Pre-Generated Conversation Matrix with Caching (PCM)

Technical Architecture Specification & Deep-Dive Whitepaper

Inventors & Systems Architects: Robert Rogers & Adrian Chadd | **Entity:** MMAP Labs (mmaplabs.com)
Provisional Patent Specification Reference: US 64/105,371 | **Date:** August 2026

Executive Overview: The AI Hardware Bottleneck & Our Solution

Modern interactive Artificial Intelligence (AI) and Natural Language Processing (NLP) models operate like a chef cooking every single meal from scratch the moment a customer places an order. They generate responses dynamically word-by-word, token-by-token at runtime through continuous, autoregressive floating-point calculations. On resource-constrained edge hardware—such as smart glasses, augmented reality (AR) headsets, smart rings, and wearables—this dynamic computation creates an insurmountable 'Inference Wall.' It drains battery life rapidly, causes uncomfortable device heating against the user's face or body, saturates memory channels, and frequently introduces probabilistic flaws known as 'hallucinations.'

Furthermore, traditional consumer wearables rely entirely on mandatory cloud streaming. When a user enters a parking garage, basement, subway, or cellular dead zone, traditional cloud-dependent wearables fail completely, leaving the user with an unresponsive device.

The **Pre-Generated Conversation Matrix with Caching (PCM)** (Provisional Patent US 64/105,371) fundamentally changes the operational paradigm. PCM acts like a master prep system where an exhaustive matrix of natural language responses, contextual variants, and dialogue trees are pre-computed offline by a massive 'Teacher Model.' At runtime on the local device, a compact neural network (such as a 1B or 2B local edge LLM) acts strictly as an intent classifier that points directly into a pre-compiled binary knowledge matrix stored on local flash storage. Using operating system virtual memory mapping (mmap), the device instantly retrieves the exact pre-written answer into active memory on-demand with zero dynamic token generation overhead.

> ***Layman Translation:*** *Instead of forcing a tiny processor inside smart glasses to write a 50-word response token-by-token while draining battery and heating up, the local AI acts as a smart index librarian. It understands what the user wants in a fraction of a second, and the operating system instantly flips to the exact pre-written page on storage.*

1. Hardware Physics & The Wearable Compute Bottleneck

Deploying conversational AI onto low-power consumer wearables presents severe physical and architectural challenges:

- **Micro-Watt Thermal & Power Budgets:** Smart glasses and AR headsets operate under strict Thermal Dissipation Factor (TDF) limits. Continuous GPU/NPU floating-point decoding generates localized heat against the temples and face while rapidly draining small 150mAh–400mAh wearable batteries.
- **Memory Bandwidth & Channel Saturation:** High-parameter LLM weights saturate memory bus channels during autoregressive decoding passes. This forces low-tier edge chips to execute heavy model quantization (e.g., 2-bit or 3-bit quantization), which severely degrades output coherence and speech quality.

- **Cadence & Network Latency:** Interactive heads-up displays (HUDs) and spatial audio demand rapid, continuous conversational cadence. Cloud-dependent round trips disrupt natural human interaction and render the interface unusable when cellular connections fluctuate.

By decoupling offline generative reasoning from online response delivery, the Pre-Generated Conversation Matrix eliminates continuous token decoding loops for routine queries, enabling rich conversational capabilities on micro-watt power budgets.

