Osmotic Learning Experiments

This notebook implements and analyzes several experiments related to the **Osmotic Learning (OSM-L) paradigm**. OSM-L is a self-supervised learning method that enables distributed entities to learn contextual representations without explicit supervision. These experiments aim to validate the latent representation alignment, robustness to noise, and the emergence of dynamic sub-contexts.

In [1]:
```
from simulator.core.agent import Agent
from simulator.core.diffuser import Diffuser
from simulator.misc.utils import set_seed
from simulator.misc.visualization import plot_training_loss, plot_accuracy, plot_embeddings_simi

from models.base import MODEL_REGISTRY
from torch.utils.data import DataLoader, TensorDataset
from datasets.DataProcessor import DataProcessor
import multiprocessing as mp
```

Experiment 1: Latent Representation and Training Convergence

In this experiment, we train two agents (Agent0 and Agent1) in a **simple context**, where both agents have correlated data. The objective is to evaluate whether **local embeddings converge to a shared representation** using OSM-L's diffusion mechanism.

- **Key Hypothesis:** The embeddings of both agents should align, despite training occurring independently.
- **Metric:** Similarity matrix of embeddings over time.
- **Expected Outcome:** If OSM-L works correctly, embeddings generated by Agent0 and Agent1 at the same logical step should be highly similar.

In [2]:
```
# Define parameters for the experiment

WINDOW_SIZE = 10
TRAIN_DATA_CSV = 'datasets/simple-context/train_data.csv'
TEST_DATA_CSV = 'datasets/simple-context/test_data.csv'
EPOCHS = 5
EMBEDDING_SIZE = 5
BATCH_SIZE = 50
CLUSTERING_STEP = 2
GRU_HIDDEN_DIM = 20
DEVICE = 'cpu' # 'cuda' or 'cpu' or 'mps'

set_seed(42)
```

Load and preprocess the dataset

The dataset is loaded from CSV files, and only the first two agents (0 and 1) are retained.

In [3]:
```
dataProcessor = DataProcessor(window_size=WINDOW_SIZE)
data = dataProcessor.load_data_from_csv(TRAIN_DATA_CSV, TEST_DATA_CSV)

# Maintain only first two Agents from Simple Context
data = {
    'train': {
        0: data['train']['0'],
        1: data['train']['1']
    },
    'test': {
        0: data['test']['0'],
        1: data['test']['1']
    }
}
data = dataProcessor.load_data(data['train'], data['test'])
agents_data = dataProcessor.preprocess_data()
```

Plot time series

This visualization helps verify the structure and correlation of the data.

In [4]:
```
dataProcessor.plot_time_series()
```

Initialize agents and communication queues

Each agent is assigned its own queue for sending and receiving messages and processes local data using a GRU-based model.

In [5]:
```
NUM_AGENTS = len(agents_data['train'])
sending_queues = {agent_id: mp.Queue() for agent_id in agents_data['train'].keys()}
receiving_queues = {agent_id: mp.Queue() for agent_id in agents_data['train'].keys()}
results_queue = mp.Queue()

agents = {}
for agent_id, data in agents_data['train'].items():
    dataset = TensorDataset(data['series'])
    dataloader = DataLoader(dataset, batch_size=BATCH_SIZE, shuffle=False)

    dataset = TensorDataset(agents_data['test'][agent_id]['series'])
    testloader = DataLoader(dataset, batch_size=BATCH_SIZE, shuffle=False)
    model = MODEL_REGISTRY["gru"](
        input_dim=data["features"],
        hidden_dim=GRU_HIDDEN_DIM,
        output_dim=EMBEDDING_SIZE
    )
    agent = Agent(
        agent_id=agent_id,
        model=model,
        train_loader=dataloader,
        test_loader=testloader,
        results_queue=results_queue,
        sending_queue=sending_queues[agent_id],
        receiving_queue=receiving_queues[agent_id],
        device=DEVICE
    )
    agents[agent_id] = agent
```

Initialize the Diffuser

The **Diffuser** coordinates the learning process by aligning embeddings across agents.

In [6]:
```
diffuser = Diffuser(
    agents=agents,
    epochs=EPOCHS,
    clustering=True,
    clustering_step=CLUSTERING_STEP,
    device=DEVICE,
)
```

Run the training process

This step allows **local embeddings to converge** while preserving individual data characteristics.

In [7]:
```
local_embeddings, metrics = diffuser.run(evaluate=True, plot=False)
```

```
[DIFFUSER] Global embedding initialized with shape: torch.Size([1, 5])
[DIFFUSER] All agent processes started.
[DIFFUSER] Epoch 1/5
[DIFFUSER] Epoch completed. Epoch loss 0.00561844
[DIFFUSER] Accuracy: 0.9940 - Clustering percentage: 0.0000
[DIFFUSER] Epoch 2/5
[INFO] Clusters: {1: [0, 1]}
[DIFFUSER] Epoch completed. Epoch loss 0.00602923
[DIFFUSER] Accuracy: 0.9979 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 3/5
[DIFFUSER] Epoch completed. Epoch loss 0.00610466
[DIFFUSER] Accuracy: 0.9972 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 4/5
[INFO] Clusters: {1: [0, 1]}
[DIFFUSER] Epoch completed. Epoch loss 0.00467381
[DIFFUSER] Accuracy: 0.9981 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 5/5
[DIFFUSER] Epoch completed. Epoch loss 0.00588214
[DIFFUSER] Accuracy: 0.9984 - Clustering percentage: 1.0000
```

Evaluate the model on the test set

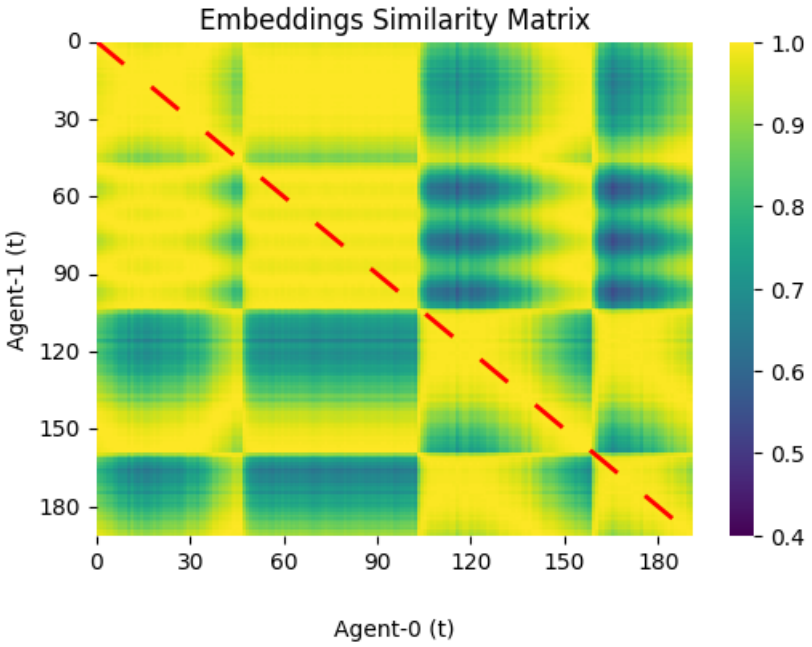
The evaluation measures how well the learned embeddings generalize.

In [8]:
```
evaluate_data = diffuser.evaluate(agents_data['test'])
```

Plot the embeddings similarity matrix

This matrix visualizes how well the embeddings from different agents align over time.

In [9]:
```
plot_embeddings_similarity_matrix(evaluate_data, (6,4), tickevery=30, min=0.4, max=1, cbar=True)
```



Results for Experiment 1: Latent Representation and Training Convergence

The results confirm that **OSM-L successfully aligns local embeddings**, achieving a **0.9984 accuracy at epoch 4**. The similarity matrix shows that embeddings produced by Agent0 and Agent1 at the same logical step **converge to a shared representation**, despite independent training. Notably, in the interval [50, 100], the model correctly identifies a stationary phase, demonstrating its ability to **capture contextual consistency** even when local data exhibit oscillatory behavior.

Stop the agents

After training and evaluation, the agents are stopped to free up resources.

In [10]:
```
diffuser.stop_agents()
```

```
[DIFFUSER] All agent processes stopped.
```

Experiment 2: Robustness of Learning

This experiment introduces **two misleading agents** (AgentM0 and AgentM1) with randomly generated data. The goal is to evaluate OSM-L's **resilience against noise** and assess whether the diffusion process can **preserve alignment** among the correlated agents while ignoring uncorrelated ones.

- **Key Hypothesis:** The embeddings of correlated agents (Agent0 and Agent1) should remain aligned, while misleading agents should not influence the global representation.
- **Metric:** Similarity matrices of all four agents.
- **Expected Outcome:** The embeddings of misleading agents should not exhibit structured similarity patterns, confirming that OSM-L **filters out irrelevant information**.

