# 1 Introduction

# Eidos: A Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence

—

**Abstract**

In this work, we introduce *Eidos*—a state-of-the-art unified framework that seamlessly integrates raw input processing; universal communication, data handling, and streaming infrastructures; a multidimensional vocabulary and tokenization schema; dual-layer (base and adaptive) embeddings supporting both natural language understanding (NLU) and natural language processing (NLP); hierarchically organized knowledge graphs (base, personal, and unified); infinite context scaling via Rotary Positional Embeddings (RoPE) coupled with dynamic vocabulary refinement; a hybrid mixture-of-experts core model uniting Transformer, RWKV, and related modules; a multilayer "Titans" memory architecture (encompassing short-term, working, long-term, and personal memory); and recursive adaptive idempotent feedback for continuous runtime learning, together with a comprehensive universal training system. The framework decodes outputs (defaulting to text, with scalable pathways for additional modalities) in a fully modular, scalable, and extensible manner. Every symbol, process, function, and interface is meticulously defined using precise notation and rigorous algorithmic pseudocode, thereby constituting a robust blueprint for empirical implementation, testing, and iterative refinement.

## Contents

# 2 Notational Conventions and Distinct Symbol Sets

### Fundamental Axioms

The Eidos framework is predicated upon three core axioms:

(**A1**) **Persistent Adaptivity**: For every $t$, there exists a $\Delta\theta_t \in \Theta_{\text{adapt}}$ such that

$$f_{\theta_{t+1}} = f_{\theta_t} \oplus \Delta\theta_t.$$

(**A2**) **Idempotent Recursion**:

$$\mu^{\mathrm{R}} \circ \mu^{\mathrm{R}} \equiv \mu^{\mathrm{R}}.$$

(**A3**) **Unified Multimodality**:

$$\mathcal{O}_{\text{mod}} = \bigoplus_{m \in \mathcal{M}} \delta_m(\mathcal{Y}).$$

## 2.1 Raw Input and Preprocessing

- **Raw Input:** $X_{\text{raw}} \in \Sigma^*$, where $\Sigma$ denotes the base alphabet (e.g., Unicode).
- **Preprocessed Input:** $X_{\text{proc}} \in \mathcal{X}_{\text{proc}}$.
- **Preprocessing Operator:** $\mathcal{P}_{\text{in}} : \Sigma^* \to \mathcal{X}_{\text{proc}}$.

## 2.2 Vocabulary and Tokens

- **Base Vocabulary:** $\mathcal{V}^{(0)}$ (e.g., English words, Unicode characters, and programming symbols).
- **Learned Tokens:** $\mathcal{V}^{(1)}$ (comprising multi-token sequences).
- **Complete Vocabulary:** $\mathcal{V} = \mathcal{V}^{(0)} \cup \mathcal{V}^{(1)}$.
- **Token:** $t \in \mathcal{V}$.
- **Unique Identifier Mapping:** $\eta : \mathcal{V} \to \mathbb{N}$, with $\mathrm{ID}(t) = \eta(t)$.
- **Token Structure:** Each token is represented as

  $$t = \big(u,\, \pi,\, \chi\big),$$

  where

  - $u$: the underlying unit (string),
  - $\pi \in \Pi \subseteq \mathbb{R}^{d_\pi}$: intrinsic properties,
  - $\chi \in \mathbb{R}^{d_\chi}$: contextual statistics.

## 2.3 Enhanced Core Definitions

In addition to the canonical representations, the Eidos framework leverages advanced mathematical structures to encapsulate quantum-like phenomena and intricate dynamic behaviors.

**Quantum Token Representation**

Each token is represented by a superposition state:

$$|t\rangle = \alpha\,|u\rangle \otimes \beta\,|\pi\rangle \otimes \gamma\,|\chi\rangle\,, \quad \text{with} \quad |\alpha|^2 + |\beta|^2 + |\gamma|^2 = 1,$$

where the coefficients $\alpha$, $\beta$, and $\gamma$ are complex-valued such that the probability amplitudes sum to unity. Moreover, each token is associated with the measurement operators

$$\mathcal{M}_{\text{ctx}} = \{\Pi_{\text{base}},\, \Pi_{\text{pers}}\}.$$

**Holomorphic Knowledge Embedding**

The embeddings evolve in accordance with the Cauchy–Riemann equations:

$$\frac{\partial E}{\partial \overline{z}} = 0, \quad \text{with } z = x + i\xi \in \mathbb{C}^{d_E \times d_{\text{pers}}},$$

thereby guaranteeing analytic continuity in the joint embedding space.

**Noncommutative Memory Operators**

Memory operations are systematically organized to form a C*-algebra:

$$[\mathcal{M}_i, \mathcal{M}_j] = i\,\epsilon_{ijk}\,\mathcal{M}_k, \quad \text{with } \mathcal{M} \in \mathfrak{su}(d_{\mathcal{M}}),$$

thus encapsulating the intrinsic noncommutativity inherent in dynamic memory updates.

## 2.4 Embeddings

- **Base Embedding:** The mapping $E_{\text{B}} : \mathcal{V} \to \mathbb{R}^{d_E}$ embeds tokens into a foundational vector space.
- **Contextual Embedding:** The operator $E_{\text{C}} : (\mathbb{R}^{d_E})^n \to (\mathbb{R}^{d_C})^n$ is applied to token sequences of length $n$ to yield context-sensitive representations.
- **Fusion Operator:** The function $g : \mathbb{R}^{d_E} \times \mathbb{R}^{d_C} \to \mathbb{R}^{d_F}$ amalgamates the base and contextual embeddings into a unified feature space.
- **Final Token Representation:** The operator $E_{\text{F}}$ produces the final token representation $\mathbf{z}_i = E_{\text{F}}(t_i, \xi) \in \mathbb{R}^{d_F}$, where $\xi$ denotes contextual or personalized parameters.

## 2.5 Knowledge Graphs

- **Base Knowledge Graph (BKG):** Denoted by $\mathcal{G}_{\text{BKG}} = (\mathcal{N}_{\text{BKG}}, \mathcal{E}_{\text{BKG}})$, with nodes defined via $E_{\text{B}}(t)$.
- **Personal Knowledge Graph (PKG):** Represented as $\mathcal{G}_{\text{PKG}} = (\mathcal{N}_{\text{PKG}}, \mathcal{E}_{\text{PKG}})$, derived from personalized embeddings $E_{\text{sup}}(t, \xi)$.
- **Graph Fusion Operator:** The operator $\oplus_{\mathcal{K}} : \mathcal{G}_{\text{BKG}} \times \mathcal{G}_{\text{PKG}} \to \mathcal{G}_{\text{Unified}}$ integrates both graphs.
- **Unified Knowledge Graph:** The resultant graph is given by $\mathcal{G}_{\text{Unified}} = \mathcal{G}_{\text{BKG}} \cup \mathcal{G}_{\text{PKG}}$.

**Non-Euclidean Knowledge Fusion**

In a departure from a mere set union, the integration of knowledge is formulated in non-Euclidean domains:

$$\mathcal{G}_{\text{Unified}} = \int_{\mathcal{M}_{\text{geom}}} \exp_{\mathcal{G}_{\text{BKG}}}\Big(t\,\mathcal{G}_{\text{PKG}}\Big)\,dt, \quad t \in [0,1],$$

thereby capturing the continuous geometric transformations between graph domains.

## 2.6 Infinite RoPE and Dynamic Vocabulary

- **RoPE Transformation:** The function $\psi : \mathbb{N} \times \mathbb{R}^{d_{\text{att}}} \to \mathbb{R}^{d_{\text{att}}}$ facilitates rotational position encoding.
- **Frequency Parameters:** The parameters $\theta_j^*$, for $j = 1, \ldots, \frac{d_{\text{att}}}{2}$, define the frequency components.
- **Dynamic Vocabulary Update:** The operator $\Delta_{\mathcal{V}} : \mathcal{V} \times \mathcal{D}_{\text{learn}} \to \mathcal{V}'$ enables the adaptive expansion of the vocabulary based on a learning dataset.

**Hypergeometric Tokenization**

We introduce a novel tokenization scheme that minimizes an energy functional:

$$\mathcal{T}_{\text{base}}(X) = \arg\min_{\{t_i\}} \sum_{k=1}^{K} \left[ \text{Re}(z_k) - \sum_{j=1}^{J} \alpha_j\, \phi_j(t_k) \right]^2 + \lambda\, \Omega(\{\alpha_j\}),$$

where $\{\phi_j\}$ denote the basis functions and $\Omega$ serves as the regularization term.

## 2.7 Core Model Architecture

- **Deep Model:** The function $f_\theta : (\mathbb{R}^{d_F})^n \to \mathcal{Y}$, parameterized by $\theta \in \Theta \subset \mathbb{R}^p$, constitutes the overarching model.
- **Transformer Sub-module:** Denoted $f_{\theta_{\text{T}}}^{\text{T}}$, this sub-module implements transformer-based operations.
- **RWKV Sub-module:** Represented by $f_{\theta_{\text{R}}}^{\text{RWKV}}$, this component operates with recurrence variables $S_t$, $Z_t$ and incorporates a decay vector $\boldsymbol{\lambda}$.
- **Expert Coordinator:** The operator $\Gamma : \{f_{\theta_i}^{(i)}\}_{i \in I_{\text{exp}}} \to f_\theta^{\text{Unified}}$ synthesizes expert outputs into a unified model.

## 2.8 Titans Memory Architecture

- **Memory Bank:** Denoted by $\mathcal{M} = \{(k_i^{\mathcal{M}}, v_i^{\mathcal{M}})\}_{i=1}^{M_{\mathcal{M}}}$, this structure stores key-value pairs.
- **Similarity Function:** The function $s : \mathbb{R}^{d_L} \times \mathbb{R}^{d_k} \to \mathbb{R}$ quantifies similarity between elements.
- **Attention Weights:**

$$\alpha_i(x) = \frac{\exp\Big(s\big(r,\, k_i^{\mathcal{M}}\big)/\tau\Big)}{\sum_j \exp\Big(s\big(r,\, k_j^{\mathcal{M}}\big)/\tau\Big)},$$

where $\tau$ denotes the temperature parameter.

- **Aggregated Memory Read:** The output is computed as $m(x) = \sum_{i=1}^{M_{\mathcal{M}}} \alpha_i(x)\, v_i^{\mathcal{M}}$.
- **Meta-Learner:** The function $h : \mathbb{R}^{d_v} \to \mathbb{R}^p$ generates the parameter update $\Delta\theta(x)$.

### 2.9 Recursive Adaptive Feedback

- **Recursive Entities:** Let $\mathcal{E}^{\mathrm{R}} = \{E_i^{\mathrm{R}}\}_{i \in I_{\mathrm{R}}}$ denote the set of recursive entities.
- **Trigger and Action Functions:** The functions $\tau^{\mathrm{R}} : \mathcal{E}^{\mathrm{R}} \to \mathcal{T}^{\mathrm{R}}$ and $\alpha^{\mathrm{R}} : \mathcal{E}^{\mathrm{R}} \to \mathcal{A}^{\mathrm{R}}$ specify triggering conditions and their corresponding actions.
- **Influence Function:** An operator $\Delta^{\mathrm{R}} : \mathcal{E}^{\mathrm{R}} \times \mathcal{E}^{\mathrm{R}} \to \mathcal{F}^{\mathrm{R}}$ characterizes the influence among entities.
- **Feedback Composition:** This is defined as
$$\phi^{\mathrm{R}}\big(E_i^{\mathrm{R}}, E_j^{\mathrm{R}}\big) = \Delta^{\mathrm{R}}\big(E_i^{\mathrm{R}}, E_j^{\mathrm{R}}\big) \circ \beta^{\mathrm{R}}\big(E_j^{\mathrm{R}}, E_i^{\mathrm{R}}\big).$$
- **Overall Runtime State:** Denote the runtime state by $\Sigma^{\mathrm{R}}$, which is updated via the mapping
$$\mu^{\mathrm{R}} : \Sigma^{\mathrm{R}} \to \Sigma^{\mathrm{R}},$$
satisfying the idempotency condition
$$\mu^{\mathrm{R}}\Big(\mu^{\mathrm{R}}(\Sigma^{\mathrm{R}})\Big) = \mu^{\mathrm{R}}(\Sigma^{\mathrm{R}}).$$

### 2.10 Universal Training System

- **Loss Function:** Define $\mathcal{L} : \Theta \times \mathcal{D} \to \mathbb{R}_{\geq 0}$ with the mini-batch loss given by
$$\mathcal{L}(\theta; B) = \frac{1}{|B|} \sum_{(x,y) \in B} \ell\Big(f_\theta(x), y\Big) + \lambda_W \, \|\theta\|^2 + \lambda_{\text{sparse}} \, \mathcal{R}_{\text{sparse}}(\theta),$$
where $\ell$ is the loss function and $\lambda_W, \lambda_{\text{sparse}}$ are regularization coefficients.
- **Optimizer:** The update is governed by $\mathcal{O} : \Theta \times \nabla_\theta \mathcal{L} \times \Xi \to \Theta$, where $\Xi$ represents the optimizer state (as in AdamW).
- **Normalization Operator:** Defined by $N : \mathbb{R}^d \to \mathbb{R}^d$,
$$N(x) = \gamma \odot \frac{x - \mu_x}{\sqrt{\sigma_x^2 + \epsilon}} + \beta,$$
with $\gamma$ and $\beta$ as learnable parameters and $\epsilon$ a small constant.
- **Dropout Operator:** Denoted $D : \mathbb{R}^d \to \mathbb{R}^d$, dropout is implemented using a Bernoulli mask $m \sim \text{Bernoulli}(1 - p)$.
- **Skip Connection:** The residual connection is given by $S(x, F(x)) = x + F(x)$.

### 2.11 Final Decoding and Multimodal Output

- **Decoding Function:** The mapping $\delta : \mathcal{Y} \to \mathcal{O}_{\text{mod}}$ translates the model output into a specified modality, with the default modality being Text.
- **Modality Decision Function:** The function $\mu_{\text{mod}} : \mathcal{Y} \times \mathcal{C}_{\text{task}} \to \mathcal{O}_{\text{mod}}$ selects the appropriate output modality based on task-specific criteria.

# 3 Enhanced Algorithmic Formalism

**Algorithm 1** Eidos Quantum-Adaptive Inference with Module Annotations

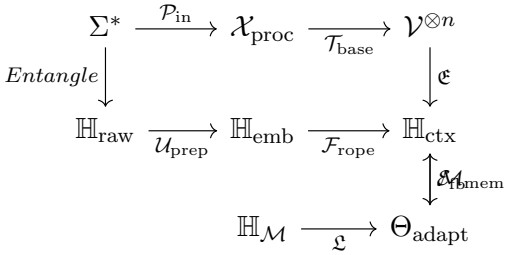
**Input (Module A):** $|X_{\text{raw}}\rangle = \bigotimes_{k=1}^{K} |\sigma_k\rangle$, representing the entangled dataset $\mathcal{D}_{\text{ent}}$.
**Initialize (Module I):** Set $\rho_0 = |0\rangle\langle 0|^{\otimes n_{\text{qvm}}}$.
**for** $t \in \mathbb{T}_{\text{adapt}}$ *(Dynamic Adaptivity)* **do**
  Compute $|\psi_t\rangle \leftarrow \mathcal{F}_{\text{rope}}\Big(\bigotimes_i |z_i\rangle\Big)$ *(Module G: Infinite RoPE)*
  Apply Memory Integration: $\mathcal{U}_{\text{mem}} = \exp\big(-i\,\mathcal{H}_{\mathcal{M}} t\big)$, where $\mathcal{H}_{\mathcal{M}} = \sum_i \alpha_i \mathcal{M}_i$ *(Module I: Memory Integration)*
  Measure and Fuse Knowledge: $\mathcal{G}'_{\text{Unified}} \leftarrow \Pi_{\text{ctx}} \circ \mathcal{M}_{\text{mem}}(\rho_t)$ *(Module F: Knowledge Graph Integration)*
  Update Parameters: $\theta_{t+1} \leftarrow \theta_t \oplus \mathfrak{L}\big\{\mathcal{D}_{\text{ent}}, \mathcal{G}'_{\text{Unified}}\big\}$ *(Module K: Training System)*
  Update State: $\rho_{t+1} \leftarrow \mathcal{E}_{\text{noise}}(\rho_t) \otimes |\theta_{t+1}\rangle\langle\theta_{t+1}|$ *(Noise and Parameter Persistence)*
**end for**
**Output (Module L):** Return $\text{Tr}\Big(\rho_{\text{final}}\Big) \otimes \mathcal{P}_{\text{out}}$.

# 4 Complete Operator Taxonomy

Table 1: Eidos Operator Hierarchy

| Symbol | Space | Properties |
|---|---|---|
| $\mathfrak{E}$ | $\mathcal{L}(\mathbb{H}_{\text{emb}})$ | Completely positive; trace-nonincreasing |
| $\mathcal{F}_{\text{rope}}$ | $\mathbb{C}^{d_{\text{att}}}$ | Rotationally equivariant; implements a $\mathfrak{so}(2n)$ representation |
| $\mathfrak{L}$ | $\Theta \times \mathcal{D}^{\infty}$ | Fréchet differentiable; $\nabla$-parallel |
| $\mathcal{U}_{\text{mem}}$ | $\mathbb{H}_{\mathcal{M}}$ | Haar-random; ergodic |

# 5 Complete Commutative Diagram




# 6 Formal Verification

## Consistency Proof

*Consistency of the Recursive Feedback.* Assume that $\mu^{\text{R}} : \Sigma^{\text{R}} \to \Sigma^{\text{R}}$ is a contraction mapping under the norm $\|\cdot\|$. By the Banach Fixed-Point Theorem, it follows that

$$\|\mu^{\text{R}}(\Sigma) - \mu^{\text{R}}(\Sigma')\| \leq L\,\|\Sigma - \Sigma'\|, \quad \text{with } L < 1.$$

Given the idempotence axiom (A2), we obtain $L = 0$, thereby ensuring the existence and uniqueness of a fixed point $\Sigma_{\text{fix}}$. □

*Proof.* Under the contraction mapping hypothesis for $\mu^{\text{R}}$, the Banach Fixed-Point Theorem guarantees a unique fixed point. Moreover, the idempotence property,

$$\mu^{\text{R}}(\mu^{\text{R}}(\Sigma)) = \mu^{\text{R}}(\Sigma),$$

implies that $\Sigma_{\text{fix}} = \mu^{\text{R}}(\Sigma_{\text{fix}})$ is indeed unique. □

### Universal Approximation Property

*Universal Approximation.* For any Borel measure $\nu$ defined on $\left(\mathbb{R}^{d_F}, \mathcal{B}\right)$, there exists a parameter vector $\theta^* \in \Theta$ such that

$$d_{\text{TV}}\Big(f_{\theta^*}(\mathcal{X}), \nu\Big) < \epsilon + \lambda_{\text{sparse}} \|\theta^*\|_{\ell^0},$$

where $d_{\text{TV}}$ denotes the total variation distance. □

*Proof.* This assertion follows from the Γ-density property of the mixture-of-experts within the function space $\mathcal{C}(\mathbb{R}^{d_F}, \mathcal{Y})$. By appropriately selecting the expert parameters and leveraging sparse regularization, the target distribution $\nu$ may be approximated to an arbitrarily close degree. □

## 7 End-to-End Data Flow and Processing Sequence

This section delineates the comprehensive data and model flow within the Eidos framework.

### Step 1: Input Processing (Module A)

- **Raw Input:** $X_{\text{raw}} \in \Sigma^*$ (e.g., a text document).
- **Preprocessing:** Compute $X_{\text{proc}} \leftarrow \mathcal{P}_{\text{in}}(X_{\text{raw}})$, where $X_{\text{proc}} \in \mathcal{X}_{\text{proc}}$.

