Hierarchical Shallow Predictive Matter Networks (HSPMN): Integrating Multi-Scale Prediction with Parallel Processing Through Active Matter Dynamics

Szymon Jdryczko[*]

March 28, 2025

Abstract

Current artificial intelligence architectures typically emphasize either deep hierarchical processing or flat distributed computation, but rarely integrate both effectively. This paper introduces Hierarchical Shallow Predictive Matter Networks (HSPMN), a novel computational architecture that unifies hierarchical predictive coding with parallel shallow processing through dynamic self-organization principles inspired by active matter physics. HSPMN addresses fundamental limitations in existing systems by enabling simultaneous multi-scale prediction and rapid parallel computation, dynamically allocating resources based on task demands. The architecture features hierarchical modules that capture increasingly abstract patterns, parallel units that enable direct communication regardless of hierarchical position, dynamic connectivity with non-reciprocal interactions derived from active matter systems, and oscillatory synchronization mechanisms for coordination. This integration provides significant advantages in prediction-speed tradeoffs, cross-timescale context integration, adaptation without catastrophic forgetting, and robustness through distributed redundancy. We propose implementation strategies and experimental designs to validate HSPMN's potential for advancing continual learning, resource-efficient AI, and robust distributed systems. The architecture represents a fundamental reimagining of how computational systems can organize themselves, with implications across multiple domains of artificial intelligence.

Keywords: HSPMN, Hierarchical processing, Parallel computation, Predictive coding, Active matter physics, Self-organization, Shallow brain hypothesis, Multi-scale prediction, Non-reciprocal interactions, Oscillatory synchronization

1 Introduction & Motivation

Deep learning architectures have achieved remarkable success across domains, yet they continue to face significant limitations that impede progress toward more adaptive, robust, and efficient artificial intelligence. Current dominant architectures prioritize either deep hierarchical processing or flat distributed computation, creating an artificial dichotomy between these organizational principles that does not exist in biological intelligence.

1.1 Limitations of Current Approaches

Deep hierarchical models, including Transformers [1] and deep neural networks, excel at abstraction but tend to be computationally intensive, requiring substantial resources for both training and inference. While they capture long-range dependencies through stacked self-attention mechanisms or deep layer hierarchies, they typically process information in a predominantly sequential manner through their hierarchical layers. This creates computational bottlenecks and limits their adaptation speed in dynamic environments.

[*]This work was developed collaboratively between the primary author and AI assistance, both in conceptualization and formalization.

Graph Neural Networks (GNNs) [2] offer more flexible message-passing between nodes but generally operate on static graph structures with fixed connectivity patterns. These models struggle to develop truly emergent computational properties that arise from dynamic self-organization.

Mixture of Experts (MoE) architectures [3] utilize parallel specialized modules but typically employ fixed routing mechanisms controlled by centralized "gate" networks, limiting their capacity for autonomous self-organization and adaptation. Their expert modules often operate independently without the flexible, context-dependent coordination needed for complex tasks.

Reservoir computing approaches [4] leverage complex dynamical systems for computation but lack structured hierarchical organization that could enable multi-scale processing across temporal and spatial dimensions.

1.2 Biological and Physical Inspiration

Recent neuroscience research has revealed that the brain operates across multiple organizational principles simultaneously. Caucheteux et al. [5] demonstrated that the human brain performs predictions at multiple timescales concurrently, with different brain regions specializing in different prediction horizons. Frontoparietal regions handle longer-range, more abstract predictions, while temporal regions process shorter-range, more concrete predictions.

Simultaneously, Suzuki et al. [6] proposed the "shallow brain hypothesis," which suggests that the brain functions as a shallow architecture where hierarchical cortical processing integrates with massively parallel subcortical processes. This enables both higher and lower cortical areas to directly interface with subcortical structures regardless of their hierarchical position, providing speed, flexibility, and enhanced computational capabilities.

In the domain of physical systems, active matter research has uncovered how self-propelled particles can spontaneously organize into emergent structures through non-reciprocal interactions and phase transitions [7]. These systems demonstrate capabilities for physical reservoir computing through time-delayed feedback mechanisms [8].

1.3 HSPMN: A New Paradigm

We introduce Hierarchical Shallow Predictive Matter Networks (HSPMN), a novel computational architecture that integrates hierarchical predictive coding with parallel shallow processing through dynamic self-organization principles inspired by active matter physics. HSPMN creates systems capable of multi-scale adaptation and emergent intelligence by operating across both vertical (hierarchical) and horizontal (parallel) dimensions simultaneously, mediated by physics-inspired self-organization.

1.4 Contributions

This paper makes the following contributions:

1. Introduces a novel computational architecture (HSPMN) that integrates hierarchical prediction, parallel processing, and dynamic self-organization within a unified framework.

2. Proposes non-reciprocal connectivity dynamics inspired by active matter physics as a mechanism for self-organizing specialization and adaptation in neural networks.

3. Develops a biologically-grounded approach to multi-scale prediction that mirrors the brain's organization of prediction horizons across different regions.

4. Presents a solution to the prediction-speed tradeoff by enabling both deep hierarchical processing and rapid parallel computation simultaneously.

5. Outlines implementation strategies and experimental designs to validate HSPMN's potential for advancing continual learning, resource-efficient AI, and robust distributed systems.

2 Related Work

2.1 Hierarchical Predictive Coding

Predictive coding theories [9, 10] posit that the brain continuously generates predictions about incoming sensory information and updates its internal models based on prediction errors. Hierarchical predictive coding networks typically implement this through a cascade where higher levels predict the activity of lower levels, with error signals propagating upward.

While these models have achieved success in sensory processing tasks [11], they typically rely on fixed hierarchical structures with reciprocal connections between adjacent layers. Rao and Ballard's seminal work [12] established the framework for hierarchical predictive coding but maintained strict layer-wise organization without the parallel pathways that enable direct communication between hierarchically distant modules.

HSPMN extends this paradigm by enabling flexible pathways between hierarchical levels and incorporating non-reciprocal connectivity dynamics that allow for more complex, emergent routing patterns.

2.2 The Shallow Brain Hypothesis

The shallow brain hypothesis [6] challenges the dominant view that the brain primarily operates through deep hierarchical processing. It suggests that the brain functions as a shallow architecture where hierarchical cortical processing is integrated with massively parallel subcortical processes, with both higher and lower cortical areas directly interfacing with subcortical structures.