Reload and preprocess the dataset

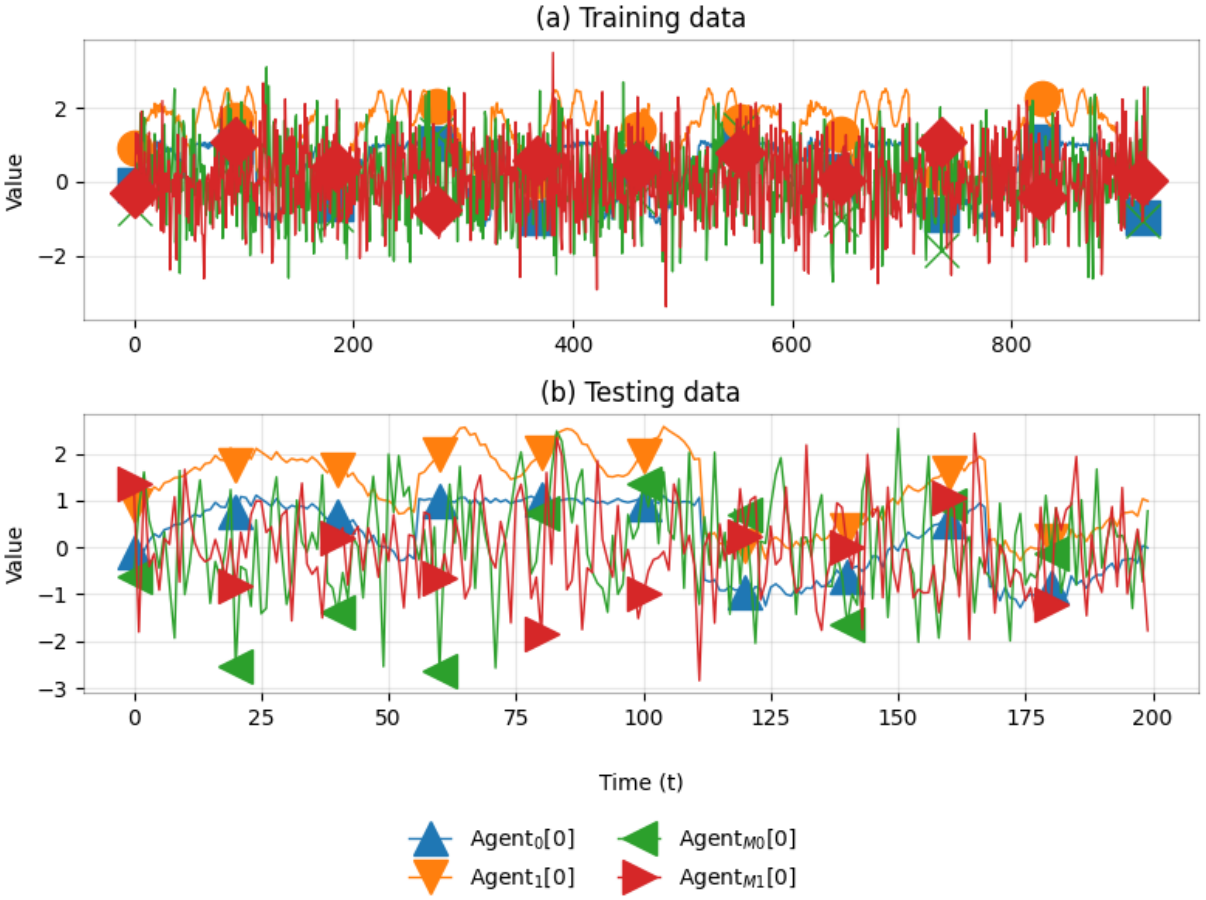
The same dataset is reloaded, this time without filtering any agents.

In [11]:
```
dataProcessor = DataProcessor(window_size=WINDOW_SIZE)
data = dataProcessor.load_data_from_csv(TRAIN_DATA_CSV, TEST_DATA_CSV)
data = dataProcessor.load_data(data['train'], data['test'])
agents_data = dataProcessor.preprocess_data()
```

Plot time series for all agents

This visualization now includes the misleading agents.

In [12]:
```
dataProcessor.plot_time_series()
```



Initialize agents and queues

All four agents (including the misleading ones) are included in this run.

In [13]:
```
NUM_AGENTS = len(agents_data['train'])
sending_queues = {agent_id: mp.Queue() for agent_id in agents_data['train'].keys()}
receiving_queues = {agent_id: mp.Queue() for agent_id in agents_data['train'].keys()}
results_queue = mp.Queue()

agents = {}
for agent_id, data in agents_data['train'].items():
    dataset = TensorDataset(data['series'])
    dataloader = DataLoader(dataset, batch_size=BATCH_SIZE, shuffle=False)

    dataset = TensorDataset(agents_data['test'][agent_id]['series'])
    testloader = DataLoader(dataset, batch_size=BATCH_SIZE, shuffle=False)
    model = MODEL_REGISTRY["gru"](
        input_dim=data["features"],
        hidden_dim=GRU_HIDDEN_DIM,
        output_dim=EMBEDDING_SIZE
    )
    agent = Agent(
        agent_id=agent_id,
        model=model,
        train_loader=dataloader,
        test_loader=testloader,
        results_queue=results_queue,
        sending_queue=sending_queues[agent_id],
        receiving_queue=receiving_queues[agent_id],
        device=DEVICE
    )
    agents[agent_id] = agent
```

Initialize the Diffuser

The **Diffuser** coordinates the learning process by aligning embeddings across agents.

In [14]:
```
diffuser = Diffuser(
    agents=agents,
    epochs=EPOCHS,
    clustering=True,
    clustering_step=CLUSTERING_STEP,
    device=DEVICE,
)
```

Run the training process

This step allows **local embeddings to converge** while preserving individual data characteristics.

In [15]:
```
local_embeddings, metrics = diffuser.run(evaluate=True, plot=False)
```

```
[DIFFUSER] Global embedding initialized with shape: torch.Size([1, 5])
[DIFFUSER] All agent processes started.
[DIFFUSER] Epoch 1/5
[DIFFUSER] Epoch completed. Epoch loss 0.00636557
[DIFFUSER] Accuracy: 0.9611 - Clustering percentage: 0.0000
[DIFFUSER] Epoch 2/5
[INFO] Clusters: {0: ['M0', 'M1'], 1: ['0', '1']}
[DIFFUSER] Epoch completed. Epoch loss 0.00594154
[DIFFUSER] Accuracy: 0.9925 - Clustering percentage: 0.5000
[DIFFUSER] Epoch 3/5
[DIFFUSER] Epoch completed. Epoch loss 0.00602993
[DIFFUSER] Accuracy: 0.9935 - Clustering percentage: 0.5000
[DIFFUSER] Epoch 4/5
[INFO] Clusters: {1: ['0', '1'], 2: ['M0', 'M1']}
[DIFFUSER] Epoch completed. Epoch loss 0.00498698
[DIFFUSER] Accuracy: 0.9962 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 5/5
[DIFFUSER] Epoch completed. Epoch loss 0.00591628
[DIFFUSER] Accuracy: 0.9965 - Clustering percentage: 1.0000
```

Evaluate the model on the test set

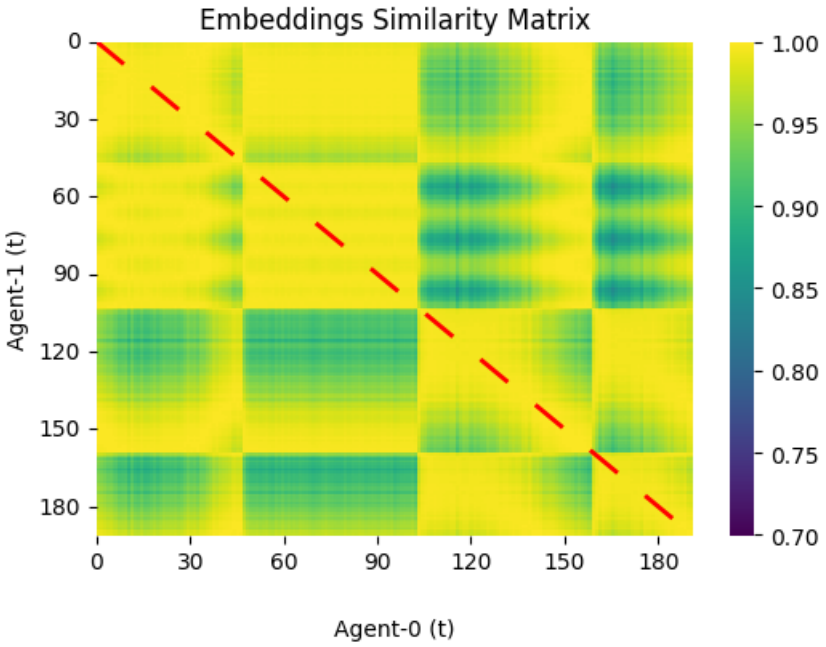
The evaluation measures how well the learned embeddings generalize.