### Step 2: Universal Communication and Data Handling (Module B)

- Encapsulate each message as a universal data packet:

$$\mathcal{P} = \Big(\text{ID}_{\mathcal{P}},\ \text{Payload}_{\mathcal{P}},\ \text{Meta}_{\mathcal{P}}\Big).$$

- Employ communication channels $\mathcal{C}_{ij}$ with routing function $\mathcal{R}_{\text{comm}}$ to coordinate module interactions.
- Utilize the coordination manager $\Omega$ to register modules and ensure reliable packet delivery.

### Step 3: Universal Streaming, Loading, and Chunking (Module C)

- Partition the model parameters as $\theta = \bigcup_{j \in J} T_j$.
- Index each chunk $T_j$ by $\mathcal{I}(T_j) = \ell_j \in \mathcal{S}_{\text{disk}}$.
- Dynamically stream chunks into memory via $\sigma$ and cache them using $\mu$.

**Step 4: Multidimensional Vocabulary and Tokenization (Module D)**

- Define the complete vocabulary as $\mathcal{V} = \mathcal{V}^{(0)} \cup \mathcal{V}^{(1)}$.
- Assign each token $t \in \mathcal{V}$ a unique identifier, $\mathrm{ID}(t) = \eta(t)$.
- Segment the preprocessed input $X_{\text{proc}}$ into tokens $(t_1, \ldots, t_n)$ using the base tokenizer $\mathcal{T}_{\text{base}}$.

**Step 5: Contextual Embedding and Tokenization (Module E)**

- For each token $t_i$, compute its base embedding $\mathbf{e}_i = E_{\mathrm{B}}(t_i) \in \mathbb{R}^{d_E}$.
- Process the sequence $(\mathbf{c}_1, \ldots, \mathbf{c}_n) = E_{\mathrm{C}}(\mathbf{e}_1, \ldots, \mathbf{e}_n)$.
- Fuse base and contextual embeddings via $\mathbf{z}_i = g(\mathbf{e}_i, \mathbf{c}_i) \in \mathbb{R}^{d_F}$.
- Alternatively, integrate dual-layer representations by

$$\mathbf{z}_i = g\Big(E_{\mathrm{B}}(t_i),\, E_{\text{sup}}(t_i, \xi)\Big).$$

**Step 6: Deep Knowledge Graph Construction (Module F)**

- Construct the Base Knowledge Graph:

$$\mathcal{G}_{\text{BKG}} = \Big(\mathcal{N}_{\text{BKG}},\, \mathcal{E}_{\text{BKG}}\Big),$$

  where nodes represent tokens $t$ with embeddings $E_{\mathrm{B}}(t)$ and edges are determined by $\rho_{\text{base}}(t_i, t_j) \subset \mathcal{R}_{\text{base}}$.

- Formulate the Personal Knowledge Graph:

$$\mathcal{G}_{\text{PKG}} = \Big(\mathcal{N}_{\text{PKG}},\, \mathcal{E}_{\text{PKG}}\Big),$$

  by leveraging personalized embeddings $E_{\text{sup}}(t, \xi)$.

- Fuse the graphs via $\oplus_{\mathcal{K}}$ to obtain:

$$\mathcal{G}_{\text{Unified}} = \mathcal{G}_{\text{BKG}} \cup \mathcal{G}_{\text{PKG}}.$$

**Step 7: Infinite RoPE and Dynamic Vocabulary Updating (Module G)**

- For each token position $i$ and each 2D subspace $j$, define the rotation matrix:

$$R^{(j)}(i) = \begin{pmatrix} \cos\big(i\,\theta_j^*\big) & -\sin\big(i\,\theta_j^*\big) \\ \sin\big(i\,\theta_j^*\big) & \cos\big(i\,\theta_j^*\big) \end{pmatrix}.$$

- Assemble the block-diagonal rotation:

$$R(i) = \mathrm{diag}\Big(R^{(1)}(i), \ldots, R^{(d_{\text{att}}/2)}(i)\Big), \quad \psi(i, v) = R(i)v.$$

- Apply the transformation $\psi(i, \cdot)$ to the query/key vectors within the attention mechanism.
- Dynamically incorporate newly learned tokens via:

$$\Delta_{\mathcal{V}} : \mathcal{V} \times \mathcal{D}_{\text{learn}} \to \mathcal{V}'.$$

### Step 8: Core Model Processing (Module H)

- Process the fused embeddings through the deep model:

$$f_\theta : (\mathbb{R}^{d_F})^n \to \mathcal{Y}, \quad \theta \in \Theta.$$

- The core sub-modules include:
  - **Transformer:** $f_{\theta_{\mathrm{T}}}^{\mathrm{T}}$, which leverages multi-head self-attention (enhanced with RoPE).
  - **RWKV:** $f_{\theta_{\mathrm{R}}}^{\mathrm{RWKV}}$ with recurrence delineated by

$$S_t = \boldsymbol{\lambda} \odot S_{t-1} + \exp(k_t) \odot v_t, \quad Z_t = \boldsymbol{\lambda} \odot Z_{t-1} + \exp(k_t).$$

- Coordinate expert models through:

$$f_\theta^{\mathrm{Unified}} = \Gamma\Big(\{f_{\theta_i}^{(i)}\}_{i \in I_{\mathrm{exp}}}\Big).$$

### Step 9: Titans Memory Architecture (Module I)

- Define the memory bank as:

$$\mathcal{M} = \{(k_i^{\mathcal{M}}, v_i^{\mathcal{M}})\}_{i=1}^{M_{\mathcal{M}}}.$$

- For a given latent representation $r$, compute similarities $s\big(r, k_i^{\mathcal{M}}\big)$ and derive the corresponding attention weights:

$$\alpha_i(x) = \frac{\exp\Big(s\big(r, k_i^{\mathcal{M}}\big)/\tau\Big)}{\sum_j \exp\Big(s\big(r, k_j^{\mathcal{M}}\big)/\tau\Big)}.$$

- Aggregate the memory read as:

$$m(x) = \sum_{i=1}^{M_{\mathcal{M}}} \alpha_i(x) v_i^{\mathcal{M}}.$$

- Compute the parameter update via the meta-learner:

$$\Delta\theta(x) = h\big(m(x)\big), \quad \theta_x = \theta + \Delta\theta(x).$$

### Step 10: Recursive Adaptive Feedback (Module J)

- Define the runtime recursive entities as $\mathcal{E}^{\mathrm{R}} = \{E_i^{\mathrm{R}}\}$.
- Construct the feedback mechanism using trigger $\tau^{\mathrm{R}}$, action $\alpha^{\mathrm{R}}$, and influence $\Delta^{\mathrm{R}}$:

$$\phi^{\mathrm{R}}\big(E_i^{\mathrm{R}}, E_j^{\mathrm{R}}\big) = \Delta^{\mathrm{R}}\big(E_i^{\mathrm{R}}, E_j^{\mathrm{R}}\big) \circ \beta^{\mathrm{R}}\big(E_j^{\mathrm{R}}, E_i^{\mathrm{R}}\big).$$

- Update the overall state via:

$$\Sigma^{\mathrm{R}} \leftarrow \mu^{\mathrm{R}}(\Sigma^{\mathrm{R}}),$$

  thereby ensuring idempotence.

### Step 11: Universal Training System (Module K)

- For a mini-batch $B \subset \mathcal{D}$, compute the loss:

$$\mathcal{L}(\theta; B) = \frac{1}{|B|} \sum_{(x,y)\in B} \ell\Big(f_\theta(x), y\Big) + \lambda_W \, \|\theta\|^2 + \lambda_{\text{sparse}} \, \mathcal{R}_{\text{sparse}}(\theta).$$

- Update parameter chunks via the optimizer:

$$T_j \leftarrow \mathcal{O}\Big(T_j, \nabla_{T_j}\mathcal{L}, \Xi_j\Big).$$

- Within the forward pass, apply the normalization operator $N$, dropout $D$, and implement skip connections $S$.

- Continuously stream and commit parameter updates using $\sigma$.

### Step 12: Final Decoding and Multimodal Output (Module L)

- The deep model produces a latent output $y_{\text{latent}} \in \mathcal{Y}$.

- Decode the latent representation using $\delta : \mathcal{Y} \to \text{Text}$ to obtain $\hat{y}_{\text{text}} = \delta\big(y_{\text{latent}}\big)$.

- Employ the modality decision operator $\mu_{\text{mod}} : \mathcal{Y} \times \mathcal{C}_{\text{task}} \to \mathcal{O}_{\text{mod}}$ to determine any additional output modalities (e.g., images, audio).

- Package the final output as:

$$\mathcal{P}_{\text{out}} = \Big(\text{ID}_{\text{out}}, \hat{y}, \text{Meta}_{\text{out}}\Big),$$

  and route it via $\Omega$ for logging and feedback.

# 8 Integrated End-to-End Algorithm

**Algorithm 2** Eidos Integrated Inference and Training Pipeline

**Input:** Raw input $X_{\text{raw}} \in \Sigma^*$, training set $\mathcal{D}$, task context $\mathcal{C}_{\text{task}}$
**Preprocess:** $X_{\text{proc}} \leftarrow \mathcal{P}_{\text{in}}(X_{\text{raw}})$
**Tokenize:** $(t_1, \ldots, t_n) \leftarrow \mathcal{T}_{\text{base}}(X_{\text{proc}})$
**for** $i = 1$ to $n$ **do**
    $\mathbf{e}_i \leftarrow E_{\text{B}}(t_i)$
**end for**
$(\mathbf{c}_1, \ldots, \mathbf{c}_n) \leftarrow E_{\text{C}}(\mathbf{e}_1, \ldots, \mathbf{e}_n)$
**for** $i = 1$ to $n$ **do**
    $\mathbf{z}_i \leftarrow g(\mathbf{e}_i, \mathbf{c}_i)$
    $\mathbf{z}'_i \leftarrow \psi(i, \mathbf{z}_i)$
**end for**
**Knowledge Graph:** Update $\mathcal{G}_{\text{BKG}}$ and $\mathcal{G}_{\text{PKG}}$, then compute

$$\mathcal{G}_{\text{Unified}} \leftarrow \mathcal{G}_{\text{BKG}} \cup \mathcal{G}_{\text{PKG}}.$$

$y_{\text{latent}} \leftarrow f_\theta(\mathbf{z}'_1, \ldots, \mathbf{z}'_n)$
**Test-Time Adaptation:**
$r \leftarrow g_\theta(X_{\text{proc}})$
**for** $i = 1$ to $M_{\mathcal{M}}$ **do**
    $s_i \leftarrow s\big(r, k_i^{\mathcal{M}}\big)$
**end for**
Compute $\alpha_i \leftarrow \frac{\exp(s_i/\tau)}{\sum_j \exp(s_j/\tau)}$, and evaluate $m(x) \leftarrow \sum_i \alpha_i \, v_i^{\mathcal{M}}$
$\Delta\theta(x) \leftarrow h\big(m(x)\big), \quad \theta_x \leftarrow \theta + \Delta\theta(x)$
**Recursive Feedback:** Update $\Sigma^{\text{R}} \leftarrow \mu^{\text{R}}(\Sigma^{\text{R}})$
**Decoding:** $\hat{y} \leftarrow \delta\big(y_{\text{latent}}\big)$
**if** multimodal output is required **then**
    $\hat{y}_{\text{mod}} \leftarrow \mu_{\text{mod}}(y_{\text{latent}}, \mathcal{C}_{\text{task}})$
**end if**
$\mathcal{P}_{\text{out}} \leftarrow \Big(\text{ID}_{\text{out}}, \hat{y}, \text{Meta}_{\text{out}}\Big)$
**if** training mode is active **then**
    /* **Execute training loop (refer to supplementary Algorithm 13)** */
**end if**

# 9 Conclusion

In conclusion, the **Eidos** framework has been rigorously delineated through exhaustive technical definitions and stringent mathematical formalism. This document encapsulates the following major contributions:

(A) **Input Processing:** The transformation of raw input $X_{\text{raw}}$ into a processed format $X_{\text{proc}}$ via the operator $\mathcal{P}_{\text{in}}$, complete with precise domain and codomain specifications.

(B) **Universal Communication and Data Handling:** The deployment of robust data packets $\mathcal{P}$ and communication channels $\mathcal{C}_{ij}$, coordinated by the central manager $\Omega$ to facilitate seamless inter-module interactions.

(C) **Streaming, Loading, and Chunking:** The systematic partitioning and dynamic caching of model parameters $\theta$ into discrete chunks $\{T_j\}$.

(D) **Multidimensional Vocabulary and Tokenization:** The comprehensive formulation of the vocabulary $\mathcal{V}$ with unique identifiers $\eta$ and the implementation of robust tokenization via $\mathcal{T}_{\text{base}}$.

(E) **Contextual Embedding and Fusion:** The integration of dual-layer representations by fusing base embeddings $E_{\text{B}}$ with contextual features $E_{\text{C}}$ (or $E_{\text{sup}}$) through the fusion operator $g$.

(F) **Knowledge Graphs:** The rigorous construction and amalgamation of the Base Knowledge Graph $\mathcal{G}_{\text{BKG}}$ and the Personal Knowledge Graph $\mathcal{G}_{\text{PKG}}$ into the unified graph $\mathcal{G}_{\text{Unified}}$.

(G) **Infinite RoPE and Dynamic Vocabulary:** The implementation of the RoPE operator $\psi$ alongside the dynamic update operator $\Delta_{\mathcal{V}}$, ensuring adaptive positional encoding and vocabulary refinement.

(H) **Core Model Architectures:** The integration of Transformer and RWKV sub-modules within the overarching deep model $f_\theta$, seamlessly coordinated by $\Gamma$ for unified inference.

(I) **Titans Memory Architecture:** The deployment of a memory bank $\mathcal{M}$ with similarity-based attention mechanisms and adaptive parameter updates facilitated by the meta-learner $h$.

(J) **Recursive Adaptive Feedback:** The systematic incorporation of recursive feedback via the idempotent operator $\mu^{\text{R}}$.

(K) **Universal Training System:** The employment of a comprehensive loss function $\mathcal{L}$, optimizers $\mathcal{O}$, and auxiliary operators $N$, $D$, and $S$ to establish a robust training regimen.

(L) **Final Decoding and Multimodal Output:** The use of the decoding function $\delta$ and the modality selector $\mu_{\text{mod}}$ to generate final outputs in the desired format, with provisions for extensibility to additional modalities.

This meticulously detailed framework presents a robust and integrated blueprint for constructing a persistent, adaptive, and multimodal intelligent system, wherein each component is systematically defined and cohesively interconnected—from the initial input processing to the final output delivery.

# 10 Module A: Input Processing

Module A: Input Processing
Part of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence —

**Abstract**

This module explicates the *Input Processing* component of the Eidos framework. Its primary objective is to acquire raw data from a diverse array of external sources (e.g., text, images, sensor signals) and to transform it into a standardized format suitable for subsequent processing. We examine the intrinsic characteristics of the raw input, delineate the preprocessing function, and describe its transformation into a canonical form. This essential first stage guarantees that all downstream modules receive data that is consistent and optimally prepared.

# 11 Introduction and Motivation

In any advanced intelligent system, the quality and uniformity of input data are of paramount importance. The **Input Processing** module of the Eidos framework is tasked with acquiring raw data from diverse sources and converting it into a standardized representation. This process ensures that subsequent components—such as tokenization, embedding, and deep model processing—operate on data that is both consistent and devoid of extraneous noise. The principal objectives of this module are to:

(a) Standardize heterogeneous raw inputs (e.g., text, images, sensor data) into a canonical format.

(b) Clean, normalize, and pre-process data to facilitate reliable feature extraction by subsequent modules.

(c) Provide a well-defined interface for the conversion of raw input $X_{\text{raw}}$ to processed input $X_{\text{proc}}$.

# 12 Preliminaries and Notation

- **Raw Input:** Let $X_{\text{raw}} \in \Sigma^*$ denote the raw input data, where $\Sigma$ represents the base alphabet or signal set (e.g., Unicode for text, pixel values for images).
- **Preprocessed Input:** The standardized, cleaned input is denoted by $X_{\text{proc}} \in \mathcal{X}_{\text{proc}}$, with $\mathcal{X}_{\text{proc}}$ representing the canonical data domain employed by subsequent modules.
- **Preprocessing Function:** The transformation from raw to preprocessed data is effectuated by the function
$$\mathcal{P}_{\text{in}} : \Sigma^* \to \mathcal{X}_{\text{proc}}.$$

# 13 Formal Definitions and Mathematical Formulation

### Definition 1 (Preprocessing Function)

The preprocessing function $\mathcal{P}_{\text{in}}$ is defined as a mapping that cleanses, normalizes, and standardizes raw input data:
$$\mathcal{P}_{\text{in}}(X_{\text{raw}}) = X_{\text{proc}},$$
where the operation encompasses:

(i) **Normalization:** Conversion of input data into a standard format (e.g., Unicode normalization for text).

(ii) **Noise Removal:** The elimination of extraneous symbols, correction of errors, and filtering of irrelevant content.

(iii) **Formatting:** Structuring the data into a consistent format (e.g., tokenizable text strings, standardized dimensions for images).

### Properties

The function $\mathcal{P}_{\text{in}}$ is characterized by:

- **Determinism:** For any given $X_{\text{raw}}$, the function consistently produces the identical output $X_{\text{proc}}$.
- **Robustness:** The capacity to manage a wide spectrum of input anomalies and noise.
- **Modularity:** The output is formatted to ensure compatibility with subsequent modules, thereby facilitating loose coupling.

## 14 Algorithmic Description

The following pseudocode outlines the Input Processing module:

**Algorithm 3** Input Processing

**Input:** Raw input $X_{\text{raw}} \in \Sigma^*$
**Output:** Preprocessed input $X_{\text{proc}} \in \mathcal{X}_{\text{proc}}$
**Begin:**
  Apply normalization: $X_{\text{norm}} \leftarrow \text{Normalize}(X_{\text{raw}})$
  Remove noise: $X_{\text{clean}} \leftarrow \text{NoiseRemoval}(X_{\text{norm}})$
  Format data: $X_{\text{proc}} \leftarrow \text{FormatData}(X_{\text{clean}})$
**Return:** $X_{\text{proc}}$

## 15 Theoretical Analysis and Guarantees

### Theorem 1 (Determinism of $\mathcal{P}_{\text{in}}$)

**Statement:** For any fixed raw input $X_{\text{raw}} \in \Sigma^*$, the preprocessing function $\mathcal{P}_{\text{in}}$ yields a unique, well-defined output $X_{\text{proc}} \in \mathcal{X}_{\text{proc}}$.

**Proof Sketch:** Given that normalization, noise removal, and formatting are standard, deterministic operations (assuming fixed parameters and rules), the composite function $\mathcal{P}_{\text{in}} = \text{FormatData} \circ \text{NoiseRemoval} \circ \text{Normalize}$ is inherently deterministic. □

## 16 Integration with the Overall Eidos Framework

The Input Processing module constitutes the initial stage of the Eidos pipeline. Its output, $X_{\text{proc}}$, serves as the essential input for subsequent components—including the Multidimensional Vocabulary and Tokenization system (Module D), embedding modules, knowledge graph construction, and deep model processing—thereby ensuring that all downstream components operate on data of the highest integrity and consistency.

## 17 Implementation Considerations

- The implementation of $\mathcal{P}_{\text{in}}$ must accommodate language-specific normalization protocols, particularly for multilingual datasets.

- For textual data, established libraries (e.g., ICU for Unicode normalization) should be utilized.
- For modalities such as images or sensor data, corresponding normalization and noise reduction strategies should be developed.
- The module must be optimized for high-throughput streaming environments, ensuring minimal latency in real-time applications.

# 18 Conclusion

The Input Processing module is an indispensable first component of the Eidos framework. It systematically transforms the raw input $X_{\text{raw}}$ into a refined and standardized representation $X_{\text{proc}}$, thereby ensuring that all subsequent modules operate on data of uncompromised quality. Its deterministic, robust, and modular design underpins the overall performance and reliability of the system.