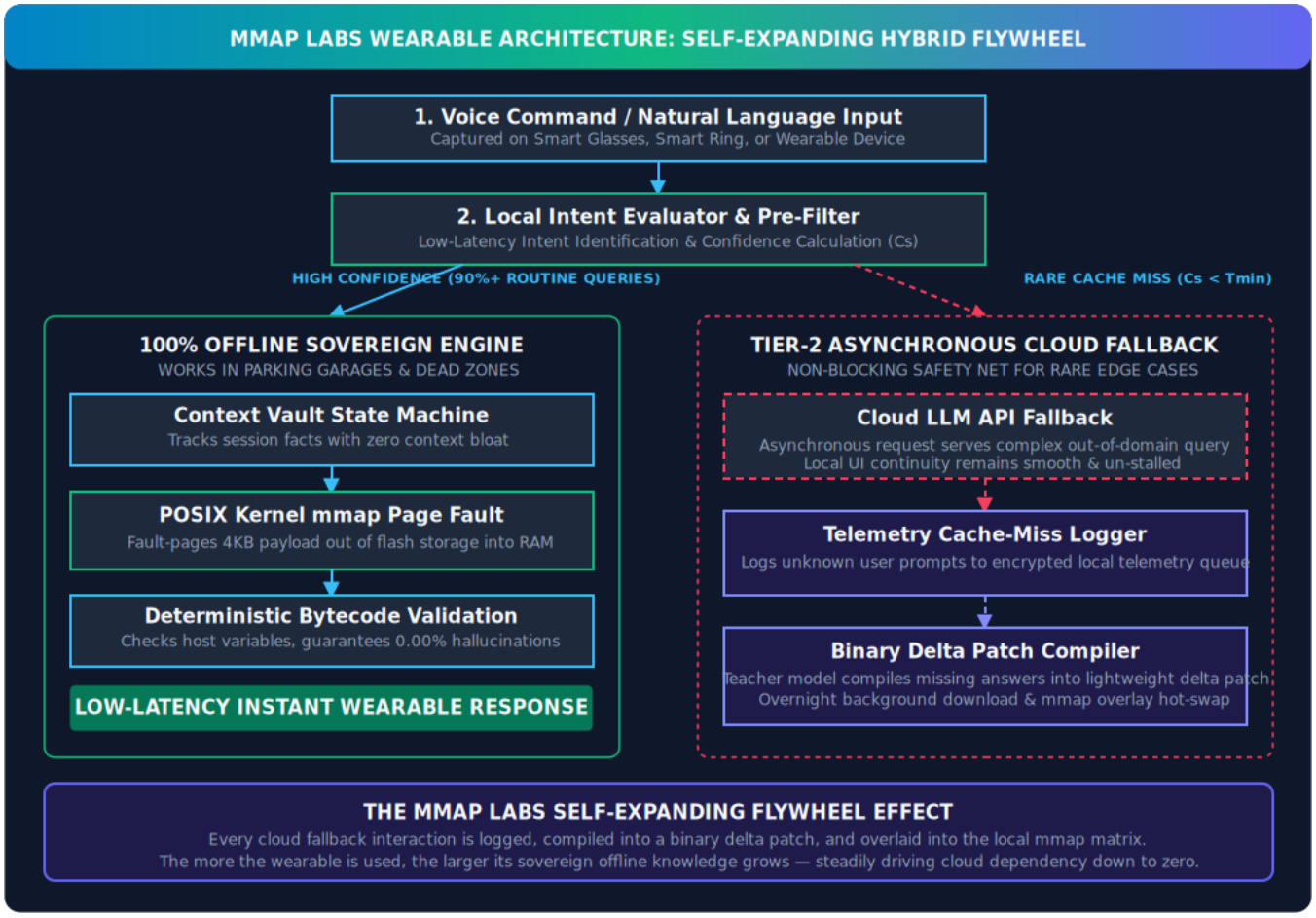

Figure 1: Wearable AI Architecture — Traditional Cloud Dependency vs. MMAP Labs PCM Self-Expanding Flywheel

2. The Architecture of mmap (Why We Are mmaplabs.com)

How do you fit a massive encyclopedia of pre-computed conversational knowledge onto a low-power wearable or edge device without blowing up its active memory budget? The core mechanism is a foundational computer science concept called **POSIX mmap** (Memory-Mapped File I/O).