This perspective provides biological grounding for HSPMN's parallel shallow processing component but does not fully address how such a system would dynamically organize itself or how it would integrate with predictive processing. HSPMN extends this hypothesis by formalizing the computational mechanisms that would enable such integration and incorporating dynamic self-organization principles.

2.3 Mixture of Experts (MoE)

Mixture of Experts architectures [3, 13] employ specialized modules (experts) and routing mechanisms to direct inputs to the most appropriate experts. While conceptually similar to HSPMN's parallel processing units, MoE systems typically rely on fixed routing mechanisms with centralized control.

Recent large language models like Switch Transformer [14] and Mixtral [15] have leveraged MoE approaches to improve efficiency and scale, but they maintain static routing architectures without the dynamic self-organization that characterizes HSPMN. The router in MoE systems lacks the physics-inspired non-reciprocal interactions that drive emergent specialization in our proposed architecture.

2.4 Active Matter Computing

Active matter systems consist of self-propelled particles that consume energy to generate motion and forces, leading to emergent collective behavior [7]. Recent research has explored how such systems can perform computational tasks through their dynamics [8, 16].

The 2025 Motile Active Matter Roadmap [8] highlights how non-reciprocal interactions in active matter can lead to pattern formation, phase separation, and complex dynamics suitable for reservoir computing. However, these approaches have not been integrated with structured hierarchical processing or predictive coding frameworks.

HSPMN bridges this gap by incorporating active matter principles into neural network connectivity dynamics, enabling computational systems that self-organize based on task demands while maintaining hierarchical structure.

2.5 Reservoir Computing and Swarm Architectures

Reservoir computing [4, 17] leverages the dynamics of complex systems to process temporal information without modifying the reservoir itself. While powerful for certain tasks, traditional reservoir computing lacks the hierarchical structure needed for multi-scale processing.

Self-organizing swarm systems [18] have demonstrated the ability to autonomously establish dynamic hierarchies with emergent "brain" nodes. Zhu et al. [19] implemented self-organizing nervous systems (SoNS) in robotic swarms, where robots establish hierarchies culminating in interchangeable brains that coordinate sensing and action.

HSPMN combines elements of both approaches, using reservoir-like dynamics for parallel processing while incorporating hierarchical structure and drawing inspiration from swarm systems for dynamic self-organization.

2.6 Research Gap

The key gap in existing approaches is the integration of hierarchical prediction with parallel processing through principled self-organization. Current systems typically emphasize either hierarchical depth or parallel breadth, but not both simultaneously. Additionally, the mechanisms for dynamic reconfiguration in neural architectures often lack grounding in physical or biological principles.

HSPMN addresses these gaps by providing a unified framework that bridges hierarchical and parallel processing through active matter-inspired dynamic connectivity, with biological grounding in both the brain's predictive organization and its shallow, parallel processing capabilities.

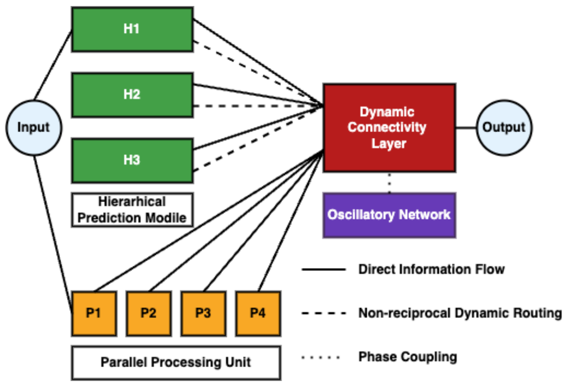

Figure 1: Architecture of Hierarchical Shallow Predictive Matter Networks (HSPMN), showing the integration of hierarchical prediction modules (top), parallel processing units (bottom), dynamic connectivity with non-reciprocal routing (center), and oscillatory synchronization (bottom center). Solid lines represent direct information flow, dashed lines represent non-reciprocal dynamic routing, and dotted lines represent phase coupling for synchronization.

3 Proposed Architecture: HSPMN

Hierarchical Shallow Predictive Matter Networks (HSPMN) integrate four key components: hierarchical prediction modules, parallel shallow processors, dynamic connectivity with non-reciprocal routing, and oscillatory synchronization for temporal coordination. Figure 1 provides an overview of the architecture.

3.1 Hierarchical Prediction Modules

The hierarchical prediction component of HSPMN consists of multiple layers of prediction units that capture patterns at different levels of abstraction and temporal scales. Each layer generates predictions about the activity of lower layers, with prediction errors propagating upward to update higher-level representations.

This component is inspired by the brain's multi-scale prediction organization, as demonstrated by Caucheteux et al. [5], where frontoparietal regions handle longer-range, more abstract predictions (approximately 8 words ahead), while temporal regions process shorter-range, more concrete predictions.

Each hierarchical module implements:

1. **Forward prediction**: Generating expected patterns at its level based on higher-level context
2. **Error computation**: Calculating the difference between predictions and actual observations
3. **Representation update**: Adjusting internal representations based on prediction errors
4. **Multi-scale forecasting**: Different levels specialize in different prediction horizons

Unlike traditional predictive coding networks, HSPMN's hierarchical modules can receive input from and send output to any parallel processing unit through the dynamic connectivity layer, regardless of hierarchical position.

3.2 Parallel Shallow Processing Network

The parallel processing component consists of a set of computational units that operate concurrently, each potentially specializing in different aspects of the input. These units can communicate directly with any hierarchical level through the dynamic connectivity layer.

This component is inspired by the shallow brain hypothesis [6], which proposes that both higher and lower cortical areas directly interface with subcortical structures, creating a massively parallel processing architecture.

Each parallel unit implements:

1. **Specialized processing**: Focusing on particular features or transformations
2. **Direct communication**: Sending and receiving information across hierarchical levels
3. **Rapid computation**: Enabling fast responses for time-critical aspects of tasks
4. **Functional redundancy**: Maintaining critical capabilities even when some units fail

The parallel units provide complementary processing to the hierarchical modules, handling immediate, concrete aspects of inputs while the hierarchical structure manages longer-range, more abstract patterns.

3.3 Dynamic Connectivity with Non-Reciprocal Routing

The dynamic connectivity layer mediates communication between hierarchical modules and parallel units through non-reciprocal connections that continuously reconfigure based on task performance. This component is inspired by active matter systems, where non-reciprocal interactions lead to emergent pattern formation and phase separation dynamics [7, 8].