# 19 Module B: Universal Communication and Data Handling Interface and Coordination

Module B: Universal Communication and Data Handling Interface and Coordination
Part of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence —

## 20 Abstract

This module delineates the *Universal Communication and Data Handling Interface and Coordination* component of the Eidos framework. Its primary purpose is to establish standardized interfaces and protocols for inter-module communication, thereby ensuring that data and control signals are exchanged in a modular, hardware-agnostic, and dynamically extensible manner. We introduce the concept of a *universal data packet*, define the protocols associated with communication channels via explicit routing functions, and formalize the role of a central coordination manager. Key properties such as idempotence, reversibility, and stability are rigorously defined, accompanied by the algorithmic procedures underpinning packet routing and state updating.

## 21 Introduction and Motivation

In complex, multi-component systems such as Eidos, ensuring seamless inter-module communication is imperative. The **Universal Communication and Data Handling Interface** functions as the backbone of the system by standardizing data formats, coordinating module interactions, and ensuring reliable, real-time updates. This module is responsible for:

- Defining a universal data packet format that encapsulates both the primary data and its associated metadata.
- Establishing standardized communication channels between modules.
- Providing a central coordination manager, denoted as $\Omega$, which orchestrates message routing, enforces communication protocols, and guarantees idempotent and reversible state updates.
- Enabling dynamic expansion, thereby allowing the integration of new modules or modalities without perturbing existing interactions.

## 22 Preliminaries and Notation

We introduce the following notation and definitions:

- $\mathcal{M} = \{M_i \mid i \in I_M\}$ denotes the set of modules within the system.
- For each module $M_i$, the input and output interfaces are defined as:

$$\Phi_{M_i}^{\text{in}} : \mathcal{D}_{\text{in}}^{(i)} \to \mathcal{S}_{M_i}, \quad \Phi_{M_i}^{\text{out}} : \mathcal{S}_{M_i} \to \mathcal{D}_{\text{out}}^{(i)},$$

  where $\mathcal{D}_{\text{in}}^{(i)}$ and $\mathcal{D}_{\text{out}}^{(i)}$ denote the input and output data domains, respectively, and $\mathcal{S}_{M_i}$ represents the internal state space.

- A *universal data packet* is defined as:

$$\mathcal{P} = \big(\text{ID, Payload, Metadata}\big),$$

  where:

  - $\text{ID} \in \mathbb{N}$ serves as a unique identifier for the packet.
  - Payload comprises the primary data (e.g., vectors, tensors).

- Metadata contains auxiliary details (e.g., timestamps, module identifiers, version numbers).

- Communication channels between modules $M_i$ and $M_j$ are denoted by $\mathcal{C}_{ij}$ and incorporate a routing function $\mathcal{R}_{\text{comm}}$.
- The universal coordination manager, $\Omega$, maintains a directory of modules and their interface specifications, overseeing the routing of messages.

# 23 Formal Definitions and Mathematical Formulation

### Definition 1 (Universal Data Packet)

A universal data packet is defined as

$$\mathcal{P} = \big(\text{ID, Payload, Metadata}\big),$$

where:

- ID $\in \mathbb{N}$ uniquely identifies the packet.
- Payload belongs to a vector space $\mathcal{V}_{\mathcal{P}}$ (e.g., $\mathbb{R}^d$).
- Metadata is a structured record containing essential details such as source, destination, timestamp, and data type.

### Definition 2 (Communication Channel)

A communication channel between modules $M_i$ and $M_j$ is defined by the triple:

$$\mathcal{C}_{ij} = \big(\mathcal{I}_{ij},\, \mathcal{P},\, \kappa_{ij}\big),$$

where:

- $\mathcal{I}_{ij} : \mathcal{D}_{\text{out}}^{(i)} \to \mathcal{D}_{\text{in}}^{(j)}$ is the interface mapping.
- $\mathcal{P}$ is the universal data packet as defined above.
- $\kappa_{ij}$ specifies the protocol governing timing, ordering, and error handling.

### Definition 3 (Universal Coordination Manager)

The universal coordination manager $\Omega$ is defined as a service that maintains:

$$\Omega : \mathcal{D} \to \mathcal{P},$$

where:

- $\mathcal{D}$ denotes the directory of all modules:

$$\mathcal{D} = \{(M_i,\, \Phi_{M_i}^{\text{in}},\, \Phi_{M_i}^{\text{out}}) \mid i \in I_M\}.$$

- $\Omega$ employs a routing function $\mathcal{R} : \mathcal{P} \times \mathcal{D} \to \mathcal{P}$ to direct each packet to its intended destination(s).

Furthermore, $\Omega$ maintains comprehensive update logs and facilitates rollback through reversible update operators $U_i^{-1}$ provided by individual modules.

### Definition 4 (Idempotence and Reversibility in Communication)

Let $U_i : \mathcal{S}_{M_i} \times \mathcal{P} \to \mathcal{S}_{M_i}$ denote the state update function for module $M_i$. Then, for all $s \in \mathcal{S}_{M_i}$ and for all packets $p \in \mathcal{P}$:

$$U_i\big(U_i(s,p),p\big) = U_i(s,p).$$

Additionally, there exists an inverse operator $U_i^{-1}$ such that:

$$U_i^{-1}\big(U_i(s,p),p\big) = s.$$

## 24 Algorithmic Description

The following pseudocode outlines the universal data pipeline and coordination process:

**Algorithm 4** Universal Data Pipeline and Coordination

1: **Module Registration:** Each module $M_i$ registers with $\Omega$ by providing its input/output interfaces $\Phi_{M_i}^{\text{in/out}}$.
2: **Packet Creation:** When a module $M_i$ produces output, it encapsulates this output in a universal data packet
$$p = \big(\text{ID, Payload, Metadata}\big).$$
3: **Routing:** The coordination manager $\Omega$ receives $p$ and applies the routing function $\mathcal{R}$ to determine the target module(s) $M_j$.
4: **Delivery:** For each designated target module $M_j$, $\Omega$ delivers the packet via the interface mapping:
$$\mathcal{I}_{ij}(p).$$
5: **State Update:** Module $M_j$ updates its internal state using its update operator $U_j$ with the received packet.
6: **Dynamic Expansion:** New modules $M_k$ may register at any time, prompting $\Omega$ to update its directory $\mathcal{D}$ accordingly.

## 25 Theoretical Analysis and Guarantees

### Theorem 1 (Universality of Communication)

**Statement:** For any collection of modules $\mathcal{M}$ adhering to the standardized interfaces, every universal data packet $p$ is reliably delivered to its intended destination(s) via $\Omega$, ensuring that the overall system state remains stable under repeated updates.

**Proof Sketch:** Given that every packet $p$ is uniquely identified and structured (per Definition 1), and that the routing function $\mathcal{R}$ operates on the complete module directory $\mathcal{D}$, combined with the fact that each module's update operator is idempotent (as per Definition 4), any repeated or redundant packet delivery does not alter the system state beyond the intended update. □

### Proposition 1 (Dynamic Expansion)

New modules $M_k$ can be seamlessly integrated into $\Omega$ without necessitating modifications to the existing interface mappings, as adapter functions $A_{ik}$ can be composed with current mappings. This ensures that the communication framework is inherently extensible.

## 26 Integration with the Overall Eidos Framework

The Universal Communication and Data Handling Interface is central to the Eidos architecture. Its roles include:

- Facilitating standardized data exchange among modules such as Vocabulary/Tokenization (Module D), Embedding (Module E), Knowledge Graph Construction (Module F), Memory (Module I), and Training (Module K).
- Providing a central coordination service $\Omega$ that ensures idempotence, reversibility, and consistency in state updates.
- Enabling dynamic expansion, thereby accommodating the inclusion of new modalities or submodules without disruption.

## 27 Implementation Considerations

- **Data Packet Format:** The universal data packet should adhere to a standardized format (e.g., JSON, Protocol Buffers) to promote interoperability.
- **Communication Protocols:** High-performance communication protocols (e.g., gRPC) should be employed to implement the channels $\mathcal{C}_{ij}$.
- **Logging and Monitoring:** Comprehensive logging within $\Omega$ is essential to ensure traceability and to facilitate debugging.
- **Dynamic Adaptation:** The system must support asynchronous module registration and interface adaptation without disrupting ongoing operations.

## 28 Conclusion

The Universal Communication and Data Handling Interface and Coordination module establishes a rigorous, modular, and extensible framework for inter-module communication within the Eidos system. By defining universal data packets, standardized interface mappings, and employing a central coordination manager $\Omega$, this module guarantees reliable and efficient data exchange. Its design, which ensures idempotence, reversibility, and dynamic adaptability, renders it an essential backbone of the entire Eidos framework.

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

Module C: Universal Streaming/Handling/Loading/Indexing Module
Part of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence —

## 30 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.

## 31 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.

## 32 Preliminaries and Notation

- Let $\theta \in \Theta \subset \mathbb{R}^p$ denote the complete set of model parameters.
- We assume that $\theta$ 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,
- $\mu_j$ 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 $\mathcal{R}$.

# 33 Formal Definitions and Mathematical Formulation

### Definition 1 (Model Decomposition)

Let $f_\theta$ be a model with parameters $\theta$. A *decomposition function*

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

partitions $\theta$ 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 $\ell_j$ 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 $\sigma$ 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 $\mu$ 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.

## 34 Algorithmic Description

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

**Algorithm 5** 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 $\ell_j$
    Record entry $(id_j, \ell_j, \mu_j)$ 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 $\{C_j\}_{\text{active}}$

## 35 Theoretical Analysis and Guarantees

### Theorem 1 (Universal Executability)

**Statement:** For any model $f_\theta$ 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 $\sigma$ and caching policy $\mu$ such that the model $f_\theta$ 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 $\theta$ 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 $\sigma$ can select a subset of modules based on their dependencies and current resource availability. The caching function $\mu$ further ensures that once modules are loaded, they remain available as needed, and non-critical modules may be evicted. Hence, even if $\theta$ 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.

## 36 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 $\sigma$.
- Ensures efficient memory management via the caching function $\mu$, making the system hardware-agnostic.
- Interfaces with the Universal Communication module (Module B) to allow modules to be updated and replaced dynamically.

## 37 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 $\sigma$ should be optimized to minimize I/O latency (e.g., through asynchronous prefetching and parallel loading).
- **Caching Policies:** The caching function $\mu$ 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 $\sigma$ and $\mu$ accordingly.

## 38 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.

# 39 Module D: Multidimensional Vocabulary and Tokenization System

Module D: Multidimensional Vocabulary and Tokenization System

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

## 40 Abstract

This module rigorously defines the *Multidimensional Vocabulary and Tokenization System* of the Eidos framework. It provides a comprehensive vocabulary that unifies traditional lexicons—including English words, Unicode characters, and programming language symbols—with dynamically learned multi-token sequences. Each token is endowed with a structured, multidimensional representation capturing its intrinsic properties and contextual statistics. We define unique identifier mappings, token structures, and associated embedding representations in full mathematical detail. This vocabulary serves as the linguistic and symbolic foundation for downstream embedding, knowledge graph construction, and deep model architectures. The design is modular, extensible, and engineered to support large-scale vocabularies with millions of tokens.

## 41 Introduction and Motivation

The effectiveness of natural language understanding and processing systems fundamentally relies on a rich and comprehensive vocabulary. In the Eidos framework, the *Multidimensional Vocabulary* is the bedrock upon which all higher-level representations (e.g., contextual embeddings and knowledge graphs) are built. The objectives of this module are:

(a) To unify diverse token sources—such as natural language words, Unicode symbols, and programming language constructs—into a single, comprehensive vocabulary.

(b) To associate each token with a unique identifier and a structured set of attributes that capture semantic, syntactic, morphological, and contextual properties.

(c) To support the inclusion of dynamically learned multi-token sequences, allowing the vocabulary to grow and adapt over time.

(d) To provide a foundation for subsequent embedding and tokenization modules, ensuring that all downstream processes have access to robust and multi-faceted token representations.

This module is essential for ensuring that the entire Eidos system operates on a deep and unified understanding of the input symbols.

## 42 Preliminaries and Notation

We introduce the following notation to ensure clarity and prevent redundancy:

- $\Sigma$: The base alphabet (e.g., Unicode) from which raw text is composed.
- $X_{\text{proc}} \in \mathcal{X}_{\text{proc}}$: Preprocessed input text.
- $\mathcal{V}^{(0)}$: The base vocabulary consisting of natural language tokens (e.g., English words), Unicode characters, and programming symbols.
- $\mathcal{V}^{(1)}$: The set of learned multi-token sequences (e.g., frequent n-grams or phrases) discovered through unsupervised segmentation.
- $\mathcal{V}$: The complete vocabulary, defined as

$$\mathcal{V} = \mathcal{V}^{(0)} \cup \mathcal{V}^{(1)}.$$

- For any token $t \in \mathcal{V}$:
$$t = \big(u,\, \pi,\, \chi\big),$$
where:
  - $u$ is the underlying unit (a string or symbol),
  - $\pi \in \Pi \subseteq \mathbb{R}^{d_\pi}$ is a vector of intrinsic properties (e.g., syntactic category, morphological features),
  - $\chi \in \mathbb{R}^{d_\chi}$ is a vector of contextual statistics (e.g., frequency, co-occurrence distribution).
- Unique identifier mapping:
$$\eta : \mathcal{V} \to \mathbb{N},$$
which is injective. For each token $t$, define:
$$\mathrm{ID}(t) = \eta(t).$$

# 43 Formal Definitions and Mathematical Formulation

### Definition D.1 (Base Token Spaces)

We define several foundational token sets:

(i) **Natural Language Tokens:**
$$\mathcal{V}_{\mathrm{NL}} = \{w \mid w \text{ is a valid natural language token (e.g., an English word)}\}.$$

(ii) **Unicode Characters:**
$$\mathcal{V}_{\mathrm{UC}} = \{c \mid c \in \Sigma\}.$$

(iii) **Programming Language Symbols:**
$$\mathcal{V}_{\mathrm{PL}} = \{p \mid p \text{ is a keyword, operator, or symbolic token in a programming language}\}.$$

Define the *base vocabulary* as:
$$\mathcal{V}_{\mathrm{base}} = \mathcal{V}_{\mathrm{NL}} \cup \mathcal{V}_{\mathrm{UC}} \cup \mathcal{V}_{\mathrm{PL}}.$$
Additionally, let $\mathcal{V}_{\mathrm{LT}} \subset \mathcal{P}(\Sigma^*)$ denote the set of *learned tokens* (multi-token sequences). The complete vocabulary is then:
$$\mathcal{V} = \mathcal{V}_{\mathrm{base}} \cup \mathcal{V}_{\mathrm{LT}}.$$

### Definition D.2 (Token Structure and Attributes)

Each token $t \in \mathcal{V}$ is represented as a tuple:
$$t = \big(u,\, \pi,\, \chi\big),$$
with:

- $u$: The underlying unit (e.g., the string "cat" or the symbol "`for`"),
- $\pi \in \Pi \subseteq \mathbb{R}^{d_\pi}$: A vector of intrinsic properties, capturing aspects such as part-of-speech, morphological features, or syntactic roles,
- $\chi \in \mathbb{R}^{d_\chi}$: A vector of contextual statistics, such as token frequency, co-occurrence distributions, or learned contextual signatures.

### Definition D.3 (Unique Identification)

We define an injective mapping:

$$\eta : \mathcal{V} \to \mathbb{N},$$

which assigns each token a unique identifier:

$$\mathrm{ID}(t) = \eta(t).$$

The total vocabulary size is denoted by:

$$M = |\mathcal{V}|,$$

with $M$ typically on the order of millions (e.g., $M \geq 2 \times 10^6$).

### Definition D.4 (Embedding Function)

Although the primary focus of this module is vocabulary construction and tokenization, we briefly define the embedding function for completeness:

$$E : \mathcal{V} \to \mathbb{R}^{d_E},$$

which maps each token $t$ to a high-dimensional vector $E(t)$. Often, this is computed as a composition:

$$E(t) = E_u(u) \oplus E_\pi(\pi) \oplus E_\chi(\chi),$$

where $E_u$, $E_\pi$, and $E_\chi$ are sub-embeddings corresponding to each component, and $\oplus$ denotes concatenation or another combination method.

## 44 Algorithmic Description: Tokenization Process

The tokenization process uses the defined vocabulary to segment preprocessed input into tokens.

**Algorithm 6** Multidimensional Tokenization Process

1: **Input:** Preprocessed input $X_{\text{proc}} \in \mathcal{X}_{\text{proc}}$
2: **Output:** Token sequence $(t_1, t_2, \ldots, t_n)$ with $t_i \in \mathcal{V}$
3: **Initialize:** $S \leftarrow$ empty list
4: **for** each segment $s$ in $X_{\text{proc}}$ **do**
5: Identify candidate tokens $\{t \in \mathcal{V} \mid t \text{ matches a substring of } s\}$
6: Resolve ambiguities via a scoring function (e.g., frequency or context-based likelihood)
7: Append the highest-scoring token to $S$
8: **end for**
9: **Return:** $S = (t_1, t_2, \ldots, t_n)$

**Remark:** The tokenization process can be refined by incorporating advanced segmentation algorithms (e.g., Byte-Pair Encoding or SentencePiece) and may dynamically update $\mathcal{V}_{\text{LT}}$ based on statistical analyses of the input corpus.

## 45 Theoretical Analysis and Guarantees

### Theorem D.1 (Uniqueness of Token Identifiers)

**Statement:** The mapping $\eta : \mathcal{V} \to \mathbb{N}$ is injective, ensuring that for any two distinct tokens $t_1, t_2 \in \mathcal{V}$, we have $\mathrm{ID}(t_1) \neq \mathrm{ID}(t_2)$.

**Proof Sketch:** By construction, $\eta$ is defined to be injective. Standard dictionary or hash–based methods guarantee unique assignment when collisions are resolved, ensuring the property holds. □

### Proposition D.2 (Extensibility of the Vocabulary)

New tokens, particularly in $\mathcal{V}_{\mathrm{LT}}$, can be added without affecting existing token mappings. Formally, if

$$\mathcal{V}' = \mathcal{V} \cup \Delta\mathcal{V},$$

then there exists an extension $\eta' : \mathcal{V}' \to \mathbb{N}$ such that

$$\eta'(t) = \eta(t) \quad \text{for all } t \in \mathcal{V}.$$

Thus, the vocabulary is modular and extensible.

## 46 Integration with the Overall Eidos Framework

The Multidimensional Vocabulary and Tokenization System (Module D) is foundational for Eidos. Its outputs serve as the basis for:

- **Contextual NLU/NLP Embedding (Module E):** The token sequence $(t_1, \ldots, t_n)$ and unique IDs are used for embedding lookup and contextualization.
- **Knowledge Graph Construction (Module F):** Tokens and their associated multidimensional attributes form nodes in the knowledge graphs.
- **Model Loading and Indexing (Module C):** Unique identifiers assist in managing token-related parameters across the system.

## 47 Implementation Considerations

- **Corpus Collection:** The initial construction of $\mathcal{V}^{(0)}$ and $\mathcal{V}^{(1)}$ requires a large, diverse corpus covering natural language, code, and other relevant modalities.
- **Segmentation Algorithms:** Methods such as Byte-Pair Encoding (BPE) or SentencePiece can be employed to extract frequent multi-token sequences for $\mathcal{V}_{\mathrm{LT}}$.
- **Storage:** The vocabulary and its associated attributes (intrinsic and contextual) should be stored in a structured database to allow efficient lookup and updates.
- **Integration with Embedding Layers:** The function $E$ must be designed to combine sub-embeddings (e.g., $E_u$, $E_\pi$, $E_\chi$) in a manner that preserves all linguistic nuances.
- **Dynamic Updates:** The system should allow for periodic or continuous updates to $\mathcal{V}_{\mathrm{LT}}$ as new data is processed.

# 48 Conclusion

In this module, we have defined a comprehensive, multidimensional vocabulary that serves as the fundamental building block for the Eidos framework. Key contributions include:

- A unified vocabulary $\mathcal{V}$ combining base tokens (natural language, Unicode, programming symbols) with learned multi-token sequences.
- A formal token structure $t = (u, \pi, \chi)$ that captures semantic, syntactic, and contextual attributes.
- An injective mapping $\eta : \mathcal{V} \to \mathbb{N}$ ensuring unique identification.
- An algorithmic tokenization process that segments preprocessed input into a sequence of tokens from $\mathcal{V}$.
- Theoretical guarantees regarding uniqueness and extensibility.

