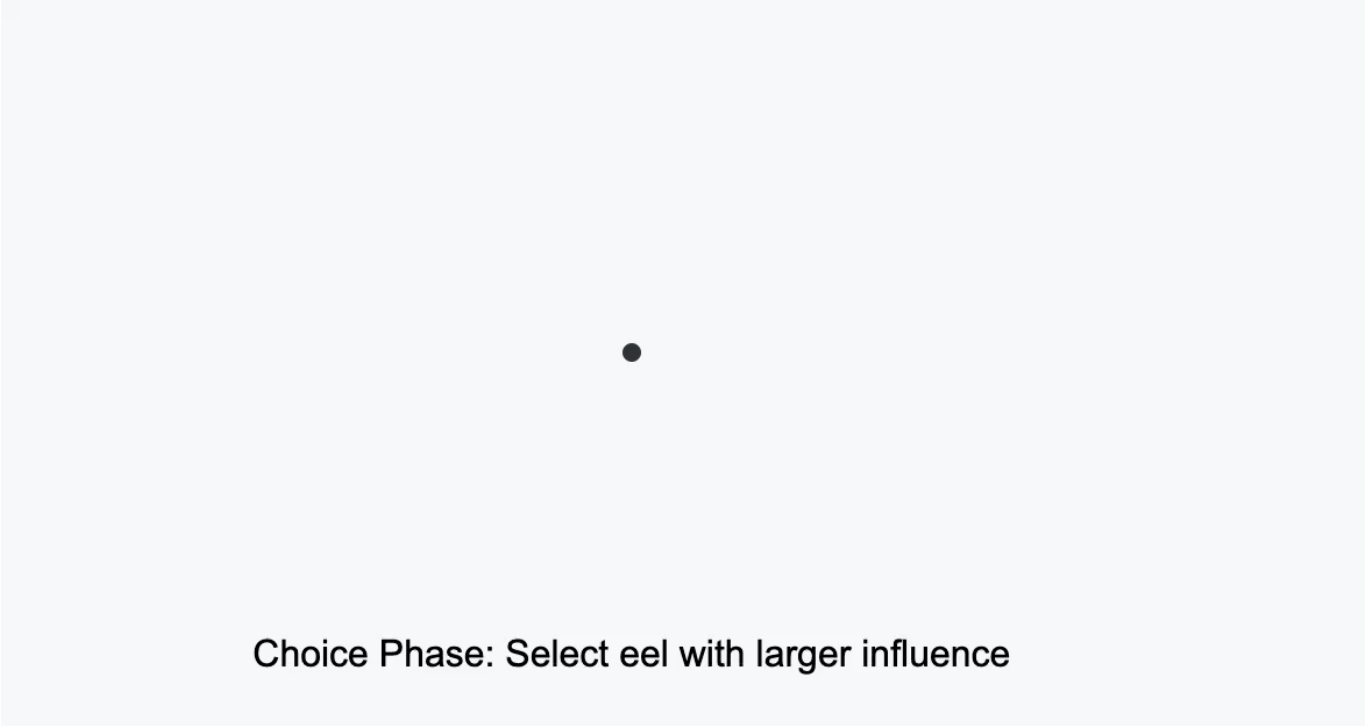
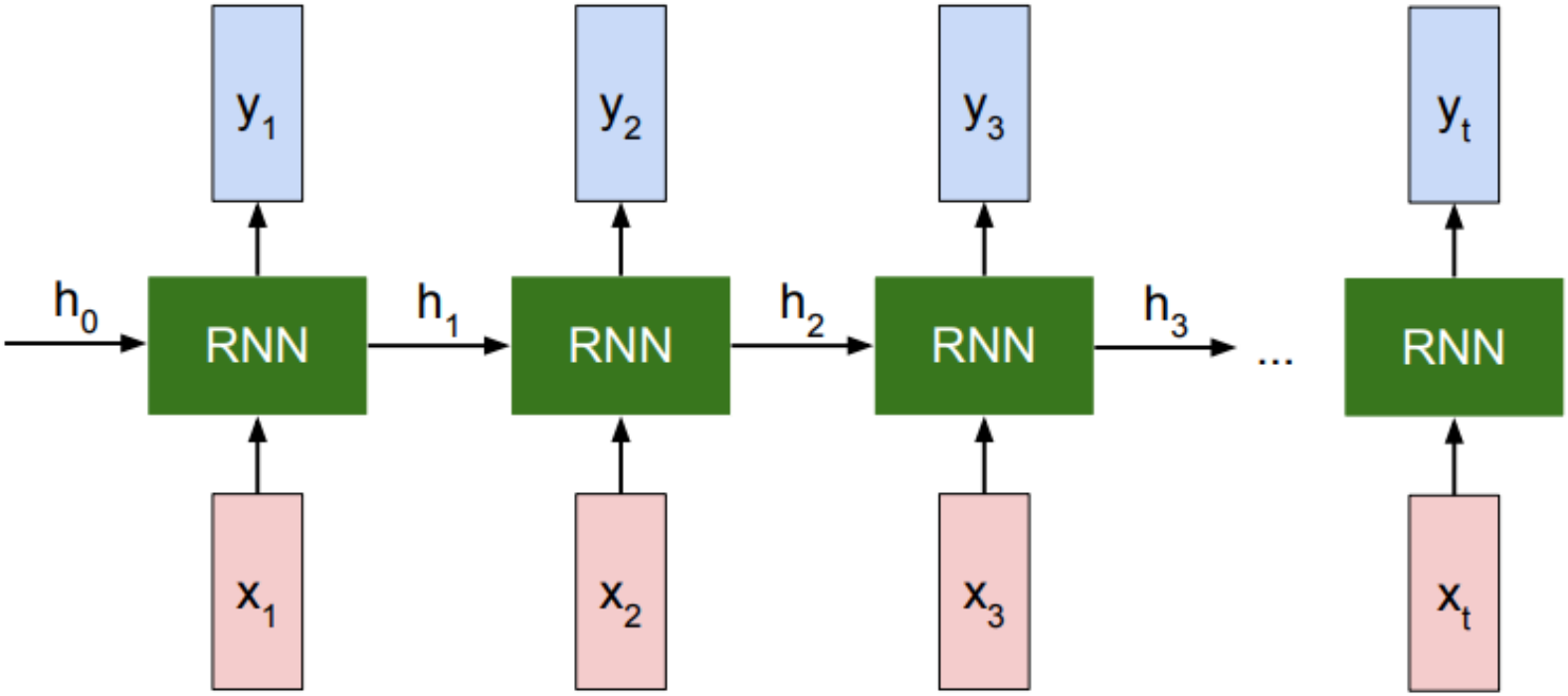