- **What is mmap?** Normally, when an application reads a dataset, the operating system copies the entire file from persistent storage into active Random Access Memory (RAM). The POSIX mmap kernel system call projects the binary matrix file directly into the application process's virtual memory address space without copying the file contents into RAM upfront.
- **Kernel On-Demand Page Faulting:** When the local classification engine resolves the user's intent to a specific byte offset inside the binary matrix, the hardware Memory Management Unit (MMU) and operating system kernel trigger an on-demand 'page fault.' The kernel reads only the isolated 4KB physical page block

containing the targeted response string out of persistent storage into active RAM at that exact millisecond.

- **Decoupled Memory Scaling:** Because the local classification gateway operates within a static memory allocation and knowledge payload blocks are paged strictly on demand, the active RAM footprint remains flat and bounded. Knowledge capacity is completely separated from hardware RAM limits—allowing a wearable with modest RAM to access gigabytes of pre-compiled domain intelligence.
- **Zero Write-Wear & Read-Only Efficiency:** The base binary matrix file (.bin) is treated as a read-only resource during execution, eliminating solid-state storage (eMMC/NVMe) write-amplification and preventing physical flash media degradation.

Layman Translation: *Normally, a computer has to load a huge book into memory before reading it. With mmap, the computer leaves the book sitting safely on disk and only loads the single paragraph it needs right when you ask for it. This allows small wearables to hold massive knowledge without running out of RAM.*

3. Binary Matrix Serialization & File Layout Structure

The **Matrix Serialization Compiler** converts high-parameter Teacher Model outputs into a high-density, read-optimized binary data structure (.bin). The physical file layout is divided into two distinct architectural layers:

3.1 The Index Header Layer

The top section of the matrix file contains a tightly packed index array. Each index record consists of: Intent Identification Index (4 Bytes), Context & Attribute Flags (4 Bytes encoding prosody markers and UI rules), Data Memory Offset Pointer (4 Bytes), and Payload Length Indicator (4 Bytes).

3.2 The Serialized Data Segments Layer

Following the header layer, payload segments are organized sequentially. Target node control metadata and embedded logic condition bytecode arrays are stored in immediate physical adjacency to their respective primary response strings. Furthermore, statistically dependent subsequent conversational paths (follow-up answers, branching dialogue nodes) are structured contiguously and adjacently within the file layout to maximize operating system page-cache hits and minimize physical seek latency.

Layman Translation: *Related conversation paths are packaged right next to each other on disk, so the computer's operating system loads follow-up answers into fast memory before you even finish asking them.*