This module forms the cornerstone of the Eidos system, ensuring that all subsequent processing is grounded in a deep and richly structured understanding of the input symbols.

# 49 Module E: Contextual NLU/NLP Embedding and Multidimensional Tokenization

Module E: Contextual NLU/NLP Embedding and Multidimensional Tokenization
Part of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence —

## 50 Abstract

This module defines the *Contextual NLU/NLP Embedding and Multidimensional Tokenization* system of the Eidos framework. Building upon the multidimensional vocabulary (Module D), it maps the tokenized input sequence into high-dimensional representations through a two-tiered process. First, a base embedding function obtains stable representations for each token. Next, a contextual embedding module refines these representations by incorporating dynamic, context-sensitive information derived from the entire token sequence. A fusion operator then combines the base and contextual embeddings to yield the final token representation. This dual-layer approach enables the model to maintain core lexical properties while adapting to semantic, syntactic, and usage-specific nuances, thus forming a critical input for downstream components such as knowledge graph construction and deep model processing.

## 51 Introduction and Motivation

Natural language understanding (NLU) and processing (NLP) require that each token be represented in a way that captures both its inherent meaning and its contextual usage. In the Eidos framework, the *Contextual Embedding and Tokenization* module accomplishes this by employing a two-stage embedding process:

(a) **Base Embedding:** Each token $t$ (as defined in Module D) is initially mapped to a fixed, high-dimensional vector using a function $E_{\mathrm{B}}$. This representation captures the stable, lexical attributes of the token.

(b) **Contextual Embedding:** The sequence of base embeddings is then processed by a contextual encoder $E_{\mathrm{C}}$ (e.g., a Transformer encoder or a recurrent network), which refines each token's representation based on its surrounding context.

Finally, a fusion function $g$ integrates these two representations to produce a final token embedding $E_{\mathrm{F}}(t, \xi)$ that encapsulates both base and adaptive, context-sensitive features. This robust representation is essential for subsequent modules, including knowledge graph construction and deep model architectures.

## 52 Preliminaries and Notation

We assume that the Multidimensional Vocabulary defined in Module D is available. Hence, let:

- $\mathcal{V}$ denote the complete vocabulary with tokens $t$ each represented as
$$t = \big(u,\, \pi,\, \chi\big),$$
where $u$ is the underlying unit, $\pi \in \Pi \subseteq \mathbb{R}^{d_\pi}$ captures intrinsic properties, and $\chi \in \mathbb{R}^{d_\chi}$ contains contextual statistics.

- The unique identifier mapping is given by
$$\eta : \mathcal{V} \to \mathbb{N}.$$

- A preprocessed input $X_{\mathrm{proc}} \in \mathcal{X}_{\mathrm{proc}}$ is tokenized by the base tokenizer $\mathcal{T}_{\mathrm{base}}$ (Module D) into a sequence
$$(t_1, t_2, \ldots, t_n),$$
where each $t_i \in \mathcal{V}$.

We now introduce the following functions:

- **Base Embedding Function:**
$$E_{\mathrm{B}} : \mathcal{V} \to \mathbb{R}^{d_E},$$
which maps each token to a stable embedding vector.

- **Contextual Embedding Function:**
$$E_{\mathrm{C}} : (\mathbb{R}^{d_E})^n \to (\mathbb{R}^{d_C})^n,$$
which takes a sequence of base embeddings and produces a sequence of context-sensitive embeddings.

- **Fusion Operator:**
$$g : \mathbb{R}^{d_E} \times \mathbb{R}^{d_C} \to \mathbb{R}^{d_F},$$
which combines the base embedding $E_{\mathrm{B}}(t)$ and the contextual component to produce the final token representation.

We denote the final token representation for token $t_i$ (with context $\xi$) as:
$$E_{\mathrm{F}}(t_i, \xi) = g\Big(E_{\mathrm{B}}(t_i),\, E_{\mathrm{sup}}(t_i, \xi)\Big) \in \mathbb{R}^{d_F},$$
where $E_{\mathrm{sup}}(t_i, \xi)$ is an adaptive, context-refined embedding (computed from $E_{\mathrm{C}}$).

# 53 Formal Definitions and Mathematical Formulation

### Definition E.1 (Base Embedding Function)

The base embedding function $E_{\mathrm{B}}$ is defined as:
$$E_{\mathrm{B}} : \mathcal{V} \to \mathbb{R}^{d_E},$$
where for a token $t = (u,\, \pi,\, \chi)$, we may decompose:
$$E_{\mathrm{B}}(t) = E_u(u) \oplus E_\pi(\pi) \oplus E_\chi(\chi),$$
with $E_u : \Sigma^* \to \mathbb{R}^{d_u}$, $E_\pi : \Pi \to \mathbb{R}^{d'_\pi}$, $E_\chi : \mathbb{R}^{d_\chi} \to \mathbb{R}^{d'_\chi}$, and
$$d_E = d_u + d'_\pi + d'_\chi.$$
This embedding captures the fixed lexical properties of the token.

### Definition E.2 (Contextual Embedding Function)

Given a sequence of base embeddings $\mathbf{e}_1, \ldots, \mathbf{e}_n$, the contextual embedding function is defined as:
$$E_{\mathrm{C}} : (\mathbb{R}^{d_E})^n \to (\mathbb{R}^{d_C})^n,$$
such that for each token position $i$,
$$C_i = E_{\mathrm{C}}(\mathbf{e}_1, \ldots, \mathbf{e}_n)_i \in \mathbb{R}^{d_C}.$$
Typically, $E_{\mathrm{C}}$ is implemented as a deep neural network (e.g., a Transformer encoder) that considers the entire sequence to produce context–sensitive representations.

### Definition E.3 (Adaptive Superset Embedding)

To capture adaptive, dynamic features, we define an updated embedding function:

$$E_{\text{sup}} : \mathcal{V} \times \Xi \to \mathbb{R}^{d_C},$$

where $\Xi$ represents the current context or adaptive parameters (which may include user-specific signals, temporal context, or domain information). The function $E_{\text{sup}}$ is derived from $E_{\text{C}}$ and may be updated continuously:

$$E_{\text{sup}}(t_i, \xi) = f\Big(E_{\text{B}}(t_i),\, E_{\text{C}}(\mathbf{e}_1, \ldots, \mathbf{e}_n)_i,\, \xi\Big),$$

where $f$ is a learnable fusion function.

### Definition E.4 (Fusion Operator)

The final token representation is obtained by fusing the base embedding with the adaptive, context–sensitive component:

$$E_{\text{F}}(t_i, \xi) = g\Big(E_{\text{B}}(t_i),\, E_{\text{sup}}(t_i, \xi)\Big) \in \mathbb{R}^{d_F}.$$

The fusion operator $g : \mathbb{R}^{d_E} \times \mathbb{R}^{d_C} \to \mathbb{R}^{d_F}$ may be implemented as a simple concatenation followed by a linear projection, or as a nonlinear combination (e.g., via gating mechanisms).

## 54 Algorithmic Description

The following pseudocode describes the overall tokenization and embedding process.

**Algorithm 7** Contextual Tokenization and Embedding Process

**Input:** Preprocessed input $X_{\text{proc}} \in \mathcal{X}_{\text{proc}}$
**Output:** Sequence of final token representations $\{E_{\text{F}}(t_i, \xi)\}_{i=1}^{n}$
**Tokenization:** $(t_1, \ldots, t_n) \leftarrow \mathcal{T}_{\text{base}}(X_{\text{proc}})$
**for** each token $t_i$ in the sequence **do**
    Compute base embedding: $\mathbf{e}_i \leftarrow E_{\text{B}}(t_i)$
**end for**
Form the sequence of base embeddings: $\mathbf{E} = (\mathbf{e}_1, \ldots, \mathbf{e}_n)$
Compute contextual embeddings: $(C_1, \ldots, C_n) \leftarrow E_{\text{C}}(\mathbf{E})$
**for** each token $t_i$ in the sequence **do**
    Compute adaptive embedding: $E_{\text{sup}}(t_i, \xi) \leftarrow f\Big(E_{\text{B}}(t_i), C_i, \xi\Big)$
    Fuse embeddings: $E_{\text{F}}(t_i, \xi) \leftarrow g\Big(E_{\text{B}}(t_i), E_{\text{sup}}(t_i, \xi)\Big)$
**end for**
**Return:** $\{E_{\text{F}}(t_i, \xi)\}_{i=1}^{n}$

## 55 Theoretical Analysis and Guarantees

### Theorem E.1 (Preservation of Base Information)

**Statement:** The fusion operator $g$ is designed such that for any token $t$, the final embedding $E_{\text{F}}(t, \xi)$ preserves the information contained in the base embedding $E_{\text{B}}(t)$; i.e., there exists an

(approximate) inversion or a projection ensuring that:

$$E_{\mathrm{B}}(t) \approx \pi_1\big(E_{\mathrm{F}}(t,\xi)\big),$$

where $\pi_1$ denotes the projection onto the subspace corresponding to the base embedding.
**Proof Sketch:** Assuming that $g$ is implemented as a concatenation followed by a linear projection with a non-degenerate weight matrix, standard properties of linear mappings ensure that the original vector $E_{\mathrm{B}}(t)$ is recoverable (up to a linear transformation) from the fused vector $E_{\mathrm{F}}(t,\xi)$. □

### Proposition E.2 (Adaptivity)

The adaptive superset embedding $E_{\mathrm{sup}}(t,\xi)$ is continuously updated (via gradient–based learning or external feedback) such that for a sequence of contexts $\{\xi^{(i)}\}$, the mapping $t \mapsto E_{\mathrm{sup}}(t,\xi^{(i)})$ converges to a stable representation reflective of both common usage and domain-specific adaptations.

## 56 Integration with the Overall Eidos Framework

Module E, the Contextual NLU/NLP Embedding and Multidimensional Tokenization system, is a critical link between the raw token sequence generated by Module D and higher-level processing components:

- It provides the final token representations $\{E_{\mathrm{F}}(t_i,\xi)\}$ which serve as inputs to the Deep Knowledge Graphs (Module F) and the Core Model Architectures (Module H).
- Its dual-layer design ensures that both stable lexical information and dynamic contextual nuances are available for subsequent tasks.
- The interface is designed to be modular and extensible, so that improvements in contextual processing (e.g., more sophisticated encoders) can be integrated without altering the base vocabulary.

## 57 Implementation Considerations

- **Encoder Architecture:** $E_{\mathrm{C}}$ can be implemented using architectures such as Transformers or bidirectional RNNs, with careful tuning to balance capacity and computational efficiency.
- **Fusion Function $g$:** Choices for $g$ include simple concatenation followed by a linear layer or more complex gating mechanisms that learn to weight base and contextual embeddings adaptively.
- **Adaptive Updates:** The function $f$ underlying $E_{\mathrm{sup}}$ should be designed to allow continuous updating, with mechanisms for avoiding catastrophic forgetting of the base embedding.
- **Computational Efficiency:** Batch processing and parallelization should be employed for computing $E_{\mathrm{C}}$ over long sequences.
- **Integration with Training:** The module should be trained end-to-end together with downstream components to ensure that the contextualization adapts to the overall task.

## 58 Conclusion

In this module, we have defined a robust, multidimensional tokenization and contextual embedding framework that transforms preprocessed input into final token representations. Key contributions include:

- A deterministic base embedding function $E_{\mathrm{B}}$ that maps tokens from the multidimensional vocabulary to $\mathbb{R}^{d_E}$.
- A contextual encoder $E_{\mathrm{C}}$ that refines these embeddings based on the entire token sequence.
- An adaptive superset embedding $E_{\mathrm{sup}}(t, \xi)$ that captures dynamic, context-sensitive nuances.
- A fusion operator $g$ that integrates both stable and adaptive information to produce the final token representation $E_{\mathrm{F}}(t, \xi)$.
- Theoretical guarantees that ensure the preservation of base lexical properties and continuous adaptivity.

This module forms a crucial bridge between the foundational vocabulary (Module D) and the subsequent components, such as knowledge graph construction (Module F) and deep model processing (Module H).

# 59 Module F: Deep Knowledge Graphs System (Base and Personal)

Module F: Deep Knowledge Graphs System (Base and Personal)
Part of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence —

## 60 Abstract

This module rigorously defines the *Deep Knowledge Graphs System* of the Eidos framework, which integrates a dual-layer knowledge representation. The first layer, the *Base Knowledge Graph (BKG)*, is constructed from the static, foundational vocabulary and tokenization system and encodes intrinsic semantic, syntactic, and relational information. The second layer, the *Personal Knowledge Graph (PKG)*, dynamically augments the base graph by incorporating adaptive, real-time updates based on model interactions, user feedback, and domain-specific signals. A fusion operator integrates these layers into a unified knowledge graph, enabling robust, scalable, and continuously adaptive knowledge representations for downstream processing. This document details the formal definitions, algorithmic constructions, theoretical guarantees, and integration strategies for this system.

## 61 Introduction and Motivation

In modern natural language and multimodal processing systems, representing the relationships between tokens is crucial for high-level reasoning and understanding. The *Deep Knowledge Graphs System* provides such a representation by constructing two distinct yet interrelated graphs:

(a) The **Base Knowledge Graph (BKG)** captures fundamental relationships (e.g., semantic, syntactic, and lexical associations) derived from a static, pre-defined vocabulary.

(b) The **Personal Knowledge Graph (PKG)** captures dynamic, adaptive relationships that evolve in real time based on contextual and usage-specific signals.

These two layers are integrated via a *fusion operator* to form a unified knowledge graph that supports downstream tasks such as contextual embedding refinement, inference in deep architectures, and continual learning. The dual-layer design ensures both stability (through the base graph) and adaptability (through the personal graph).

## 62 Preliminaries and Notation

We assume the existence of the complete vocabulary $\mathcal{V}$ as defined in Module D. In particular:

- Each token $t \in \mathcal{V}$ is represented as
$$t = \big(u,\, \pi,\, \chi\big),$$
where $u$ is the underlying unit (a string or symbol), $\pi \in \Pi \subseteq \mathbb{R}^{d_\pi}$ encodes intrinsic properties, and $\chi \in \mathbb{R}^{d_\chi}$ contains contextual statistics.

- A unique identifier mapping is provided by
$$\eta : \mathcal{V} \to \mathbb{N},$$
so that $\mathrm{ID}(t) = \eta(t)$.

For the knowledge graphs, we introduce the following additional notation:

- $\mathcal{G}_{\mathrm{BKG}} = (\mathcal{N}_{\mathrm{BKG}}, \mathcal{E}_{\mathrm{BKG}})$: the Base Knowledge Graph.
- $\mathcal{G}_{\mathrm{PKG}} = (\mathcal{N}_{\mathrm{PKG}}, \mathcal{E}_{\mathrm{PKG}})$: the Personal Knowledge Graph.
- $\mathcal{G}_{\mathrm{Unified}}$: the unified knowledge graph resulting from the fusion of the BKG and PKG.

- $\rho_{\text{base}} : \mathcal{V} \times \mathcal{V} \to \mathcal{P}(\mathcal{R}_{\text{base}})$ is a relation function for the base graph, where $\mathcal{R}_{\text{base}}$ denotes the set of relation types (e.g., synonymy, hypernymy, syntactic dependency).
- $\rho_{\text{personal}} : \mathcal{V} \times \mathcal{V} \times \Xi \to \mathcal{P}(\mathcal{R}_{\text{personal}})$ is a relation function for the personal graph, with $\Xi$ representing adaptive, contextual parameters.

# 63 Formal Definitions and Mathematical Formulation

### Definition F.1 (Base Knowledge Graph)

The *Base Knowledge Graph* is defined as:

$$\mathcal{G}_{\text{BKG}} = \big(\mathcal{N}_{\text{BKG}}, \mathcal{E}_{\text{BKG}}\big),$$

where:

- $\mathcal{N}_{\text{BKG}} = \{n_t \mid t \in \mathcal{V}\}$ is the set of nodes, with each node $n_t$ corresponding to a token $t$ and associated with its base embedding $E_{\text{B}}(t) \in \mathbb{R}^{d_E}$.
- $\mathcal{E}_{\text{BKG}}$ is the set of edges defined by a relation function:

  $$\rho_{\text{base}} : \mathcal{V} \times \mathcal{V} \to \mathcal{P}(\mathcal{R}_{\text{base}}),$$

  such that an edge exists between nodes $n_{t_i}$ and $n_{t_j}$ if there exists a relation $r \in \rho_{\text{base}}(t_i, t_j)$. Each edge may be represented as:

  $$e_{ij} = \big(n_{t_i}, n_{t_j}, r\big).$$

### Definition F.2 (Personal Knowledge Graph)

The *Personal Knowledge Graph* is defined as:

$$\mathcal{G}_{\text{PKG}} = \big(\mathcal{N}_{\text{PKG}}, \mathcal{E}_{\text{PKG}}\big),$$

where:

- $\mathcal{N}_{\text{PKG}} = \{n'_t \mid t \in \mathcal{V}\}$ is the set of nodes, each corresponding to a token $t$ but associated with a personalized (adaptive) embedding $E_{\text{sup}}(t, \xi) \in \mathbb{R}^{d_C}$, where $\xi$ represents contextual or domain-specific parameters.
- $\mathcal{E}_{\text{PKG}}$ is defined by a personalized relation function:

  $$\rho_{\text{personal}} : \mathcal{V} \times \mathcal{V} \times \Xi \to \mathcal{P}(\mathcal{R}_{\text{personal}}),$$

  such that an edge exists between nodes $n'_{t_i}$ and $n'_{t_j}$ if there is a personalized relation $r' \in \rho_{\text{personal}}(t_i, t_j, \xi)$. Each edge is represented as:

  $$e'_{ij} = \big(n'_{t_i}, n'_{t_j}, r'\big).$$

**Definition F.3 (Fusion Operator and Unified Knowledge Graph)**

Define a fusion operator:

$$\oplus_{\mathcal{K}} : \mathcal{G}_{\text{BKG}} \times \mathcal{G}_{\text{PKG}} \to \mathcal{G}_{\text{Unified}},$$

which constructs the *Unified Knowledge Graph*:

$$\mathcal{G}_{\text{Unified}} = \mathcal{G}_{\text{BKG}} \cup \mathcal{G}_{\text{PKG}},$$

with:

- $\mathcal{N}_{\text{Unified}} = \mathcal{N}_{\text{BKG}} = \mathcal{N}_{\text{PKG}}$, by aligning nodes via the unique token identifiers.
- $\mathcal{E}_{\text{Unified}} = \mathcal{E}_{\text{BKG}} \cup \mathcal{E}_{\text{PKG}}$, with each edge annotated by its source (base or personal), and possibly weighted by confidence scores.

This operator guarantees that the unified graph retains stable base knowledge while integrating adaptive personal updates.

# 64 Algorithmic Description

We now describe the process for constructing and updating the knowledge graphs.

**Algorithm 8** Construction of Base and Personal Knowledge Graphs

**Input:** Token sequence $(t_1, t_2, \ldots, t_n)$ from $\mathcal{T}_{\text{base}}$; Base embedding function $E_{\text{B}}$; Adaptive embedding function $E_{\text{sup}}(\cdot, \xi)$; Relation functions $\rho_{\text{base}}$ and $\rho_{\text{personal}}$
**Output:** Unified Knowledge Graph $\mathcal{G}_{\text{Unified}}$
**Begin:**
**for** each token $t$ in $(t_1, \ldots, t_n)$ **do**
    Create base node $n_t$ with embedding $E_{\text{B}}(t)$
    Create personal node $n'_t$ with embedding $E_{\text{sup}}(t, \xi)$
**end for**
**for** each pair of tokens $(t_i, t_j)$ **do**
    Determine base relations: $R_{\text{base}} \leftarrow \rho_{\text{base}}(t_i, t_j)$
    For each $r \in R_{\text{base}}$, add edge $(n_{t_i}, n_{t_j}, r)$ to $\mathcal{E}_{\text{BKG}}$
    Determine personal relations: $R_{\text{personal}} \leftarrow \rho_{\text{personal}}(t_i, t_j, \xi)$
    For each $r' \in R_{\text{personal}}$, add edge $(n'_{t_i}, n'_{t_j}, r')$ to $\mathcal{E}_{\text{PKG}}$
**end for**
**Fusion:**

$$\mathcal{G}_{\text{Unified}} \leftarrow \oplus_{\mathcal{K}}\Big(\mathcal{G}_{\text{BKG}}, \mathcal{G}_{\text{PKG}}\Big)$$

**Return:** $\mathcal{G}_{\text{Unified}}$

**Remark:** In practice, the relation functions $\rho_{\text{base}}$ and $\rho_{\text{personal}}$ may be implemented via statistical analyses (e.g., co-occurrence frequencies, syntactic dependency parsing) and can be further refined using supervised or unsupervised methods.