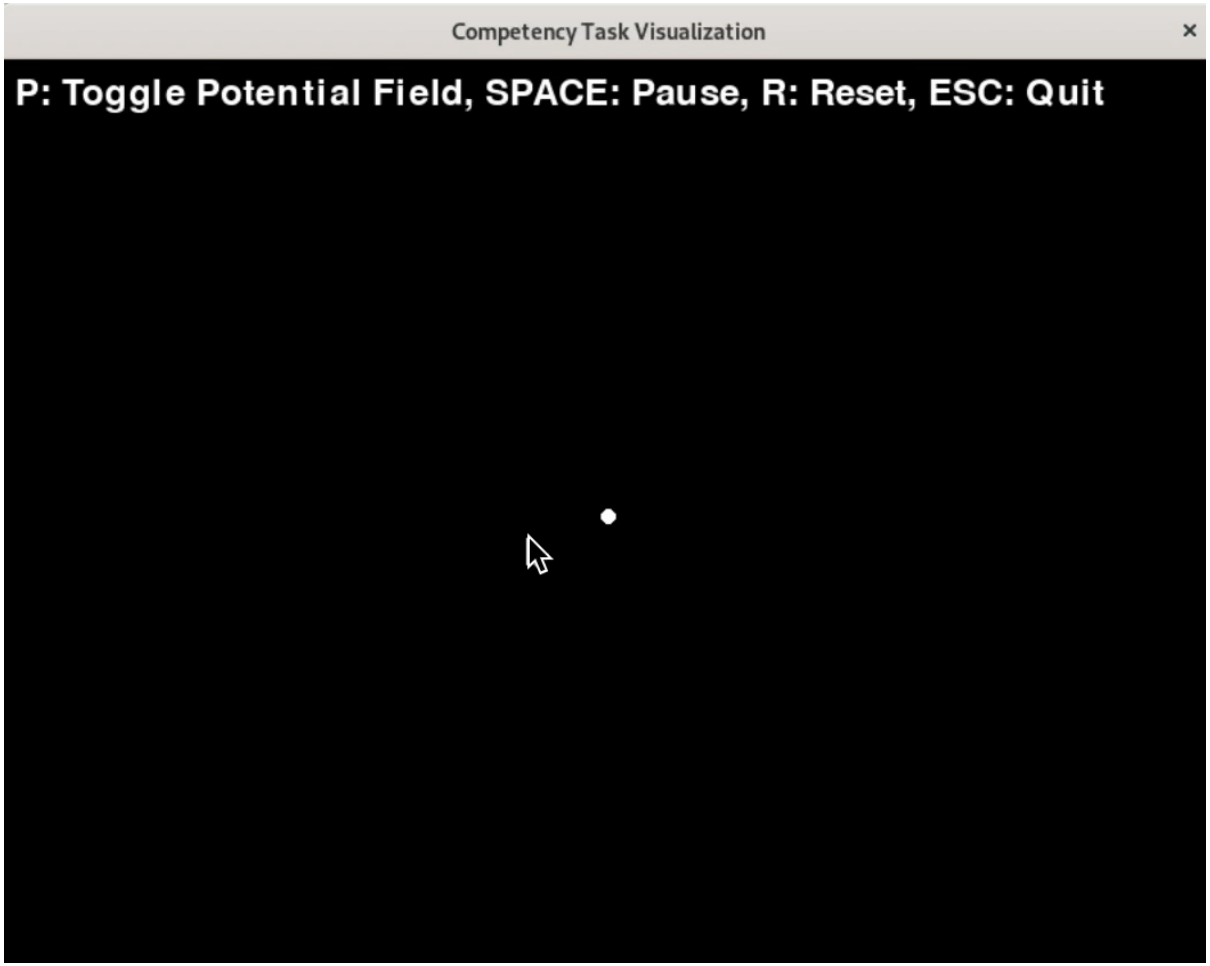
Task-Optimized RNN in PyTorch