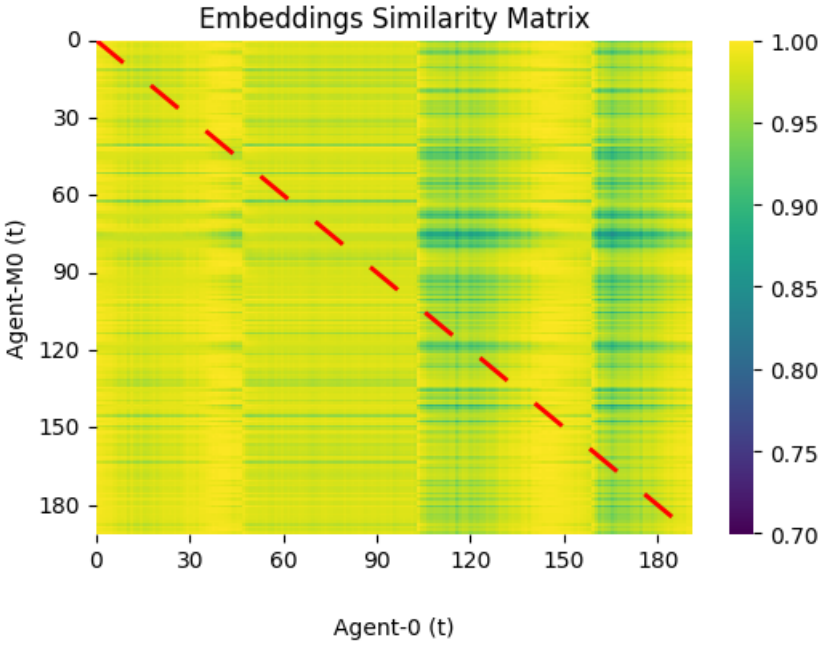
In [16]:
```
evaluate_data = diffuser.evaluate(agents_data['test'])
```

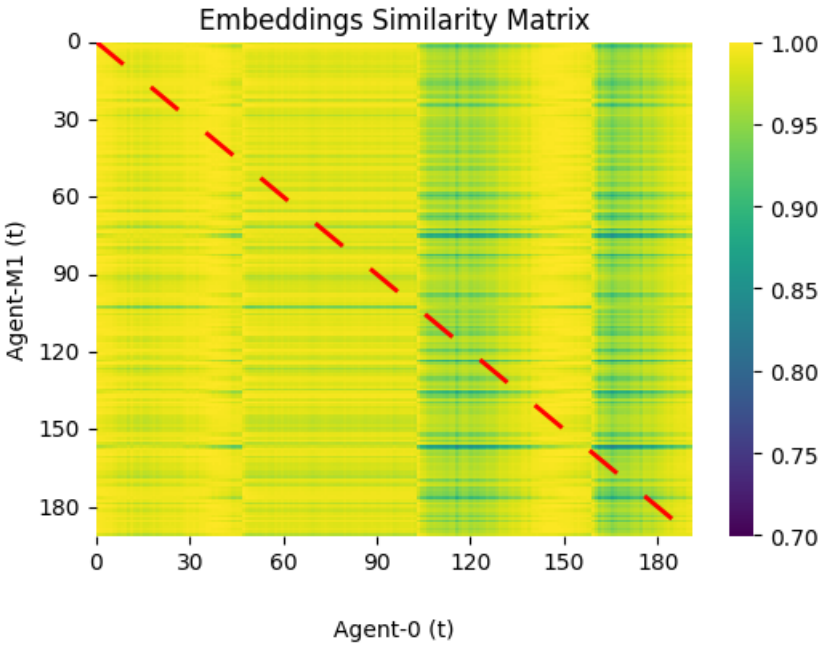
Plot the embeddings similarity matrix

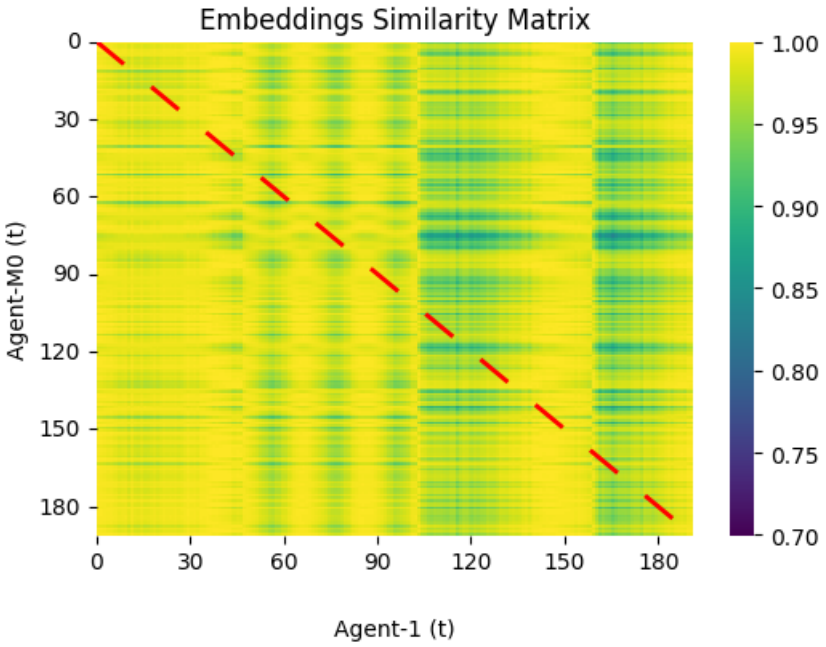
This matrix visualizes how well the embeddings from different agents align over time.

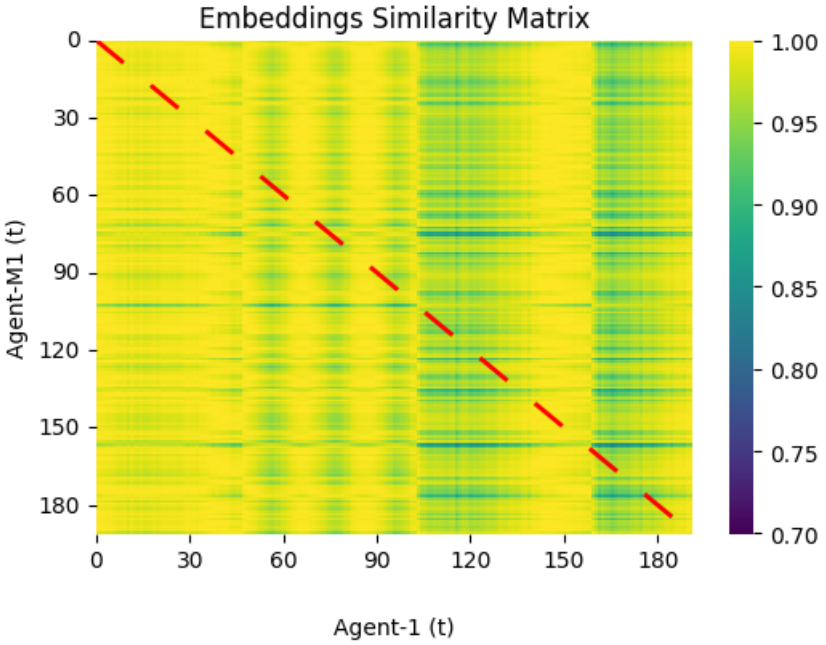
In [17]:
```
plot_embeddings_similarity_matrix(evaluate_data, (6,4), tickevery=30, min=0.7, max=1, cbar=True)
```

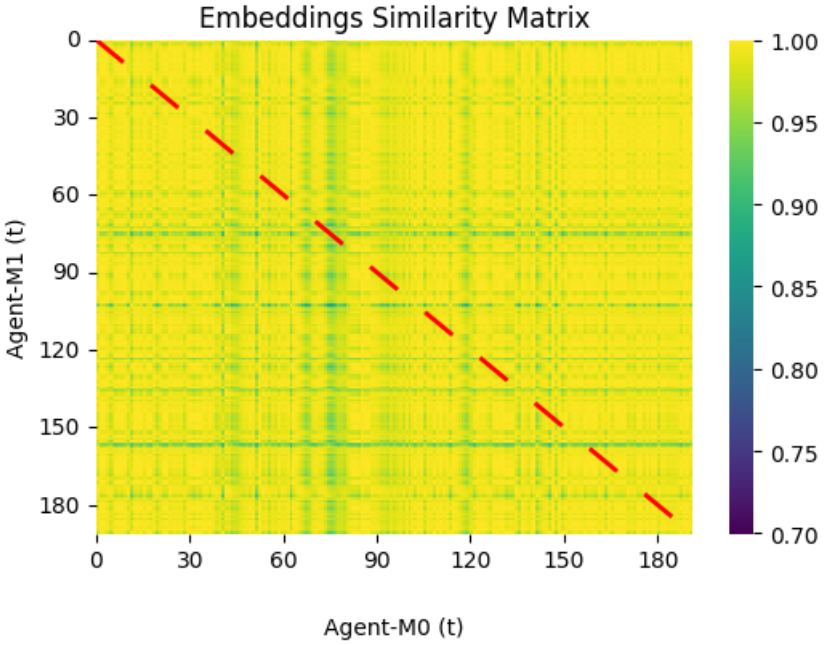

Embeddings Similarity Matrix
Agent-1 (t)
Agent-0 (t)
0
30
60
90
120
150
180
1.00
0.95
0.90
0.85
0.80
0.75
0.70


Embeddings Similarity Matrix
Agent-M0 (t)
Agent-0 (t)
0
30
60
90
120
150
180
1.00
0.95
0.90
0.85
0.80
0.75
0.70


Embeddings Similarity Matrix
Agent-M1 (t)
Agent-0 (t)
0
30
60
90
120
150
180
1.00
0.95
0.90
0.85
0.80
0.75
0.70


Embeddings Similarity Matrix
Agent-M0 (t)
Agent-1 (t)
0
30
60
90
120
150
180
1.00
0.95
0.90
0.85
0.80
0.75
0.70


Embeddings Similarity Matrix
Agent-M1 (t)
Agent-1 (t)
0
30
60
90
120
150
180
1.00
0.95
0.90
0.85
0.80
0.75
0.70


Embeddings Similarity Matrix
Agent-M1 (t)
Agent-M0 (t)
0
30
60
90
120
150
180
1.00
0.95
0.90
0.85
0.80
0.75
0.70

Results for Experiment 2: Robustness of Learning

Even in the presence of misleading agents, OSM-L **preserves representation alignment** among the correlated agents (Agent0 and Agent1). The similarity matrix of these agents **retains its structured pattern**, whereas the misleading agents (AgentM0 and AgentM1) exhibit **randomized embeddings without a clear diagonal structure**. This confirms that **OSM-L effectively filters out uncorrelated information**, preventing the diffusion process from being influenced by noise.