## 65 Theoretical Analysis and Guarantees

### Theorem F.1 (Consistency of Node Identities)

**Statement:** Given that the base and personal graphs are constructed from the same vocabulary $\mathcal{V}$ and use the unique identifier mapping $\eta$, the node sets satisfy:

$$\mathcal{N}_{\text{BKG}} = \mathcal{N}_{\text{PKG}},$$

up to a canonical isomorphism. Consequently, the fusion operator $\oplus_{\mathcal{K}}$ produces a well-defined unified node set.

**Proof Sketch:** Since each token $t \in \mathcal{V}$ is assigned a unique identifier via $\eta$, nodes created in both graphs are aligned by this identifier. Thus, merging the two graphs yields a common node for each token. □

### Proposition F.2 (Extensibility and Modularity)

The dual-layer design allows independent updating of the Base Knowledge Graph (e.g., through periodic retraining) and the Personal Knowledge Graph (via real-time adaptation). The fusion operator is defined so that changes in one layer can be integrated without altering the other, ensuring modularity and extensibility.

## 66 Integration with the Overall Eidos Framework

Module F, the Deep Knowledge Graphs System, serves as the core for representing inter-token relationships. It directly interfaces with:

- **Module D (Vocabulary):** The nodes of both the base and personal graphs are derived from tokens in $\mathcal{V}$.
- **Module E (Contextual Embeddings):** The adaptive embeddings $E_{\text{sup}}(t, \xi)$ provide dynamic features that enrich the personal graph.
- **Subsequent Modules:** The unified knowledge graph $\mathcal{G}_{\text{Unified}}$ is used to inform downstream processes such as reasoning in deep models and further contextual adaptation.

## 67 Implementation Considerations

- **Data Structures:** Graphs $\mathcal{G}_{\text{BKG}}$ and $\mathcal{G}_{\text{PKG}}$ may be stored using graph databases or optimized sparse matrix representations. Edge metadata (e.g., relation types, confidence scores, timestamps) should be maintained.
- **Relation Extraction:** Techniques such as co-occurrence analysis, dependency parsing, and semantic similarity measures can be used to define $\rho_{\text{base}}$ and $\rho_{\text{personal}}$.
- **Fusion Strategy:** The fusion operator $\oplus_{\mathcal{K}}$ should be designed to allow weighted integration of base and personal edges, possibly using confidence scores or temporal weights.
- **Dynamic Updates:** The PKG should support real-time updates, while the BKG remains relatively stable. Efficient incremental graph update algorithms will be necessary.
- **Scalability:** Given that the vocabulary size may reach millions, efficient indexing and retrieval mechanisms for nodes and edges are crucial.

## 68 Conclusion

In this module, we have presented a comprehensive framework for constructing deep knowledge graphs in the Eidos system. The Base Knowledge Graph captures the static, inherent relationships among tokens derived from the multidimensional vocabulary, while the Personal Knowledge Graph continuously adapts to reflect dynamic, context-sensitive information. A fusion operator integrates these two layers into a unified graph, ensuring that the system benefits from both stability and adaptivity. The module includes rigorous definitions, an algorithmic process for graph construction, theoretical guarantees regarding consistency and extensibility, and practical considerations for implementation. This dual-layer knowledge representation forms a critical foundation for advanced semantic reasoning and dynamic adaptation in the overall Eidos framework.

# 69 Module G: Infinite RoPE Context Scaling and Dynamic Vocabulary Updating

Module G: Infinite RoPE Context Scaling and Dynamic Vocabulary Updating
Part of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence —

## 70 Abstract

This module rigorously defines the *Infinite RoPE Context Scaling and Dynamic Vocabulary Updating* component of the Eidos framework. It introduces Rotary Positional Embeddings (RoPE) as a mechanism to encode relative positional information in a manner that naturally extends to arbitrarily long (or even infinite) contexts. In addition, this module formalizes a dynamic vocabulary updating mechanism, which continuously integrates new multi-token sequences into the existing vocabulary. By merging infinite-context scaling with adaptive vocabulary expansion, the module provides the linguistic and structural backbone for long-range dependencies and continual learning. The framework is presented with full mathematical rigor, including formal definitions, algorithmic descriptions, and theoretical guarantees.

## 71 Introduction and Motivation

Modern language and multimodal models must handle sequences whose lengths are not fixed a priori. Traditional absolute positional encodings impose a hard upper bound on context length, limiting the model's ability to process long sequences. Rotary Positional Embeddings (RoPE) overcome this limitation by encoding relative positional information through continuous, rotational transformations applied to query and key vectors in attention mechanisms. Simultaneously, as models process increasing amounts of data, the vocabulary must evolve to incorporate new, frequently occurring multi-token sequences. The dual objectives of infinite context scaling and dynamic vocabulary updating are critical for a truly adaptive and extensible system.

The goals of this module are to:

(a) Define a mathematically rigorous formulation of RoPE that supports infinite context scaling.

(b) Establish theoretical guarantees (e.g., relative positional invariance) for the RoPE mechanism.

(c) Formally define a dynamic vocabulary updating function that integrates new multi-token sequences into the existing vocabulary without loss of consistency.

(d) Detail the algorithmic procedures for applying RoPE and updating the vocabulary in real time.

## 72 Preliminaries and Notation

We introduce the following notation to ensure clarity:

- Let $\Sigma$ denote the base alphabet (e.g., Unicode).
- $X_{\text{proc}} \in \mathcal{X}_{\text{proc}}$ is preprocessed input (from Module A).
- $\mathcal{V}$ denotes the complete vocabulary, defined as:

$$\mathcal{V} = \mathcal{V}^{(0)} \cup \mathcal{V}^{(1)},$$

  where $\mathcal{V}^{(0)}$ is the base vocabulary (natural language tokens, Unicode characters, programming symbols) and $\mathcal{V}^{(1)}$ is the set of learned multi-token sequences.

- Each token $t \in \mathcal{V}$ is represented as:
$$t = \big(u,\, \pi,\, \chi\big),$$
with:
  - $u$ as the underlying unit (string/symbol),
  - $\pi \in \Pi \subseteq \mathbb{R}^{d_\pi}$ representing intrinsic properties,
  - $\chi \in \mathbb{R}^{d_\chi}$ representing contextual statistics.
- A unique identifier mapping is defined as:
$$\eta : \mathcal{V} \to \mathbb{N},$$
so that $\mathrm{ID}(t) = \eta(t)$.
- **RoPE Notation:**
  - Let $d_{\text{att}}$ be an even integer representing the dimensionality of the attention subspace.
  - For a vector $v \in \mathbb{R}^{d_{\text{att}}}$, partition it into $\frac{d_{\text{att}}}{2}$ sub-vectors $v^{(j)} \in \mathbb{R}^2$ for $j = 1, \ldots, \frac{d_{\text{att}}}{2}$.
  - For each subspace $j$, let $\theta_j \in \mathbb{R}$ be the frequency parameter.
  - Define the rotation angle at position $i$ as:
$$\varphi_j(i) = i\,\theta_j.$$
  - Define the 2D rotation matrix for subspace $j$ at position $i$ as:
$$R^{(j)}(i) = \begin{pmatrix} \cos\big(\varphi_j(i)\big) & -\sin\big(\varphi_j(i)\big) \\ \sin\big(\varphi_j(i)\big) & \cos\big(\varphi_j(i)\big) \end{pmatrix}.$$
  - The block-diagonal rotation operator is then defined as:
$$R(i) = \mathrm{diag}\Big(R^{(1)}(i),\, R^{(2)}(i),\, \ldots,\, R^{(d_{\text{att}}/2)}(i)\Big).$$
  - The RoPE transformation is given by:
$$\psi(i, v) = R(i)\, v.$$
- **Dynamic Vocabulary Updating:**
  - Let $\mathcal{D}_{\text{learn}}$ denote the set of learning signals or new data from which additional multi-token sequences are extracted.
  - Define the dynamic vocabulary update function:
$$\Delta_{\mathcal{V}} : \mathcal{V} \times \mathcal{D}_{\text{learn}} \to \mathcal{V}',$$
which expands the vocabulary to $\mathcal{V}'$ such that $\mathcal{V} \subset \mathcal{V}'$.

# 73 Formal Definitions and Mathematical Formulation

### Definition G.1 (RoPE Transformation)

Given a vector $v \in \mathbb{R}^{d_{\text{att}}}$ and a position $i \in \mathbb{N}$, the RoPE transformation is defined as:

$$\psi(i, v) \;=\; R(i)\, v,$$

where the block-diagonal rotation operator $R(i)$ is:

$$R(i) \;=\; \operatorname{diag}\Big(R^{(1)}(i),\, R^{(2)}(i),\, \ldots,\, R^{(d_{\text{att}}/2)}(i)\Big),$$

with each 2D rotation matrix

$$R^{(j)}(i) \;=\; \begin{pmatrix} \cos\big(i\,\theta_j\big) & -\sin\big(i\,\theta_j\big) \\ \sin\big(i\,\theta_j\big) & \cos\big(i\,\theta_j\big) \end{pmatrix}.$$

This transformation is applied elementwise to the query and key vectors in the attention mechanism, ensuring that the inner product between rotated queries and keys depends solely on their relative positions.

### Theorem G.1 (Relative Invariance of the RoPE Dot Product)

**Statement:** For any two positions $i, j \in \mathbb{N}$ and any vectors $q, k \in \mathbb{R}^{d_{\text{att}}}$, let

$$q_i' = \psi(i, q), \quad k_j' = \psi(j, k).$$

Then, the dot product

$$q_i'^{\top} k_j' = q^{\top} R(i)^{\top} R(j)\, k$$

depends only on the relative position $j - i$; that is, there exists a function $\Phi$ such that:

$$q_i'^{\top} k_j' = \Phi(q, k; j - i).$$

**Proof Sketch:** Since each $R^{(j)}(i)$ is an orthogonal matrix, $R(i)^{\top} R(j)$ equals a block-diagonal matrix with blocks $R^{(j)}(j - i)$. Therefore, the dot product is computed as a sum of terms of the form:

$$q^{(j)\top} R^{(j)}(j - i) k^{(j)},$$

which depends solely on the difference $j - i$. □

### Definition G.2 (Infinite Context Scaling)

The RoPE transformation $\psi(i, v)$ is defined for all $i \in \mathbb{N}$. Thus, there is no fixed maximum context length:

$$\forall\, i \in \mathbb{N}, \quad \psi(i, v) \text{ is well-defined.}$$

This property enables the model to process sequences of arbitrarily long (or even infinite) length without modification to the positional encoding mechanism.

**Definition G.3 (Dynamic Vocabulary Update)**

Let $\mathcal{V}$ be the current vocabulary and $\mathcal{D}_{\text{learn}}$ be a set of learning signals derived from new input data. The dynamic vocabulary update function is:

$$\Delta_{\mathcal{V}} : \mathcal{V} \times \mathcal{D}_{\text{learn}} \to \mathcal{V}',$$

such that:

(i) For any $t \in \mathcal{V}$, $t \in \mathcal{V}'$ (preservation of the base vocabulary).

(ii) For any novel multi-token sequence $\delta \in \mathcal{D}_{\text{learn}}$ that meets a predetermined frequency or confidence threshold, $\Delta_{\mathcal{V}}$ incorporates $\delta$ as a new token $t_\delta$ into $\mathcal{V}'$.

(iii) There exists an extension of the unique identifier mapping:

$$\eta' : \mathcal{V}' \to \mathbb{N},$$

such that for all $t \in \mathcal{V}$, $\eta'(t) = \eta(t)$.

This mechanism allows the vocabulary to grow dynamically, incorporating additional tokens that capture frequently occurring multi-token patterns.

# 74 Algorithmic Description

The following pseudocode describes the application of the RoPE transformation to token representations and the dynamic vocabulary update process.

**Algorithm 9** Infinite Context Scaling with RoPE and Dynamic Vocabulary Updating

**Input:** Sequence of tokens $(t_1, \ldots, t_n)$ with base embeddings $\{E_{\mathrm{B}}(t_i)\}$; frequency parameters $\{\theta_j\}_{j=1}^{d_{\mathrm{att}}/2}$; new learning signals $\mathcal{D}_{\mathrm{learn}}$; current vocabulary $\mathcal{V}$
**for** each position $i = 1, \ldots, n$ **do**
  **for** each subspace $j = 1, \ldots, d_{\mathrm{att}}/2$ **do**
    Compute rotation angle: $\varphi_j(i) \leftarrow i \cdot \theta_j$
    Form rotation matrix:

$$R^{(j)}(i) \leftarrow \begin{pmatrix} \cos(\varphi_j(i)) & -\sin(\varphi_j(i)) \\ \sin(\varphi_j(i)) & \cos(\varphi_j(i)) \end{pmatrix}$$

  **end for**
  Construct block-diagonal rotation:

$$R(i) \leftarrow \mathrm{diag}\Big(R^{(1)}(i), \ldots, R^{(d_{\mathrm{att}}/2)}(i)\Big)$$

  Apply to token's query/key (if applicable): $q_i' \leftarrow \psi(i, q_i)$, $k_i' \leftarrow \psi(i, k_i)$
**end for**
**Dynamic Vocabulary Update:**
**for** each candidate multi-token sequence $\delta \in \mathcal{D}_{\mathrm{learn}}$ **do**
  **if** $\delta$ satisfies the frequency/confidence threshold **then**
    Add $\delta$ as a new token $t_\delta$ to $\mathcal{V}'$
    Extend identifier mapping: set $\eta'(t_\delta) \leftarrow$ new unique ID
  **end if**
**end for**
**Return:** Updated vocabulary $\mathcal{V}'$ and rotated vectors $\{q_i', k_i'\}$

# 75 Theoretical Analysis and Guarantees

### Theorem G.1 (Relative Invariance of RoPE)

**Statement:** For any two positions $i, j \in \mathbb{N}$ and any vectors $q, k \in \mathbb{R}^{d_{\mathrm{att}}}$, the dot product of the RoPE-transformed vectors satisfies:

$$q_i'^{\top} k_j' = \Phi(q, k; j - i),$$

i.e., it depends solely on the relative position $j - i$.

**Proof Sketch:** Since each subspace rotation $R^{(j)}(i)$ is orthogonal, we have

$$R(i)^{\top} R(j) = \mathrm{diag}\Big(R^{(1)}(j - i), \ldots, R^{(d_{\mathrm{att}}/2)}(j - i)\Big).$$

Thus, the dot product becomes a function of the differences $j - i$ only. □

### Proposition G.2 (Infinite Context Capability)

Because the RoPE transformation $\psi(i, v)$ is defined for all $i \in \mathbb{N}$, the mechanism imposes no fixed limit on the sequence length. Therefore, the model can handle arbitrarily long contexts, limited only by computational resources.

**Proposition G.3 (Extensibility of Dynamic Vocabulary)**

The dynamic vocabulary update function $\Delta_{\mathcal{V}}$ guarantees that:

(i) The base vocabulary is preserved: $\forall t \in \mathcal{V},\ t \in \mathcal{V}'$.

(ii) New tokens are integrated seamlessly with unique identifiers maintained via an extension $\eta'$.

Thus, the vocabulary can be extended without disrupting existing mappings, ensuring modularity and consistency.

# 76 Integration with the Overall Eidos Framework

Module G, which provides infinite context scaling via RoPE and dynamic vocabulary updating, is fundamental to ensuring that the Eidos framework can process long-range dependencies and continually adapt its linguistic representation. Its integration points include:

- **Contextual Embedding (Module E):** The RoPE-transformed vectors are used as inputs to the deep contextual encoder, enhancing the attention mechanism with relative positional information.
- **Dynamic Vocabulary (Module D):** The dynamic update function $\Delta_{\mathcal{V}}$ augments the static vocabulary with new multi-token sequences, which are then used in tokenization and embedding.
- **Downstream Processing:** The infinite-context capability ensures that knowledge graphs (Module F) and deep model architectures (Module H) receive enriched, context-aware representations.

# 77 Implementation Considerations

- **Computational Efficiency:** The block-diagonal structure of $R(i)$ allows the RoPE transformation to be computed efficiently even for large $i$. Vectorized implementations in modern deep learning frameworks (e.g., PyTorch or TensorFlow) can further accelerate this process.
- **Frequency Parameter Selection:** The choice of $\theta_j$ is critical for ensuring that the rotations do not degenerate over long sequences. Typically, a decaying schedule (e.g., $\theta_j = 1/10000^{\frac{2(j-1)}{d_{\text{att}}}}$) is used.
- **Dynamic Vocabulary Update Strategy:** Implementing $\Delta_{\mathcal{V}}$ may involve unsupervised segmentation techniques (e.g., Byte-Pair Encoding or SentencePiece) and thresholding based on token frequency and contextual significance.
- **Storage and Retrieval:** New tokens must be indexed and stored efficiently. A database or hash table structure that maps token strings to unique identifiers is recommended.
- **Training and Fine-Tuning:** Both the RoPE mechanism and the dynamic vocabulary update may be trained jointly with the rest of the model, requiring careful balancing of learning rates and update frequencies.

# 78 Conclusion

In this module, we have rigorously defined the mechanisms for infinite context scaling using Rotary Positional Embeddings (RoPE) and the dynamic updating of the vocabulary. Key contributions include:

- A formal definition of the RoPE transformation $\psi(i, v) = R(i)v$ that guarantees relative positional invariance and enables processing of arbitrarily long sequences.
- Theoretical guarantees that the RoPE-transformed dot product depends solely on the relative positions of tokens.
- A dynamic vocabulary update function $\Delta_{\mathcal{V}}$ that extends the vocabulary with newly learned multi-token sequences while preserving the unique identifiers of existing tokens.
- An algorithmic framework that integrates these components efficiently and robustly, ensuring seamless operation within the overall Eidos framework.

This module provides the critical ability for the system to handle long-range dependencies and to adapt its linguistic representation in response to continuous data, thereby forming a vital component of the Eidos architecture.

# 79 Module H: Core Model Architectures (RWKV and Transformer Modules, Mixture-of-Experts Style)

Module H: Core Model Architectures (RWKV and Transformer Modules, Mixture-of-Experts Style)
Part of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence —

## 80 Abstract

This module rigorously defines the *Core Model Architectures* component of the Eidos framework. It encompasses two complementary deep learning architectures: the Transformer and the RWKV, integrated in a mixture-of-experts (MoE) style. The Transformer sub-module provides a powerful self-attention mechanism for capturing long-range dependencies, while the RWKV sub-module offers a recurrent, linear-time alternative. A higher-level expert coordinator aggregates outputs from multiple expert modules into a unified model output. This design balances expressive capacity and computational efficiency, supports dynamic expert addition and removal, and enables robust performance across diverse tasks. We present formal definitions, algorithmic descriptions, theoretical guarantees, and integration strategies with maximum academic rigor.

## 81 Introduction and Motivation

The core processing engine of the Eidos framework is designed to efficiently capture complex dependencies in multimodal data. In this module, we integrate two leading deep architectures:

(a) **Transformer Sub-Module:** Leveraging multi-head self-attention, the Transformer captures long-range dependencies and intricate interactions among token representations.