RNN

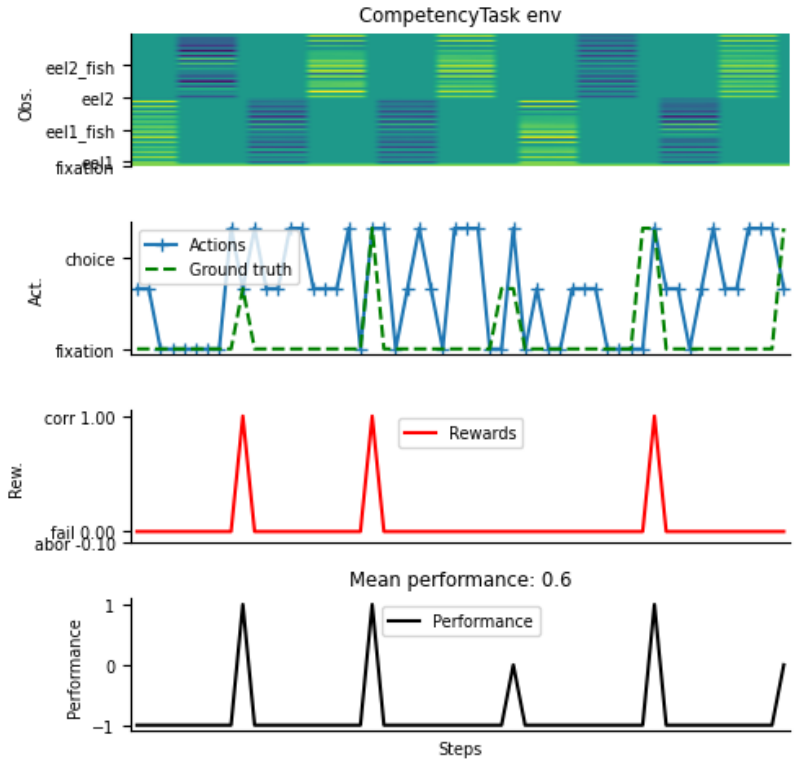


Task


Competency Task Visualization
P: Toggle Potential Field, SPACE: Pause, R: Reset, ESC: Quit

