Module C: Universal Streaming/Handling/Loading/Indexing Module

Part of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence

—

Contents

1 Abstract

This module defines the *Universal Streaming/Handling/Loading/Indexing* component of the Eidos framework. Its primary purpose is to manage the storage and on-demand retrieval of large-scale model data (including model weights, biases, parameters, intermediate representations, and graphs) in a hardware-agnostic manner. By decomposing a model into minimal, self-contained modules (or chunks), this system creates a persistent, disk-resident index that supports streaming the necessary components during inference, evaluation, and training. The module is designed to minimize in-memory footprint while ensuring rapid, reliable loading of model components in accordance with available computational resources.

2 Introduction and Motivation

Modern deep learning systems frequently require handling models whose size exceeds the available RAM, particularly when deployed on heterogeneous hardware ranging from low-memory CPUs to high-end GPUs. The **Universal Streaming/Handling/Loading/Indexing Module** addresses this challenge by decomposing a model into minimal modules and indexing them on persistent storage. Key motivations include:

- **Scalability:** Enable the execution of models of arbitrary size by streaming components on demand.
- **Hardware-Agnostic Deployment:** Allow model execution on diverse platforms regardless of available memory.
- **Modularity and Extensibility:** Decompose the model into self-contained chunks that can be updated, reloaded, or replaced without affecting the overall architecture.
- **Efficiency:** Optimize I/O operations and caching strategies to minimize latency and maximize throughput.

3 Preliminaries and Notation

- Let $\theta \in \Theta \subset \mathbb{R}^p$ denote the complete set of model parameters.
- We assume that θ is composed of a collection of multi-dimensional tensors, and can be decomposed as:
$$\theta = \bigcup_{j \in J} T_j,$$
where J is a finite index set and each T_j represents a subset of parameters corresponding to a particular layer or functional subcomponent.
- A *module* (or *chunk*) is defined as a tuple:
$$C_j = \big(id_j,\, T_j,\, \mu_j\big), \quad \text{for } j \in J,$$
where:
  - id_j is a unique identifier for the module,

- T_j is the parameter subset for the module,
- μ_j is metadata describing the module (including tensor shapes, data types, dependencies, and size).

- The persistent storage is abstractly denoted by $\mathcal{S}_{\text{disk}}$. We assume that $\mathcal{S}_{\text{disk}}$ is sufficiently large to hold all model modules.
- The current runtime resource context (e.g., available RAM, GPU memory, CPU cores) is denoted by $\mathcal{R}$.

4 Formal Definitions and Mathematical Formulation

Definition 1 (Model Decomposition)

Let f_θ be a model with parameters θ. A *decomposition function*

$$\mathcal{D} : \Theta \to \{C_j\}_{j \in J}$$

partitions θ into modules $C_j = (id_j, T_j, \mu_j)$ such that:

$$\theta = \bigcup_{j \in J} T_j,$$

and for all $j, k \in J$ with $j \neq k$, the sets T_j and T_k are disjoint (or minimally overlapping).

Definition 2 (Indexing Function)

We define an *indexing function*

$$\mathcal{I} : \{C_j\}_{j \in J} \to \{\ell_j \in \mathcal{S}_{\text{disk}}\},$$

which maps each module C_j to a location ℓ_j in persistent storage. The mapping is assumed to be injective so that:

$$\mathcal{I}(C_j) = \ell_j, \quad \text{with } \ell_j \text{ uniquely identifying the storage location of } C_j.$$

The complete index is then given by:

$$\mathcal{I} = \{(id_j, \ell_j, \mu_j) \mid j \in J\}.$$

Definition 3 (Streaming Function)

Let the *streaming function* be defined as:

$$\sigma : \mathcal{I} \times \mathcal{R} \to \{C_j\}_{j \in J'},$$

where $J' \subseteq J$ is the subset of modules that are loaded into fast memory (e.g., RAM or GPU memory) given the current runtime resource context $\mathcal{R}$. The function σ selects modules based on:

- **Dependency:** Only modules required for the current computation are loaded.
- **Resource Constraints:** The total memory used by the loaded modules does not exceed the available resources specified by $\mathcal{R}$.

Definition 4 (Caching Function)

The *caching function* is a mapping:

$$\mu : \{C_j\} \times \mathcal{R} \to \{C_j\}_{\text{active}},$$

which governs the residency of modules in fast memory. The function μ implements a caching policy (e.g., least-recently-used, priority-based, or predictive prefetching) that determines which modules remain loaded in memory for fast access and which may be evicted.