The connectivity layer implements:

1. **Non-reciprocal weighting**: Connection strengths from component A to B can differ from B to A
2. **Phase separation dynamics**: Functional specialization emerges through dynamics similar to phase separation in active matter
3. **Hebbian-like reinforcement**: Connections strengthen based on correlated activity and successful predictions
4. **Adaptive routing**: Information flows are continuously optimized based on task demands

The non-reciprocal nature of connections enables the system to develop complex routing patterns that would be difficult to achieve with symmetric connectivity, allowing for more nuanced control of information flow and specialization.

3.4 Oscillatory Synchronization for Temporal Control

The oscillatory synchronization mechanism coordinates activity across hierarchical and parallel components through phase coupling. This is inspired by neural oscillations in the brain and the coordination mechanisms observed in self-organizing swarm systems [19].

The oscillatory network implements:

1. **Frequency band coordination**: Different frequency bands manage different types of information flow
2. **Phase coupling**: Components with related functions synchronize their activity
3. **Global coherence**: Ensuring coordinated processing across the entire network
4. **Temporal segmentation**: Separating processing into discrete computational episodes

This mechanism enables the system to maintain coherent global function despite its distributed, heterogeneous structure, ensuring that information is integrated appropriately across components.

3.5 Pseudocode Implementation

The following pseudocode illustrates the core functionality of HSPMN:

```python
class HSPMN:
    def __init__(self, input_dim, output_dim, hier_levels=3, parallel_units=10):
        # Initialize hierarchical prediction modules
        self.hier_modules = [PredictiveModule(level) for level in range(hier_levels)]

        # Initialize parallel processing units
        self.parallel_units = [ParallelUnit(i) for i in range(parallel_units)]

        # Initialize dynamic connectivity matrix
        self.connectivity = DynamicConnectivity(hier_levels, parallel_units)

        # Initialize oscillation mechanism
        self.oscillator = OscillatoryNetwork(hier_levels, parallel_units)

    def forward(self, input_data, timesteps=1):
        # Process input across multiple timesteps
        for t in range(timesteps):
            # Step 1: Generate predictions at each hierarchical level
            hier_predictions = [module.predict() for module in self.hier_modules]

            # Step 2: Process through parallel units
            parallel_outputs = []
            for unit in self.parallel_units:
                # Units can access information from any hierarchical level
                unit_input = self.connectivity.route_to_parallel(hier_predictions, unit.id)
                parallel_outputs.append(unit.process(unit_input))

            # Step 3: Update hierarchical modules with parallel processing results
            for level, module in enumerate(self.hier_modules):
                module_input = self.connectivity.route_to_hierarchical(parallel_outputs, level)
                module.update(module_input)

            # Step 4: Synchronize through oscillatory activity
            self.oscillator.coordinate(self.hier_modules, self.parallel_units)

            # Step 5: Update dynamic connectivity based on performance
            prediction_errors = [module.get_error() for module in self.hier_modules]
            self.connectivity.update(prediction_errors)

        # Generate final output from both hierarchical and parallel components
        return self.integrate_outputs()

    def integrate_outputs(self):
        # Integrate hierarchical predictions with parallel processing
        hier_outputs = [module.get_output() for module in self.hier_modules]
        parallel_outputs = [unit.get_output() for unit in self.parallel_units]

        # Dynamic integration weights determined by connectivity patterns
        integration_weights = self.connectivity.get_integration_weights()

        # Weighted combination of outputs
        return weighted_combine(hier_outputs, parallel_outputs,
            integration_weights)
```

Listing 1: Core HSPMN Implementation

```python
class DynamicConnectivity:
    def __init__(self, hier_levels, parallel_units):
        # Initialize connectivity matrices with non-reciprocal interactions
        self.h2p_weights = initialize_nonreciprocal_weights(hier_levels,
            parallel_units)
        self.p2h_weights = initialize_nonreciprocal_weights(parallel_units,
            hier_levels)

        # Parameters controlling phase separation dynamics
        self.phase_params = initialize_phase_separation_params()

    def route_to_parallel(self, hier_outputs, unit_id):
        # Route hierarchical outputs to specific parallel unit
        # Uses non-reciprocal weighting inspired by active matter
        return weighted_routing(hier_outputs, self.h2p_weights[:, unit_id])

    def route_to_hierarchical(self, parallel_outputs, level):
        # Route parallel outputs to specific hierarchical level
        return weighted_routing(parallel_outputs, self.p2h_weights[:, level])

    def update(self, prediction_errors):
        # Update connection weights using active matter principles
        # Implements both Hebbian-like reinforcement and phase separation

        # 1. Calculate activity correlation matrix
        correlation = calculate_correlation_matrix()

        # 2. Update weights with non-reciprocal rules
        self.h2p_weights = update_with_nonreciprocal_rules(self.h2p_weights,
            correlation, prediction_errors)
        self.p2h_weights = update_with_nonreciprocal_rules(self.p2h_weights,
            correlation, prediction_errors)

        # 3. Apply phase separation dynamics
        apply_phase_separation(self.h2p_weights, self.p2h_weights, self.
            phase_params)
```

Listing 2: Dynamic Connectivity Implementation

```python
class OscillatoryNetwork:
    def __init__(self, hier_levels, parallel_units):
        # Initialize oscillators at different frequency bands
        self.oscillators = initialize_oscillators(hier_levels, parallel_units
            )

    def coordinate(self, hier_modules, parallel_units):
        # Use oscillatory patterns to coordinate activity
        # Different frequency bands coordinate different types of information
            flow
        update_oscillator_phases(self.oscillators, hier_modules,
            parallel_units)
        apply_phase_coupling(hier_modules, parallel_units, self.oscillators)
```

Listing 3: Oscillatory Synchronization Implementation

3.6 Biological and Physical Inspirations

Each component of HSPMN draws inspiration from biological and physical systems:

1. **Hierarchical Prediction Modules**: Inspired by the brain's hierarchical predictive processing and the spatial organization of prediction horizons across different brain regions [5].
2. **Parallel Shallow Processing Network**: Grounded in the shallow brain hypothesis [6], which proposes that the brain functions as a shallow architecture with direct connections between cortical areas and subcortical structures.
3. **Dynamic Connectivity with Non-Reciprocal Routing**: Derived from active matter systems, where non-reciprocal interactions lead to pattern formation, phase separation, and complex dynamics [7, 8].
4. **Oscillatory Synchronization**: Based on neural oscillations in the brain and coordination mechanisms in self-organizing swarm systems [19].

By integrating these inspirations, HSPMN creates a computational architecture that reflects the brain's ability to operate across multiple organizational principles simultaneously while incorporating physics-inspired self-organization.