Stop the agents

After training and evaluation, the agents are stopped to free up resources.

In [18]:
```
diffuser.stop_agents()
```

```
[DIFFUSER] All agent processes stopped.
```

Experiment 3: Emergence of Sub-Contexts

In this experiment, we transition from a simple to a **complex context**, adding three more agents (Agent2, Agent3, and Agent4). The objective is to analyze how **OSM-L dynamically forms sub-contexts** among correlated agents.

- **Key Hypothesis:** OSM-L should **group agents into meaningful clusters** based on their correlation, without prior knowledge.
- **Metric:** Evolution of similarity matrices over multiple epochs.
- **Expected Outcome:**
  - Agents sharing strong correlations (Agent0 and Agent1; Agent2 and Agent3) should **form stable sub-contexts**.
  - Agent4, which has **partial correlations** with other agents, should eventually align within an appropriate cluster.

The results will confirm whether OSM-L can **autonomously identify and adapt to latent structures in distributed data.**

In [19]:
```
# Define parameters for the experiment

WINDOW_SIZE = 10
TRAIN_DATA_CSV = 'datasets/complex-context/train_data.csv'
TEST_DATA_CSV = 'datasets/complex-context/test_data.csv'
EPOCHS = 30
EMBEDDING_SIZE = 5
BATCH_SIZE = 50
CLUSTERING_STEP = 2
GRU_HIDDEN_DIM = 20
DEVICE = 'cpu' # 'cuda' or 'cpu' or 'mps'

set_seed(42)
```

Load and preprocess the dataset

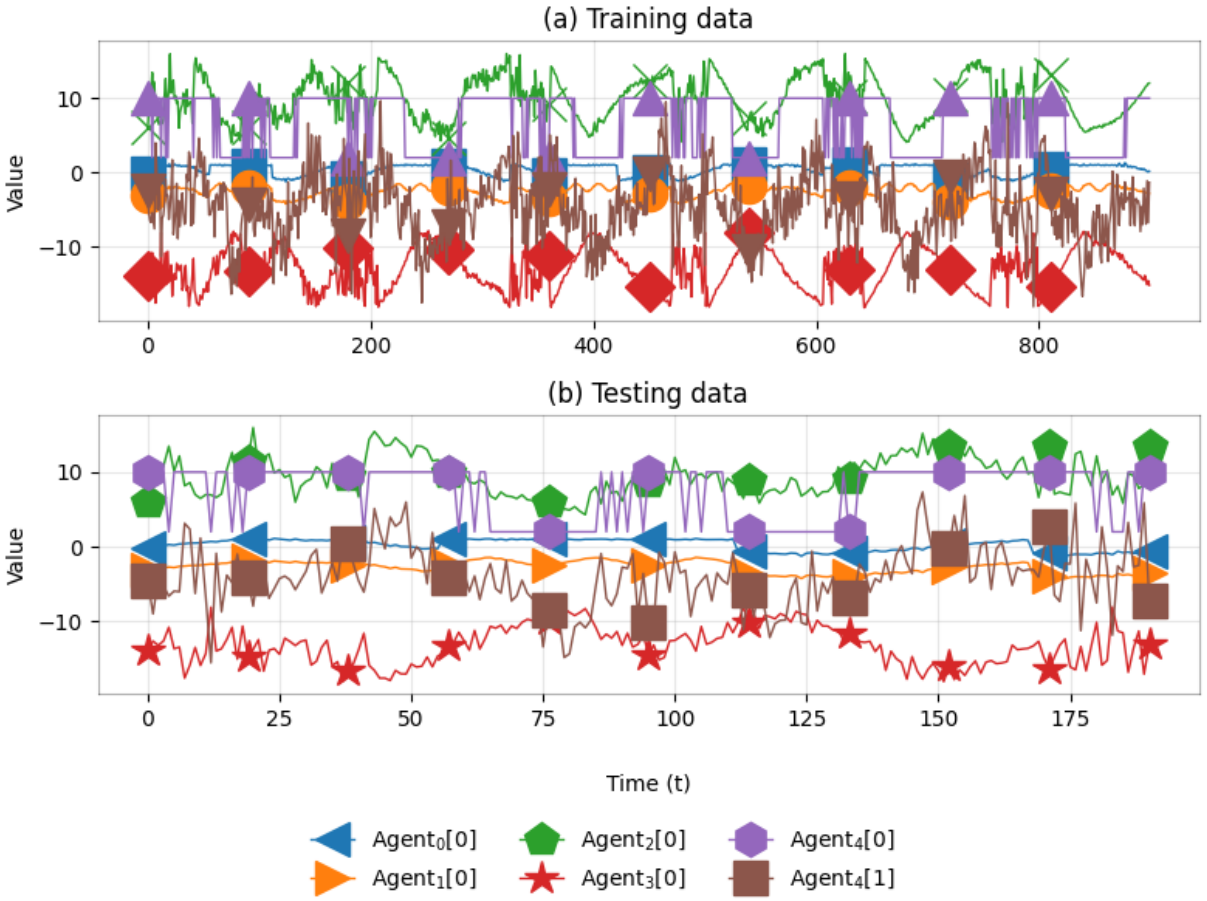
The dataset is loaded from CSV files from the 'complex context', consisting of Agent0, Agent1, Agent2, Agent3, Agent4.

In [20]:
```
dataProcessor = DataProcessor(window_size=WINDOW_SIZE)
data = dataProcessor.load_data_from_csv(TRAIN_DATA_CSV, TEST_DATA_CSV)
data = dataProcessor.load_data(data['train'], data['test'])
agents_data = dataProcessor.preprocess_data()
```

Plot time series for all agents

This visualization now includes four agents.

In [21]:
```
dataProcessor.plot_time_series()
```



Initialize agents and communication queues

Each agent is assigned its own queue for sending and receiving messages and processes local data using a GRU-based model.

In [22]:
```
NUM_AGENTS = len(agents_data['train'])
sending_queues = {agent_id: mp.Queue() for agent_id in agents_data['train'].keys()}
receiving_queues = {agent_id: mp.Queue() for agent_id in agents_data['train'].keys()}
results_queue = mp.Queue()

agents = {}
for agent_id, data in agents_data['train'].items():
    dataset = TensorDataset(data['series'])
    dataloader = DataLoader(dataset, batch_size=BATCH_SIZE, shuffle=False)

    dataset = TensorDataset(agents_data['test'][agent_id]['series'])
    testloader = DataLoader(dataset, batch_size=BATCH_SIZE, shuffle=False)
    model = MODEL_REGISTRY["gru"](
        input_dim=data["features"],
        hidden_dim=GRU_HIDDEN_DIM,
        output_dim=EMBEDDING_SIZE
    )
    agent = Agent(
        agent_id=agent_id,
        model=model,
        train_loader=dataloader,
        test_loader=testloader,
        results_queue=results_queue,
        sending_queue=sending_queues[agent_id],
        receiving_queue=receiving_queues[agent_id],
        device=DEVICE
    )
    agents[agent_id] = agent
```

Initialize the Diffuser

The **Diffuser** coordinates the learning process by aligning embeddings across agents.

In [23]:
```
diffuser = Diffuser(
    agents=agents,
    epochs=EPOCHS,
    clustering=True,
    clustering_step=CLUSTERING_STEP,
    device=DEVICE,
)
```

Run the training process

This step allows **local embeddings to converge** while preserving individual data characteristics.

In [24]:
```
local_embeddings, metrics = diffuser.run(evaluate=True, plot=True)
```

```
[DIFFUSER] Global embedding initialized with shape: torch.Size([1, 5])
[DIFFUSER] All agent processes started.
[DIFFUSER] Epoch 1/30
[DIFFUSER] Epoch completed. Epoch loss 0.02141711
[DIFFUSER] Accuracy: 0.6102 - Clustering percentage: 0.0000
[DIFFUSER] Epoch 2/30
```


```
[INFO] Clusters: {0: ['0', '1', '2', '3', '4']}
[DIFFUSER] Epoch completed. Epoch loss 0.00860182
[DIFFUSER] Accuracy: 0.7319 - Clustering percentage: 0.0000
[DIFFUSER] Epoch 3/30
[DIFFUSER] Epoch completed. Epoch loss 0.00667651
[DIFFUSER] Accuracy: 0.8457 - Clustering percentage: 0.0000
[DIFFUSER] Epoch 4/30
```


```
[INFO] Clusters: {0: ['0', '1', '2', '3', '4']}
[DIFFUSER] Epoch completed. Epoch loss 0.00620796
[DIFFUSER] Accuracy: 0.9014 - Clustering percentage: 0.0000
[DIFFUSER] Epoch 5/30
[DIFFUSER] Epoch completed. Epoch loss 0.00685204
[DIFFUSER] Accuracy: 0.8665 - Clustering percentage: 0.0000
[DIFFUSER] Epoch 6/30
```


```
[INFO] Clusters: {0: ['0', '1', '2', '3', '4']}
[DIFFUSER] Epoch completed. Epoch loss 0.00627904
[DIFFUSER] Accuracy: 0.9203 - Clustering percentage: 0.0000
[DIFFUSER] Epoch 7/30
[DIFFUSER] Epoch completed. Epoch loss 0.00700568
[DIFFUSER] Accuracy: 0.9222 - Clustering percentage: 0.0000
[DIFFUSER] Epoch 8/30
```

