



- (51) **International Patent Classification:**
Not classified
- (21) **International Application Number:**
PCT/EP2024/080780
- (22) **International Filing Date:**
30 October 2024 (30.10.2024)
- (25) **Filing Language:** English
- (26) **Publication Language:** English
- (30) **Priority Data:**
2023/0359 30 August 2023 (30.08.2023) IE
2023/0365 31 August 2023 (31.08.2023) IE
- (72) **Inventor; and**
- (71) **Applicant: HYNES, Cathal Ryan** [IE/IE]; 2 Haven View,
Malahide, Dublin, K36 X048 (IE).
- (74) **Agent: KELTIE LIMITED;** Portershed a Dó 15 Market
Street, Galway, H91 TCX3 (IE).

- (81) **Designated States** (*unless otherwise indicated, for every kind of national protection available*): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG,

KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

- (84) **Designated States** (*unless otherwise indicated, for every kind of regional protection available*): ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

- Published:**
- *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*
 - *the filing date of the international application is within two months from the date of expiration of the priority period (Rule 26bis.3)*
 - *in black and white; the international application as filed contained color or greyscale and is available for download from PATENTSCOPE*

(54) **Title:** METHODS AND APPARATUS FOR PROJECTOR CAMERA SYSTEMS

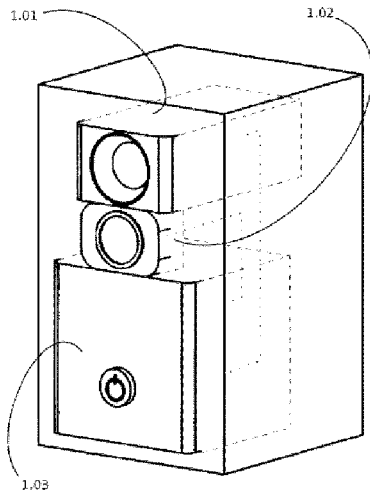


Figure 1

(57) **Abstract:** A method of transforming data transmitted through a physical environment comprises the following steps An emitted signal is emitted into the physical environment. A detected signal is then detected from detection of the emitted signal after interaction with the physical environment. The detected signal is then transformed to modify a further emitted signal. These steps are repeated until a pre-determined objective is met. A system suitable for performing such a method is also described. In particular embodiments, the emitted signal is emitted by a projector, and the detected signal is detected by a camera.

METHODS AND APPARATUS FOR PROJECTOR CAMERA SYSTEMS

Field of Invention

The present invention relates to methods and apparatus for projector camera systems. While embodiments described in detail are particularly relevant to systems including both a projector and a camera, the principles described are applicable to other real-world systems.

Background to Invention

Emitted and detected signals – particularly projected images, and images captured from the environment – are typically processed using classical analytical techniques. It would be desirable to achieve new capabilities by using more modern analytical and data processing techniques, particularly in using additional capabilities provided by machine learning and artificial intelligence generally.

Summary of Invention

In a first aspect, the invention provides a method of transforming data transmitted through a physical environment, the method comprising:

- emitting an emitted signal into the physical environment;
- detecting a detected signal from detection of the emitted signal after interaction with the physical environment;
- transforming the detected signal to modify a further emitted signal; and
- repeating the emitting, detecting, and transforming steps until a predetermined objective is met.

Using iterative processing loops of this kind for emitted and detected signals – as discussed below, this approach is particularly effective with images, but can also be used with signals of other types (such as sound) – can be used to provide a number of new capabilities in image processing and information security. Starting typically with a null signal initially, use of such loops can achieve wide range of desired results. In embodiments, this approach can even be used to develop new types of computing engine.

In one embodiment, the emitted signal is an image emitted by a projector, such as a video projector (in which case the image is part of a video stream), and the detected signal is an image captured by a camera. Image capture may be synchronized with image projection – for a

typical digital video projector the refresh rate may involve refresh approximately every 16ms. The transforming steps can be made at this rate for the most effective response – this involves significant data processing in the case of a typical 1920x1080 pixel projected colour image with a 5360x4600 detected colour image (these are exemplary values for typical systems) – however the loop could also be carried out at some multiple of the refresh rate.