Breaking Down the CompetencyTask

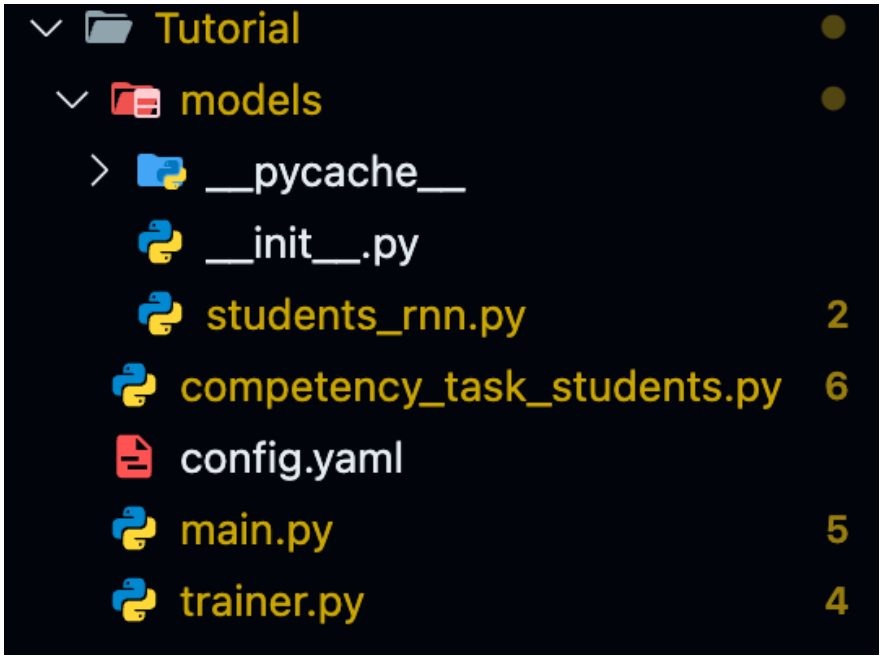
Example Network



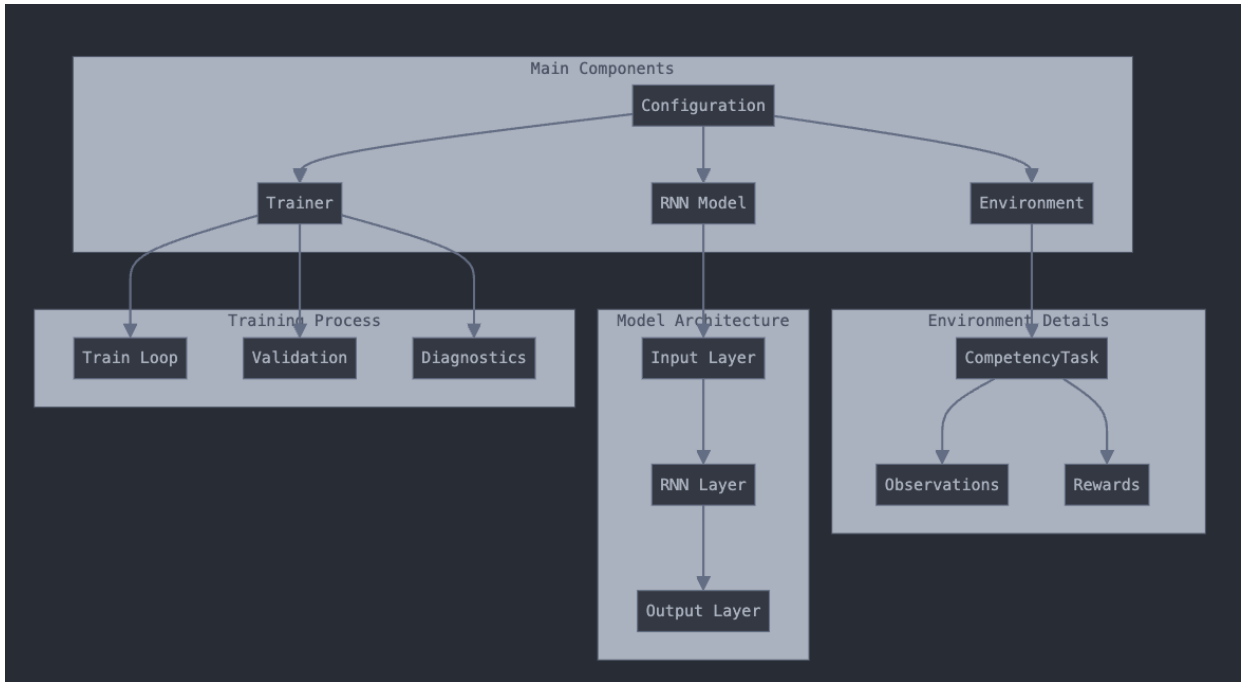
Code

Main

Project Structure



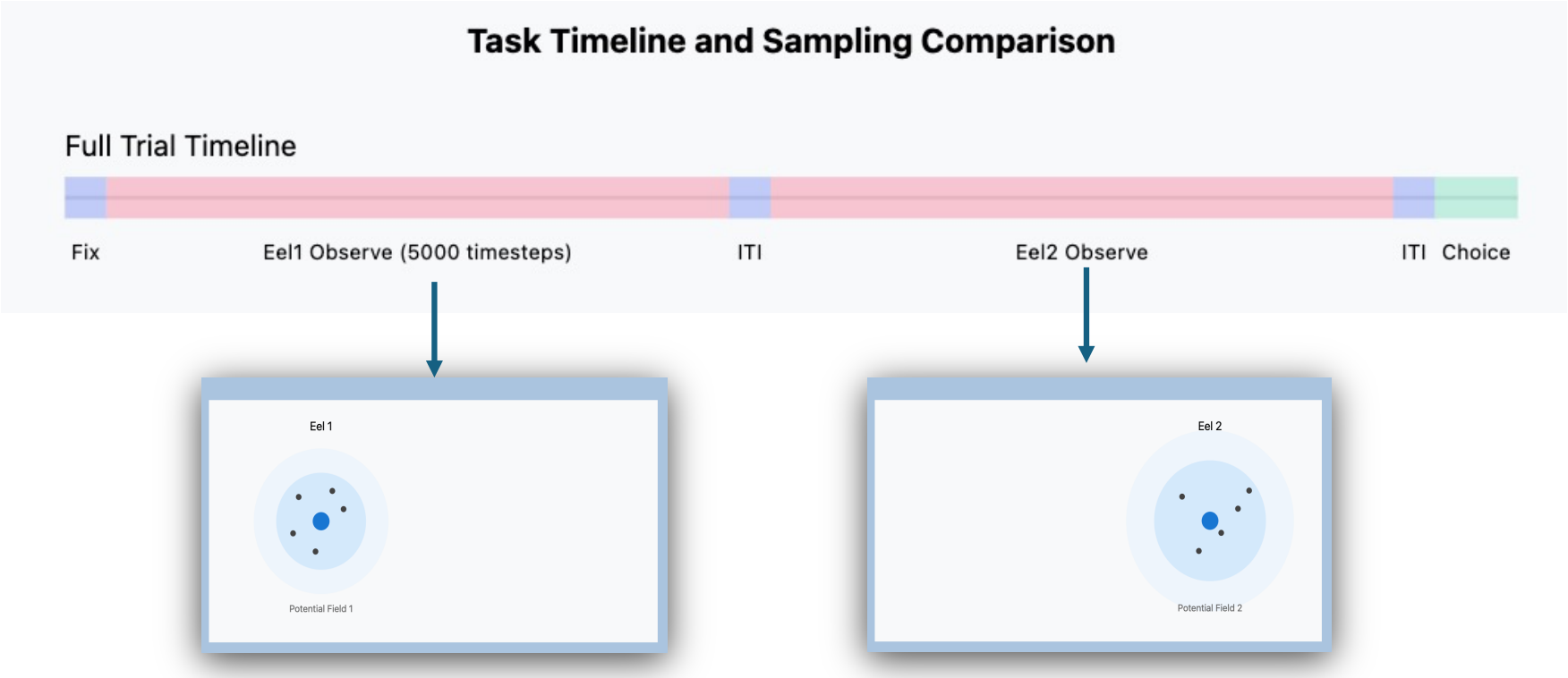
Main


Main Components
Configuration
Trainer
RNN Model
Environment
Training Process
Train Loop
Validation
Diagnostics
Model Architecture
Input Layer
RNN Layer
Output Layer
Environment Details
CompetencyTask
Observations
Rewards