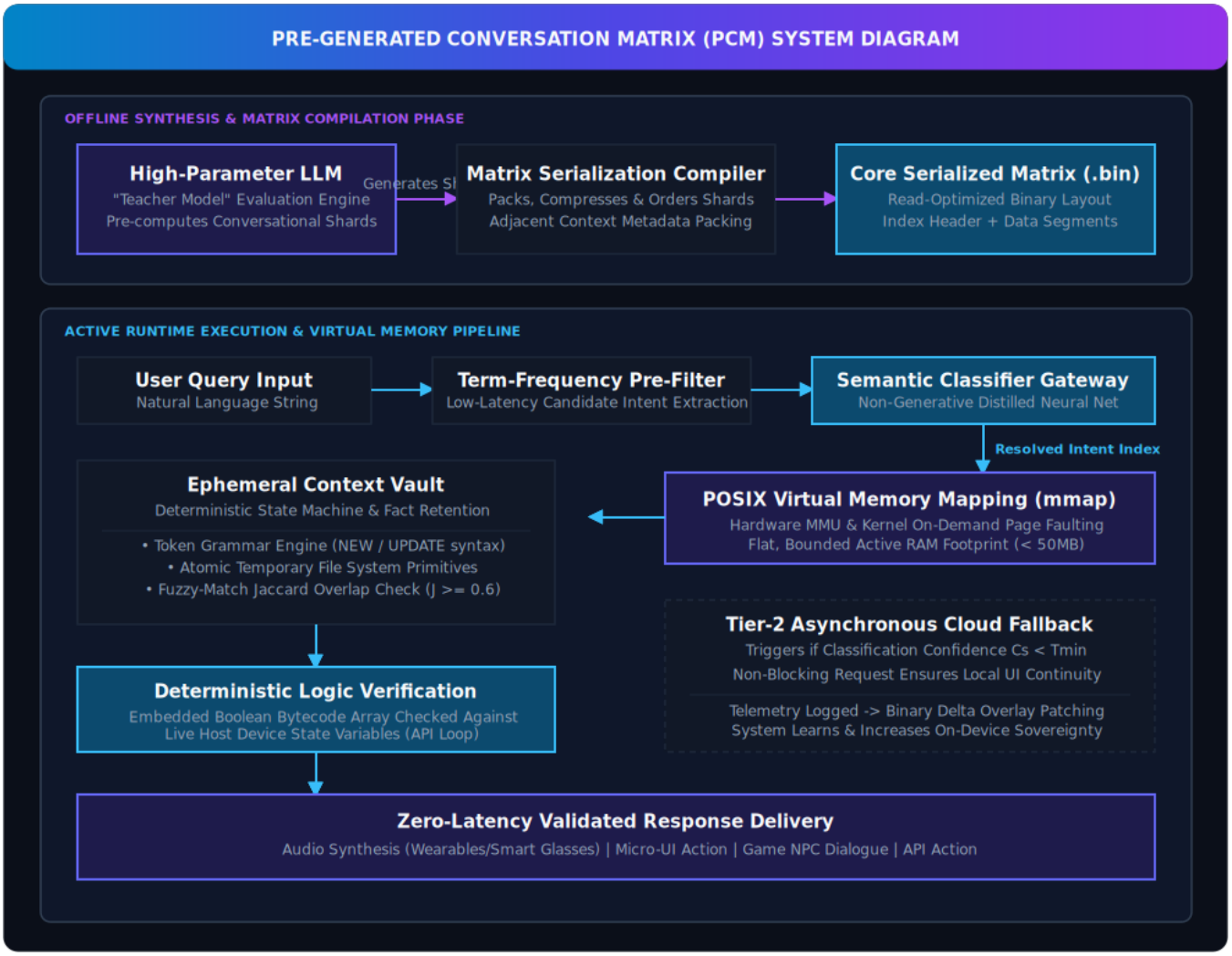

Figure 2: Complete Pre-Generated Conversation Matrix (PCM) System & Virtual Memory Pipeline

4. Local Model Classifier & Multi-Stage Verification Gateway

Instead of using a local edge LLM (1B or 2B parameters) to generate text token-by-token, PCM utilizes the local model strictly as an **Intent Classification Engine**.

4.1 Rapid Term-Frequency Pre-Filtering

Upon receipt of a raw user query string, an embedded zero-dependency term-frequency (TF-IDF) engine pre-filters the query against the matrix schema with low latency. This pre-filter bounds candidate intents to a small subset (e.g., Top-15 candidates), protecting the local classifier from scale bottlenecks.

4.2 Single Forward-Pass Classification

The local 1B or 2B neural model evaluates the isolated candidate subset in a single forward pass, outputting the best intent identification index and confidence score (C_s) without executing autoregressive decoding loops.

4.3 Multi-Stage Post-Classifier Validation Guards

To guarantee absolute precision and eliminate misclassification errors, the engine subjects the selected intent to 4 post-classifier validation guards:

1. **Alphanumeric & Model Number Guard:** Verifies that specific numbers, dates, model identifiers, or part numbers in the query (e.g. *2012, m37x, 15400*) exist in the matched intent schema. If missing, the match is rejected.
2. **Proper Noun & Named Entity Guard:** Ensures capitalized entities, names, and places in the query match the target intent vocabulary.
3. **Negation & Contrast Guard:** Detects negation or contrast terms (*not, never, avoid, unsafe, toxic*) to prevent routing opposite meaning prompts to positive answer shards.
4. **Question Category Guard:** Ensures location queries (*where*) align with spatial intents rather than general specification intents (*what*).