In embodiments, the transforming step is performed using a generator model adapted to predict an emitted signal from a detected signal. This generator model may be a trained machine learning model. A suitable machine learning model may be an artificial neural network – this may be a temporal model, and could be implemented as a convolutional neural network.

In certain embodiments, the machine learning model is trained to blend a predetermined image into the further emitted image. In certain cases, this predetermined image is predetermined text, though it could be any other targeted loss function.

In embodiments, the method may further comprise modifying the generator model in real time – this may in principle be with each detected signal, though in practice it may occur less frequently.

In embodiments, the transforming step may comprise applying a predetermined style to the further emitted signal.

In embodiments, the transforming step may comprise operating on the detected signal using a one-way function. This one-way function may comprise a structured noise generator function. The emitting, detecting and transforming steps may be repeated a predetermined number of times to achieve the predetermined objective. In other embodiments, this process may run continuously. This predetermined objective may comprise providing a cryptographic hash, but may involve any other one-way objective. The output of such a method may be a secure data recording.

In embodiments, the method may be performed in parallel by a plurality of emitter-detector pairs. Here, the plurality of emitter-detector pairs may be configured to act as an adversarial autoencoder, with at least one of the plurality of emitter-detector pairs adversarial to the other emitter-detector pairs. There are various ways in which this structure can be used. In one,

multiple agents achieve an outcome (optionally a one-way function) by shining on a shared scene – this is exemplified by encryption of a “real” scene. Another is emission-recording pair verification via autoencoder. In generation of a cryptographic hash as described above, such an adversarial autoencoder may be adapted to verify integrity of the cryptographic hash.

In further embodiments, an output of the method may be production of a 3D image. This may be achieved by Neural Radiance Fields (NeRF), or by other techniques such as point clouds or depth maps.

In further embodiments, the predetermined objective may be completion of a computation process.

In a second aspect, the invention provides a system for transformation of data transmitted through a physical environment, the system comprising: at least one emitter for emitting an emitted signal into the physical environment; at least one detector for detecting a detected signal from detection of the emitted signal after interaction with the environment; and computing means, wherein the computing means is adapted to transform the detected signal to modify a further emitted signal.

In embodiments, the emitter is a projector and the detector is a camera, and the computing means is a suitably programmed processor. The projector may be a video projector. In such a case, image capture by the camera may be matched to a refresh rate of the video projector.

In embodiments, the system is adapted to perform the method of one or more of the embodiments of the first aspect.

Figures

The following figures are provided with this specification for illustrative purposes.

FIGURE 1

1.01: Projector

1.02: Camera

1.03: Agent

FIGURE 2

2.04: Speaker

2.05: Microphone

2.06: Agent

FIGURE 3

3.07: Stereo Projector

3.08: Right Channel Filtered Camera

3.09: Left Channel Filtered Camera

3.10: Agent

3.11: Microphone

3.12: Speaker

FIGURE 4

4.13: Right Channel Filtered Camera

4.14: Left Channel Filtered Camera

4.15: Stereo Projector

FIGURE 5

5.16: Right Channel Unfiltered Camera

5.17: Left Channel Unfiltered Camera

5.18: Projector

FIGURE 6

6.19: Right Channel Filtered Camera

6.20: Left Channel Filtered Camera

FIGURE 7

7.21: Projector

7.22: Camera

7.23: Agent

FIGURE 8

8.24: Emitter, e.g. Projector or Speaker

8.25: Reactor, e.g. Non-Linear Optical or Acoustic Medium

8.26: Recorder, e.g. Camera or Microphone

FIGURE 9

9.27: Emitter, e.g. Projector or Speaker

9.28: Reactor, e.g. Non-Linear Optical or Acoustic Medium

9.29: Recorder, e.g. Camera or Microphone

9.30: Emitter, e.g. Projector or Speaker

FIGURE 10

10.31: Recorder, e.g. Camera or Microphone

10.32: Reactor, e.g. Non-Linear Optical or Acoustic Medium

10.33: Emitter, e.g. Projector or Speaker

FIGURE 11

11.34: Fibonacci Spiral

11.35: High-Dimensional Recorder Array e.g. Cameras or Microphones

FIGURE 12

12.36: High-Dimensional Recorder Array e.g. Cameras or Microphones

12.37: Smooth Mating Between Recorders and Reactor, Optional Signal Damping e.g. Black Paint or Acoustic Foam