Code

Environment

Environment

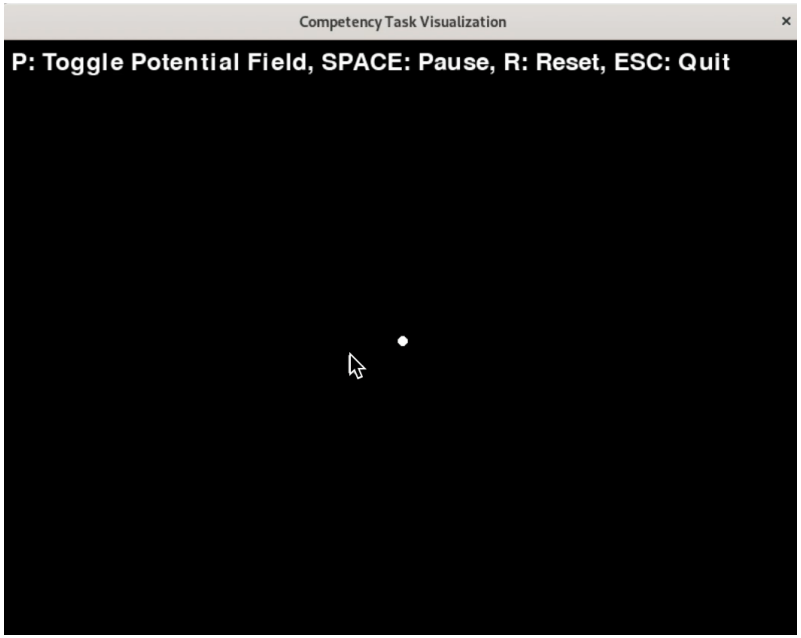


Environment

Task

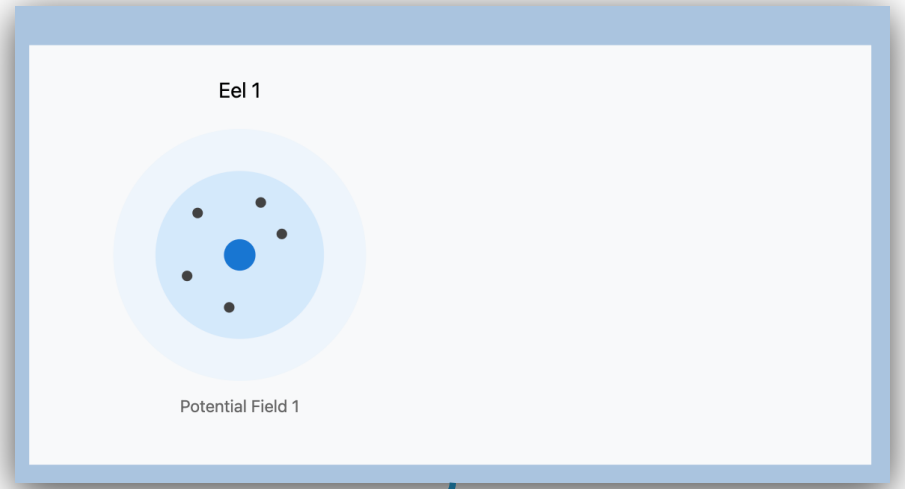
- **What is the Task?**
  - The agent must:
    - Observe two eels and their influence on nearby fish
    - Determine which eel has a stronger influence
    - Make a correct choice to receive a reward

- **Trial Structure**
  1. **Fixation Phase**
     1. Nothing shown
  2. **First Observation**
     1. Watch first eel and its fish
  3. **Second Observation**
     1. Watch second eel and its fish
  4. **Decision Phase**
     1. Choose which eel has stronger influence
     2. Receive reward if correct



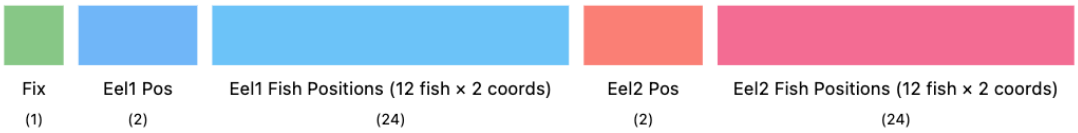
Environment

Timepoint Example




```
# During eel1_observe:
obs = [1, x1, y1, fish1_x1, fish1_y1, ..., fish12_x1, fish12_y1, 0, 0, ..., 0]

# During eel2_observe:
obs = [1, 0, 0, ..., 0, x2, y2, fish1_x2, fish1_y2, ..., fish12_x2, fish12_y2]
```



Total Dimensions: 53

Code

Configuration

Configuration (config.yaml)

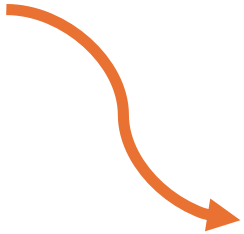
Model

```yaml
# config.yaml

# Model Configuration
model:
  hidden_size: 64
  dtype: float32
```

- **hidden_size: Controls model**
  - **capacity Larger** = more complex patterns but slower training

- **Smaller** = faster training but might miss subtle patterns



```python
# main.py

# Initialize model
model = RNNNet(
    input_size=env.observation_space.shape[0],
    hidden_size=config['model']['hidden_size'],
    output_size=env.action_space.n,
).to(device)
```

Configuration (config.yaml)

Trainer

```yaml
training:
  batch_size: 64
  learning_rate: 4e-5
  num_epochs: 10
  num_trials_per_epoch: 500
  validation_trials: 500
```

- **batch_size:** Number of trials processed together
  - **Larger** = more stable gradients but more memory
  - **Smaller** = less stable but faster updates
- **learning_rate**: Controls step size in optimization
  - **Too high** = unstable training
  - **Too low** = slow convergence

num_epochs: Total training iterations

validation_frequency: How often to check performance

Definitions

Gradients

Gradients drive optimization by:
1.Quantifying how each parameter affects the loss
2.Determining parameter updates during backpropagation
3.Enabling efficient weight optimization through gradient descent

```yaml
# Configuration parameters affecting gradients
training:
    learning_rate: 4e-5  # Step size for updates
    optimizer:
        type: 'adamw'    # Adaptive optimization
        weight_decay: 1e-4  # L2 regularization
```