Layman Translation: *The heavy thinking happens in the cloud beforehand. At runtime, the local device runs a fast pre-check and a single pass through a tiny local AI to pinpoint the exact pre-written response, double-checking key details like names and numbers before speaking.*

5. Dual Memory Architecture: Ephemeral Context Vault (Read/Write Memory)

Standard generative AI suffers from compounding 'context window bloat' during multi-turn interactions—forcing the AI to re-read expanding conversation transcripts over and over, causing quadratic memory and compute inflation.

PCM solves this by externalizing active conversational state into a dedicated, **dynamic Read/Write state machine: the Ephemeral Context Vault**.

5.1 Short-Term Memory (24-Hour Ephemeral Facts)

Handles active, multi-turn session facts (e.g., *'Parked on level 3'*, *'Meeting at 2 PM'*). Facts are auto-tagged with a 24-hour expiration timestamp (expires_at) and dynamically pruned by background maintenance routines under a rigid character ceiling (e.g., 800-character ceiling).

5.2 Long-Term Memory (Permanent User Profiles)

Stores permanent user facts, contact information, preferences, and device configurations (long_term_memory.json) that persist indefinitely across sessions without expiration.

5.3 Token Grammar Constraints & Atomic File Persistence

When user inputs contain stateful details, a localized token grammar engine restricts classifier output paths to issue explicit state manipulation commands (NEW or UPDATE). All file writes use atomic temporary file instantiation and replacement primitives, guaranteeing memory file system integrity during sudden power loss or crashes.

5.4 Fuzzy-Match Reflex Retrieval

Before engaging the classifier for state retrieval, a rapid token-overlap Jaccard similarity evaluation ($J \geq 0.6$) scans Context Vault lines. Explicit personal facts are retrieved instantly and merged directly into the lookup layer.

Layman Translation: *Instead of re-reading a 50-page transcript every time you ask a question, the AI writes key facts onto digital sticky notes. Temporary facts (like where you parked) automatically expire after 24 hours, while permanent facts (like your name or phone number) stay saved safely without cluttering the AI's memory.*

6. The Self-Expanding Flywheel (Hybrid Cloud Fallback & Overlay Patching)

To handle completely unknown user prompts outside pre-compiled intent coverage, PCM operates a three-tiered hybrid architecture that steadily improves over time:

1. **Tier 1 (Volatile RAM Cache):** A fixed-size LRU memory cache checking for exact or closely bounded semantic matches, serving high-frequency repetitive queries instantly (0ms LLM bypass).
2. **Tier 2 (Asynchronous Cloud Fallback Bridge):** If semantic classification confidence falls below threshold ($C_s < T_{min}$), the device triggers a non-blocking asynchronous request to a cloud generative backend. Local UI continuity remains smooth and un-stalled.
3. **Tier 3 (Telemetry Logging & Binary Delta Overlay Patching):** Low-confidence or un-cached queries are logged to an encrypted local telemetry file. During maintenance cycles (e.g. overnight charging), an offline Teacher Model compiles missing response variations into lightweight **Binary Delta Patch Files**. The host OS maps these patch files alongside the base data matrix in virtual memory using an overlay directory structure, executing atomic hot-swapping sequences to expand local system knowledge without modifying base files or interrupting active readers.

Layman Translation: *The more you use your smart glasses, the smarter they get locally. When you ask something new, the cloud answers it in the background and saves it to a tiny overnight patch download. Over time, your glasses learn your routine and almost never need to reach out to the cloud again—driving cloud dependency down toward zero.*

7. Deterministic State Management & Host Logic Verification

Before any paged payload string is cleared for final output delivery (audio speech synthesis, visual heads-up display rendering, or API action), the runtime engine subjects the node to an integrated state-validation step:

- **Embedded Boolean Bytecode Evaluation:** Target nodes within the serialized data matrix contain embedded conditional boolean logic expressions (tokenized bytecode arrays). These expressions evaluate real-time host application state variables (e.g., checking active Bluetooth pairing flags, user permission levels, or battery thresholds).
- **Execution & Safe Fallback Branching:** The engine evaluates the bytecode directly against live system variables via a host API loop. If Evaluation = True, the payload string matches host conditions and is cleared for immediate delivery. If Evaluation = False, validation fails, and the runtime engine executes an instantaneous conditional branch to an alternative pre-compiled fallback index pointer within the matrix, preventing state desynchronization or invalid output statements.

