Documentación RNTA.jl - Módulos de Inferencia

inference/ReasoningPathways.jl

Propósito: Implementa trayectorias de razonamiento para inferencia

Funciones:

- `ReasoningNode(node_type::Symbol, tensor::Array{T,3}; metadata::Dict{Symbol, Any}=Dict{Symbol, Any}(), confidence::Float32=1.0f0)` : Constructor para nodos de razonamiento
- `ReasoningEdge(source_id::UUID, target_id::UUID, edge_type::Symbol; strength::Float32=1.0f0, metadata::Dict{Symbol, Any}=Dict{Symbol, Any}())` : Constructor para conexiones entre nodos
- `ReasoningPathway(name::String; metadata::Dict{Symbol, Any}=Dict{Symbol, Any}())` : Constructor para trayectorias de razonamiento
- `PathwayTemplate(name::String, description::String, node_types::Vector{Symbol}, edge_types::Vector{Symbol}, structure::Dict{Symbol, Any}; node_initializer::Function=default_node_initializer, node_processor::Function=default_node_processor)` : Constructor para plantillas de trayectorias
- `ReasoningEngine(brain::BrainSpace; semantic_space::Union{Semantic3DSpace, Nothing}=nothing, config::Dict{Symbol, Any}=Dict{Symbol, Any}())` : Constructor para motor de inferencia
- `initialize_templates()` : Inicializa plantillas predefinidas para trayectorias de razonamiento
- `default_node_initializer(engine, node_type, input_tensor)` : Inicializador de nodos por defecto
- `default_node_processor(engine, node, input_tensors)` : Procesador de nodos por defecto
- `combine_tensors(tensors::Vector{Array{T,3}})` : Combina múltiples tensores en uno solo
- `create_pathway(engine::ReasoningEngine, template_key::Symbol, input_tensor::Array{T,3}; name::String="")` : Crea una nueva trayectoria de razonamiento a partir de una plantilla
- `initialize_pathway_structure!(engine::ReasoningEngine, pathway::ReasoningPathway, template::PathwayTemplate)` : Inicializa la estructura de una trayectoria según la plantilla
- `add_node!(pathway::ReasoningPathway, node::ReasoningNode)` : Añade un nodo a una trayectoria de razonamiento

- `add_edge!(pathway::ReasoningPathway, edge::ReasoningEdge)` : Añade una conexión a una trayectoria de razonamiento
- `run_pathway!(engine::ReasoningEngine, pathway::ReasoningPathway)` : Ejecuta una trayectoria de razonamiento completa
- `create_dependency_graph(pathway::ReasoningPathway)` : Crea un grafo de dependencias para la trayectoria
- `topological_sort(dependencies::Dict{UUID, Vector{UUID}})` : Ordena los nodos según sus dependencias (orden topológico)
- `process_node!(engine::ReasoningEngine, pathway::ReasoningPathway, node_id::UUID)` : Procesa un nodo en la trayectoria
- `get_input_nodes(pathway::ReasoningPathway, node_id::UUID)` : Obtiene los IDs de los nodos de entrada para un nodo dado
- `calculate_pathway_confidence(pathway::ReasoningPathway)` : Calcula la confianza global de una trayectoria
- `get_pathway_result(pathway::ReasoningPathway)` : Obtiene el resultado final de una trayectoria de razonamiento
- `reason(engine::ReasoningEngine, input_tensor::Array{T,3}, template_key::Symbol=:analysis)` : Función principal para razonar sobre un tensor de entrada
- `visualize_pathway(pathway::ReasoningPathway)` : Genera una visualización de una trayectoria de razonamiento
- `compare_pathways(pathway1::ReasoningPathway, pathway2::ReasoningPathway)` : Compara dos trayectorias de razonamiento
- `combine_pathways(engine::ReasoningEngine, pathway1::ReasoningPathway, pathway2::ReasoningPathway)` : Combina dos trayectorias de razonamiento en una nueva
- `save_pathway(pathway::ReasoningPathway, filename::String)` : Guarda una trayectoria de razonamiento en un archivo
- `load_pathway(filename::String)` : Carga una trayectoria de razonamiento desde un archivo
- `create_custom_pathway(engine::ReasoningEngine, nodes_config::Vector{Dict{Symbol, Any}}, edges_config::Vector{Dict{Symbol, Any}}, input_tensor::Array{T,3}; name::String="Custom Pathway")` : Crea una trayectoria personalizada con configuración específica
- `analyze_reasoning(pathway::ReasoningPathway)` : Analiza una trayectoria de razonamiento para extraer estadísticas

- `has_cycles(pathway::ReasoningPathway)` : Verifica si una trayectoria tiene ciclos

inference/UncertaintyEstimation.jl

Propósito: Implementa estimación de incertidumbre para el sistema RNTA

Funciones:

- `UncertaintyMetrics(aleatoric::Float32, epistemic::Float32, distributional::Float32, structural::Float32, uncertainty_map::Array{T,3}; details::Dict{Symbol, Any}=Dict{Symbol, Any}())` : Constructor para métricas de incertidumbre
- `UncertaintyEstimator(brain::BrainSpace; num_samples::Int=10, noise_level::Float32=0.05f0, config::Dict{Symbol, Any}=Dict{Symbol, Any}())` : Constructor para estimador de incertidumbre
- `estimate_uncertainty(estimator::UncertaintyEstimator, input_tensor::Array{T,3})` : Estima la incertidumbre para un tensor de entrada
- `estimate_aleatoric_uncertainty(estimator::UncertaintyEstimator, input_tensor::Array{T,3})` : Estima la incertidumbre aleatoria mediante múltiples pases con ruido
- `estimate_epistemic_uncertainty(estimator::UncertaintyEstimator, input_tensor::Array{T,3})` : Estima la incertidumbre epistémica mediante diálogo interno
- `estimate_distributional_uncertainty(estimator::UncertaintyEstimator, input_tensor::Array{T,3})` : Estima la incertidumbre distribucional mediante comparación con datos conocidos
- `estimate_structural_uncertainty(estimator::UncertaintyEstimator, input_tensor::Array{T,3})` : Estima la incertidumbre estructural mediante análisis de la arquitectura
- `estimate_structural_uncertainty_fallback(estimator::UncertaintyEstimator, input_tensor::Array{T,3})` : Método alternativo para estimar incertidumbre estructural
- `calculate_uncertainty_map(estimator::UncertaintyEstimator, input_tensor::Array{T,3}, aleatoric::Float32, epistemic::Float32, distributional::Float32, structural::Float32)` : Calcula un mapa espacial de incertidumbre
- `calculate_uncertainty_metrics(estimator::UncertaintyEstimator, outputs::Vector{Array{T,3}})` : Calcula métricas de incertidumbre para un conjunto de salidas
- `calculate_entropy(tensor::Array{T,3})` : Calcula la entropía de un tensor, una medida de incertidumbre
- `estimate_uncertainty_from_pathway(estimator::UncertaintyEstimator, pathway::ReasoningPathway)` : Estima la incertidumbre a partir de una trayectoria de razonamiento

- `calibrate_uncertainty(estimator::UncertaintyEstimator, ground_truth::Vector{Array{T,3}}, predictions::Vector{Array{S,3}}, uncertainties::Vector{UncertaintyMetrics})` : Calibra el estimador de incertidumbre usando datos de referencia
- `correlation(x::Vector{T}, y::Vector{S})` : Calcula la correlación entre dos vectores
- `apply_calibration(estimator::UncertaintyEstimator, metrics::UncertaintyMetrics)` : Aplica factores de calibración a métricas de incertidumbre
- `visualize_uncertainty(metrics::UncertaintyMetrics; projection::Symbol=:max, colormap=:viridis)` : Genera una visualización de la incertidumbre
- `uncertainty_threshold(metrics::UncertaintyMetrics, threshold::Float32=0.7f0)` : Determina si la incertidumbre supera un umbral crítico
- `monte_carlo_uncertainty(estimator::UncertaintyEstimator, input_tensor::Array{T,3}, num_samples::Int=20)` : Estima incertidumbre con mayor precisión usando muestreo Monte Carlo

inference/InternalDialogue.jl

Propósito: Implementa mecanismos de diálogo interno y deliberación

Funciones:

- `DialogueAgent(name::String, brain::BrainSpace; traits::Dict{Symbol, Float32}=Dict{Symbol, Float32}(), influence::Float32=1.0f0)` : Constructor para agentes de diálogo
- `DialogueContext(stimulus::Array{T,3}; metadata::Dict{Symbol, Any}=Dict{Symbol, Any}(), objective_function::Union{Function, Nothing}=nothing)` : Constructor para contexto de diálogo
- `DialogueSystem(brain::BrainSpace; num_agents::Int=3, config::Dict{Symbol, Any}=Dict{Symbol, Any}(), semantic_space::Union{Semantic3DSpace, Nothing}=nothing, context_mapper::Union{ContextMapper, Nothing}=nothing)` : Constructor para sistema de diálogo
- `create_default_agents(brain::BrainSpace, num_agents::Int)` : Crea un conjunto predeterminado de agentes para el diálogo
- `start_dialogue!(system::DialogueSystem, stimulus::Array{T,3}; metadata::Dict{Symbol, Any}=Dict{Symbol, Any}(), objective_function::Union{Function, Nothing}=nothing)` : Inicia un nuevo diálogo interno con un estímulo dado
- `initialize_agents!(system::DialogueSystem, stimulus::Array{T,3})` : Inicializa los agentes con el estímulo inicial
- `initialize_agent!(agent::DialogueAgent, stimulus::Array{T,3}, config::Dict{Symbol, Any})` : Inicializa un agente con un estímulo

- `apply_agent_bias(stimulus::Array{T,3}, traits::Dict{Symbol, Float32})` : Aplica sesgos al estímulo basado en los rasgos del agente
- `run_dialogue_step!(system::DialogueSystem)` : Ejecuta un paso del diálogo interno
- `select_next_agent(system::DialogueSystem)` : Selecciona el próximo agente para contribuir al diálogo
- `process_with_agent(agent::DialogueAgent, context::DialogueContext)` : Procesa el estímulo o estado actual con el cerebro del agente
- `record_contribution!(context::DialogueContext, agent_id::UUID, response::Array{T,3})` : Registra una contribución al diálogo
- `update_consensus!(system::DialogueSystem)` : Actualiza el tensor de consenso basado en las contribuciones de los agentes
- `check_convergence!(system::DialogueSystem)` : Verifica si el diálogo ha convergido
- `tensor_similarity(tensor1::Array{T,3}, tensor2::Array{S,3})` : Calcula la similitud entre dos tensores
- `run_dialogue!(system::DialogueSystem, stimulus::Array{T,3}, max_steps::Int=10; convergence_threshold::Float32=0.9f0, metadata::Dict{Symbol, Any}=Dict{Symbol, Any}(), objective_function::Union{Function, Nothing}=nothing)` : Ejecuta un diálogo completo hasta convergencia o número máximo de pasos
- `finish_dialogue!(system::DialogueSystem)` : Finaliza el diálogo actual
- `apply_consensus_to_brain!(system::DialogueSystem)` : Aplica el consenso final al cerebro base
- `get_dialogue_result(system::DialogueSystem)` : Obtiene el resultado del último diálogo
- `analyze_dialogue(context::DialogueContext)` : Analiza un diálogo para extraer información sobre las contribuciones
- `internal_dialogue(brain::BrainSpace, input_tensor::Array{T,3}; max_steps::Int=5, num_agents::Int=3, convergence_threshold::Float32=0.9f0, config::Dict{Symbol, Any}=Dict{Symbol, Any}())` : Función principal para diálogo interno de un cerebro con un tensor de entrada

nlp/LanguageGeneration.jl

Propósito: Implementa mecanismos de generación de lenguaje natural a partir de tensores