- **Gradient Magnitude**
  - Large gradients → Risk of unstable training
  - Very small gradients → Slow convergence or vanishing gradient problem
  - Clipping prevents extreme updates while preserving direction
- **Optimization Strategy**
  - **Learning rate:** scales gradient-based updates
  - **AdamW:** Adaptive optimizer with momentum
  - **Weight decay:** Prevents weights from growing too large

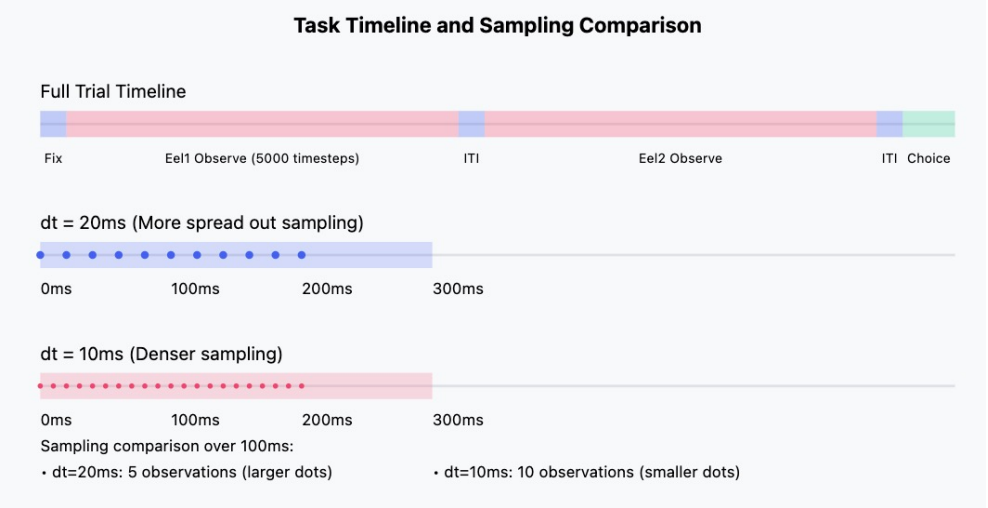
Configuration(config.yaml)

Task

```yaml
# Task Configuration
task:
  dt: 10  # Keep 50hz sampling rate
  timing:
    fixation: 100
    eel1_observe: 5000  # Doubled observation time
    iti1: 100
    eel2_observe: 5000  # Doubled observation time
    iti2: 100
    choice: 100

  # Environment Parameters
  environment:
    eel:
      speed: 0.02
      size: 0.1
      wiggle: 0.01

    fish:
      count: 12
      size: 0.05
      speed:
        slow: 0.002
        fast: 0.05
```