12.38: Reactor, e.g. Mirrored Cavity or Acoustic Cavity

12.39: High-Dimensional Emitter Array e.g. Projectors or Speakers

FIGURE 13

13.40: Low-Dimensional Recorder Array e.g. Camera or Microphone

13.41: Dimensional Reduction Reactor e.g. Mirrored Cavity or Acoustic Cavity

13.42: Optionally Smooth Mating Between Emitters and Reactor, Optional Signal Damping e.g. Black Paint or Acoustic Foam

13.43: High-Dimensional Emitter Array e.g. Projectors or Speakers

FIGURE 14

14.44: Low-Dimensional Emitter Array e.g. Projector or Speaker

14.45: Dimensional Expansion Reactor e.g. Mirrored Cavity or Acoustic Cavity

14.46: Optionally Smooth Mating Between Recorders and Reactor, Optional Signal Damping e.g. Black Paint or Acoustic Foam

14.47: High-Dimensional Recorder Array e.g. Cameras or Microphone

FIGURE 15

15.48: Matrix of Agents, e.g. Array of Projector-Scene-Camera-Agent Loops

15.49: Scene, e.g. Matte White-Coloured Mannequin

15.50: Addressed Subset of Agent elements at $i=1$

15.51: Lower Bound of Addressed Agent elements at $i=1$

15.52: Upper Bound of Addressed Agent elements at $i=1$

15.53: Area of Overlap between Addressed Agent elements between $i=1$ and $i=2$

15.54: Novel Area of Addressed Agents at $i=2$

15.55: Area of overlap between $i=3$ and $i=2$

15.56: Novel Area of Addressed Agents at $i=3$

Specific Description - Introduction

Embodiments of the invention will now be described by way of example with reference to the accompanying figures.

The description takes the following structure:

- Description of the general operating principles of the invention and its embodiments, including discussion of various processing types;
- Provision and explanation of pseudocode for specific processing types; and
- Detailed description and mathematical explanation for specific aspects of processing according to embodiments of the invention.

General Operating Principles

The general approach described is for repeated loops of emission into and detection from an environment of a signal. In most embodiments described below, this is a visual signal, but as is indicated in certain embodiments, it may be a more complex signal (for example, comprising image and sound) or it may be a non-visual signal. Between detection and further emission there is processing of the signal. This may involve use of a machine learning system, such as an artificial neural network (particularly a convolutional neural network). This may be thought of as a series of filters applied to an image (“image” will be used for convenience for the discussion that follows – from the statements above, it is clear that this discussion is not limited to images). These filters may distort, resize or recolour the image – how they do this may be adjusted during a training process.

Training involves manipulating these filters until they provide the desired output for a given input. One approach involves training of these filters to predict a resulting image. In essence, this involves treating the external world as equivalent to one of these filters (or sets of filter). This general approach provides a system that is readily applicable to a wide range of real-world problems.

A glossary of terms and key concepts will be provided below. This is followed by basic implementations of a variety of key concepts, with discussion of applications that become available using this approach. More detailed implementations are then described for specific examples, followed by an extended description with mathematical underpinning.

Glossary of Terms and Key Concepts

Projection Signal: The visual pattern or image emitted by the projector onto the scene.

Captured Image: The image captured by the camera, which includes the projection signal and the influence of the scene.

Generator Model: A machine learning model that predicts projection signals from captured images.

Comparative Loss: A loss function used to measure the difference between the predicted projection signal and the actual projection signal.

Style Function: A function that applies a specific style or transformation to an image or signal.

Text Prompt: A textual input that guides the generator model to blend visual outputs with textual information.

One-Way Hash Function: A cryptographic function that maps input data to a fixed-size hash value, which is computationally infeasible to reverse.

Structured Noise Generator: A function that generates noise patterns based on a given hash value, used to create unique projection signals.

Adversarial Autoencoder: A neural network architecture that combines autoencoding with adversarial training to learn data representations and detect anomalies.