Funciones:

- `DecoderConfig(; temperature::Float32=1.0f0, threshold::Float32=0.05f0, repetition_penalty::Float32=1.2f0, context_window::Int=50, max_tokens::Int=100, decoding_strategy::Symbol=:sampling, normalize_tensors::Bool=true, context_weight::Float32=0.3f0)` : Constructor para configuración del decodificador
- `LanguageDecoder(tokenizer::TensorialTokenizer, dimensions::NTuple{3,Int}; semantic_space::Union{Semantic3DSpace, Nothing}=nothing, context_mapper::Union{ContextMapper, Nothing}=nothing, config::DecoderConfig=DecoderConfig())` : Constructor para decodificador de lenguaje
- `generate_text(decoder::LanguageDecoder, input_tensor::Array{T,3}; max_length::Int=100, context::Union{Array{S,3}, Nothing}=nothing)` : Genera texto a partir de un tensor de entrada
- `greedy_decoding(decoder::LanguageDecoder, input_tensor::Array{T,3}, max_length::Int)` : Implementa decodificación voraz (selecciona siempre el token más probable)
- `sampling_decoding(decoder::LanguageDecoder, input_tensor::Array{T,3}, max_length::Int)` : Implementa decodificación por muestreo (selecciona tokens con probabilidad proporcional)
- `beam_search_decoding(decoder::LanguageDecoder, input_tensor::Array{T,3}, max_length::Int; beam_width::Int=5)` : Implementa decodificación por búsqueda en haz
- `compute_token_probabilities(decoder::LanguageDecoder, tensor::Array{T,3}, previous_tokens::Vector{Int})` : Calcula las probabilidades de cada token para el siguiente paso
- `compute_semantic_probabilities(decoder::LanguageDecoder, tensor::Array{T,3}, previous_tokens::Vector{Int})` : Calcula probabilidades de tokens usando el espacio semántico
- `compute_projection_probabilities(decoder::LanguageDecoder, tensor::Array{T,3}, previous_tokens::Vector{Int})` : Calcula probabilidades proyectando el tensor directamente a espacio de vocabulario
- `adjust_with_ngrams(probabilities::Vector{Float32}, previous_tokens::Vector{Int}, tokenizer::TensorialTokenizer)` : Ajusta probabilidades usando información de n-gramas
- `softmax(logits::Vector{T})` : Aplica función softmax para convertir logits a probabilidades
- `apply_temperature(probabilities::Vector{Float32}, temperature::Float32)` : Aplica temperatura para ajustar la distribución de probabilidad
- `apply_threshold(probabilities::Vector{Float32}, threshold::Float32)` : Filtra tokens con probabilidad por debajo del umbral
- `apply_repetition_penalty(probabilities::Vector{Float32}, previous_tokens::Vector{Int}, penalty::Float32, window_size::Int)` : Aplica penalización a tokens repetidos recientemente

- `normalize_probabilities(probabilities::Vector{Float32})` : Normaliza vector de probabilidades para que sumen 1
- `sample_from_distribution(probabilities::Vector{Float32})` : Muestrea un índice según la distribución de probabilidad
- `get_top_k_tokens(probabilities::Vector{Float32}, k::Int)` : Obtiene los k tokens más probables con sus probabilidades
- `update_generation_state(decoder::LanguageDecoder, current_state::Array{T,3}, next_token::Int)` : Actualiza el estado de generación después de seleccionar un token
- `get_token_tensor(tokenizer::TensorialTokenizer, token_id::Int)` : Obtiene representación tensorial de un token
- `is_end_token(tokenizer::TensorialTokenizer, token_id::Int)` : Verifica si un token es de fin de secuencia
- `decode_tokens(tokenizer::TensorialTokenizer, tokens::Vector{Int})` : Convierte una secuencia de tokens a texto
- `normalize_tensor(tensor::Array{T,3})` : Normaliza un tensor para evitar valores extremos
- `generate_text_from_semantic(decoder::LanguageDecoder, concept_id::String; max_length::Int=100)` : Genera texto a partir de un concepto en el espacio semántico
- `generate_text_from_context(decoder::LanguageDecoder, context_mapper::ContextMapper; max_length::Int=100)` : Genera texto a partir del estado de contexto actual
- `generate_continuation(decoder::LanguageDecoder, text::String; max_length::Int=100)` : Genera una continuación para un texto dado

nlp/SemanticSpace.jl

Propósito: Implementa un espacio semántico tridimensional para representación de lenguaje

Funciones:

- `SemanticRepresentation(label::String, tensor::Array{T,3}; id::String="", metadata::Dict{Symbol, Any}=Dict{Symbol, Any}(), confidence::Float32=1.0f0)` : Constructor para representación semántica
- `Semantic3DSpace(dimensions::NTuple{3,Int}; brain::Union{BrainSpace, Nothing}=nothing, tokenizer::Union{TensorialTokenizer, Nothing}=nothing, dimension_weights::NTuple{3,Float32}=(1.0f0, 1.0f0, 1.0f0))` : Constructor para espacio semántico 3D