5 Algorithmic Description

Below is the pseudocode for the model loading, indexing, and streaming process:

Algorithm 1 Model Decomposition, Indexing, and Streaming

Input: Model parameters $\theta \in \Theta$; Resource context $\mathcal{R}$; Decomposition function $\mathcal{D}$
Output: Active module set $\{C_j\}_{j \in J'}$ loaded into memory
Decomposition: Compute $\{C_j\}_{j \in J} \leftarrow \mathcal{D}(\theta)$
for each module $C_j \in \{C_j\}_{j \in J}$ **do**
    Store T_j to persistent storage at location ℓ_j
    Record entry (id_j, ℓ_j, μ_j) in index $\mathcal{I}$
end for
Streaming: Based on the current resource context $\mathcal{R}$, select modules to load:

$$\{C_j\}_{j \in J'} \leftarrow \sigma(\mathcal{I}, \mathcal{R}).$$

Caching: Manage active modules via:

$$\{C_j\}_{\text{active}} \leftarrow \mu\big(\{C_j\}_{j \in J'}, \mathcal{R}\big).$$

Return: Active module set $\{C_j\}_{\text{active}}$

6 Theoretical Analysis and Guarantees

Theorem 1 (Universal Executability)

Statement: For any model f_θ decomposed into modules $\{C_j\}_{j \in J}$ and any execution context $\mathcal{R}$ with sufficient persistent storage $\mathcal{S}_{\text{disk}}$, there exists a streaming strategy σ and caching policy μ such that the model f_θ can be executed (for inference, evaluation, or training) with active modules $\{C_j\}_{j \in J'}$ loaded in fast memory.

Proof Sketch: Since the decomposition function $\mathcal{D}$ partitions θ into minimal modules that are independent or minimally coupled, and since the index $\mathcal{I}$ provides a unique mapping to persistent storage, a streaming function σ can select a subset of modules based on their dependencies and current resource availability. The caching function μ further ensures that once modules are loaded, they remain available as needed, and non-critical modules may be evicted. Hence, even if θ is extremely large, only a manageable subset is needed at any time, guaranteeing universal executability. □

Proposition 1 (Scalability)

The modular design ensures that the in-memory resource requirement is proportional to the number of active modules rather than the total model size. Thus, the system scales efficiently on hardware with limited RAM, provided that persistent storage is adequate.

7 Integration with the Overall Eidos Framework

The Universal Streaming/Handling/Loading/Indexing Module is a key backbone of the Eidos framework. It:

- Facilitates the deployment of large models by decomposing parameters into independent modules.
- Provides a disk-resident index ($\mathcal{I}$) that allows for on-demand loading via the streaming function σ.
- Ensures efficient memory management via the caching function μ, making the system hardware-agnostic.
- Interfaces with the Universal Communication module (Module B) to allow modules to be updated and replaced dynamically.

8 Implementation Considerations

- **Data Structures:** The index $\mathcal{I}$ may be implemented as a database or a structured file (e.g., JSON, Protocol Buffers) that maps module IDs to disk locations and metadata.
- **Streaming Optimization:** The streaming function σ should be optimized to minimize I/O latency (e.g., through asynchronous prefetching and parallel loading).
- **Caching Policies:** The caching function μ should implement advanced cache replacement algorithms (such as least-recently-used or priority-based caching) to keep the most critical modules in memory.
- **Dependency Graphs:** To determine which modules are needed at any time, a dependency graph of the model must be maintained and updated as modules are loaded or updated.
- **Hardware Interfaces:** The system must be designed to query available resources in $\mathcal{R}$ (e.g., memory size, GPU capacity) and adjust σ and μ accordingly.

9 Conclusion

The Universal Streaming/Handling/Loading/Indexing Module provides a rigorously defined, modular, and scalable method for managing large-scale models in the Eidos framework. By decomposing model parameters into minimal modules, mapping these to persistent storage via an index, and dynamically streaming them into fast memory based on current resource constraints, this module ensures hardware-agnostic and efficient model execution. Its design guarantees that even extremely large models can be loaded and executed in a streaming manner, forming a critical backbone for the overall framework.

Module Summary:
Completed:

- Module A: Input Processing.
- Module B: Universal Communication & Data Handling Interface and Coordination.
- Module C: Universal Streaming/Handling/Loading/Indexing Module.

Remaining Modules:

- Module D: Multidimensional Vocabulary and Tokenization System.
- Module E: Contextual NLU/NLP Embedding and Multidimensional Tokenization.
- Module F: Deep Knowledge Graphs System (Base and Personal).
- Module G: Infinite RoPE Context Scaling and Dynamic Vocabulary Updating.
- Module H: Core Model Architectures (RWKV and Transformer Modules, Mixture-of-Experts Style).
- Module I: Titans Memory Architecture (Multi-Layer Memory Module).
- Module J: Recursive Adaptive Dynamic Idempotent Feedback and State-Based Runtime Learning and Inference.
- Module K: Universal Training System.
- Module L: Final Decoding and Multimodal Output.