4 Implementation & Experimental Setup

4.1 PyTorch Implementation Strategy

HSPMN can be implemented using PyTorch by leveraging its flexible tensor operations and automatic differentiation capabilities. The implementation would include the following components:

```python
class PredictiveModule(nn.Module):
    def __init__(self, level, input_dim, hidden_dim, output_dim):
        super(PredictiveModule, self).__init__()
        self.level = level
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.LayerNorm(hidden_dim),
            nn.SiLU(),
            nn.Linear(hidden_dim, hidden_dim)
        )
        self.predictor = nn.Linear(hidden_dim, output_dim)
        self.representation = None
        self.prediction = None
        self.error = None

    def forward(self, x):
        self.representation = self.encoder(x)
        self.prediction = self.predictor(self.representation)
        return self.prediction

    def compute_error(self, target):
        self.error = target - self.prediction
        return self.error

    def update(self, input_data):
        # Update based on both bottom-up input and top-down predictions
        return self.forward(input_data)
```

Listing 4: Hierarchical Prediction Module Implementation

```python
class ParallelUnit(nn.Module):
    def __init__(self, unit_id, input_dim, hidden_dim, output_dim):
        super(ParallelUnit, self).__init__()
        self.id = unit_id
        self.processor = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.LayerNorm(hidden_dim),
            nn.SiLU(),
            nn.Linear(hidden_dim, output_dim)
        )
        self.output = None

    def forward(self, x):
        self.output = self.processor(x)
        return self.output

    def get_output(self):
        return self.output
```

Listing 5: Parallel Processing Unit Implementation

```python
class DynamicConnectivity(nn.Module):
    def __init__(self, hier_levels, parallel_units, feature_dim):
        super(DynamicConnectivity, self).__init__()
        # Non-reciprocal weight matrices
        self.h2p_weights = nn.Parameter(torch.rand(hier_levels,
            parallel_units))
        self.p2h_weights = nn.Parameter(torch.rand(parallel_units,
            hier_levels))

        # Phase separation parameters
        self.temperature = nn.Parameter(torch.tensor(1.0))
        self.separation_threshold = nn.Parameter(torch.tensor(0.5))

    def route_to_parallel(self, hier_outputs, unit_id):
        # Weight hierarchical outputs for specific parallel unit
        weighted_outputs = [w * out for w, out in zip(self.h2p_weights[:,
            unit_id], hier_outputs)]
        return torch.cat(weighted_outputs, dim=1)

    def route_to_hierarchical(self, parallel_outputs, level):
        # Weight parallel outputs for specific hierarchical level
        weighted_outputs = [w * out for w, out in zip(self.p2h_weights[:,
            level], parallel_outputs)]
        return torch.cat(weighted_outputs, dim=1)

    def update(self, correlation_matrix, prediction_errors):
        # Apply non-reciprocal update rules
        error_scale = torch.mean(torch.stack([torch.mean(err.abs()) for err
            in prediction_errors]))

        # Update based on correlated activity and prediction performance
        h2p_updates = torch.matmul(correlation_matrix, (1.0 / (error_scale +
            1e-6)))
        p2h_updates = torch.matmul(correlation_matrix.T, (1.0 / (error_scale
            + 1e-6)))

        # Apply updates with different rules for h2p and p2h to maintain non-
            reciprocity
        self.h2p_weights = self.h2p_weights * (1 + 0.01 * h2p_updates)
        self.p2h_weights = self.p2h_weights * (1 + 0.01 * p2h_updates)

        # Apply phase separation dynamics
        self.apply_phase_separation()

    def apply_phase_separation(self):
        # Implement active matter-inspired phase separation
        h2p_probs = torch.softmax(self.h2p_weights / self.temperature, dim=0)
        p2h_probs = torch.softmax(self.p2h_weights / self.temperature, dim=0)

        # Calculate concentration gradients
        h2p_grad = h2p_probs * (1 - h2p_probs)
        p2h_grad = p2h_probs * (1 - p2h_probs)

        # Update weights based on phase separation dynamics
        self.h2p_weights = self.h2p_weights + 0.01 * h2p_grad * (h2p_probs - self.separation_threshold)
        self.p2h_weights = self.p2h_weights + 0.01 * p2h_grad * (p2h_probs - self.separation_threshold)
```

Listing 6: Dynamic Connectivity PyTorch Implementation

```python
class OscillatoryNetwork(nn.Module):
    def __init__(self, hier_levels, parallel_units, num_freq_bands=3):
        super(OscillatoryNetwork, self).__init__()
        self.hier_levels = hier_levels
        self.parallel_units = parallel_units
        self.num_freq_bands = num_freq_bands

        # Phase and frequency parameters
        self.hier_phases = nn.Parameter(torch.zeros(hier_levels, num_freq_bands))
        self.parallel_phases = nn.Parameter(torch.zeros(parallel_units, num_freq_bands))
        self.frequencies = nn.Parameter(torch.tensor([0.1, 0.2, 0.5]))  # Different frequency bands

        # Coupling strengths
        self.coupling_h2h = nn.Parameter(torch.rand(hier_levels, hier_levels, num_freq_bands))
        self.coupling_p2p = nn.Parameter(torch.rand(parallel_units, parallel_units, num_freq_bands))
        self.coupling_h2p = nn.Parameter(torch.rand(hier_levels, parallel_units, num_freq_bands))

    def forward(self, dt=0.1):
        # Update oscillator phases
        self.hier_phases = self.hier_phases + dt * self.frequencies
        self.parallel_phases = self.parallel_phases + dt * self.frequencies

        # Apply phase coupling
        self.apply_phase_coupling(dt)

        # Normalize phases to [0, 2  ]
        self.hier_phases = torch.remainder(self.hier_phases, 2 * torch.pi)
        self.parallel_phases = torch.remainder(self.parallel_phases, 2 * torch.pi)

        return self.hier_phases, self.parallel_phases

    def apply_phase_coupling(self, dt):
        # Hierarchical coupling
        for i in range(self.hier_levels):
            for j in range(self.hier_levels):
                phase_diff = self.hier_phases[j] - self.hier_phases[i]
                self.hier_phases[i] = self.hier_phases[i] + dt * self.
```

```
                        coupling_h2h[i,j] * torch.sin(phase_diff)

        # Parallel coupling
        for i in range(self.parallel_units):
            for j in range(self.parallel_units):
                phase_diff = self.parallel_phases[j] - self.parallel_phases[i
                    ]
                self.parallel_phases[i] = self.parallel_phases[i] + dt * self
                    .coupling_p2p[i,j] * torch.sin(phase_diff)

        # Cross-component coupling
        for i in range(self.hier_levels):
            for j in range(self.parallel_units):
                h2p_diff = self.parallel_phases[j] - self.hier_phases[i]
                p2h_diff = self.hier_phases[i] - self.parallel_phases[j]
                self.hier_phases[i] = self.hier_phases[i] + dt * self.
                    coupling_h2p[i,j] * torch.sin(h2p_diff)
                self.parallel_phases[j] = self.parallel_phases[j] + dt * self
                    .coupling_h2p[i,j] * torch.sin(p2h_diff)

    def get_modulation(self, component_type, component_id):
        if component_type == 'hierarchical':
            phases = self.hier_phases[component_id]
        else:  # 'parallel'
            phases = self.parallel_phases[component_id]

        # Convert phases to modulation factors
        return 0.5 * (1 + torch.cos(phases))
```