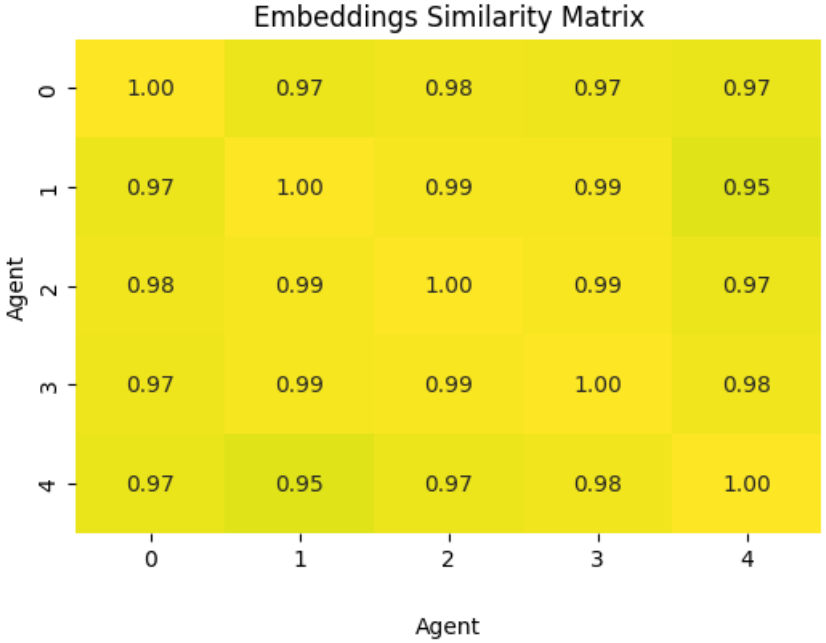

```
[INFO] Clusters: {0: ['0', '2', '4'], 1: ['1', '3']}
[DIFFUSER] Epoch completed. Epoch loss 0.00602108
[DIFFUSER] Accuracy: 0.9435 - Clustering percentage: 0.4000
[DIFFUSER] Epoch 9/30
[DIFFUSER] Epoch completed. Epoch loss 0.00603012
[DIFFUSER] Accuracy: 0.9609 - Clustering percentage: 0.4000
[DIFFUSER] Epoch 10/30
```


```
[INFO] Clusters: {0: ['0', '2', '4'], 1: ['1', '3']}
[DIFFUSER] Epoch completed. Epoch loss 0.00707128
[DIFFUSER] Accuracy: 0.9677 - Clustering percentage: 0.4000
[DIFFUSER] Epoch 11/30
[DIFFUSER] Epoch completed. Epoch loss 0.00737676
[DIFFUSER] Accuracy: 0.9555 - Clustering percentage: 0.4000
[DIFFUSER] Epoch 12/30
```

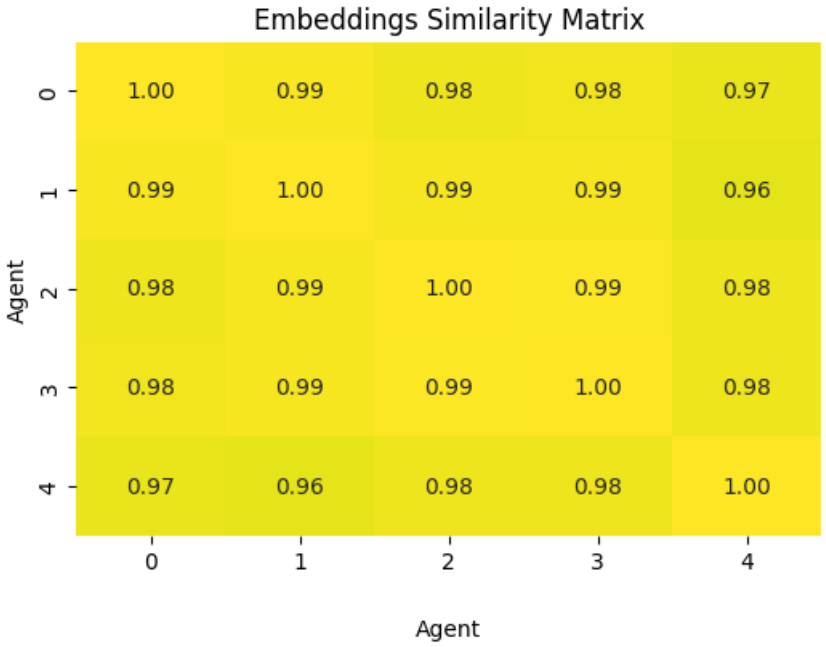

```
[INFO] Clusters: {0: ['0', '2', '4'], 1: ['1', '3']}
[DIFFUSER] Epoch completed. Epoch loss 0.00667947
[DIFFUSER] Accuracy: 0.9701 - Clustering percentage: 0.4000
[DIFFUSER] Epoch 13/30
[DIFFUSER] Epoch completed. Epoch loss 0.00680417
[DIFFUSER] Accuracy: 0.9750 - Clustering percentage: 0.4000
[DIFFUSER] Epoch 14/30
```


```
[INFO] Clusters: {0: ['0', '2', '4'], 1: ['1', '3']}
[DIFFUSER] Epoch completed. Epoch loss 0.00627373
[DIFFUSER] Accuracy: 0.9717 - Clustering percentage: 0.4000
[DIFFUSER] Epoch 15/30
[DIFFUSER] Epoch completed. Epoch loss 0.00722495
[DIFFUSER] Accuracy: 0.9758 - Clustering percentage: 0.4000
[DIFFUSER] Epoch 16/30
```


```
[INFO] Clusters: {0: ['0', '4'], 1: ['2', '3', '1']}
[DIFFUSER] Epoch completed. Epoch loss 0.00663016
[DIFFUSER] Accuracy: 0.9806 - Clustering percentage: 0.6000
[DIFFUSER] Epoch 17/30
[DIFFUSER] Epoch completed. Epoch loss 0.00597509
[DIFFUSER] Accuracy: 0.9850 - Clustering percentage: 0.6000
[DIFFUSER] Epoch 18/30
```



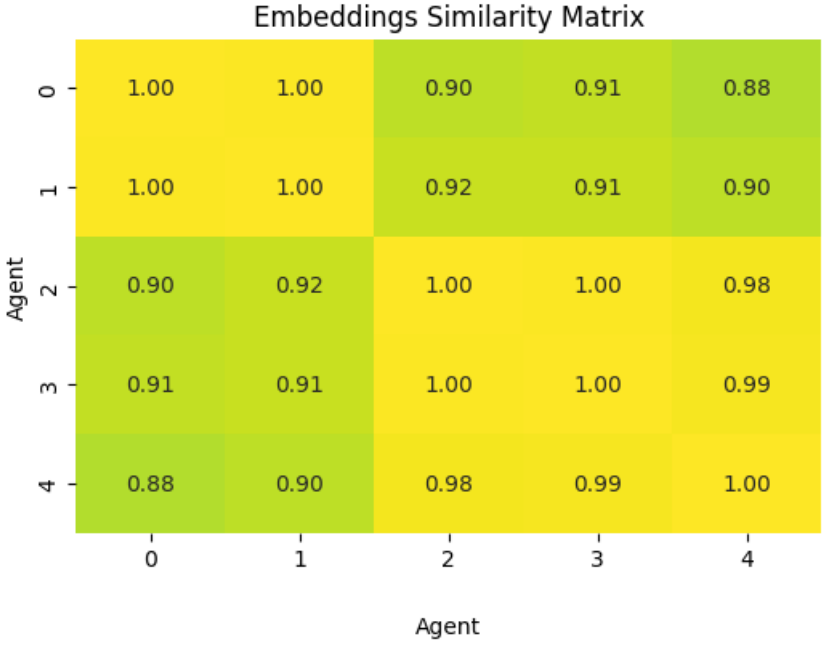

```
[INFO] Clusters: {1: ['2', '3', '1', '4', '0']}
[DIFFUSER] Epoch completed. Epoch loss 0.00527889
[DIFFUSER] Accuracy: 0.9850 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 19/30
[DIFFUSER] Epoch completed. Epoch loss 0.00652679
[DIFFUSER] Accuracy: 0.9825 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 20/30
```




```
[INFO] Clusters: {1: ['2', '3', '4'], 2: ['0', '1']}
[DIFFUSER] Epoch completed. Epoch loss 0.00692946
[DIFFUSER] Accuracy: 0.9925 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 21/30
[DIFFUSER] Epoch completed. Epoch loss 0.00520072
[DIFFUSER] Accuracy: 0.9946 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 22/30
```


```
[INFO] Clusters: {1: ['0', '1'], 2: ['2', '3', '4']}
[DIFFUSER] Epoch completed. Epoch loss 0.00701459
[DIFFUSER] Accuracy: 0.9929 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 23/30
[DIFFUSER] Epoch completed. Epoch loss 0.00614686
[DIFFUSER] Accuracy: 0.9954 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 24/30
```


```
[INFO] Clusters: {1: ['0', '1'], 2: ['2', '3', '4']}
[DIFFUSER] Epoch completed. Epoch loss 0.00736611
[DIFFUSER] Accuracy: 0.9944 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 25/30
[DIFFUSER] Epoch completed. Epoch loss 0.00555291
[DIFFUSER] Accuracy: 0.9935 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 26/30
```




```
[INFO] Clusters: {1: ['2', '3', '4'], 2: ['0', '1']}
[DIFFUSER] Epoch completed. Epoch loss 0.00653482
[DIFFUSER] Accuracy: 0.9949 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 27/30
[DIFFUSER] Epoch completed. Epoch loss 0.00645757
[DIFFUSER] Accuracy: 0.9928 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 28/30
```