- `add_concept!(space::Semantic3DSpace, representation::SemanticRepresentation)` : Añade un concepto al espacio semántico
- `remove_concept!(space::Semantic3DSpace, concept_id::String)` : Elimina un concepto del espacio semántico
- `update_spatial_index!(space::Semantic3DSpace, representation::SemanticRepresentation)` : Actualiza el índice espacial para una representación
- `remove_from_spatial_index!(space::Semantic3DSpace, representation::SemanticRepresentation)` : Elimina una representación del índice espacial
- `find_significant_regions(tensor::Array{T,3}; threshold::Float32=0.5f0)` : Encuentra regiones con valores significativos en un tensor
- `find_connected_regions(mask::BitArray{3})` : Encuentra regiones conectadas en una máscara binaria 3D
- `explore_region!(mask::BitArray{3}, labels::Array{Int,3}, start_x::Int, start_y::Int, start_z::Int, label::Int)` : Explora una región conectada usando búsqueda en anchura
- `query(space::Semantic3DSpace, text_query::String; top_k::Int=5)` : Realiza una consulta semántica basada en texto
- `query(space::Semantic3DSpace, tensor_query::Array{T,3}; top_k::Int=5)` : Realiza una consulta semántica basada en tensor
- `update_attention!(space::Semantic3DSpace, query_tensor::Array{T,3})` : Actualiza el mapa de atención basado en una consulta
- `semantic_search(space::Semantic3DSpace, query_tensor::Array{T,3}; top_k::Int=5)` : Realiza una búsqueda semántica en el espacio
- `apply_dimension_weights(tensor::Array{T,3}, weights::NTuple{3,Float32})` : Aplica ponderación por dimensiones a un tensor
- `compute_similarity(tensor1::Array{T,3}, tensor2::Array{S,3}, attention_map::SpatialAttentionMap)` : Calcula similitud entre dos tensores, ponderada por atención
- `create_concept_from_text(text::String, tokenizer::TensorialTokenizer; label::String="", metadata::Dict{Symbol,Any}=Dict{Symbol,Any}(), confidence::Float32=1.0f0)` : Crea una representación semántica a partir de texto
- `merge_representations(rep1::SemanticRepresentation, rep2::SemanticRepresentation; weight1::Float32=0.5f0, weight2::Float32=0.5f0)` : Combina dos representaciones semánticas

- `extract_concept_field(space::Semantic3DSpace, concept_id::String)` : Extrae la representación de un concepto como campo espacial
- `find_related_concepts(space::Semantic3DSpace, concept_id::String; top_k::Int=5)` : Encuentra conceptos relacionados con un concepto dado
- `process_query_in_brain!(space::Semantic3DSpace, query::String)` : Procesa una consulta semántica en el cerebro asociado
- `visualize_semantic_space(space::Semantic3DSpace; show_concepts::Bool=true, show_attention::Bool=true, highlight_concept::String="")` : Genera una visualización del espacio semántico
- `save_semantic_space(space::Semantic3DSpace, filename::String)` : Guarda el espacio semántico en un archivo
- `load_semantic_space(filename::String; brain::Union{BrainSpace, Nothing}=nothing, tokenizer::Union{TensorialTokenizer, Nothing}=nothing)` : Carga un espacio semántico desde un archivo

nlp/TensorialTokenizer.jl

Propósito: Implementa tokenización de texto a representaciones tensoriales 3D

Funciones:

- `TensorialTokenizer(vocabulary::Vector{String}; embedding_dims::NTuple{3,Int}=(5, 5, 5), init_scale::Float32=0.1f0, context_dim::Int=10, padding_strategy::Symbol=:right, unk_token::String="<UNK>")` : Constructor principal para TensorialTokenizer
- `tokenize(tokenizer::TensorialTokenizer, text::String)` : Convierte texto en secuencia de índices de tokens
- `encode_tensor(tokenizer::TensorialTokenizer, token_indices::Vector{Int}; max_length::Int=nothing)` : Convierte secuencia de índices en representación tensorial 3D
- `apply_context(tokenizer::TensorialTokenizer, token_indices::Vector{Int}, seq_tensor::Array{Float32,6})` : Aplica información contextual al tensor de secuencia
- `reduce_dimensions(tensor6d::Array{Float32,6})` : Reduce un tensor 6D a 3D para representación final
- `process_text(tokenizer::TensorialTokenizer, text::String; max_length::Int=nothing)` : Procesa texto directamente a representación tensorial

- `create_default_tokenizer(vocabulary_size::Int=10000)` : Crea un tokenizador tensorial con configuración por defecto

nlp/ContextualMapping.jl

Propósito: Implementa mapeo contextual para procesamiento de lenguaje natural

Funciones:

- `ContextState(dimensions::NTuple{3,Int}; history_length::Int=5, decay_factor::Float32=0.8f0, inertia_factor::Float32=0.3f0)` : Constructor para estado de contexto
- `ContextMapper(dimensions::NTuple{3,Int}; semantic_space::Union{Semantic3DSpace, Nothing}=nothing, tokenizer::Union{TensorialTokenizer, Nothing}=nothing, brain::Union{BrainSpace, Nothing}=nothing, config::Dict{Symbol, Any}=Dict{Symbol, Any}())` : Constructor para mapeador contextual
- `initialize_context_operators()` : Inicializa los operadores contextuales predefinidos
- `process_text(mapper::ContextMapper, text::String; operation::Symbol=:add, weight::Float32=1.0f0)` : Procesa texto y actualiza el estado de contexto
- `process_tensor(mapper::ContextMapper, tensor::Array{T,3}; operation::Symbol=:add, weight::Float32=1.0f0)` : Procesa un tensor de entrada y actualiza el estado de contexto
- `update_context!(state::ContextState, input_tensor::Array{T,3}; operation::Symbol=:add, weight::Float32=1.0f0)` : Actualiza el estado de contexto con un tensor de entrada
- `update_attention!(state::ContextState, input_tensor::Array{T,3})` : Actualiza el mapa de atención del contexto basado en el tensor de entrada
- `find_significant_regions(tensor::Array{T,3}; threshold::Float32=0.5f0)` : Encuentra regiones con valores significativos en un tensor
- `find_connected_regions(mask::BitArray{3})` : Encuentra regiones conectadas en una máscara binaria 3D
- `explore_region!(mask::BitArray{3}, labels::Array{Int,3}, start_x::Int, start_y::Int, start_z::Int, label::Int)` : Explora una región conectada usando búsqueda en anchura
- `extract_entities!(mapper::ContextMapper, text::String)` : Extrae entidades del texto y las añade al contexto
- `decay_entities!(state::ContextState)` : Aplica decaimiento a entidades activas en el contexto
- `normalize_tensor(tensor::Array{T,3})` : Normaliza un tensor para evitar valores extremos