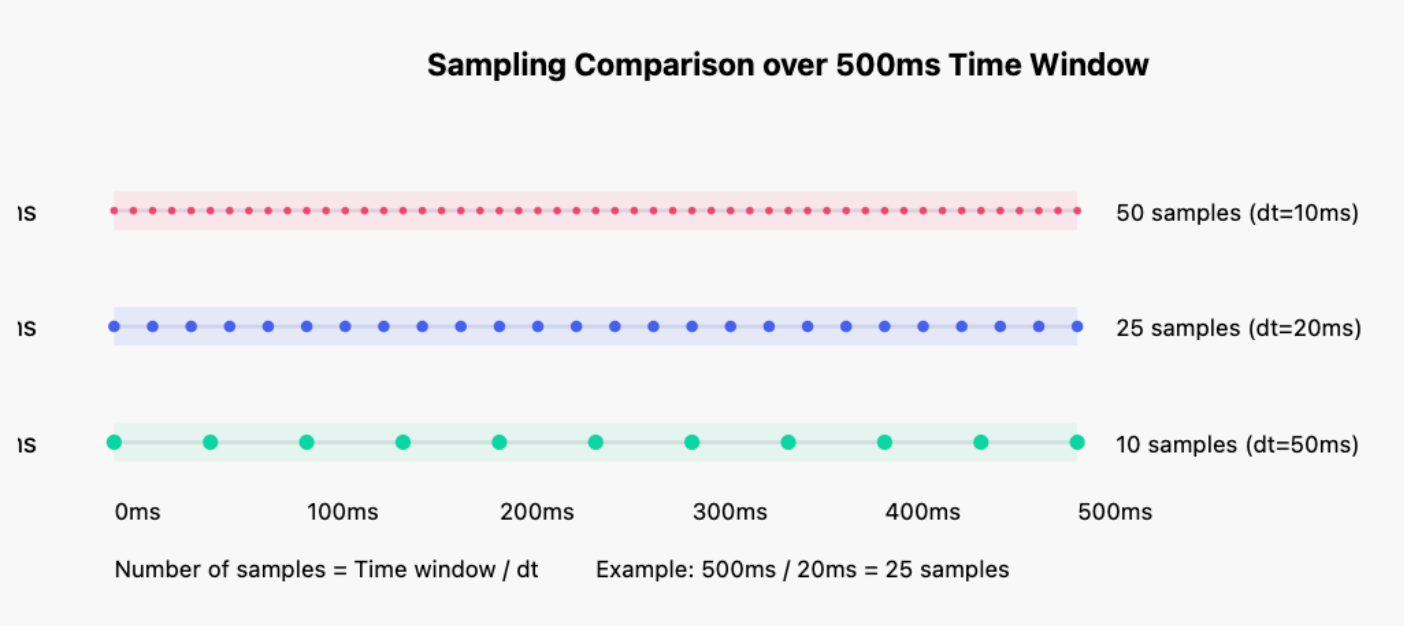


- **dt = 20ms**
  - For fixation period (100 timesteps):
    - 2000ms total duration
    - 100 observations (2000/20)
- **dt = 10ms**
  - For fixation period (100 timesteps):
    - 1000ms total duration
    - 200 observations (1000/10)

Configuration

DT delta time

- Time step used to advance the environment's state in each simulation step.
- Specifies the amount of that passes between updates in the environment, defining the resolution of time.



Code

Trainer

```python
def __init__(self, model, env, config):
    self.model = model          # Neural network model
    self.env = env              # Task environment
    self.config = config        # Configuration parameters

    # Training metrics
    self.training_losses = []
    self.training_accuracies = []
    self.validation_accuracies = []

    # Key parameters from config
    self.num_epochs = config['training']['num_epochs']
    self.batch_size = config['training']['batch_size']
    self.num_trials_per_epoch = config['training']['num_trials_per_epoch']
```

Trainer

train

```python
def train(self):
    for epoch in range(self.num_epochs):
        # 1. Process batches
        for batch_idx in range(num_batches):
            # Get batch of trials
            inputs_batch, labels_batch = self.get_batch()

            # Train on this batch
            loss = self.train_step(inputs_batch, labels_batch)

            # Track performance
            accuracy = self.evaluate_batch()

        # 2. Run validation periodically
        if (epoch + 1) % self.val_frequency == 0:
            val_accuracy = self.validate()

        # 3. Run diagnostics periodically
        if (epoch + 1) % 10 == 0:
            self.run_diagnostics()
```

- Epoch management
- Batch processing
- Periodic validation
- Progress tracking

Trainer

train_step

```python
def train_step(self, inputs, labels):
    """Single training step with improvements."""
    # Zero gradients
    self.optimizer.zero_grad()

    # Normalize & forward pass
    inputs = (inputs - inputs.mean()) / (inputs.std() + 1e-8)
    outputs, _ = self.model(inputs)

    # Loss & backprop
    loss = self.criterion(outputs, labels)
    loss.backward()

    self.optimizer.step()

    return loss.item()
```

- Forward: Get prediction, loss
  Backward: Compute gradients
  Update: Improve weights

Trainer

train_step

```python
def train_step(self, inputs, labels):
    # Zero gradients
    self.optimizer.zero_grad()

    # Forward pass
    outputs, _ = self.model(inputs)

    # Calculate loss
    loss = self.criterion(outputs, labels)

    # Backward pass with gradient clipping
    loss.backward()
    torch.nn.utils.clip_grad_norm_(
        self.model.parameters(),
        self.config['training']['gradient_clip']
    )

    # Update weights
    self.optimizer.step()
```

Trainer

Validation

```python
def validate(self):
    self.model.eval()
    correct = 0
    total = 0

    with torch.no_grad():
        for _ in range(self.validation_trials):
            # Run trial
            trial_info = self.env.new_trial()

            # Get prediction
            inputs = self.process_inputs(self.env.ob)
            outputs, _ = self.model(inputs)

            # Check accuracy
            prediction = self.get_prediction(outputs)
            if prediction == trial_info['ground_truth']:
                correct += 1
            total += 1

    return correct / total
```

- Evaluate generalization
- Track learning progress
- Prevent overfitting

Code

Model

Assignment

Model