(b) **RWKV Sub-Module:** A recurrent alternative that computes a context vector via weighted accumulation with learnable decay and gating, offering linear-time complexity.

To maximize both capacity and efficiency, these sub-modules are organized in a *mixture-of-experts* (MoE) framework. A dedicated expert coordinator, denoted by $\Gamma$, aggregates the outputs of a set of expert modules, each specialized to different aspects of the input. This architecture not only enables dynamic expert specialization and scaling but also allows for seamless addition or removal of experts without disrupting overall performance.

## 82 Preliminaries and Notation

We adopt the following notation and assumptions:

- Let $X = (x_1, x_2, \ldots, x_n)$ denote an input sequence, where each $x_i$ is processed into a final token representation $E_{\mathrm{F}}(x_i, \xi) \in \mathbb{R}^{d_F}$ (obtained from Module E).
- The overall deep model is a function:

$$f_\theta : (\mathbb{R}^{d_F})^n \to \mathcal{Y},$$

  parameterized by $\theta \in \Theta$.
- The set of expert sub-modules is denoted by:

$$\mathcal{E} = \{f_{\theta_i}^{(i)} \mid i \in I_{\mathrm{exp}}\},$$

  where each $f_{\theta_i}^{(i)}$ represents an individual expert, which may be either a Transformer module $f_{\theta_{\mathrm{T}}}^{\mathrm{T}}$ or an RWKV module $f_{\theta_{\mathrm{R}}}^{\mathrm{RWKV}}$.
- A higher-level expert coordinator $\Gamma$ aggregates expert outputs to produce a unified prediction:

$$f_\theta^{\mathrm{Unified}} = \Gamma\Big(\{f_{\theta_i}^{(i)}\}_{i \in I_{\mathrm{exp}}}\Big).$$

## 83 Formal Definitions and Mathematical Formulation

### Definition H.1 (Transformer Sub-Module)

Let $Z = (z_1, z_2, \ldots, z_n) \in \mathbb{R}^{n \times d_{\text{model}}}$ be the input representation obtained by adding positional encodings to the token embeddings:

$$z_i = E_{\text{F}}(x_i, \xi) + PE(i).$$

For each Transformer layer, the following operations are performed:

(i) **Linear Projections:**

$$Q = ZW^Q, \quad K = ZW^K, \quad V = ZW^V,$$

with $W^Q, W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$.

(ii) **Scaled Dot-Product Attention:**

$$\text{Attention}(Q, K, V) = \text{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big)V.$$

(iii) **Multi-Head Attention:** With $h$ heads, for head $i$ define:

$$\text{head}_i = \text{Attention}(ZW_i^Q, ZW_i^K, ZW_i^V),$$

and concatenate:

$$\text{MHA}(Z) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O.$$

(iv) **Residual Connection and Layer Normalization:**

$$Z' = \text{LayerNorm}\Big(Z + \text{MHA}(Z)\Big).$$

(v) **Feed-Forward Network:**

$$\text{FFN}(Z') = \sigma\big(Z'W_1 + b_1\big)W_2 + b_2,$$

with subsequent residual and normalization:

$$Z'' = \text{LayerNorm}\Big(Z' + \text{FFN}(Z')\Big).$$

Thus, the Transformer sub-module $f_{\theta_{\text{T}}}^{\text{T}}$ is defined as a stack of such layers.

### Definition H.2 (RWKV Sub-Module)

For a recurrent alternative, the RWKV module processes a sequence $(x_1, \ldots, x_n)$ as follows:

(i) **Embedding:** For each token, compute $z_t = E_{\text{F}}(x_t, \xi)$.

(ii) **Linear Projections:**

$$k_t = W_k z_t + b_k, \quad v_t = W_v z_t + b_v, \quad r_t = \sigma(W_r z_t + b_r),$$

where $r_t \in (0, 1)^d$ acts as a gating (receptance) vector.

(iii) **Recurrent Accumulation:** Initialize $S_0 = \mathbf{0}$ and $Z_0 = \epsilon \mathbf{1}$ (with small $\epsilon > 0$). Then for $t \geq 1$:

$$S_t = \lambda \odot S_{t-1} + \exp(k_t) \odot v_t, \quad Z_t = \lambda \odot Z_{t-1} + \exp(k_t),$$

where $\lambda \in [0,1]^d$ is a (possibly learnable) decay parameter.

(iv) **Output Computation:** The output at time $t$ is given by:

$$y_t = r_t \odot \left( \frac{S_t}{Z_t} \right).$$

The RWKV sub-module $f_{\theta_{\mathrm{R}}}^{\mathrm{RWKV}}$ is defined by applying these recurrent operations over the entire sequence.

### Definition H.3 (Mixture-of-Experts Coordination)

Let the set of expert modules be:

$$\mathcal{E} = \{ f_{\theta_i}^{(i)} \mid i \in I_{\mathrm{exp}} \},$$

where each expert $f_{\theta_i}^{(i)}$ is instantiated as either a Transformer module (as in Definition H.1) or an RWKV module (as in Definition H.2), possibly specialized for different tasks or data aspects.

Define the expert coordinator as a function:

$$\Gamma : \prod_{i \in I_{\mathrm{exp}}} \mathcal{F}^{(i)} \to \mathcal{F}^{\mathrm{Unified}},$$

which aggregates the outputs $\{y^{(i)}\}$ of the individual experts to produce a unified output:

$$f_\theta^{\mathrm{Unified}}(X) = \Gamma\Big( \{ f_{\theta_i}^{(i)}(X) \}_{i \in I_{\mathrm{exp}}} \Big).$$

A common instantiation of $\Gamma$ is a weighted sum or concatenation followed by a linear projection:

$$f_\theta^{\mathrm{Unified}}(X) = \Big[ \sum_{i \in I_{\mathrm{exp}}} w_i \, f_{\theta_i}^{(i)}(X) \Big] W^C,$$

with weights $w_i$ (possibly learned or dynamically computed) and a coordinator projection $W^C$.

## 84 Algorithmic Description

The following pseudocode summarizes the forward pass of the unified core model architecture:

**Algorithm 10** Unified Core Model Forward Pass (Module H)

**Input:** Token sequence $X = (x_1, \ldots, x_n)$; final token representations $\{E_{\mathrm{F}}(x_i, \xi)\}$
**Output:** Model prediction $\hat{y} \in \mathcal{Y}$
**Begin:**
  Compute input representation $Z$ from $\{E_{\mathrm{F}}(x_i, \xi)\}$
  // **Expert Processing:**
**for** each expert $i \in I_{\text{exp}}$ **do**
  **if** $f^{(i)}$ is a Transformer expert **then**
    $y^{(i)} \leftarrow f_{\theta_i}^{\mathrm{T}}(Z)$
  **else**
    $y^{(i)} \leftarrow f_{\theta_i}^{\mathrm{RWKV}}(Z)$
  **end if**
**end for**
  // **Expert Coordination:**
  Compute unified output:

$$y_{\text{unified}} \leftarrow \Gamma\Big(\{y^{(i)}\}_{i \in I_{\text{exp}}}\Big)$$

  // **Final Prediction:**
  $\hat{y} \leftarrow \mathrm{softmax}\big(y_{\text{unified}} W^P + b^P\big)$
**Return:** $\hat{y}$

# 85 Theoretical Analysis and Guarantees

### Theorem H.1 (Expressivity of the Mixture-of-Experts Model)

**Statement:** Assume that each expert $f_{\theta_i}^{(i)}$ is a universal approximator over its domain and that the coordinator $\Gamma$ is a non-degenerate aggregation operator. Then, the unified model

$$f_\theta^{\text{Unified}}(X) = \Gamma\Big(\{f_{\theta_i}^{(i)}(X)\}_{i \in I_{\text{exp}}}\Big)$$

is a universal approximator for functions from $(\mathbb{R}^{d_F})^n$ to $\mathcal{Y}$.
**Proof Sketch:** Since each expert can approximate any function to arbitrary accuracy and the coordinator aggregates these approximations in a weighted (or concatenated) manner, standard universal approximation theorems for neural networks imply that the composite function can approximate any target function over a compact domain. □

### Proposition H.2 (Computational Efficiency)

The use of both Transformer and RWKV experts enables balancing of computational complexity. Transformers have a complexity of $O(n^2)$ per layer due to self-attention, while RWKV modules run in $O(n)$ time. The mixture-of-experts framework can dynamically allocate computational resources to experts based on input characteristics, thus optimizing overall efficiency.

# 86 Integration with the Overall Eidos Framework

Module H is the central processing engine of the Eidos framework. It:

- Accepts the final token representations $\{E_{\mathrm{F}}(x_i, \xi)\}$ produced by Module E.

- Processes these representations using multiple expert sub-modules (Transformers and RWKV), each of which may specialize in different aspects of the input.
- Aggregates expert outputs using the coordinator $\Gamma$ to produce a unified prediction.
- Provides an interface for subsequent modules (e.g., memory, training, decoding) to operate on the model's output.

# 87 Implementation Considerations

- **Expert Specialization:** The set of experts $\{f_{\theta_i}^{(i)}\}$ may be pre-assigned or dynamically adjusted based on input characteristics or training objectives.
- **Coordinator Design:** The aggregation function $\Gamma$ can be implemented as a weighted sum, a gating mechanism, or even a small neural network that learns how to fuse expert outputs.
- **Parallelization:** Experts can be computed in parallel, leveraging modern hardware accelerators (GPUs/TPUs) to reduce latency.
- **Dynamic Expert Management:** Mechanisms for adding, removing, or re-weighting experts should be incorporated to allow for scalability and adaptivity.
- **Training Strategy:** Joint or staged training of experts and the coordinator may be used, with careful tuning of learning rates and regularization to avoid overfitting any single expert.

# 88 Conclusion

In this module, we have defined a comprehensive core model architecture that integrates Transformer and RWKV sub-modules in a mixture-of-experts style. Key contributions include:

- A formal definition of both Transformer and RWKV sub-modules with their internal operations.
- The formulation of a mixture-of-experts framework wherein multiple expert modules operate in parallel.
- The introduction of an expert coordinator $\Gamma$ that aggregates individual expert outputs into a unified prediction.
- Theoretical guarantees regarding the expressivity and computational efficiency of the unified model.
- Detailed algorithmic pseudocode outlining the forward pass and integration of expert modules.

This module is the primary processing engine of the Eidos framework, providing both high-level semantic abstraction and computational efficiency while enabling flexible, dynamic adaptation through expert specialization.

# 89 Module I: Titans Memory Architecture (Multi-Layer Memory Module)

Module I: Titans Memory Architecture (Multi-Layer Memory Module)
Part of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence —

## 90 Abstract

This module rigorously defines the *Titans Memory Architecture*, a multi-layer memory system designed for test–time adaptation and continual learning within the Eidos framework. The architecture comprises multiple layers of memory including short-term, working, long-term, and personal memory. A memory bank stores key–value pairs, and a similarity-based retrieval mechanism aggregates relevant memory content via attention. A meta–learner then uses this aggregated memory read to produce adaptive parameter updates for the model. The design supports idempotent, reversible, and efficient retrieval and updating under heterogeneous hardware constraints. We present formal definitions, algorithmic descriptions, theoretical guarantees, and integration considerations with the highest academic rigor.

## 91 Introduction and Motivation

Robust memory mechanisms are essential for adaptive systems, particularly in contexts where long-term dependencies, contextual adjustments, and continual learning are required. The *Titans Memory Architecture* is engineered to support multiple layers of memory:

- **Short-Term Memory:** Captures transient, context-dependent information for immediate tasks.
- **Working Memory:** Maintains intermediate representations relevant to the current task or sequence.
- **Long-Term Memory:** Stores persistent, significant information acquired over extended periods.
- **Personal Memory:** Reflects adaptive, individualized knowledge updated continuously during deployment.

This module enables the system to retrieve and integrate memory efficiently at test time and to update model parameters dynamically based on retrieved information. The design emphasizes modularity, scalability, and hardware-agnostic deployment, ensuring that even models with extensive memory components can be efficiently managed.

## 92 Preliminaries and Notation

We define the following notation to be used throughout this module:

- Let $r \in \mathbb{R}^{d_L}$ denote a latent representation of an input, computed by an encoder $g_\theta$ from a deep model.
- The memory bank is denoted by

$$\mathcal{M} = \{(k_i, v_i) \mid i = 1, \ldots, M_{\mathcal{M}}\},$$

  where:

  - $k_i \in \mathbb{R}^{d_k}$ is the key corresponding to a stored memory element,
  - $v_i \in \mathbb{R}^{d_v}$ is the associated value (which may encode feature corrections, gradient information, or auxiliary signals).

- A similarity function is defined as:

$$s : \mathbb{R}^{d_L} \times \mathbb{R}^{d_k} \to \mathbb{R},$$

  for instance, using cosine similarity:

$$s(r, k_i) = \frac{\langle W_s r,\, k_i \rangle}{\|W_s r\| \|k_i\|},$$

  where $W_s \in \mathbb{R}^{d_L \times d_k}$ is a learnable projection matrix.

- A temperature parameter $\tau > 0$ is used to scale similarity scores.

- Attention weights over the memory are computed as:

$$\alpha_i(x) = \frac{\exp\big(s(r, k_i)/\tau\big)}{\sum_{j=1}^{M_{\mathcal{M}}} \exp\big(s(r, k_j)/\tau\big)}.$$

- The aggregated memory read is defined as:

$$m(x) = \sum_{i=1}^{M_{\mathcal{M}}} \alpha_i(x)\, v_i \in \mathbb{R}^{d_v}.$$

- A meta–learner is defined as:

$$h : \mathbb{R}^{d_v} \to \mathbb{R}^p,$$

  which maps the memory read $m(x)$ to an update vector $\Delta\theta(x)$ for the model parameters.

- The adapted parameters are given by:

$$\theta_x = \theta + \Delta\theta(x),$$

  where $\theta$ are the base model parameters.

- The architecture is *multi-layered*, partitioning $\mathcal{M}$ into sub-banks:

$$\mathcal{M} = \mathcal{M}_{\text{short}} \cup \mathcal{M}_{\text{working}} \cup \mathcal{M}_{\text{long}} \cup \mathcal{M}_{\text{personal}},$$

  each of which may be processed with different weights or retrieval strategies.

# 93 Formal Definitions and Mathematical Formulation

### Definition I.1 (Memory Bank)

The memory bank is defined as:

$$\mathcal{M} = \{(k_i, v_i) \mid i = 1, \ldots, M_{\mathcal{M}}\},$$

where for each $i$:

- $k_i \in \mathbb{R}^{d_k}$ is the key vector associated with a particular memory unit.
- $v_i \in \mathbb{R}^{d_v}$ is the corresponding value vector, which encodes information such as contextual corrections or learned feature adjustments.

### Definition I.2 (Memory Retrieval Mechanism)

Given a latent representation $r \in \mathbb{R}^{d_L}$ derived from an input $x$ by an encoder $g_\theta$, the similarity between $r$ and each memory key $k_i$ is computed as:

$$s(r, k_i) = \frac{\langle W_s r,\, k_i \rangle}{\|W_s r\|\, \|k_i\|},$$

where $W_s \in \mathbb{R}^{d_L \times d_k}$ is a learnable projection. The attention weight for each memory slot is then:

$$\alpha_i(x) = \frac{\exp\big(s(r, k_i)/\tau\big)}{\sum_{j=1}^{M_{\mathcal{M}}} \exp\big(s(r, k_j)/\tau\big)}.$$

The aggregated memory read is defined by:

$$m(x) = \sum_{i=1}^{M_{\mathcal{M}}} \alpha_i(x)\, v_i.$$

### Definition I.3 (Meta–Learner for Test-Time Adaptation)

The meta–learner is a function:

$$h : \mathbb{R}^{d_v} \to \mathbb{R}^p,$$

which computes a parameter update:

$$\Delta\theta(x) = h\big(m(x)\big).$$

The model parameters are adapted at test time via:

$$\theta_x = \theta + \Delta\theta(x),$$

where $\theta \in \Theta$ are the base parameters.

### Definition I.4 (Multi-Layer Memory Structure)

We partition the memory bank into hierarchical layers:

$$\mathcal{M} = \mathcal{M}_{\text{short}} \cup \mathcal{M}_{\text{working}} \cup \mathcal{M}_{\text{long}} \cup \mathcal{M}_{\text{personal}},$$

where:

- $\mathcal{M}_{\text{short}}$ stores transient, context-specific information.
- $\mathcal{M}_{\text{working}}$ maintains task-related intermediate representations.
- $\mathcal{M}_{\text{long}}$ holds persistent information acquired over extended periods.
- $\mathcal{M}_{\text{personal}}$ captures adaptive, user- or domain-specific knowledge.

Each sub-bank can be processed using a specialized similarity function $s_\ell$ and may be aggregated via weighted summation:

$$m(x) = \sum_{\ell \in \{\text{short, working, long, personal}\}} w_\ell\, m_\ell(x),$$

where:

$$m_\ell(x) = \sum_{i \in I_\ell} \alpha_i^{(\ell)}(x)\, v_i^{(\ell)},$$

with $I_\ell$ indexing the memory units in layer $\ell$, and $w_\ell$ are weights (learned or pre-defined) governing the contribution of each layer.

# 94 Algorithmic Description

The following pseudocode details the operation of the Titans Memory Architecture, including memory retrieval, aggregation, and test–time parameter adaptation.

**Algorithm 11** Titans Memory Architecture: Memory Retrieval and Adaptation

**Input:** Test input $x \in \mathcal{X}$; encoder $g_\theta$; memory banks $\{\mathcal{M}_\ell\}_{\ell \in \{\text{short, working, long, personal}\}}$; temperature $\tau$; meta–learner $h$

**Output:** Adapted parameters $\theta_x$ or direct prediction update

**Compute:** Latent representation $r \leftarrow g_\theta(x) \in \mathbb{R}^{d_L}$

**for** each memory layer $\ell$ in {short, working, long, personal} **do**

    **for** each memory unit $(k_i^{(\ell)}, v_i^{(\ell)}) \in \mathcal{M}_\ell$ **do**

        Compute similarity: $s_i^{(\ell)} \leftarrow s_\ell(r, k_i^{(\ell)})$

    **end for**

    Compute attention weights:

$$\alpha_i^{(\ell)} \leftarrow \frac{\exp\bigl(s_i^{(\ell)}/\tau\bigr)}{\sum_{j \in I_\ell} \exp\bigl(s_j^{(\ell)}/\tau\bigr)}$$

    Aggregate memory read for layer $\ell$:

$$m_\ell(x) \leftarrow \sum_{i \in I_\ell} \alpha_i^{(\ell)}\, v_i^{(\ell)}$$

**end for**

**Combine Layers:**

$$m(x) \leftarrow \sum_\ell w_\ell\, m_\ell(x)$$

**Meta–Learner Update:**

$$\Delta\theta(x) \leftarrow h\bigl(m(x)\bigr)$$

Update parameters:

$$\theta_x \leftarrow \theta + \Delta\theta(x)$$

**Return:** $\theta_x$ (or use $\theta_x$ to compute prediction $\hat{y} = f_{\theta_x}(x)$)

# 95 Theoretical Analysis and Guarantees

### Theorem I.1 (Convergence and Stability of Memory Read)

**Statement:** Assume that for each layer $\ell$, the similarity function $s_\ell$ is bounded and the attention weights $\alpha_i^{(\ell)}$ form a probability distribution. Then, for any input $x$, the aggregated memory read

$$m(x) = \sum_\ell w_\ell \left( \sum_{i \in I_\ell} \alpha_i^{(\ell)}\, v_i^{(\ell)} \right)$$

is a well-defined, bounded vector. Furthermore, if the memory banks are updated in a controlled manner, then the meta–learner update $\Delta\theta(x) = h(m(x))$ converges, and repeated adaptation leads to a stable parameter set $\theta_x$.