- `get_context_vector(mapper::ContextMapper)` : Obtiene una representación vectorial del contexto actual
- `reset_context!(mapper::ContextMapper)` : Reinicia el estado de contexto a cero
- `get_active_entities(mapper::ContextMapper; threshold::Float32=0.1f0)` : Obtiene las entidades actualmente activas en el contexto
- `get_context_history(mapper::ContextMapper; n::Int=3)` : Obtiene los últimos n estados de contexto
- `compare_contexts(context1::Array{T,3}, context2::Array{S,3})` : Compara dos estados de contexto y calcula su similitud
- `save_context_state(state::ContextState, filename::String)` : Guarda el estado de contexto en un archivo
- `load_context_state(filename::String, dimensions::NTuple{3,Int})` : Carga un estado de contexto desde un archivo

operations/SpatialAttention.jl

Propósito: Implementa mecanismos de atención volumétrica para el espacio cerebral

Funciones:

- `SpatialAttentionMap(dimensions::NTuple{3,Int}; focus_factor::Float32=2.0f0, effective_radius::Float32=5.0f0, decay_type::Symbol=:gaussian)` : Constructor principal para mapa de atención espacial
- `shift_attention!(attention_map::SpatialAttentionMap, new_center::NTuple{3,Float32})` : Desplaza el centro de atención a una nueva posición
- `adjust_focus!(attention_map::SpatialAttentionMap, new_focus_factor::Float32)` : Ajusta el factor de enfoque (concentración) de la atención
- `adjust_radius!(attention_map::SpatialAttentionMap, new_radius::Float32)` : Ajusta el radio efectivo de atención
- `apply_attention(tensor::Array{T,3}, attention_map::SpatialAttentionMap)` : Aplica el mapa de atención a un tensor de entrada
- `recalculate_attention_field!(attention_map::SpatialAttentionMap)` : Recalcula el campo de atención basado en los parámetros actuales
- `create_attention_from_activity(activity_tensor::Array{T,3}; threshold::Float32=0.5f0, radius::Float32=3.0f0, focus_factor::Float32=2.0f0, decay_type::Symbol=:gaussian)` : Crea un

mapa de atención basado en un tensor de actividad

- `multi_focus_attention(activity_tensor::Array{T,3}, num_foci::Int=3; threshold::Float32=0.3f0, radius::Float32=3.0f0, focus_factor::Float32=1.5f0)` : Crea un mapa de atención con múltiples focos basado en puntos de actividad
- `attention_guided_convolution(input::Array{T,3}, kernel::Array{S,3}, attention_map::SpatialAttentionMap; stride::NTuple{3,Int}=(1,1,1), padding::Int=0)` : Realiza una convolución tensorial guiada por atención

operations/TensorTransformations.jl

Propósito: Implementa transformaciones fundamentales para tensores 3D

Funciones:

- `tensor_convolution(input::Array{T,3}, kernel::Array{S,3}; stride::NTuple{3,Int}=(1,1,1), padding::Int=0)` : Aplica una convolución 3D adaptativa al tensor de entrada
- `adaptive_pooling(input::Array{T,3}, output_size::NTuple{3,Int}; mode::Symbol=:max)` : Aplica pooling adaptativo para redimensionar el tensor a output_size
- `tensor_interpolation(input::Array{T,3}, output_size::NTuple{3,Int}; mode::Symbol=:linear)` : Redimensiona un tensor mediante interpolación
- `spatial_attention_transform(input::Array{T,3}, attention_map::Array{S,3})` : Aplica una transformación atencional ponderando regiones según un mapa de atención
- `zero_pad(tensor::Array{T,3}, padding::Int)` : Añade padding de ceros alrededor del tensor
- `distance_weights(region_size::NTuple{3,Int})` : Genera pesos basados en distancia al centro para pooling ponderado

operations/VolumetricActivations.jl

Propósito: Implementa funciones de activación que operan en volúmenes completos

Funciones:

- `volumetric_activation(tensor::Array{T,3}; type::Symbol=:adaptive_tanh, parameters=nothing)` : Aplica una función de activación volumétrica al tensor 3D
- `adaptive_tanh_activation(tensor::Array{T,3}, parameters=nothing)` : Implementación de tanh con pendiente adaptativa basada en la magnitud local
- `volumetric_swish(tensor::Array{T,3}, parameters=nothing)` : Implementación volumétrica de Swish con modulación contextual