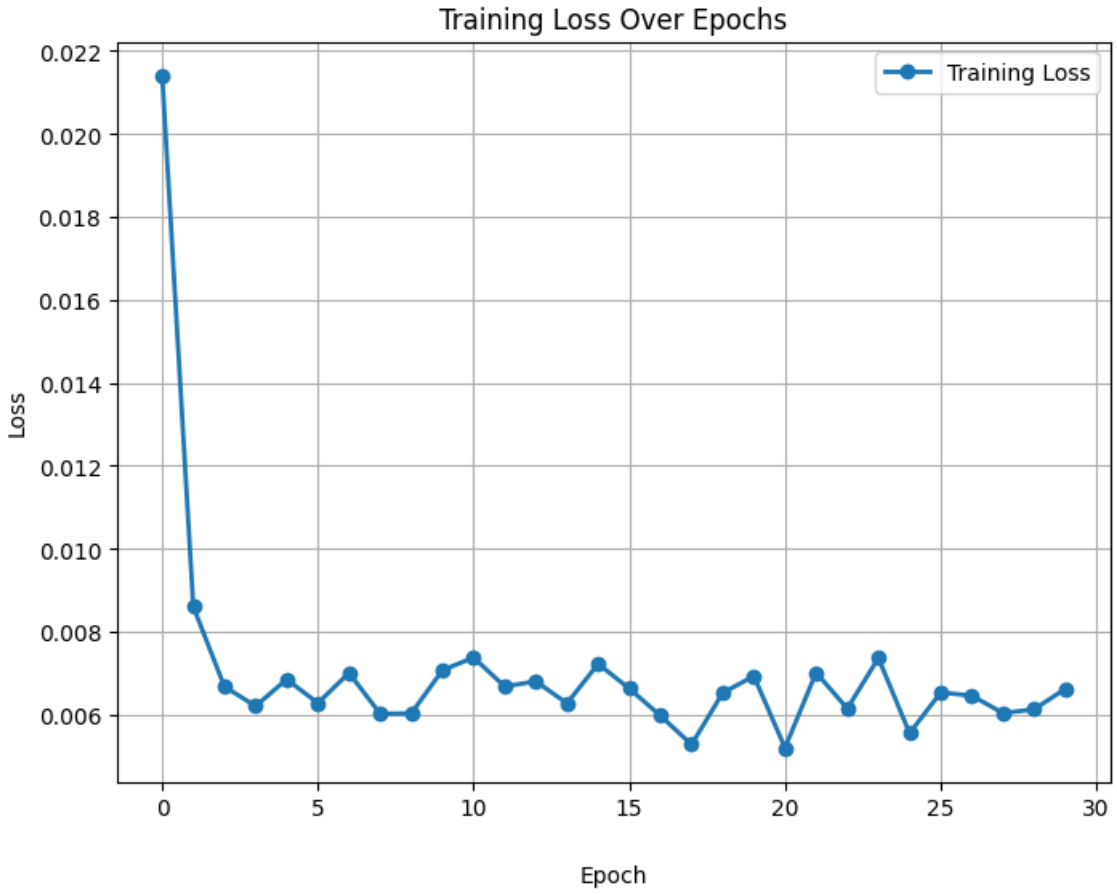

```
[INFO] Clusters: {1: ['0', '1'], 2: ['2', '3', '4']}
[DIFFUSER] Epoch completed. Epoch loss 0.00603878
[DIFFUSER] Accuracy: 0.9896 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 29/30
[DIFFUSER] Epoch completed. Epoch loss 0.00612860
[DIFFUSER] Accuracy: 0.9890 - Clustering percentage: 1.0000
[DIFFUSER] Epoch 30/30
```

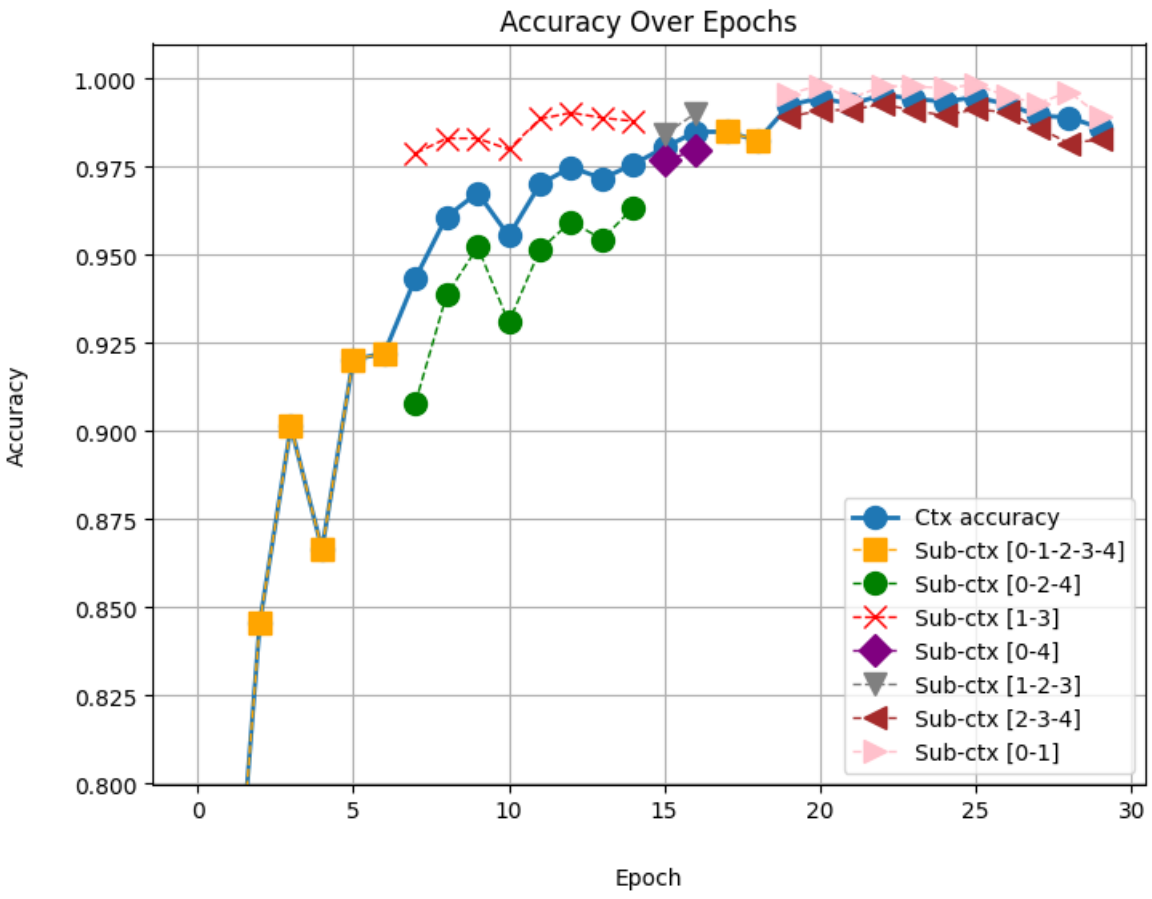

```
[INFO] Clusters: {1: ['2', '3', '4'], 2: ['0', '1']}
[DIFFUSER] Epoch completed. Epoch loss 0.00662311
[DIFFUSER] Accuracy: 0.9861 - Clustering percentage: 1.0000
```

Evaluate Training Loss and Accuracy over Epochs

Plot training loss and accuracy showing their evolution during the training.

In [25]:
```
training_losses = metrics["training_losses"]
plot_training_loss(training_losses)
plot_accuracy(metrics)
```





Evaluate the model on the test set

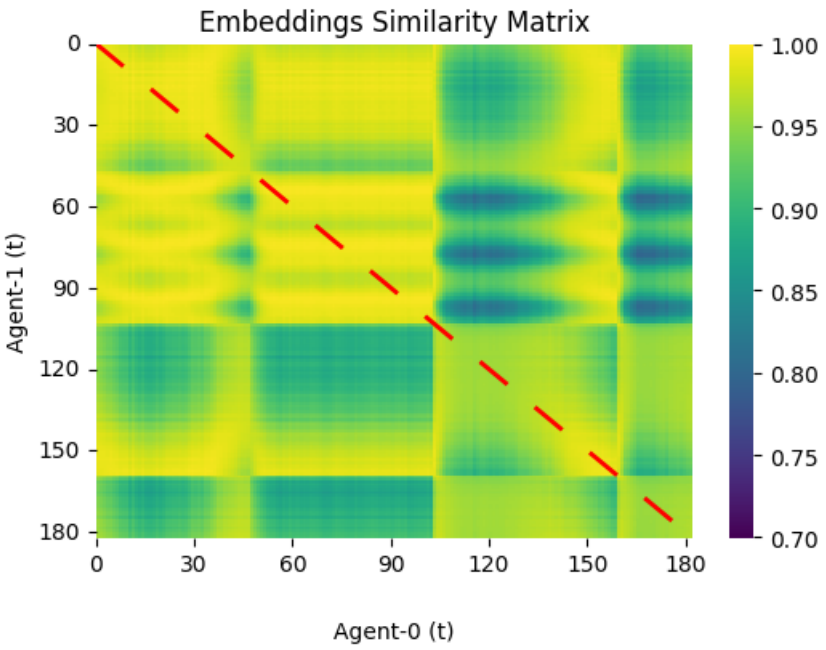
The evaluation measures how well the learned embeddings generalize.