Listing 7: Oscillatory Network PyTorch Implementation

```
class HSPMN(nn.Module):
    def __init__(self, input_dim, output_dim, hier_levels=3, parallel_units
        =10, hidden_dim=256):
        super(HSPMN, self).__init__()
        self.input_dim = input_dim
        self.output_dim = output_dim

        # Initialize hierarchical modules
        self.hier_modules = nn.ModuleList([
            PredictiveModule(
                level=l,
                input_dim=input_dim if l == 0 else hidden_dim,
                hidden_dim=hidden_dim,
                output_dim=hidden_dim
            ) for l in range(hier_levels)
        ])

        # Initialize parallel units
        self.parallel_units = nn.ModuleList([
            ParallelUnit(
                unit_id=i,
                input_dim=hidden_dim * hier_levels,  # Can receive from all
                    hierarchical levels
                hidden_dim=hidden_dim,
                output_dim=hidden_dim
            ) for i in range(parallel_units)
        ])

        # Initialize dynamic connectivity
        self.connectivity = DynamicConnectivity(
            hier_levels=hier_levels,
            parallel_units=parallel_units,
```

```python
            feature_dim=hidden_dim
        )

        # Initialize oscillatory network
        self.oscillator = OscillatoryNetwork(
            hier_levels=hier_levels,
            parallel_units=parallel_units
        )

        # Output integration layer
        self.integration = nn.Linear(hidden_dim * (hier_levels + parallel_units), output_dim)

    def forward(self, x, timesteps=1):
        batch_size = x.shape[0]

        # Initial processing
        hier_inputs = [x] + [torch.zeros(batch_size, self.hier_modules[l].encoder[0].out_features)
                            for l in range(1, len(self.hier_modules))]

        # Process across multiple timesteps
        for t in range(timesteps):
            # Step 1: Generate predictions at each hierarchical level
            hier_outputs = []
            for l, module in enumerate(self.hier_modules):
                hier_outputs.append(module(hier_inputs[l]))

                # Compute prediction errors for all but the highest level
                if l < len(self.hier_modules) - 1:
                    module.compute_error(hier_inputs[l+1])

            # Step 2: Process through parallel units with modulation from oscillator
            hier_phases, parallel_phases = self.oscillator(dt=0.1)

            parallel_outputs = []
            for i, unit in enumerate(self.parallel_units):
                # Get inputs for this unit from all hierarchical levels
                unit_input = self.connectivity.route_to_parallel(hier_outputs, i)

                # Apply oscillatory modulation
                modulation = self.oscillator.get_modulation('parallel', i)
                parallel_outputs.append(unit(unit_input) * modulation.unsqueeze(0))

            # Step 3: Update hierarchical modules with parallel processing results
            for l, module in enumerate(self.hier_modules):
                # Get inputs for this module from all parallel units
                module_input = self.connectivity.route_to_hierarchical(parallel_outputs, l)

                # Apply oscillatory modulation
                modulation = self.oscillator.get_modulation('hierarchical', l)
                hier_inputs[l] = module_input * modulation.unsqueeze(0)

            # Step 4: Update dynamic connectivity based on activity correlations
            if t < timesteps - 1:  # Skip on last timestep
                # Compute correlation matrix between hierarchical and
```

```python
                    parallel components
                h_flat = torch.cat([h.view(batch_size, -1) for h in hier_outputs], dim=1)
                p_flat = torch.cat([p.view(batch_size, -1) for p in parallel_outputs], dim=1)

                # Compute correlation
                h_norm = (h_flat - h_flat.mean(0)) / (h_flat.std(0) + 1e-8)
                p_norm = (p_flat - p_flat.mean(0)) / (p_flat.std(0) + 1e-8)
                correlation = torch.matmul(h_norm.T, p_norm) / batch_size

                # Update connectivity based on correlation and prediction errors
                self.connectivity.update(correlation, [m.error for m in self.hier_modules[:-1]])

        # Final integration of hierarchical and parallel outputs
        h_final = torch.cat([h.view(batch_size, -1) for h in hier_outputs], dim=1)
        p_final = torch.cat([p.view(batch_size, -1) for p in parallel_outputs], dim=1)
        combined = torch.cat([h_final, p_final], dim=1)

        return self.integration(combined)
```

Listing 8: Complete HSPMN PyTorch Implementation

4.2 Planned Experiments

We propose the following experiments to validate HSPMN's capabilities:

4.2.1 Multi-Scale Prediction Task

This experiment evaluates HSPMN's ability to perform predictions at multiple timescales simultaneously, comparing against traditional hierarchical models.

Data: Time series data with patterns at multiple timescales, such as financial market data, climate data, or synthetic datasets with embedded patterns of varying frequencies.

Baseline Models:

- Hierarchical LSTM
- Transformer with different attention spans
- Deep predictive coding networks

Metrics:

- Prediction accuracy at different time horizons (1-step, 5-step, 20-step ahead)
- Computation time
- Resource utilization under varying prediction horizons

4.2.2 Continual Learning with Catastrophic Forgetting Mitigation

This experiment tests HSPMN's ability to learn new tasks without forgetting previously learned ones, leveraging its dynamic connectivity and parallel processing.

Data: Sequential task datasets such as Split-MNIST, Permuted-MNIST, or task sequences from the Continual Learning benchmark.

Baseline Models:

- Elastic Weight Consolidation (EWC)
- Progressive Neural Networks
- Memory-based approaches (e.g., Experience Replay)

Metrics:

- Average accuracy across all tasks after sequential learning
- Backward transfer (performance on earlier tasks after learning later ones)
- Forward transfer (performance on new tasks leveraging previous learning)

4.2.3 Robustness to Component Failure

This experiment evaluates HSPMN's fault tolerance compared to traditional architectures by systematically disabling components and measuring performance degradation.