- `tensor_relu(tensor::Array{T,3}, parameters=nothing)` : ReLU tensorial con fugas adaptativas y respuesta sinusoidal
- `spatial_activation(tensor::Array{T,3}, parameters=nothing)` : Activación que preserva relaciones espaciales y refuerza gradientes espaciales
- `contextual_activation(tensor::Array{T,3}, parameters=nothing)` : Activación que modula respuestas basadas en el contexto local
- `temporal_activation(tensor::Array{T,3}, parameters)` : Activación que simula dinámica temporal dentro del espacio tensorial
- `feature_activation(tensor::Array{T,3}, parameters=nothing)` : Activación que enfatiza características distintivas en el tensor
- `general_activation(tensor::Array{T,3}, parameters=nothing)` : Activación general que combina diferentes tipos según el contexto

operations/PropagationDynamics.jl

Propósito: Implementa dinámica de propagación de activación en el espacio neuronal 3D

Funciones:

- `PropagationParameters(; temporal_decay::Float32=0.9f0, spatial_attenuation::Float32=0.2f0, propagation_speed::Float32=1.0f0, activation_threshold::Float32=0.3f0, refractory_period::Int=2, integration_factor::Float32=0.5f0, propagation_type::Symbol=:wave)` : Constructor con valores por defecto para parámetros de propagación
- `propagate_activation(brain::BrainSpace, input_activation::Array{T,3}; params::PropagationParameters=PropagationParameters())` : Propaga la activación a través del espacio cerebral según los parámetros especificados
- `propagate_wave(activation_field::Array{T,3}, refractory_state::BitArray{3}, refractory_counter::Array{Int,3}, params::PropagationParameters)` : Implementa propagación de activación tipo onda
- `propagate_diffusion(activation_field::Array{T,3}, params::PropagationParameters)` : Implementa propagación de activación por difusión
- `propagate_saltatory(activation_field::Array{T,3}, neurons::Dict{NTuple{3,Int}, TensorNeuron}, connections::Vector{TensorConnection}, params::PropagationParameters)` : Implementa propagación de activación saltatorio a través de conexiones

- `extract_temporal_dynamics(activation_history::Vector{Array{T,3}}, position::NTuple{3,Int})` : Extrae la dinámica temporal de activación en una posición específica
- `calculate_temporal_features(time_series::Vector{T})` : Calcula características temporales de una serie de activación
- `visualize_propagation(activation_history::Vector{Array{T,3}})` : Genera una visualización de la propagación de activación a través del tiempo

training/MultidimensionalLoss.jl

Propósito: Implementa funciones de pérdida para tensores 3D

Funciones:

- `MultidimensionalLoss(; base_type::Symbol=:mse, coherence_weight::Float32=0.2f0, regularization_weight::Float32=0.01f0, focus_factor::Float32=2.0f0, error_threshold::Float32=0.5f0)` : Constructor principal para función de pérdida multidimensional
- `calculate_loss(loss_function::MultidimensionalLoss, prediction::Array{T,3}, target::Array{S,3})` : Calcula la pérdida y su gradiente entre predicción y objetivo
- `calculate_base_loss(type::Symbol, prediction::Array{T,3}, target::Array{S,3})` : Calcula la pérdida base según el tipo especificado
- `calculate_coherence_loss(prediction::Array{T,3}, target::Array{S,3})` : Calcula la pérdida de coherencia espacial basada en gradientes locales
- `spatial_gradients(tensor::Array{T,3})` : Calcula los gradientes espaciales de un tensor 3D
- `calculate_regularization(tensor::Array{T,3})` : Calcula un término de regularización para prevenir valores extremos
- `apply_focus(gradient::Array{T,3}, prediction::Array{S,3}, target::Array{U,3}, focus_factor::Float32, threshold::Float32)` : Amplifica el gradiente en regiones con error por encima del umbral
- `default_loss()` : Crea una función de pérdida multidimensional con parámetros por defecto

training/SpatialOptimizers.jl

Propósito: Implementa optimizadores especializados para espacios tensoriales 3D

Funciones:

- `SpatialOptimizer(brain::BrainSpace; alpha::Float32=0.001f0, beta1::Float32=0.9f0, beta2::Float32=0.999f0, lambda::Float32=0.2f0, gamma::Float32=1.0f0, delta::Float32=0.1f0,`

`period::Int=1000, epsilon::Float32=1f-8)` : Constructor principal para optimizador espacial

- `optimization_step!(neuron::TensorNeuron, gradient::Array{T,3}, optimizer::SpatialOptimizer)` : Aplica un paso de optimización a una neurona usando el optimizador espacial
- `update_spatial_modulator!(modulator::Array{Float32,3}, gradient::Array{T,3}, lambda::Float32)` : Actualiza el modulador espacial basado en gradientes locales
- `default_optimizer(brain::BrainSpace)` : Crea un optimizador con parámetros por defecto para el espacio cerebral

training/GradientPropagation.jl

Propósito: Implementa propagación de gradientes en el espacio tensorial 3D

Funciones:

- `GradientConfig(; backprop_factor::Float32=1.0f0, lateral_factor::Float32=0.3f0, update_method::Symbol=:adam, momentum::Float32=0.9f0, gradient_clip::Float32=5.0f0, use_tensor_gradients::Bool=true, regularization_lambda::Float32=0.0001f0, regularization_type::Symbol=:L2)` : Constructor con valores por defecto para configuración de gradientes
- `initialize_adam_state(tensor_size::NTuple{3,Int})` : Inicializa el estado de Adam para un tensor del tamaño dado
- `initialize_adam_state(scalar::Bool=true)` : Inicializa el estado de Adam para un valor escalar
- `compute_gradients(brain::BrainSpace, loss_gradient::Array{T,3}, config::GradientConfig=GradientConfig())` : Computa gradientes para todas las neuronas y conexiones en el cerebro
- `extract_local_gradient(global_gradient::Array{T,3}, position::NTuple{3,Int}, config::GradientConfig)` : Extrae gradiente local para una neurona en la posición dada
- `compute_neuron_gradient(neuron::TensorNeuron, local_gradient::Array{T,3}, config::GradientConfig)` : Computa el gradiente para una neurona específica
- `compute_connection_gradient(connection::TensorConnection, brain::BrainSpace, neuron_gradients::Dict{UUID, Array{T,3}}, config::GradientConfig)` : Computa el gradiente para una conexión específica
- `apply_gradients!(brain::BrainSpace, neuron_gradients::Dict{UUID, Array{T,3}}, connection_gradients::Dict{UUID, Array{S,3}}, learning_rate::Float32,`