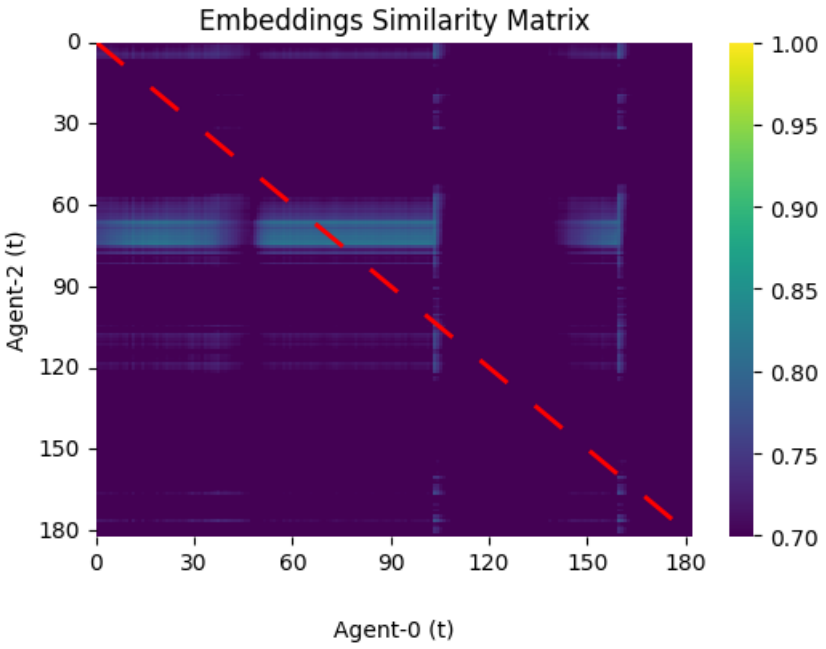
In [26]:
```
evaluate_data = diffuser.evaluate(agents_data['test'])
```

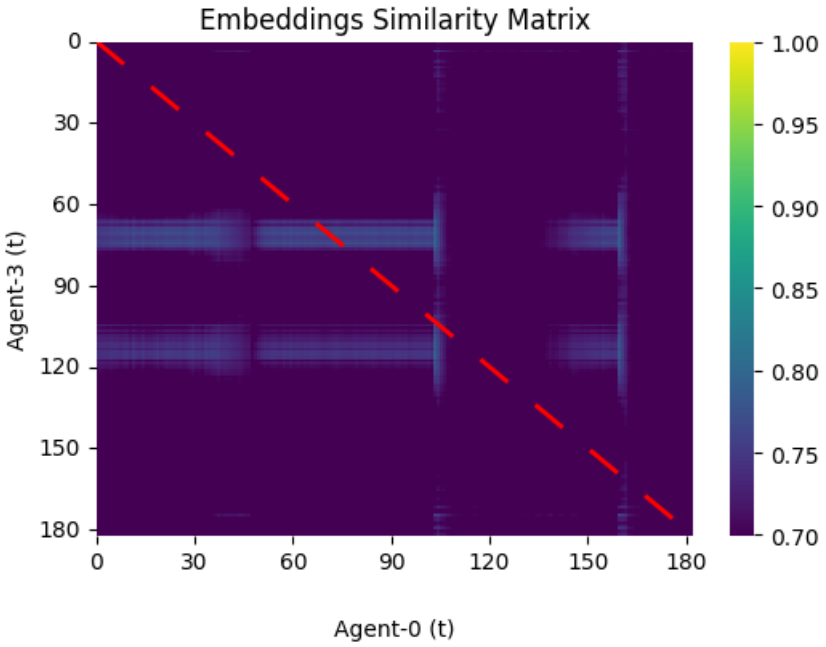
Plot the embeddings similarity matrix

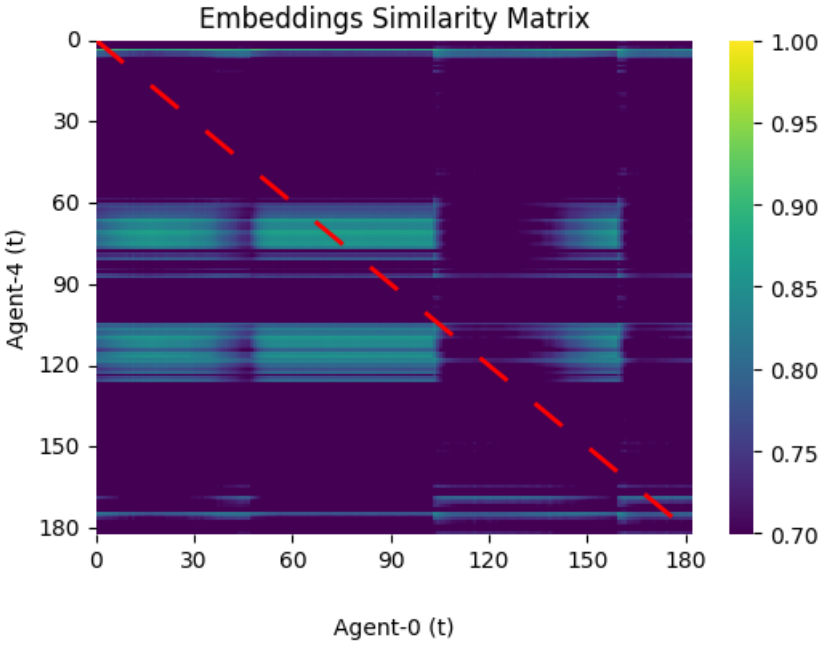
This matrix visualizes how well the embeddings from different agents align over time.

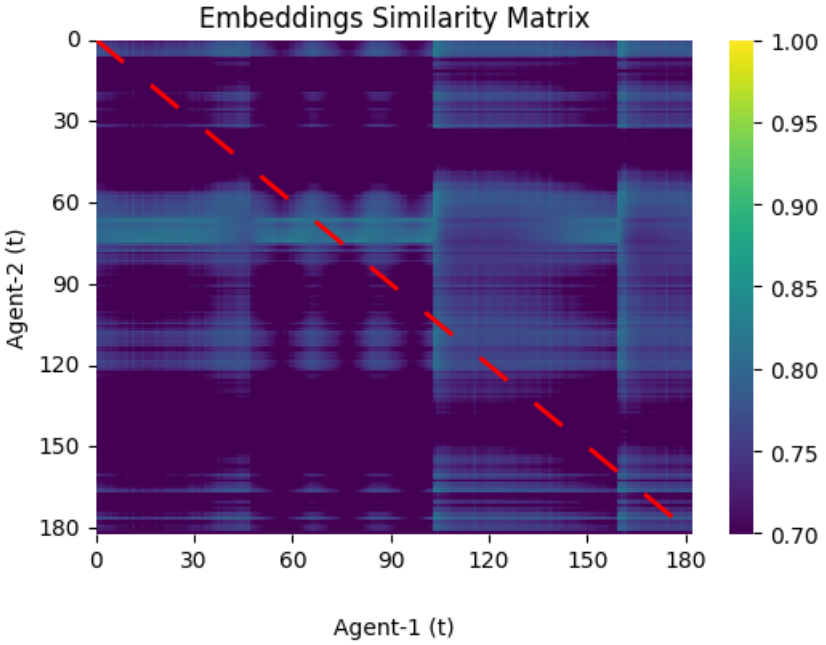
In [27]:
```
plot_embeddings_similarity_matrix(evaluate_data, (6,4), tickevery=30, min=0.7, max=1, cbar=True)
```