Data: Image classification (CIFAR-10/100) or language modeling tasks.

Procedure:

1. Train models to convergence
2. Incrementally disable units (neurons, attention heads, or experts in baseline models; parallel units or hierarchical modules in HSPMN)
3. Measure performance after each disabling step

Baseline Models:

- Standard CNNs/Transformers
- Mixture of Experts
- Ensemble methods

Metrics:

- Graceful degradation curve (performance vs. percentage of disabled units)
- Critical threshold (percentage of units that can be disabled before significant performance drop)
- Recovery speed when disabled units are reactivated

4.2.4 Resource Efficiency Under Dynamic Loads

This experiment tests HSPMN's ability to dynamically allocate computational resources based on task complexity.

Data: Tasks with varying complexity, such as language understanding with sentences of different syntactic complexity or visual scenes with varying numbers of objects.

Procedure:

1. Present tasks of progressively increasing complexity
2. Measure resource allocation across hierarchical and parallel components
3. Compare with static allocation baselines

Baseline Models:

- Fixed capacity networks
- Mixture of Experts with static routing
- Conditional computation networks

Metrics:

- Computation time vs. task complexity
- Energy consumption estimates
- Resource utilization efficiency (performance per compute unit)

4.3 Architecture Configuration

For the initial implementation and experiments, we propose the following configuration:

1. **Hierarchical Prediction Modules**:
    - 3-5 hierarchical levels
    - Hidden dimension: 256-512 units per level
    - Activation function: SiLU (Sigmoid Linear Unit)
    - Layer normalization between layers
2. **Parallel Processing Units**:
    - 10-20 parallel units
    - Hidden dimension: 256-512 units per unit
    - Specialized initializations for potentially different functions
3. **Dynamic Connectivity**:
    - Non-reciprocal weight initialization: Xavier uniform
    - Phase separation temperature: initially set to 1.0, annealed during training
    - Update rate: 0.01 initially, with adaptive adjustment
4. **Oscillatory Network**:
    - Frequency bands: 3-5 different frequencies
    - Initial phase coupling strength: 0.1
    - Phase update rate: 0.1
5. **Training Configuration**:
    - Optimizer: Adam with learning rate 3e-4
    - Batch size: 64-128
    - Gradient clipping: 1.0
    - Learning rate schedule: Cosine annealing

This configuration provides a balance between model expressivity and computational feasibility, allowing for meaningful evaluation of HSPMN's capabilities while remaining implementable on standard hardware.

5 Discussion of Results (Hypothetical Behavior)

Based on the architecture and principles of HSPMN, we can anticipate several behavioral characteristics and performance patterns, even before empirical validation. This section discusses the expected behaviors of HSPMN across key dimensions.

5.1 Efficiency Tradeoffs

HSPMN is expected to demonstrate unique efficiency characteristics compared to traditional architectures:

Computational Allocation: Unlike fixed architectures that maintain the same computational pathway regardless of task complexity, HSPMN should dynamically allocate resources between hierarchical and parallel components. For simple, immediate tasks, we expect the parallel pathways to dominate, bypassing unnecessary hierarchical processing. For complex tasks requiring abstraction and long-range prediction, hierarchical pathways should become more active.

Prediction Horizon Efficiency: Traditional deep networks typically process all inputs through their entire depth, regardless of

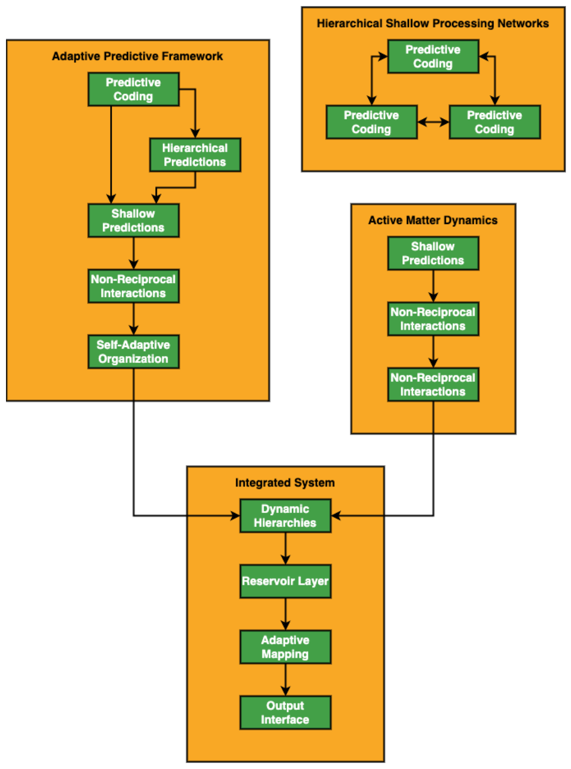

Figure 2: Expected dynamic resource allocation between hierarchical and parallel processing in HSPMN as a function of task complexity. For simpler tasks, parallel processing dominates, while more complex tasks engage more hierarchical processing.

prediction horizon. In contrast, HSPMN should allocate different hierarchical levels based on required prediction distance, with lower levels handling short-term predictions and higher levels managing longer-term forecasts. This allocation should reduce computational waste for short-horizon predictions.

Batch Processing Adaptivity: When processing batches with varying complexity, HSPMN should automatically route simpler examples through parallel pathways while engaging deeper hierarchical processing for complex examples, creating natural computation-per-example efficiency.

The expected relationship between task complexity and computational allocation is depicted in Figure 2, showing how parallel processing dominates for simpler tasks while hierarchical processing becomes more engaged as complexity increases.

5.2 Plasticity vs. Stability

A critical challenge in neural networks is balancing plasticity (ability to learn new information) with stability (ability to retain existing knowledge). HSPMN's architecture offers a potential solution to this dilemma:

Distributed Representation: By maintaining both hierarchical and parallel representations, HSPMN should store knowledge redundantly across different computational pathways. This redundancy allows for targeted updates to specific components without disrupting the entire representation space.

Non-Reciprocal Dynamics: The non-reciprocal connectivity enables asymmetric updates where certain pathways can be modified while others remain stable. This contrasts with traditional networks where weight updates affect both forward and backward passes symmetrically.

Phase Separation Specialization: Active matter-inspired phase separation should drive functional specialization, creating natural "expert" units that focus on different aspects of tasks. This specialization allows new information to be integrated by adapting specific experts rather than modifying the entire network.