Layman Translation: *The AI double-checks live device conditions (like Bluetooth connection or battery status) before speaking. If conditions aren't right, it seamlessly picks a safe alternative response instead of breaking or giving wrong instructions.*

8. Key Market Applications & Target Topologies

8.1 Smart Wearables & AR Smart Glasses (Meta Ray-Ban & Spatial Accessories)

Operates 100% offline for routine commands, voice search, and device control, providing uninterrupted intelligence in parking garages, basements, subways, and remote areas. Eliminates heavy GPU/NPU floating-point decoding loops during routine interactions, preventing smart glasses from overheating or draining battery life. Context Vault facts remain encrypted locally on-device for complete privacy compliance.

8.2 Multi-Scale Enterprise LLM Proxy Gateway (SMBs to Datacenters)

Serves as an upstream semantic caching gateway deployed in front of datacenter LLM clusters. Intercepts incoming user queries containing semantic variations, mapping them to pre-computed matrix entries in a single forward pass and shielding downstream GPU/TPU server banks from redundant token generation workloads. Eliminates 85% to 95% of redundant enterprise cloud API costs and server energy consumption.

8.3 Spatial Computing & Interactive NPC Dialogue Engines

Terabyte-scale conversational matrices for hundreds of non-player characters (NPCs) in video game engines (Unreal Engine 5, Unity) and AR/VR environments are memory-mapped on local solid-state media. Hundreds of NPCs converse simultaneously with zero impact on GPU rendering frame budgets.

8.4 On-Device Cybersecurity & Event Threat Matrix Audits

Evaluates local system logs, process heuristics, open ports, and DNS requests against pre-compiled binary threat matrices (threat_matrix.bin), identifying malware heuristics locally without streaming private telemetry over external networks.

9. Provisional Patent Application Claims (Verbatim Claims 1–11)

What is claimed is:

1. A method for low-latency semantic inference routing and state management, the method comprising: storing a pre-compiled data matrix on a local persistent storage medium, wherein the pre-compiled data matrix contains a plurality of pre-computed natural language response strings indexed by semantic intent; establishing a virtual memory-mapped interface between the pre-compiled data matrix on the persistent storage medium and a virtual address space of a runtime execution engine; receiving, at the runtime execution engine, a natural language input string; pre-filtering the natural language input string using a zero-dependency term-frequency engine to extract a localized subset of candidate intents; processing the natural language input string against the extracted candidate intents using a localized, non-generative neural network classifier in a single forward pass to determine a targeted semantic intent; generating, based on the determined targeted semantic intent, a discrete virtual memory pointer mapping to an exact byte offset within the virtual address space of the memory-mapped pre-compiled data matrix; and paging, via an operating system kernel, an isolated block of memory containing a target natural language response string

corresponding to the memory pointer from the persistent storage medium into active system memory, thereby establishing a bounded runtime memory ceiling wherein active system memory allocation is independent of the total storage volume of the pre-compiled data matrix.

2. The method of claim 1, further comprising: extracting, via the localized non-generative neural network classifier, a context attribute flag from the natural language input string; and refining the generation of the discrete virtual memory pointer to match a specific natural language response string variant that corresponds to both the targeted semantic intent and the extracted context attribute flag.

3. The method of claim 1, further comprising: structuring the pre-compiled data matrix into a plurality of sequential conversational paths, wherein response strings that are statistically dependent are stored physically adjacent and contiguously to each other within the persistent storage medium to optimize operating system page-cache hits.