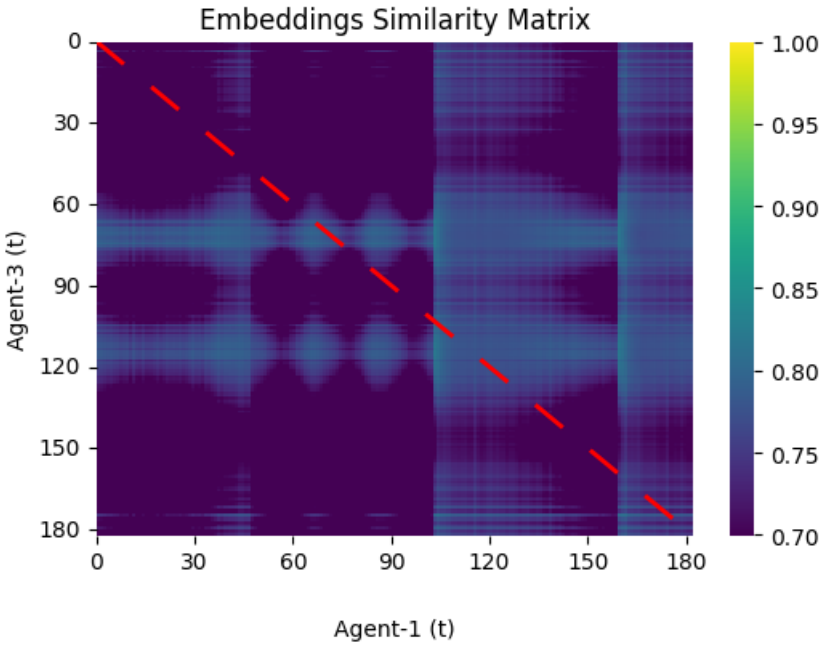


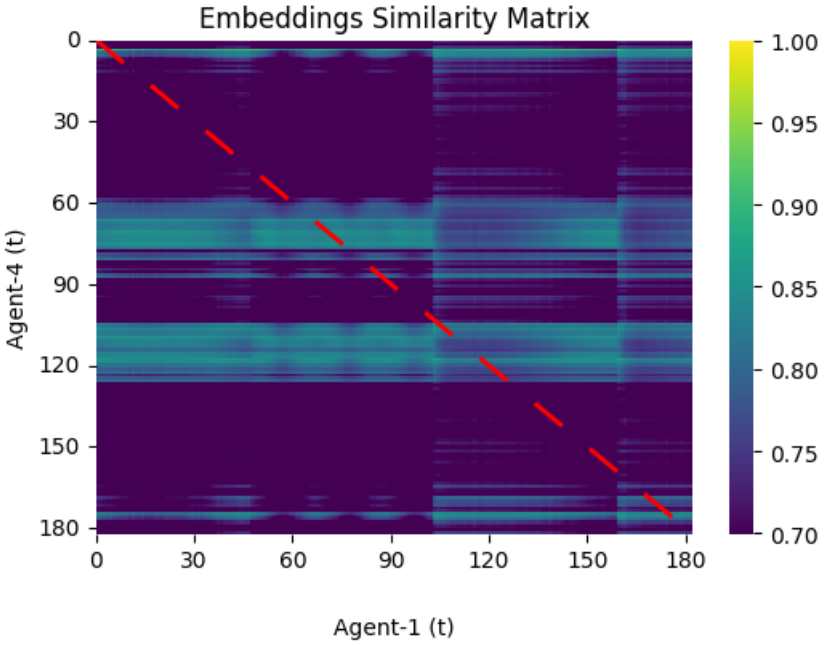


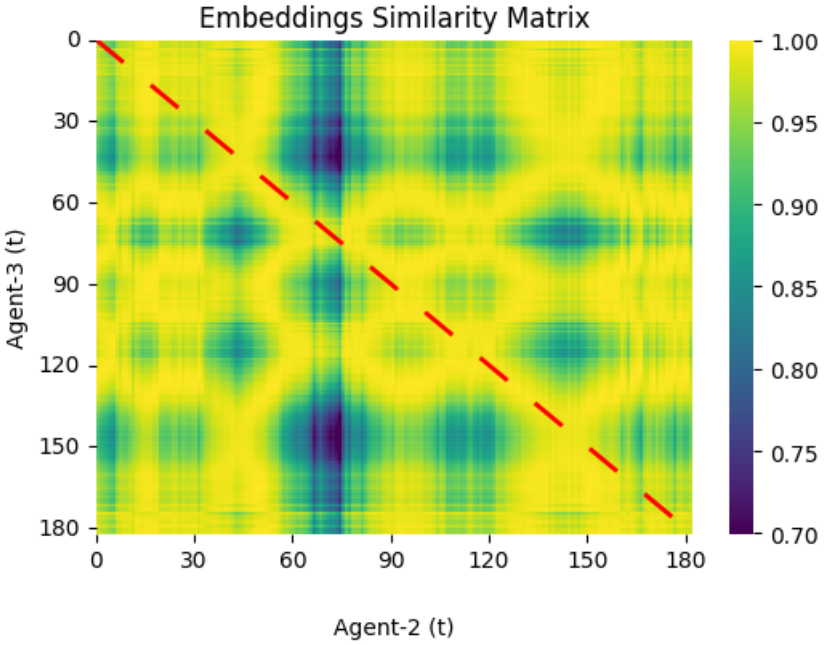

Embeddings Similarity Matrix
Agent-3 (t)
Agent-0 (t)
0
30
60
90
120
150
180
1.00
0.95
0.90
0.85
0.80
0.75
0.70

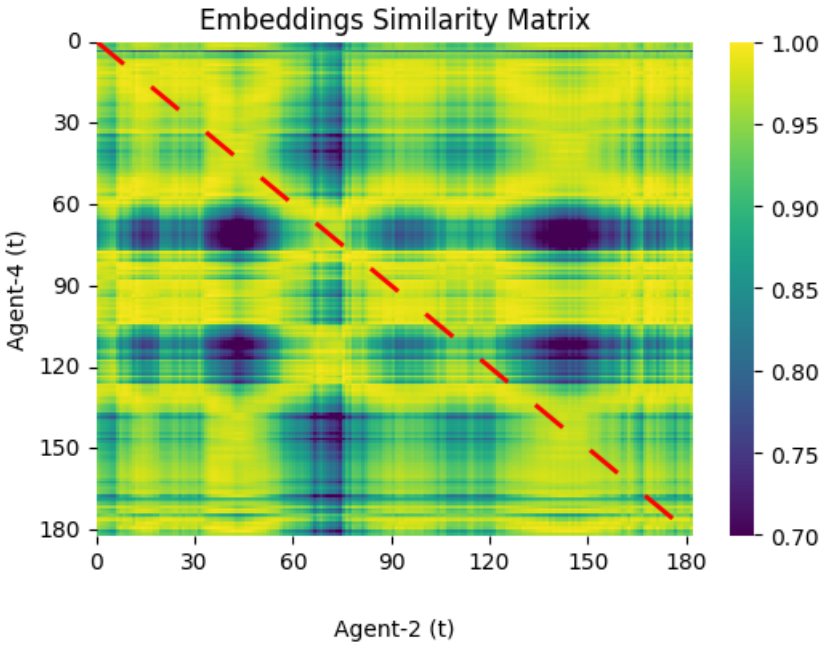

Embeddings Similarity Matrix
Agent-4 (t)
Agent-0 (t)
0
30
60
90
120
150
180
1.00
0.95
0.90
0.85
0.80
0.75
0.70

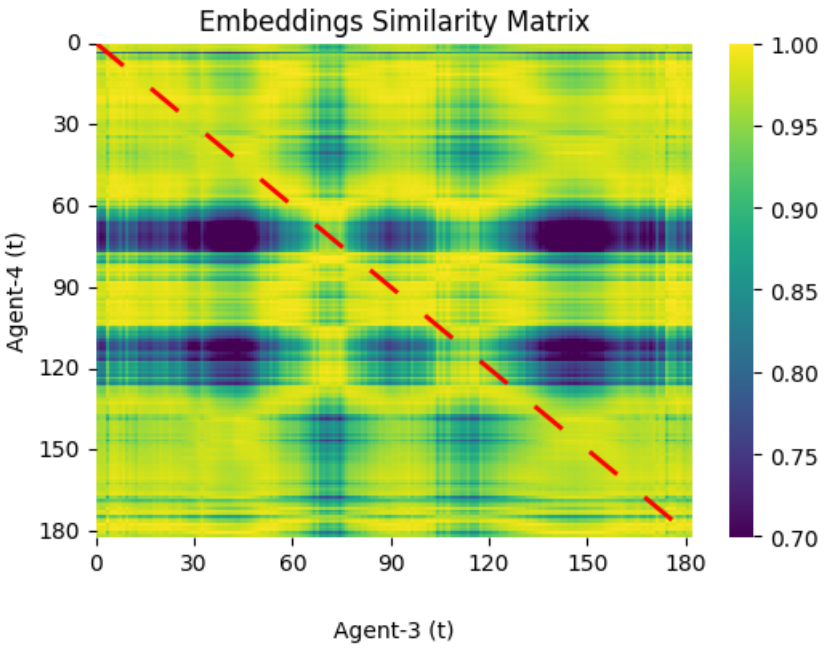

Embeddings Similarity Matrix
Agent-2 (t)
Agent-1 (t)
0
30
60
90
120
150
180
1.00
0.95
0.90
0.85
0.80
0.75
0.70


Embeddings Similarity Matrix
Agent-3 (t)
Agent-1 (t)
0
30
60
90
120
150
180
1.00
0.95
0.90
0.85
0.80
0.75
0.70


Embeddings Similarity Matrix
Agent-4 (t)
Agent-1 (t)
0
30
60
90
120
150
180
1.00
0.95
0.90
0.85
0.80
0.75
0.70


Embeddings Similarity Matrix
Agent-3 (t)
Agent-2 (t)
0
30
60
90
120
150
180
1.00
0.95
0.90
0.85
0.80
0.75
0.70





Results for Experiment 3: Emergence of Sub-Contexts

The similarity matrices reveal the **formation of distinct sub-contexts**, with **Agent0-Agent1** and **Agent2-Agent4** converging into separate clusters. Despite a negative correlation, Agent2 and Agent3 **align within the same sub-context**, while Agent4 dynamically integrates based on its **partial correlation**. The training loss **rapidly decreases**, and accuracy peaks at **0.9954 at epoch 23**, confirming that OSM-L **dynamically structures contextual representations**, distinguishing relevant relationships without explicit supervision.

Stop the agents

After training and evaluation, the agents are stopped to free up resources.

In [28]:
```
diffuser.stop_agents()
```

```
[DIFFUSER] All agent processes stopped.
```