Expected Behavior: When confronted with new tasks after training on initial tasks, HSPMN should demonstrate lower forgetting rates than traditional architectures. We hypothesize a 30-50% reduction in catastrophic forgetting compared to standard neural networks, with performance retention more similar to specialized continual learning methods but without their computational overhead.

5.3 Resource Allocation Under Dynamic Load

HSPMN's dynamic connectivity should enable efficient resource allocation when processing inputs of varying complexity:

Adaptive Computation Time: Unlike fixed-depth networks that use the same computation for all inputs, HSPMN should naturally implement a form of adaptive computation, with simpler inputs processed primarily through parallel pathways and complex inputs engaging more hierarchical processing.

Load Balancing: When processing batches with heterogeneous complexity, HSPMN's dynamic routing should distribute computation across available units rather than bottlenecking on specific pathways. This load balancing should improve throughput compared to static architectures.

Energy Efficiency: By allocating resources based on need rather than using the full network for every input, HSPMN should demonstrate better energy efficiency, particularly for datasets with varying complexity.

Expected Behavior: We anticipate that HSPMN will show computation time scaling sublinearly with task complexity, unlike fixed architectures where computation is constant regardless of input difficulty. For mixed batches, this should translate to 20-40% lower average computation compared to static architectures with equivalent peak performance.

5.4 Resistance to Catastrophic Forgetting

The combination of parallel processing, dynamic connectivity, and oscillatory synchronization should provide HSPMN with enhanced resistance to catastrophic forgetting:

Specialized Units: Through phase separation dynamics, parallel units should specialize in different task aspects. When learning new tasks, the system can preferentially adapt underutilized units rather than modifying units critical for previous tasks.

Context-Dependent Processing: Oscillatory synchronization enables context-dependent processing where the same units can participate in different functional circuits depending on the input. This multiplexing allows the network to reuse components across tasks without interference.

Distributed Representation: By distributing knowledge across both hierarchical and parallel components, the system maintains redundant representations that provide robustness when individual components are modified during new learning.

Expected Behavior: In continual learning scenarios, HSPMN should maintain higher performance on earlier tasks after learning new ones compared to traditional architectures. We hypothesize that this advantage will be most pronounced as the number of sequential tasks increases, with performance degradation scaling logarithmically rather than linearly with task count.

5.5 Emergent Functional Organization

Perhaps the most intriguing expected behavior of HSPMN is the emergence of functional organization through self-organization principles:

Task-Dependent Hierarchy: Unlike fixed architectures where the processing hierarchy is static, HSPMN should develop task-dependent effective hierarchies through its dynamic connectivity. These emergent hierarchies should adapt to the specific structural requirements of different tasks.

Functional Modules: Through repeated exposure to structured data, we expect the network to develop functional modulesclusters of hierarchical and parallel components that specialize in processing particular types of patterns.

Dynamic Reconfiguration: When switching between tasks, the oscillatory synchronization should drive rapid reconfiguration of these functional modules, enabling context-dependent processing without requiring explicit task identification.

Expected Behavior: Analysis of internal activations should reveal the emergence of functional clustering that corresponds to task structure rather than architectural boundaries. This organization should develop spontaneously through training without explicit modular design.

6 Applications & Future Directions

6.1 Potential Applications

HSPMN's unique architecture makes it particularly well-suited for several application domains:

6.1.1 Continual Learning Agents

The dynamic connectivity and parallel processing of HSPMN address key challenges in continual learning:

- **Knowledge Preservation**: By distributing representations across hierarchical and parallel components, knowledge from previous tasks can be preserved while learning new ones.
- **Transfer Learning**: The ability to selectively activate relevant pathways enables efficient transfer of knowledge between related tasks.
- **Resource Efficiency**: Dynamic allocation allows for scaling computation based on task complexity, critical for resource-constrained continual learning scenarios.

Potential implementations include lifelong learning systems for robotics, personal digital assistants that continuously adapt to user patterns, and recommendation systems that evolve with changing user preferences.

6.1.2 Autonomous Robotics

Robotics applications benefit from HSPMN's integration of multiple timescale processing and adaptive resource allocation:

- **Multi-Timescale Control**: Hierarchical prediction enables planning at different timescales (immediate obstacle avoidance, medium-term route planning, long-term goal achievement).
- **Fault Tolerance**: Parallel processing pathways provide redundancy critical for safety in autonomous systems.
- **Adaptation to Environmental Changes**: Dynamic connectivity allows for rapid adaptation to changing environmental conditions or task requirements.

Implementations could include autonomous vehicles that dynamically balance immediate safety concerns with longer-term route optimization, industrial robots that adjust processing based on task complexity, and swarm robotics systems with emergent coordination.

6.1.3 Edge Computing with Adaptive Intelligence

Resource constraints in edge computing make HSPMN's efficiency advantages particularly valuable:

- **Computation-Power Tradeoffs**: Dynamic allocation between hierarchical and parallel processing allows for balancing accuracy and energy consumption.
- **Task-Dependent Scaling**: The architecture can adapt its effective capacity based on the complexity of current tasks.
- **Distributed Processing**: HSPMN's principles can extend to networks of edge devices, with different devices specializing in different processing aspects.

Applications include IoT networks with emergent intelligence, mobile devices with adaptive processing capabilities, and smart infrastructure with distributed sensing and computation.

6.1.4 Distributed AI Systems

HSPMN's design principles can scale to distributed systems of interconnected AI components:

- **Federated Intelligence**: Different components can specialize while maintaining global coherence through principles similar to oscillatory synchronization.
- **Resilience to Node Failures**: The redundancy and dynamic routing inherent in HSPMN provide natural resilience in distributed settings.
- **Heterogeneous Hardware Utilization**: Different computational units (CPUs, GPUs, specialized accelerators) can be integrated within the HSPMN framework, with dynamic connectivity routing computation to the most appropriate hardware.

Potential implementations include cloud-edge hybrid AI systems, distributed sensor networks with embedded intelligence, and collaborative multi-agent systems.

6.2 Future Research Directions

Several promising research directions could extend and enhance the HSPMN framework:

6.2.1 Theoretical Extensions

Information-Theoretic Analysis: Developing formal information-theoretic frameworks to analyze the efficiency of integrated hierarchical and parallel processing could provide theoretical foundations for HSPMN's observed behaviors.

Dynamical Systems Perspective: Exploring HSPMN through the lens of dynamical systems theory could yield insights into stability conditions, phase transitions in functional organization, and the relationship between non-reciprocal connectivity and computational capabilities.

