh (Contested Urban Zones)** In dense or contested environments, MKCORE units can be dispersed via UAV, artillery shell, or loitering munition into rooftops, alleyways, or ventilation shafts. Once embedded, each node:

  - Conducts real-time audio/visual parsing
  - Filters relevant behavioral patterns (e.g., target identification, group formation)
  - Relays only high-value events to central command via mesh radio

  This turns dumb sensors into **edge-native cognitive sentinels**, reducing bandwidth and analyst fatigue while increasing actionable intelligence throughput.

- **Battlefield Roleplay Systems (Deceptive AI Agents)** With sufficient training, MKCORE can power compact LLMs that simulate:

  - Local civilians (for Turing-class patrol interrogation drills)
  - Captured combatants (for extracting response protocols)

– Realistic enemy chatter (to mask friendly ops)

These agents can be deployed in training simulators or live theaters as part of red team deception operations.

Conclusion

MKCORE represents a radical departure from the conventional approach to machine learning deployment. By collapsing the abstraction layers between model and machine, and reimagining inference as physical logic rather than floating-point software emulation, MKCORE delivers unmatched performance-per-watt and battlefield resilience. Whether embedded on autonomous drones, buried in environmental sensors, or orbiting aboard microsatellites, its reconfigurable and secure architecture opens the door to real-time, local AI across every domain—without the burden of centralized compute or continuous connectivity. As global demand surges for trustworthy, lightweight, and mission-adaptable intelligence systems, MKCORE offers a scalable foundation for the next generation of cognitive infrastructure at the edge.

Claims

Claim 1: A system for performing domain-specific language model inference using only FPGA logic, external memory, and SSD storage.

Claim 2: A method for compiling transformer models into bit-serial logic suitable for FPGA implementation.

Claim 3: A storage schema and memory layout for streaming quantized weights into FPGA-based inference logic with minimal overhead and deterministic timing.

Claim 4: A tamper-resistant, software-free execution path for embedded language model inference, eliminating reliance on operating systems, drivers, or dynamic memory allocation.

Claim 5: A tactical compute fabric wherein transformer inference is performed without CPUs or GPUs, designed to integrate within MILSPEC-compliant systems including drones, munitions, mobile command units, and hardened edge compute devices.