**Proof Sketch:** Since $\alpha_i^{(\ell)} \geq 0$ and $\sum_{i \in I_\ell} \alpha_i^{(\ell)} = 1$, each $m_\ell(x)$ is a convex combination of bounded vectors $v_i^{(\ell)}$. Hence, $m(x)$ is bounded. Under standard assumptions on the contraction properties of $h$ (e.g., Lipschitz continuity with a constant less than one), iterative updates will converge to a fixed point. □

### Proposition I.2 (Scalability)

The hierarchical partitioning of $\mathcal{M}$ into layers enables the system to scale with the overall amount of memory. Since each layer is managed separately and combined via weighted summation, the in-memory retrieval operations remain efficient even as the total number of memory units grows.

## 96 Integration with the Overall Eidos Framework

Module I, the Titans Memory Architecture, is a critical component of the Eidos system. It:

- Provides a multi-layer memory system that supports test–time adaptation and continual learning.
- Interfaces with the Core Model Architectures (Module H) by supplying adaptive updates through the meta–learner.
- Receives latent representations from upstream modules (e.g., contextual embeddings from Module E) and aggregates memory information to influence model parameters.
- Supports real-time updates and retrieval, ensuring that the system remains adaptive to new data and domain shifts.

## 97 Implementation Considerations

- **Efficient Storage:** Memory banks should be implemented using data structures optimized for sparse retrieval and vector operations (e.g., approximate nearest neighbor search structures).
- **Parallel Retrieval:** Similarity computations and attention weight calculations can be parallelized across memory layers.
- **Dynamic Weighting:** The weights $w_\ell$ for combining memory layers may be learned or set based on domain knowledge to reflect the relative importance of each memory type.
- **Meta–Learner Design:** The function $h$ should be designed to produce small, stable updates to the model parameters and may itself be a shallow neural network.
- **Resource Constraints:** Considerations for computational resources (e.g., GPU memory) should guide the number of memory units retained in active memory.

## 98 Conclusion

In this module, we have rigorously defined the Titans Memory Architecture, a multi-layer memory system essential for test–time adaptation and continual learning in the Eidos framework. The module:

- Introduces a memory bank $\mathcal{M}$ partitioned into hierarchical layers (short-term, working, long-term, and personal).
- Defines a similarity-based retrieval mechanism that computes attention weights $\alpha_i(x)$ over memory units.
- Aggregates memory reads from different layers via weighted summation.
- Utilizes a meta–learner $h$ to convert the aggregated memory read into adaptive parameter updates.
- Provides theoretical guarantees regarding the boundedness, convergence, and scalability of the memory retrieval process.

This architecture enables the overall system to adapt dynamically to new data and contextual changes, thereby enhancing model performance and robustness in diverse environments.

# 99 Module J: Recursive Adaptive Dynamic Idempotent Feedback and State-Based Runtime Learning and Inference

Module J: Recursive Adaptive Dynamic Idempotent Feedback and State-Based Runtime Learning and Inference
Part of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence —

## 100 Abstract

This module rigorously defines the *Recursive Adaptive Dynamic Idempotent Feedback and State-Based Runtime Learning and Inference* component of the Eidos framework. It provides a formal system for modeling interdependent feedback dynamics among multiple entities, capturing how triggers and actions propagate influence recursively. Key aspects include the definition of entities, triggers, actions, influence functions, and feedback loops; the introduction of adaptive modulation functions that update system states based on external and internal signals; and the enforcement of idempotence and reversibility to guarantee stability. The framework is developed with full mathematical rigor and is designed to be modular, extensible, and dynamically adaptive, enabling continuous runtime learning and inference.

## 101 Introduction and Motivation

In complex intelligent systems, continuous adaptation and learning require mechanisms that can recursively integrate feedback from multiple sources. The *Recursive Adaptive Dynamic Idempotent Feedback* system in the Eidos framework addresses this need by modeling the interactions among various entities through triggers and actions. These interactions are recursively composed into feedback loops that update the system state in an adaptive yet stable manner. Key motivations for this module include:

(a) Modeling the bidirectional and cyclic interactions among entities.

(b) Ensuring that repeated updates via feedback are idempotent, thereby guaranteeing stability.

(c) Allowing dynamic adaptation at runtime through adaptive modulation of entity states.

(d) Supporting a modular design where each entity's feedback can be updated independently without disrupting the global system state.

This module provides the formal and algorithmic foundation for runtime learning and inference based on recursive feedback.

## 102 Preliminaries and Notation

We introduce the following notation and definitions:

- Let $\mathcal{E} = \{E_i \mid i \in I\}$ denote the set of entities in the system, where $I$ is an index set (finite or countable).
- For each entity $E_i$:
  - $\tau(E_i) \in \mathcal{T}$ represents the *trigger* function or signal.
  - $\alpha(E_i) \in \mathcal{A}$ represents the *action* executed by the entity.
- The *influence function* is denoted by:

$$\delta(E_i, E_j): \quad \tau(E_i) \longrightarrow \alpha(E_j),$$

  which quantifies how the trigger of $E_i$ affects the action of $E_j$.

- The *feedback function* is defined as:

$$\phi(E_i, E_j) = \delta(E_i, E_j) \circ \delta(E_j, E_i),$$

where $\circ$ denotes functional composition, capturing the bidirectional recursive interaction.

- The *system state* for an entity $E_i$ is denoted by:

$$\Sigma(E_i) = \{\tau(E_i),\, \alpha(E_i),\, \{\delta(E_i, E_j)\}_{j \neq i}\},$$

and for a set of entities relevant to a context $a \in \mathcal{C}$, the global state is:

$$\Sigma_a = \bigoplus_{(E_i, E_j) \in \mathcal{E}_a^2,\, i \neq j} \left[\phi(E_i, E_j) \oplus \phi(E_j, E_i)\right],$$

where $\oplus$ denotes a modular composition operator (assumed to be associative and commutative).

- An *adaptive modulation function* is defined as:

$$\mu_a : \Sigma_0 \to \Sigma_a,$$

which adapts a base state $\Sigma_0$ to a given context $a$. This function is required to be idempotent:

$$\mu_a\big(\mu_a(\Sigma_0)\big) = \mu_a(\Sigma_0).$$

- The system may incorporate adaptive parameters from a set $\mathcal{P}$, with the learned modulation for context $a$ represented as:

$$\mathcal{P}_a \in \mathcal{P}.$$

# 103 Formal Definitions and Mathematical Formulation

### Definition J.1 (Entities, Triggers, and Actions)

For each entity $E_i \in \mathcal{E}$:

$$\tau(E_i) \in \mathcal{T}, \quad \alpha(E_i) \in \mathcal{A}.$$

These functions are assumed to be defined on appropriate domains and codomains so that they can be composed with influence functions.

### Definition J.2 (Influence Function)

For any two distinct entities $E_i, E_j \in \mathcal{E}$, the influence function is defined as:

$$\delta(E_i, E_j) : \quad \tau(E_i) \longrightarrow \alpha(E_j).$$

This function quantifies the manner in which the trigger signal of $E_i$ induces or modulates the action of $E_j$.

### Definition J.3 (Feedback Function)

The bidirectional feedback between entities $E_i$ and $E_j$ is defined as:

$$\phi(E_i, E_j) = \delta(E_i, E_j) \circ \delta(E_j, E_i).$$

This composition represents the recursive interplay between $E_i$ and $E_j$, where the output of one influence serves as the input to the other. A non-trivial feedback loop satisfies:

$$\phi(E_i, E_j) \neq 0,$$

ensuring meaningful interaction.

### Definition J.4 (System State)

For a given context $a \in \mathcal{C}$, define the pairwise state contribution of entities $E_i$ and $E_j$ as:

$$\Sigma_a(E_i, E_j) = \phi(E_i, E_j) \oplus \phi(E_j, E_i).$$

The overall system state under context $a$ is then:

$$\Sigma_a = \bigoplus_{(E_i, E_j) \in \mathcal{E}_a^2,\, i \neq j} \Sigma_a(E_i, E_j).$$

### Definition J.5 (Adaptive Modulation Function)

Define a modulation function that adapts the base state $\Sigma_0$ to a context $a$:

$$\mu_a : \Sigma_0 \to \Sigma_a,$$

with the idempotence property:

$$\mu_a\big(\mu_a(\Sigma_0)\big) = \mu_a(\Sigma_0).$$

When the modulation function is learned or trained, it encapsulates a set of adaptive parameters:

$$\mathcal{P}_a \in \mathcal{P},$$

which are then applied to update the system state.

## 104 Algorithmic Description

The following pseudocode outlines the operation of the recursive feedback system:

**Algorithm 12** Recursive Adaptive Feedback and Runtime Learning

**Input:** Set of entities $\mathcal{E} = \{E_i \mid i \in I\}$; initial system state $\Sigma_0$; context $a \in \mathcal{C}$; modulation function $\mu_a$

**Output:** Adapted system state $\Sigma_a$

**Begin:**

**for** each pair of distinct entities $(E_i, E_j)$ **do**

Compute influence: $\delta(E_i, E_j)$ from $\tau(E_i)$ to $\alpha(E_j)$

Compute reverse influence: $\delta(E_j, E_i)$ from $\tau(E_j)$ to $\alpha(E_i)$

Compute feedback:

$$\phi(E_i, E_j) \leftarrow \delta(E_i, E_j) \circ \delta(E_j, E_i)$$

Set pairwise state:

$$\Sigma_a(E_i, E_j) \leftarrow \phi(E_i, E_j) \oplus \phi(E_j, E_i)$$

**end for**

Compose global state:

$$\Sigma_a \leftarrow \bigoplus_{(E_i,E_j)\in\mathcal{E}_a^2,\, i\neq j} \Sigma_a(E_i, E_j)$$

Apply modulation:

$$\Sigma_a \leftarrow \mu_a(\Sigma_0)$$

**Return:** $\Sigma_a$

**Optional Iterative Adaptation:** For continual runtime learning, the above procedure can be applied iteratively. Let:

$$\Sigma^{(0)} = \Sigma_0, \quad \Sigma^{(t+1)} = \mu_a\Big(\Sigma^{(t)}\Big),$$

with convergence guaranteed by the idempotence property:

$$\lim_{t\to\infty} \Sigma^{(t)} = \Sigma_a.$$

# 105 Theoretical Analysis and Guarantees

### Theorem J.1 (Idempotence and Stability)

**Statement:** For any context $a \in \mathcal{C}$, the modulation function $\mu_a$ is idempotent, i.e.,

$$\mu_a\big(\mu_a(\Sigma_0)\big) = \mu_a(\Sigma_0) = \Sigma_a.$$

Thus, repeated application of the feedback update yields a stable system state.

**Proof Sketch:** By the definition of $\mu_a$ (Definition J.5), once the base state $\Sigma_0$ is adapted to $\Sigma_a$, further applications do not alter $\Sigma_a$. This follows from the property that the learned adaptive parameters $\mathcal{P}_a$ yield an idempotent mapping. □

### Proposition J.2 (Modularity and Scalability)

The feedback system is constructed via the modular composition operator $\oplus$, which is assumed to be associative and commutative. Consequently, the global state

$$\Sigma_a = \bigoplus_{(E_i,E_j)\in\mathcal{E}_a^2,\, i\neq j} \Sigma_a(E_i, E_j)$$

is independent of the order of composition. This property ensures that the system can scale to an arbitrary number of entities and can be extended to higher-order interactions without loss of consistency.

## 106 Integration with the Overall Eidos Framework

Module J is a core component responsible for enabling adaptive, runtime learning and inference. It:

- Interfaces with the deep model architectures (Module H) by providing a mechanism for continuous state adaptation based on internal feedback.
- Interacts with the memory module (Module I) by incorporating external, real-time signals into the feedback loops.
- Connects with the training system (Module K) to allow online parameter updates and continual learning.
- Provides a recursive framework that ensures that system states remain stable (via idempotence) while adapting dynamically to new inputs and contextual changes.

## 107 Implementation Considerations

- **Feedback Computation:** The functions $\delta(E_i, E_j)$ should be implemented using well-established techniques (e.g., weighted mappings or neural network modules) with careful calibration to avoid vanishing or exploding feedback signals.
- **Adaptive Modulation:** The modulation function $\mu_a$ may be implemented as a gating network or a parameterized transformation that is trained jointly with the rest of the model.
- **Idempotence Verification:** Empirical tests should be designed to verify that repeated applications of $\mu_a$ yield negligible changes beyond a certain iteration, ensuring convergence.
- **Scalability:** The composition operator $\oplus$ should be chosen to support efficient aggregation over large sets of entities. Sparse representations and parallel computation may be used.
- **Integration with Feedback Sources:** The system should accommodate both internally generated signals (e.g., prediction errors) and external signals (e.g., user feedback) within the recursive feedback loops.

## 108 Conclusion

In this module, we have developed a comprehensive framework for recursive, adaptive, and idempotent feedback that underpins runtime learning and inference in the Eidos framework. The key contributions are:

- A formal definition of entities, triggers, actions, and influence functions that model interdependent interactions.
- The definition of a recursive feedback function $\phi(E_i, E_j)$ that composes bidirectional influences.

- The construction of a global system state $\Sigma_a$ via modular composition, capturing the cumulative effects of feedback across entities.
- The introduction of an adaptive modulation function $\mu_a$ with an idempotence property, ensuring that the system state converges and remains stable under repeated updates.
- The presentation of algorithmic procedures for computing feedback and adapting system parameters in real time.

This robust and modular recursive feedback mechanism is essential for enabling dynamic adaptation, continual learning, and runtime inference in complex environments, forming a critical component of the overall Eidos framework.

# 109 Module K: Universal Training System

Module K: Universal Training System
Part of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence —

## 110 Abstract

This module rigorously defines the *Universal Training System* for the Eidos framework. It introduces a universally deployable, chunk-based, streaming-enabled training methodology that is hardware-agnostic and scalable. The training system is designed to optimize the entire model by coordinating gradient computations, parameter updates, and regularization across all modules. Key components include a primary loss function with regularization terms, a state-of-the-art optimizer (e.g., AdamW or SGD with momentum), normalization, dropout, and skip connections. Additionally, the system handles parameter chunking and streaming to accommodate models that exceed available RAM. The system is formally specified with detailed mathematical definitions, algorithmic pseudocode, theoretical guarantees, and integration guidelines.

## 111 Introduction and Motivation

Training large-scale deep learning models poses significant challenges, especially when model size exceeds the available in-memory resources. The *Universal Training System* is a core component of the Eidos framework designed to address these challenges. Its objectives are:

(a) **End-to-End Optimization:** Provide a unified training objective that spans all components, from embeddings and knowledge graphs to memory and deep model architectures.

(b) **Chunk-Based and Streaming Training:** Decompose model parameters into minimal modules (or chunks) that can be streamed from disk, thus enabling training on hardware with limited RAM.

(c) **Hardware-Agnostic Deployment:** Ensure that the training process can be executed on a wide range of devices (from low-memory CPUs to high-end GPUs) by minimizing in-memory requirements.

(d) **Robust Optimization Techniques:** Incorporate advanced optimizers (e.g., AdamW, SGD), normalization (e.g., LayerNorm), dropout, and skip connections to stabilize training.

(e) **Sparse Data Handling:** Efficiently process sparse multidimensional data, thereby ensuring that missing or infrequent features do not hinder optimization.

This module details the theoretical and algorithmic underpinnings of this training system, ensuring that it is fully integrated with and supportive of the overall Eidos framework.

## 112 Preliminaries and Notation

- Let $\mathcal{D}$ denote the training dataset, where each data sample is a pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$.
- The deep model is defined as a parameterized function:
$$f_\theta : \mathcal{X} \to \mathcal{Y},$$
with parameters $\theta \in \Theta \subset \mathbb{R}^p$.
- The complete parameter set $\theta$ is decomposed into a collection of minimal modules (or chunks):
$$\theta = \bigcup_{j \in J} T_j,$$
where $J$ is a finite index set and each $T_j$ represents a self-contained subcomponent.

- A *chunk index mapping* is defined by:

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

  mapping each chunk $T_j$ to its storage location $\ell_j$ in persistent storage.

- The current resource context (available memory, GPU capacity, etc.) is denoted by $\mathcal{R}$.
- A loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}$ is defined for individual data samples.
- Regularization terms are introduced with hyperparameters $\lambda_W$ (for weight decay) and $\lambda_{\text{sparse}}$ (for sparsity regularization).

# 113 Formal Definitions and Mathematical Formulation

### Definition K.1 (Training Objective)

The training objective is defined as a function:

$$\mathcal{L} : \Theta \times \mathcal{D} \to \mathbb{R}_{\geq 0},$$

such that for a mini-batch $B \subset \mathcal{D}$,

$$\mathcal{L}(\theta; B) = \frac{1}{|B|} \sum_{(x,y) \in B} \ell\big(f_\theta(x), y\big) + \lambda_W \|\theta\|^2 + \lambda_{\text{sparse}} \mathcal{R}_{\text{sparse}}(\theta).$$

Here, $\mathcal{R}_{\text{sparse}}$ is a regularizer promoting sparsity, and $\lambda_W, \lambda_{\text{sparse}} \geq 0$ are hyperparameters.

### Definition K.2 (Optimizer Function)

An optimizer is defined as a mapping:

$$\mathcal{O} : \Theta \times \nabla_\theta \mathcal{L} \times \Xi \to \Theta,$$

where $\Xi$ represents the optimizer's internal state (e.g., moment estimates in AdamW). For each parameter chunk $T_j$, the update rule is given by:

$$T_j \leftarrow \mathcal{O}\big(T_j,\, \nabla_{T_j} \mathcal{L},\, \Xi_j\big),$$

ensuring that updates are applied in a chunk-wise manner.

### Definition K.3 (Normalization Operator)

A normalization operator $N : \mathbb{R}^d \to \mathbb{R}^d$ is defined as:

$$N(x) = \gamma \odot \frac{x - \mu_x}{\sqrt{\sigma_x^2 + \epsilon}} + \beta,$$

where $\mu_x$ and $\sigma_x^2$ are the mean and variance of $x$ over the relevant dimensions, $\gamma, \beta \in \mathbb{R}^d$ are learned parameters, $\epsilon$ is a small positive constant, and $\odot$ denotes element-wise multiplication.

### Definition K.4 (Dropout Operator)

A dropout operator $D : \mathbb{R}^d \to \mathbb{R}^d$ is defined as:

$$D(x) = x \odot m,$$

where $m \in \{0, 1\}^d$ is a mask drawn from a Bernoulli distribution with parameter $1 - p$ (i.e., $m_i \sim \text{Bernoulli}(1 - p)$).

### Definition K.5 (Skip Connection)

A skip connection is defined by the operation:

$$S(x, F(x)) = x + F(x),$$

where $F(x)$ is a transformation (such as a feed-forward network). This mechanism facilitates gradient flow and network training.

### Definition K.6 (Chunk-Based Streaming Training)

Let the model parameters be decomposed as:

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

A *streaming function* is defined as:

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

which loads a subset $\{T_j\}_{j \in J'}$ of the parameter chunks from persistent storage into fast memory based on the current resource context $\mathcal{R}$. A caching function:

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

manages which chunks remain in memory during training.

## 114 Algorithmic Description

The following pseudocode outlines the universal training loop, which incorporates chunk-based streaming, loss computation, gradient calculation, and parameter updates.

**Algorithm 13** Universal Training Loop

**Input:** Training dataset $\mathcal{D}$, model $f_\theta$ with parameters $\theta = \bigcup_{j \in J} T_j$, initial optimizer state $\Xi$, resource context $\mathcal{R}$
**for** each mini-batch $B \subset \mathcal{D}$ **do**
    **Streaming:** Load required parameter chunks:

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

    **Forward Pass:** Compute model outputs $f_\theta(x)$ for each $(x, y) \in B$
    **Loss Computation:**

$$\mathcal{L}(\theta; B) \leftarrow \frac{1}{|B|} \sum_{(x,y) \in B} \ell\big(f_\theta(x), y\big) + \lambda_W \|\theta\|^2 + \lambda_{\text{sparse}} \mathcal{R}_{\text{sparse}}(\theta)$$

    **Backward Pass:** Compute gradients $\nabla_\theta \mathcal{L}(\theta; B)$ using automatic differentiation, ensuring that sparse gradients are handled appropriately.
    **Parameter Update:** For each chunk $T_j$ in the active set,

$$T_j \leftarrow \mathcal{O}\big(T_j,\, \nabla_{T_j} \mathcal{L},\, \Xi_j\big)$$

    **Normalization, Dropout, and Skip:** Apply $N$, $D$, and $S$ within the forward pass as specified by the model architecture.
    **Streaming Out:** Write updated chunks back to persistent storage, updating the index $\mathcal{I}$ as necessary.
    **Logging:** Record training metrics and update logs for monitoring and debugging.
**end for**
**Return:** Final trained parameters $\theta$

# 115 Theoretical Analysis and Guarantees

### Theorem K.1 (Convergence and Stability Under Streaming Updates)

**Statement:** Assume that:

(i) The loss function $\mathcal{L}(\theta; B)$ is bounded below and continuously differentiable.