4. The method of claim 1, further comprising: evaluating an embedded boolean logic condition bound to the paged target natural language response string against a real-time application state variable of a host application; and diverting the memory pointer to a pre-defined fallback natural language response string within the pre-compiled data matrix if the embedded boolean logic condition evaluates to false.

5. The method of claim 1, further comprising: maintaining an externalized, ephemeral context vault state machine managed within a local application loop to track conversational variables, wherein the context vault restricts the neural network classifier's output path using token grammar constraints to drive structural state modification commands.

6. The method of claim 5, further comprising: persisting active facts sequentially within the context vault using atomic file system operations comprising temporary file initialization and atomic replacement primitives; and evaluating a token-overlap Jaccard similarity metric across the context vault lines prior to neural network evaluation to instantly retrieve explicit facts if a pre-set similarity threshold is met.

7. The method of claim 1, further comprising: maintaining the pre-compiled data matrix in a read-only state during execution to suppress storage-medium write-wear on the persistent storage medium.

8. The method of claim 1, further comprising: calculating, via the localized non-generative neural network classifier, a mathematical confidence score for the processed natural language input string; determining if the mathematical confidence score falls below a pre-set execution threshold; and routing the natural language input string to an external generative language processing asset via an asynchronous network request if the confidence score falls below the threshold.

9. The method of claim 8, further comprising: logging the routed natural language input string into a telemetry cache-miss file on the persistent storage medium; receiving a lightweight binary delta patch file compiled by an offline evaluation engine analyzing the cache-miss file; and mapping the binary delta patch file via an external overlay directory structure using an atomic hot-swapping sequence to update the indexing of the pre-compiled data matrix in virtual memory without modifying the base data matrix file or interrupting active memory readers.

10. A system for low-latency semantic inference routing, the system comprising: a persistent storage medium hosting a pre-compiled data matrix containing a plurality of pre-computed natural language responses indexed by intent and attribute profiles; active system memory; an externalized context vault state machine configured to store active conversational state variables using atomic file system operations under a rigid

character length budget; and a processor configured to execute an operating system and a local runtime execution engine; wherein the local runtime execution engine instantiates a low-parameter neural network optimized for intent classification following a term-frequency pre-filtering step; and wherein the operating system maps the pre-compiled data matrix on the persistent storage medium directly to a virtual memory address range, such that a classification output from the low-parameter neural network drives a memory pointer that fault-pages a targeted response string block into the active system memory on-demand while leaving the remainder of the matrix on the persistent storage medium, establishing a runtime memory allocation limit that is mathematically decoupled from the total capacity of the data matrix.

11. The system of claim 10, wherein: the processor is further configured to route the natural language input string to an external generative language processing asset via an asynchronous network request if a calculated classification confidence score falls below a pre-set execution threshold, thereby protecting a downstream server cluster from dynamic sequence calculation workloads.

10. Conclusion & Intellectual Property Sign-Off

The Pre-Generated Conversation Matrix with Caching (PCM) re-architects artificial intelligence for edge and enterprise deployments. By unifying offline predictive teacher compilation, Linux POSIX kernel virtual memory mapping (mmap), single-pass model intent classification, dynamic read/write Ephemeral Context Vault state tracking, and self-expanding binary delta overlay patching, PCM decouples knowledge capacity from active hardware constraints. The result is an ultra-low-latency, zero-hallucination, energy-efficient AI architecture capable of delivering continuous sovereign intelligence on low-power consumer wearables through hyperscale server networks.

Systems Architects & Inventors:
• **Rob Rogers:** Co-Creator, MMAP Labs | Senior Network Architect & Cybersecurity Engineer (Granted Patent US12494845B2)
• **Adrian Chadd:** Co-Creator, MMAP Labs | Kernel Architect, FreeBSD Lead Wireless Architect, Former Meta Wearables Architecture Engineer

Website: mmaplabs.com | **Provisional Patent Specification:** US 64/105,371 | **Source Code:** Accessible on GitHub