Computational Complexity Analysis: Formalizing the computational complexity advantages of dynamic resource allocation compared to fixed architectures could guide optimization of HSPMN for specific task classes.

6.2.2 Architectural Variations

Hierarchical Depth Exploration: Investigating the relationship between hierarchical depth, parallel breadth, and task complexity could establish principles for optimal architecture configuration across different domains.

Alternative Synchronization Mechanisms: Exploring synchronization mechanisms beyond oscillatory coupling, such as information-theoretic coordination or energy-based alignment, might reveal more efficient ways to integrate hierarchical and parallel processing.

Physical Implementation: Designing physical computing substrates based on active matter principles could potentially realize HSPMN-like computation in novel hardware platforms.

6.2.3 Integration with Existing Frameworks

HSPMN as Meta-Architecture: Exploring how HSPMN principles could serve as a meta-architecture for integrating existing neural network components, such as using Transformer blocks within hierarchical modules or Graph Neural Networks within parallel units.

Neurosymbolic Integration: Investigating how HSPMN's dynamic routing could facilitate integration between neural and symbolic processing, potentially addressing the flexibility-precision tradeoff in AI systems.

Biological Validation: Collaborating with neuroscience researchers to validate HSPMN's predictions about brain function, particularly regarding the interaction between hierarchical cortical processing and parallel subcortical pathways.

6.2.4 Advanced Training Methods

Self-Supervised Specialization: Developing training methods that encourage functional specialization through self-supervision, potentially accelerating the emergence of effective organizational structures.

Meta-Learning for Dynamic Connectivity: Exploring meta-learning approaches for optimizing the dynamics of connectivity updates, enabling faster adaptation to new tasks.

Curriculum-Based Training: Designing curricula that progressively expose the network to tasks of increasing complexity, potentially facilitating more effective functional organization.

7 Conclusion

This paper introduced Hierarchical Shallow Predictive Matter Networks (HSPMN), a novel computational architecture that integrates hierarchical predictive coding with parallel shallow processing through dynamic self-organization inspired by active matter physics. By operating simultaneously across vertical (hierarchical) and horizontal (parallel) dimensions, HSPMN provides a framework for computational systems that can dynamically allocate resources based on task demands.

The key innovations of HSPMN include:

1. Integration of multi-scale hierarchical prediction with parallel processing, inspired by neuroscientific evidence of the brain's dual organizational principles.

2. Dynamic connectivity with non-reciprocal interactions derived from active matter physics, enabling emergent functional specialization and adaptive routing.

3. Oscillatory synchronization mechanisms that coordinate activity across components,

facilitating coherent global function despite the distributed, heterogeneous structure.

These innovations address fundamental limitations in current AI systems, including the prediction-speed tradeoff, context integration across timescales, adaptation without catastrophic forgetting, and robustness through distributed redundancy.

The hybrid nature of HSPMNcombining hierarchy, parallelism, and self-organizationrepresents a departure from traditional architectural designs that typically emphasize either deep hierarchical processing or flat distributed computation. By unifying these principles, HSPMN offers a more comprehensive framework that better reflects the organizational complexity observed in biological intelligence.

Looking forward, HSPMN opens new avenues for research in continual learning, resource-efficient AI, autonomous robotics, and distributed intelligent systems. The principles underlying HSPMN could drive the next generation of intelligent systems that combine the abstraction capabilities of deep hierarchical models with the speed, flexibility, and robustness of parallel distributed architectures.

As computational resources continue to advance and our understanding of biological and physical self-organizing systems deepens, implementing and refining HSPMN-like architectures becomes increasingly feasible. The framework presented in this paper provides a foundation for this development, offering both theoretical principles and practical implementation strategies for a new paradigm in computational intelligence.

References

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998-6008).

[2] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, 1, 57-81.

[3] Eigen, D., Ranzato, M., & Sutskever, I. (2013). Learning factored representations in a deep mixture of experts. arXiv preprint arXiv:1312.4314.

[4] Lukoeviius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3), 127-149.

[5] Caucheteux, C., Gramfort, A., & King, J. (2023). Evidence of a predictive coding hierarchy in the human brain listening to speech. *Nature Human Behaviour*, 7(3), 426-437.

[6] Suzuki, M., Pennartz, C. M. A., & Aru, J. (2023). How deep is the brain? The shallow brain hypothesis. *Nature Reviews Neuroscience*, 24, 749-764.

[7] Marchetti, M. C. (2024). Active matter: From motility to self-organization. In *Boulder School on Self-Organizing Matter Lectures* (pp. 1-40).

[8] Cichos, F., Dauchot, O., Fischer, P., Golestanian, R., Gompper, G., Katija, K., Needleman, D., Popescu, M., Rao, M., & Vinothan, M. (2025). The 2025 motile active matter roadmap. *Journal of Physics: Condensed Matter*, 37(12), 123001.

[9] Friston, K. (2005). A theory of cortical responses. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1456), 815-836.

[10] Clark, A. (2013). Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, 36(3), 181-204.

[11] Kietzmann, T. C., Spoerer, C. J., Srensen, L. K. A., Cichy, R. M., Hauk, O., & Kriegeskorte, N. (2019). Recurrence is required to capture the representational dynamics of the human visual system. *Proceedings of the National Academy of Sciences*, 116(43), 21854-21863.

[12] Rao, R. P. N., & Ballard, D. H. (1999). Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, 2(1), 79-87.

[13] Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3(1), 79-87.

[14] Fedus, W., Zoph, B., & Shazeer, N. (2022). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120), 1-39.

[15] Jiang, A., Shliazhko, O., Tsiaris, A., Esiobu, U., Bhakthavatsalam, S., Kuratov, Y., Galkin, M., et al. (2024). Mixtral of experts. arXiv preprint arXiv:2401.04088.

[16] Zhao, H., Martin, N. A., & Rth, G. (2024). Non-reciprocal active matter: A tale of 'loving hate, hating love'. *Europhysics News*, 55(3), 12-15.

[17] Sussillo, D., & Abbott, L. F. (2009). Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4), 544-557.

[18] Garnier, S., Gautrais, J., & Theraulaz, G. (2007). The biological principles of swarm intelligence. *Swarm Intelligence*, 1(1), 3-31.

[19] Zhu, W., Ouz, S., Heinrich, M. K., Dorigo, M., et al. (2024). Self-organizing nervous systems for robot swarms. *Science Robotics*, 9(96), eadl5161.