`config::GradientConfig=GradientConfig())` : Aplica los gradientes calculados a las neuronas y conexiones

- `apply_sgd_update!(neuron::TensorNeuron, gradient::Union{Array{T,3}, T}, learning_rate::Float32, config::GradientConfig)` : Aplica actualización SGD a una neurona
- `apply_sgd_update!(connection::TensorConnection, gradient::Union{Array{T,3}, T}, learning_rate::Float32, config::GradientConfig)` : Aplica actualización SGD a una conexión
- `apply_momentum_update!(neuron::TensorNeuron, gradient::Union{Array{T,3}, T}, learning_rate::Float32, optimizer_state::AdamState, config::GradientConfig)` : Aplica actualización con momentum a una neurona
- `apply_momentum_update!(connection::TensorConnection, gradient::Union{Array{T,3}, T}, learning_rate::Float32, optimizer_state::AdamState, config::GradientConfig)` : Aplica actualización con momentum a una conexión
- `apply_adam_update!(neuron::TensorNeuron, gradient::Union{Array{T,3}, T}, learning_rate::Float32, optimizer_state::AdamState, config::GradientConfig)` : Aplica actualización Adam a una neurona
- `apply_adam_update!(connection::TensorConnection, gradient::Union{Array{T,3}, T}, learning_rate::Float32, optimizer_state::AdamState, config::GradientConfig)` : Aplica actualización Adam a una conexión
- `backpropagate_gradients!(brain::BrainSpace, loss_gradient::Array{T,3}, learning_rate::Float32, config::GradientConfig=GradientConfig())` : Propaga gradientes a través del cerebro y aplica actualizaciones
- `process_batch!(brain::BrainSpace, input_batch::Vector{Array{T,3}}, target_batch::Vector{Array{S,3}}, loss_function, learning_rate::Float32, config::GradientConfig=GradientConfig())` : Procesa un batch de entrenamiento completo
- `calculate_loss(loss_function, prediction::Array{T,3}, target::Array{S,3})` : Calcula pérdida y gradiente dado un par predicción-objetivo
- `mse_loss(prediction::Array{T,3}, target::Array{S,3})` : Calcula pérdida de error cuadrático medio y su gradiente
- `mae_loss(prediction::Array{T,3}, target::Array{S,3})` : Calcula pérdida de error absoluto medio y su gradiente
- `huber_loss(prediction::Array{T,3}, target::Array{S,3}, delta::Float32=1.0f0)` : Calcula pérdida Huber y su gradiente

training/ModelCloning.jl

Propósito: Implementa mecanismos de clonación y deliberación interna

Funciones:

- `refine_with_internal_dialogue!(brain::BrainSpace, input::Array{T,3}, target::Array{S,3}; num_clones::Int=3)` : Refina el procesamiento mediante deliberación interna
- `evaluate_clone_confidences(clone_outputs::Vector{Array{T,3}}, target::Array{S,3})` : Evalúa la confianza de cada clon basada en su cercanía al objetivo
- `evaluate_internal_coherence(output::Array{T,3})` : Evalúa la coherencia interna de un output cuando no hay objetivo conocido
- `generate_consensus(clone_outputs::Vector{Array{T,3}}, confidences::Vector{Float32})` : Genera un consenso ponderado por confianza de los outputs de los clones
- `calculate_dialogue_gradient(original_output::Array{T,3}, consensus::Array{S,3})` : Calcula el gradiente de diálogo interno basado en la diferencia entre el output original y el consenso
- `apply_dialogue_gradient!(brain::BrainSpace, dialogue_gradient::Array{T,3}, learning_rate::Float32)` : Aplica el gradiente de diálogo interno al espacio cerebral
- `calculate_neuron_gradient(brain::BrainSpace, neuron::TensorNeuron, global_gradient::Array{T,3})` : Calcula el gradiente para una neurona específica basado en el gradiente global

utils/TensorIO.jl

Propósito: Implementa funcionalidades de entrada/salida para tensores y modelos

Funciones:

- `save_tensor(tensor, filename::String; format=:jld2, compression=true, metadata=nothing)` : Guarda un tensor en un archivo con el formato especificado
- `load_tensor(filename::String; format=nothing)` : Carga un tensor desde un archivo
- `export_model(brain_space::BrainSpace, filename::String; include_weights=true, include_activations=false, format=:jld2)` : Exporta un modelo RNTA completo a un archivo
- `import_model(filename::String; format=nothing)` : Importa un modelo RNTA desde un archivo
- `tensor_to_hdf5(tensor, filename::String; dataset_name="tensor", compression=true)` : Guarda un tensor específicamente en formato HDF5 con opciones avanzadas