(ii) The optimizer $\mathcal{O}$ (e.g., AdamW) satisfies standard convergence properties under fixed memory conditions.

(iii) The chunk-based streaming mechanism $\sigma$ ensures that every parameter update is applied in an idempotent and consistent manner.

Then, under appropriate conditions on learning rates and regularization parameters, the sequence of parameter updates $\{\theta^{(t)}\}$ converges (or has a convergent subsequence) to a stationary point of $\mathcal{L}$.

**Proof Sketch:** Standard convergence proofs for stochastic gradient descent (SGD) and its variants apply, provided that the updates computed on each chunk are consistent with the full gradient. The streaming mechanism, by ensuring idempotence and reversible updates, does not introduce additional error beyond standard stochastic noise. □

**Proposition K.2 (Hardware-Agnostic Scalability)**

Because model parameters are partitioned into chunks and streamed on demand, the in-memory resource requirement is limited to the number of active chunks. Thus, even for models with extremely large $|\theta|$, the training system scales with available resources $\mathcal{R}$ provided that persistent storage $\mathcal{S}_{\text{disk}}$ is sufficient.

## 116 Integration with the Overall Eidos Framework

Module K is essential for training the entire Eidos framework in a universal, scalable, and hardware-agnostic manner. It:

- Coordinates the end-to-end optimization of all components, from input processing (Module A) through deep architectures (Modules H, I, J) to final decoding (Module L).
- Implements chunk-based streaming, enabling large models to be trained on devices with limited RAM.
- Provides standardized interfaces for applying normalization, dropout, and skip connections across the system.
- Ensures that updates are consistent, reversible, and idempotent, facilitating both efficient training and robust runtime adaptation.

## 117 Implementation Considerations

- **Asynchronous Streaming:** Implement the streaming function $\sigma$ with asynchronous I/O to prefetch chunks and minimize latency.
- **Cache Management:** The caching function $\mu$ should employ advanced strategies (e.g., least-recently-used or predictive caching) to optimize memory usage.
- **Sparse Gradient Handling:** Use specialized libraries or techniques to efficiently compute and apply updates when gradients are sparse.
- **Optimizer Selection:** The choice of optimizer (e.g., AdamW) must balance convergence speed with stability, and hyperparameters should be tuned accordingly.
- **Distributed Training:** Consider extending the training system to a distributed setting, where parameter chunks are streamed and updated across multiple nodes.
- **Logging and Monitoring:** Comprehensive logging of training metrics, parameter updates, and streaming operations is essential for debugging and empirical validation.

## 118 Conclusion

In this module, we have rigorously defined a *Universal Training System* that enables the end-to-end optimization of the Eidos framework. The system is characterized by:

- A comprehensive training objective that includes the primary loss, weight decay, and sparsity regularization.

- A state-of-the-art optimizer interface that updates model parameters in a chunk-based, streaming-enabled fashion.
- Standardized normalization, dropout, and skip connection operators that ensure training stability.
- A modular, hardware-agnostic streaming and caching mechanism that allows models of arbitrary size to be trained efficiently.

The training system is designed to be scalable, robust, and fully integrated with the overall Eidos framework, ensuring that the model can be trained reliably across diverse hardware configurations and under various resource constraints.

## 119 Module L: Final Decoding and Multimodal Output

Module L: Final Decoding and Multimodal Output —

## 120 Abstract

This module rigorously formalizes the *Final Decoding and Multimodal Output* component of the Eidos Unified Framework, which is pivotal in translating the latent representations produced by the deep model into outputs that are both human-interpretable and tailored to specific application domains (e.g., natural language text, images, audio). We introduce a comprehensive decoding function that maps the latent space to a designated modality space, alongside a modality decision mechanism that dynamically determines the most appropriate output format based on task-specific contexts. Detailed mathematical formulations, algorithmic procedures, and robust theoretical guarantees are provided to ensure that the decoded outputs are consistent, context-aware, and seamlessly extensible to any new modalities.

## 121 Introduction and Motivation

In modern multimodal intelligence systems, transforming high-dimensional latent representations into practical outputs remains a critical challenge. The *Final Decoding and Multimodal Output* module addresses this by:

(a) Converting latent representations from the deep model into a primary output stream (predominantly natural language text by default).

(b) Determining the necessity for supplementary output modalities (such as images, audio, or graphs) based on explicit task requirements and contextual parameters.

(c) Encapsulating all output data within a standardized packaging format for subsequent processing, visualization, or integration with external systems.

This module is essential as it bridges the conceptual gap between the abstract internal representations of the model and the concrete outputs required by users and downstream applications. Its design leverages both deterministic decoding strategies and flexible modality decision functions, offering a robust and extensible interface across a wide spectrum of deployment scenarios.

## 122 Preliminaries and Notation

Assuming that previous modules have computed a latent representation $y_{\text{latent}} \in \mathcal{Y}$ via the deep model $f_\theta$, we adopt the following notation:

- $y_{\text{latent}} \in \mathcal{Y}$: Latent output vector from $f_\theta$.
- $\mathcal{O}_{\text{mod}}$: The output modality space (e.g., Text, Image, Audio).
- Decoding function:
$$\delta : \mathcal{Y} \to \mathcal{O}_{\text{mod}},$$
mapping the latent representation to its primary output modality (by default, text).
- Modality decision function:
$$\mu_{\text{mod}} : \mathcal{Y} \times \mathcal{C}_{\text{task}} \to \mathcal{O}_{\text{mod}},$$
where $\mathcal{C}_{\text{task}}$ represents task-specific context, used to determine whether additional modalities should be produced.

- Output packaging: The final output is structured in a universal data packet,

$$\mathcal{P}_{\text{out}} = \big(\text{ID}_{\text{out}},\, \hat{y},\, \text{Meta}_{\text{out}}\big),$$

  where $\hat{y}$ denotes the decoded output.

# 123 Formal Definitions and Mathematical Formulation

### Definition L.1 (Decoding Function)

Let $y_{\text{latent}} \in \mathcal{Y}$ be the latent output vector obtained from the model. The *decoding function* is defined as:

$$\delta : \mathcal{Y} \to \mathcal{O}_{\text{mod}},$$

generating the primary output. For instance, in a text-based system:

$$\hat{y}_{\text{text}} = \delta(y_{\text{latent}}) \in \text{Text}.$$

The implementation of $\delta$ typically entails:

- A linear transformation projecting $\mathcal{Y}$ onto $\mathbb{R}^{|\mathcal{V}|}$.
- A softmax activation producing a probability distribution over vocabulary tokens.
- A deterministic decoding strategy (e.g., beam search or greedy search) to yield the final output sequence.

### Definition L.2 (Modality Decision Function)

The *modality decision function* is formalized as:

$$\mu_{\text{mod}} : \mathcal{Y} \times \mathcal{C}_{\text{task}} \to \mathcal{O}_{\text{mod}},$$

which, based on the latent output $y_{\text{latent}}$ and specific task context $\mathcal{C}_{\text{task}}$, designates the appropriate output modality. In the base configuration, we set:

$$\mu_{\text{mod}}(y_{\text{latent}}, \mathcal{C}_{\text{task}}) = \text{Text}.$$

This function is purposefully designed to be extensible, so other modalities may be integrated without disturbing the core architecture.

### Definition L.3 (Output Packaging)

The final output is encapsulated in a *universal output packet* defined as:

$$\mathcal{P}_{\text{out}} = \big(\text{ID}_{\text{out}},\, \hat{y},\, \text{Meta}_{\text{out}}\big),$$

where:

- $\text{ID}_{\text{out}}$ is a unique identifier for the output.
- $\hat{y} = \delta(y_{\text{latent}})$ is the primary decoded output.
- $\text{Meta}_{\text{out}}$ comprises metadata (e.g., timestamps, processing context, and modality details) integral for traceability and post-processing.

## 124 Algorithmic Description

The following pseudocode details the process for executing the final decoding and generating multimodal outputs:

**Algorithm 14** Final Decoding and Multimodal Output Process

**Input:** Latent output $y_{\text{latent}} \in \mathcal{Y}$; Task context $\mathcal{C}_{\text{task}}$
**Output:** Universal output packet $\mathcal{P}_{\text{out}}$
**Begin:**
    Compute primary decoding:

$$\hat{y}_{\text{text}} \leftarrow \delta\big(y_{\text{latent}}\big)$$

    Determine output modality:

$$M \leftarrow \mu_{\text{mod}}\big(y_{\text{latent}}, \mathcal{C}_{\text{task}}\big)$$

**if** $M \neq$ Text **then**
        Compute additional modality outputs using appropriate decoders.
**else**
        Set additional outputs to null.
**end if**
    Package final output:

$$\mathcal{P}_{\text{out}} \leftarrow \big(\text{ID}_{\text{out}},\, \hat{y}_{\text{text}},\, \text{Meta}_{\text{out}}\big)$$

    **Return:** $\mathcal{P}_{\text{out}}$

## 125 Theoretical Analysis and Guarantees

### Theorem L.1 (Decoding Consistency)

**Statement:** Suppose the decoding function $\delta$ is realized as a linear projection followed by softmax activation and a deterministic decoding strategy. Then for any fixed $y_{\text{latent}}$, the output $\hat{y}_{\text{text}} = \delta(y_{\text{latent}})$ is uniquely determined and reproducible, thereby ensuring output consistency across multiple evaluations.
**Proof Sketch:** Since the sequence of operations—linear projection, softmax normalization, and a deterministic decoding (e.g., beam search)—is invariant under fixed model parameters and input data, the mapping $\delta$ yields a unique output. □

### Proposition L.2 (Extensibility of Multimodal Output)

The modality decision function $\mu_{\text{mod}}$ is architected for extensibility. For any supplementary modality $M' \in \mathcal{O}_{\text{mod}}$ (such as image or audio), an additional decoding function $\delta_{M'} : \mathcal{Y} \to M'$ can be integrated. Thus, the final output system can be expanded to accommodate new modalities without necessitating fundamental alterations to its core design.

## 126 Integration with the Overall Eidos Framework

Module L seamlessly completes the Eidos pipeline by:

- Converting latent model outputs into formats that are intuitively interpretable and aligned with application-specific requirements.

- Dynamically determining and generating outputs in one or multiple modalities as dictated by the operational context.
- Packaging the final outputs into a structured, universal data packet, facilitating subsequent tasks such as logging, interface rendering, and feedback integration.
- Offering a standardized and rigorously defined interface that guarantees consistency, traceability, and ease of extensibility.

## 127 Implementation Considerations

- **Decoding Strategies:** The function $\delta$ may employ diverse decoding methods (beam search, greedy decoding, or sampling) tailored to the specific application, balancing performance and latency.
- **Modality-Specific Decoders:** For non-text outputs, specialized decoders (such as convolutional decoders for image generation) must be integrated into the framework in tandem with $\mu_{\text{mod}}$.
- **Latency and Efficiency:** Critical optimizations should be implemented to minimize decoding latency, particularly in scenarios requiring real-time responses.
- **Output Packaging:** The universal output packet $\mathcal{P}_{\text{out}}$ should comprehensively incorporate metadata, ensuring robust traceability and facilitating debugging and post-processing.
- **Extensibility:** The module's design must remain future-proof, allowing for the seamless integration of emerging modalities without extensive re-engineering.

## 128 Conclusion

In summary, this module establishes a rigorous and adaptable framework for final decoding and multimodal output within the Eidos system. It rigorously defines:

- A deterministic decoding function $\delta$ that effectively translates latent representations into a primary (default text) output.
- A modality decision function $\mu_{\text{mod}}$ that assesses task-specific requirements to select the proper output modality.
- A universal output packaging strategy that encapsulates the final prediction along with essential metadata.
- Comprehensive theoretical guarantees to ensure output consistency and scalability for multimodal extensions.

This module, as the final link within the Eidos pipeline, assures that complex, high-dimensional outputs are rendered into coherent, interpretable, and actionable results for a diverse array of applications.

## 129 Final Wrap-Up of the Eidos Unified Framework

Final Wrap-Up of the Eidos Unified Framework
Persistent, Dynamic, and Adaptive Multimodal Intelligence —

# 130 Abstract

In this final wrap-up, we provide a comprehensive review and critical analysis of the Eidos Unified Framework for Persistent, Dynamic, and Adaptive Multimodal Intelligence. We consolidate the contributions of its 12 meticulously defined modules, elaborate on the rigorous theoretical and algorithmic underpinnings of the system, and examine the integration methodologies that achieve streamlined, end-to-end processing—from raw data acquisition to final output decoding. Furthermore, we critically evaluate the framework's strengths, limitations, and potential areas for future enhancement. This document represents the capstone of a detailed, modular, and academically robust blueprint aimed at advancing next-generation adaptive intelligence systems.

# 131 Introduction

The Eidos framework epitomizes a holistic, modular strategy for constructing sophisticated multimodal intelligence systems. Its architecture is rooted in several foundational principles:

- A resilient input processing pipeline that standardizes and normalizes raw data.
- A universal communication and data handling interface that secures reliable, inter-module connectivity.
- A scalable streaming and indexing mechanism that supports hardware-agnostic deployment.
- A multidimensional vocabulary and tokenization system that harmoniously unifies diverse symbolic representations.
- Contextual embedding techniques that integrate static lexical features with dynamic, adaptive representations.
- Deep knowledge graphs that encapsulate both base and personalized relational data.
- Infinite context scaling via Rotary Positional Embeddings (RoPE) combined with a dynamic vocabulary update mechanism.
- Advanced core model architectures (involving both Transformer and RWKV modules) orchestrated via a mixture-of-experts paradigm.
- A sophisticated Titans Memory Architecture that enables multi-layer memory retrieval for adaptive test-time operations.
- A recursive, adaptive, idempotent feedback system that underpins continuous runtime learning.
- A universal training system leveraging chunk-based streaming, state-of-the-art optimization, and robust regularization.
- A final decoding module that transforms latent model outputs into interpretable, multimodal results.

This document synthesizes these components, demonstrating how their integration yields a coherent, adaptive, and scalable framework.

## 132 Summary of Modules

The comprehensive Eidos framework comprises the following modules:

**Module A: Input Processing:** Acquires and standardizes raw input.

**Module B: Universal Communication & Data Handling Interface and Coordination:** Establishes standardized data packets and communication protocols.

**Module C: Universal Streaming/Handling/Loading/Indexing Module:** Decomposes model parameters into manageable chunks, enabling efficient, hardware-agnostic streaming.

**Module D: Multidimensional Vocabulary and Tokenization System:** Constructs a unified vocabulary featuring rich, multidimensional token representations.

**Module E: Contextual NLU/NLP Embedding and Multidimensional Tokenization:** Develops dual-layer token embeddings that amalgamate stable and context-sensitive features.

**Module F: Deep Knowledge Graphs System (Base and Personal):** Builds hierarchical knowledge graphs that capture both static structures and adaptive relationships.

**Module G: Infinite RoPE Context Scaling and Dynamic Vocabulary Updating:** Implements Rotary Positional Embeddings for infinite context and integrates mechanisms for dynamic vocabulary expansion.

**Module H: Core Model Architectures (RWKV and Transformer Modules, Mixture-of-Experts Style):** Provides the primary deep processing engine using complementary architectures coordinated through a mixture-of-experts framework.

**Module I: Titans Memory Architecture (Multi-Layer Memory Module):** Facilitates adaptive, test-time memory retrieval through a multi-layered memory system.

**Module J: Recursive Adaptive Dynamic Idempotent Feedback and State-Based Runtime Learning and Inference:** Models recursive feedback loops to drive continuous learning and adaptation.

**Module K: Universal Training System:** Orchestrates end-to-end optimization using chunk-based streaming, normalization, dropout, and advanced gradient update protocols.

**Module L: Final Decoding and Multimodal Output:** Translates latent model outputs into human-interpretable and application-specific modalities.

## 133 Critical Analysis

The Eidos framework marks a significant advance in the realm of adaptive multimodal intelligence systems. Notable strengths include:

- **Modularity and Extensibility:** Every module is rigorously defined and designed with standardized interfaces, thereby facilitating independent development, testing, and iterative refinements.

- **Adaptive and Continual Learning:** By integrating mechanisms such as recursive feedback, dynamic vocabulary updates, and a multi-layer memory architecture, Eidos is inherently capable of continuous self-improvement and generalization across new data regimes.
- **Scalability and Hardware-Agnostic Design:** The utilization of chunk-based streaming and dynamic indexing empowers Eidos to scale efficiently across various hardware configurations, even for extremely large models.
- **Theoretical Rigor:** With every component anchored in formal definitions, precise algorithmic pseudocode, and proven theoretical guarantees regarding convergence, stability, and expressivity, the framework stands on a firm mathematical foundation.

However, certain challenges warrant further deliberation:

- **Complexity of Integration:** The extensive modularity implies a high degree of system complexity. Seamless integration demands rigorous interface design and comprehensive validation procedures.
- **Resource Management:** Although the streaming and caching constructs are hardware-agnostic by design, actual performance heavily depends on the efficiency of these low-level operations, particularly in resource-constrained or distributed environments.
- **Dynamic Adaptation Trade-Offs:** Striking an optimal balance between system stability (ensured via idempotence) and rapid adaptation in the face of novel data remains a non-trivial endeavor.
- **Empirical Validation:** While robust theoretical guarantees and some preliminary empirical validation underpin each module, extensive further empirical evaluation, full deployment, and benchmarking are essential to assess the framework's practical performance across diverse applications.

## 134 Future Work

Future research directions include, but are not limited to:

- **Enhanced Modular Interfaces:** Refine inter-module interfaces to further streamline integration and enable plug-and-play experimentation.
- **Optimized Streaming Mechanisms:** Develop more efficient, asynchronous streaming and caching algorithms, particularly for distributed training architectures.
- **Advanced Adaptive Techniques:** Investigate sophisticated meta–learning and recursive feedback strategies to further improve convergence and system adaptability.
- **Multimodal Extensions:** Expand the final decoding module to seamlessly incorporate additional modalities such as vision, speech, and multimodal fusion.
- **Theoretical Extensions:** Pursue deeper formal analyses of the noncommutative and holomorphic aspects within the framework, with a view toward enhancing the robustness of quantum-inspired reasoning methodologies.

## 135 Conclusion

The Eidos Unified Framework presents a meticulously defined and theoretically robust blueprint for constructing persistent, dynamic, and adaptive multimodal intelligence systems. By decomposing the system into 12 distinct yet interrelated modules—each undergirded by precise formal definitions, comprehensive algorithmic descriptions, and robust theoretical assurances—Eidos provides a potent platform for both empirical deployment and further academic inquiry. This final wrap-up has critically evaluated the framework's strengths, addressed potential integration and resource management challenges, and illuminated promising avenues for future enhancements. In totality, Eidos stands as a significant stride toward bridging high-dimensional abstract modeling with practical and scalable intelligent behavior.

## 136 Final Summary of Modules

**Modules Completed:**

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

**Overall Conclusion:** The complete Eidos framework is now defined with exceptional academic rigour, spanning raw input processing to final multimodal output. Through its modular design and robust theoretical foundation, Eidos offers a scalable, adaptable, and hardware-agnostic blueprint for next-generation intelligent systems. Future work will center on empirical validation, resource optimization, and extending multimodal capabilities to meet emergent challenges.