- `tensor_from_hdf5(filename::String; dataset_name="tensor")` : Carga un tensor desde un archivo HDF5 específico
- `metadata_to_json(metadata, filename::String)` : Guarda metadatos en un archivo JSON separado
- `metadata_from_json(filename::String)` : Carga metadatos desde un archivo JSON
- `batched_tensor_save(tensors::Dict, base_filename::String; format=:jld2, compression=true, batch_size=10)` : Guarda múltiples tensores en archivos por lotes para eficiencia
- `batched_tensor_load(base_filename::String)` : Carga múltiples tensores guardados por lotes
- `tensor_checkpoint(tensor, checkpoint_dir::String, name::String; keep_last=5, metadata=nothing)` : Guarda un punto de control de un tensor con historial limitado
- `model_versioning(brain_space::BrainSpace, version::String, repo_dir::String; include_weights=true, include_activations=false, description="")` : Guarda una versión específica de un modelo RNTA en un repositorio versionado
- `model_conversion(brain_space::BrainSpace, format::Symbol, output_file::String; framework=:default, config=Dict())` : Convierte un modelo RNTA a otro formato o framework
- `validate_tensor_file(filename::String; format=nothing)` : Valida un archivo de tensor, verificando su integridad y estructura

utils/ConfigurationSystem.jl

Propósito: Implementa sistema de configuración para RNTA

Funciones:

- `configure_brain_space(brain_space::BrainSpace, config::RNTAConfig)` : Aplica una configuración a un espacio cerebral existente
- `load_configuration(filename::String; format::Symbol=:toml)` : Carga una configuración desde un archivo
- `save_configuration(config::RNTAConfig, filename::String; format::Symbol=:toml)` : Guarda una configuración en un archivo
- `merge_configurations(base_config::RNTAConfig, override_config::RNTAConfig)` : Combina dos configuraciones, con la segunda tomando precedencia
- `validate_configuration(config::RNTAConfig)` : Valida que una configuración sea correcta y consistente
- `apply_presets(config::RNTAConfig, preset_name::Symbol)` : Aplica un preset predefinido a una configuración

- `available_presets()` : Devuelve la lista de presets disponibles
- `register_preset(name::Symbol, config::Dict{Symbol,Any})` : Registra un nuevo preset de configuración
- `dump_configuration(config::RNTAConfig; format::Symbol=:readable)` : Convierte una configuración a texto para visualización o debugging
- `diff_configurations(config1::RNTAConfig, config2::RNTAConfig)` : Compara dos configuraciones y muestra las diferencias
- `configure_for_hardware(config::RNTAConfig, hardware_type::Symbol)` : Configura optimizaciones específicas para un tipo de hardware
- `configure_for_task(config::RNTAConfig, task_type::Symbol)` : Configura optimizaciones específicas para un tipo de tarea
- `required_configuration(module_name::Symbol)` : Obtiene la configuración mínima requerida para un módulo específico

utils/PerformanceMetrics.jl

Propósito: Implementa sistema de métricas de rendimiento y benchmarking

Funciones:

- `track_performance(tracker::MetricsTracker, key::Symbol, value::Number)` : Registra una métrica de rendimiento específica
- `start_timing(tracker::MetricsTracker, key::Symbol)` : Inicia la medición de tiempo para una operación específica
- `end_timing(tracker::MetricsTracker, key::Symbol)` : Finaliza la medición de tiempo para una operación específica y registra el resultado
- `track_loss(tracker::MetricsTracker, loss_value::Number)` : Registra un valor de pérdida durante el entrenamiento
- `track_accuracy(tracker::MetricsTracker, accuracy::Union{Nothing,Number})` : Registra un valor de precisión durante el entrenamiento
- `track_epoch_time(tracker::MetricsTracker, seconds::Number)` : Registra el tiempo de procesamiento de una época
- `compute_statistics(values::Vector{<:Number})` : Calcula estadísticas descriptivas para un conjunto de valores

- `create_performance_report(tracker::MetricsTracker)` : Genera un informe completo de rendimiento a partir de las métricas registradas
- `log_metrics(logger::MetricsLogger, metrics::Dict)` : Registra métricas en el archivo de log y en el historial
- `benchmark_model(brain_space::BrainSpace, input_dims::Tuple, num_iterations::Int=100; warmup::Int=10, use_cuda::Bool=CUDA.functional())` : Realiza un benchmark exhaustivo del modelo
- `memory_profile(brain_space::BrainSpace, input_dims::Tuple; track_detailed::Bool=false, use_cuda::Bool=CUDA.functional())` : Analiza el uso de memoria del modelo durante la ejecución
- `latency_profile(brain_space::BrainSpace, input_dims::Tuple, num_iterations::Int=10; breakdown::Bool=true, use_cuda::Bool=CUDA.functional())` : Analiza la latencia del modelo con desglose por componentes
- `throughput_test(brain_space::BrainSpace, batch_sizes::Vector{Int}, sequence_length::Int, num_iterations::Int=5; use_cuda::Bool=CUDA.functional())` : Mide el rendimiento del modelo con diferentes tamaños de batch
- `compare_configurations(configs::Vector{Dict}, brain_space::BrainSpace, input_dims::Tuple, num_iterations::Int=10)` : Compara múltiples configuraciones del modelo para rendimiento

utils/Serialization.jl

Propósito: Implementa funcionalidades para guardar y cargar modelos RNTA

Funciones:

- `save_brain(brain::BrainSpace, filename::String)` : Guarda un modelo RNTA en un archivo
- `load_brain(filename::String)` : Carga un modelo RNTA desde un archivo
- `save_checkpoint(brain::BrainSpace, base_filename::String; max_checkpoints::Int=5)` : Guarda un checkpoint del modelo, manteniendo un número limitado de versiones
- `to_tensor(data)` : Convierte diferentes tipos de datos a representación tensorial 3D
- `from_tensor(tensor::Array{T,3}, output_type::Symbol=:auto)` : Convierte un tensor 3D de vuelta a un tipo de datos más simple si es posible
- `brain_summary(brain::BrainSpace)` : Genera un resumen estadístico del estado del cerebro