Domain Discriminator: A neural network component that distinguishes between real and generated data.

Comparative Discriminator: A function that compares two inputs and outputs a similarity score, used for loss computation.

Cryptographic Loop: A sequence where each step depends on the cryptographic output of the previous step, ensuring data integrity and security.

Hash Chain: A sequence of hash values where each hash includes the previous hash, creating a chain that verifies the integrity of a sequence of data.

Anomaly Detection: The process of identifying unusual patterns or data points that deviate from the norm.

Backpropagation: A method used in training neural networks where gradients are computed and propagated backward to update model parameters.

Optimizer: An algorithm that adjusts the model's parameters to minimize the loss function.

Fine-tuning: The process of further training a pre-trained model on new data to adapt it to specific tasks or environments.

Modification - First of all, various types of modification process will now be described, with reference to pseudocode. All of these processes are performed with an emitter-detector pair – that is, for an image, a projector-camera pair. In the case of later processes, multiple emitter-detector pairs may be used.

1. Dataset Recording with Projection Signals

```
Import Projection_Signal_Dataset
Import Captured_Image_Dataset
From Projector Import Emit_Projection_Signal
From Camera Import Capture_Image

Initialize count = 0
While count < length(Projection_Signal_Dataset):
    # Emit the current projection signal onto the scene
    Emit_Projection_Signal(Projection_Signal_Dataset[count])

    # Capture the corresponding image from the camera
    Captured_Image = Capture_Image()
    Captured_Image_Dataset[count] = Captured_Image

    count = count + 1
```

This enables the collection of synchronized datasets of projection signal and captured images. This may be employed, for example, with a travelling system – the datasets may represent images as the projector-camera system travels through space. Here, the projector emits known signals onto the scene, and the camera captures images as influenced by the scene and the projection signals. Only images and projection signals are captured in this example – no position data or inertial measurements are recorded (though this is in principle also possible).

2. Training Generator to Predict Emissions from Captured Images

```
Import Captured_Image_Dataset
Import Projection_Signal_Dataset
From Generator Import Initialize_Model, Predict_Emission
From Loss_Functions Import Comparative_Loss
From Optimizer Import Backpropagate, Optimizer

# Initialize the generator model
Generator_Model = Initialize_Model()

# Initialize the optimizer for the generator model
Generator_Optimizer = Optimizer(Generator_Model)

Initialize count = 0
While count < length(Captured_Image_Dataset):
    # Get the current captured image
    Captured_Image = Captured_Image_Dataset[count]

    # Use the generator to predict the emission from the captured image

    Predicted_Emission = Predict_Emission(Generator_Model, Captured_Image)

    # Calculate the loss between the predicted emission and the actual projection
    signal
    Loss = Comparative_Loss(Projection_Signal_Dataset[count], Predicted_Emission)

    # Backpropagate the loss to update the generator model parameters
    Generator_Optimizer.zero_grad()
    Backpropagate(Loss, Generator_Model)
    Generator_Optimizer.step()

    count = count + 1
```

This describes the training of a generator model to predict projection signals – emissions – from the captured images. The generator here takes a captured image as input, and predicts the corresponding projection signal. A loss value is computed by comparison of the predicted emission with the actual projection signal. Model parameters are then updated to minimize this loss using backpropagation and optimization techniques.

3. Continuous Processing and Emitting of Styled Outputs

```

From Generator Import Predict_Emission
From Style_Function Import Apply_Style
From Projector Import Emit_Projection_Signal
From Camera Import Capture_Image

While True:
    # Capture the current image from the camera
    Captured_Image = Capture_Image()

    # Use the generator to predict the emission from the captured image
    Predicted_Emission = Predict_Emission(Generator_Model, Captured_Image)

    # Apply a style function to the predicted emission
    Styled_Output = Apply_Style(Predicted_Emission)

    # Emit the styled output as the new projection signal
    Emit_Projection_Signal(Styled_Output)

```

This generator function continuously processes captured images, applies a style to the predicted emissions, and projects them back onto the scene. Captured images are used to predict emissions, and these are then modified by a style function. These styled outputs are projected onto the scene. A feedback loop results.

4. Training a Model to Blend Emission with Text Prompt

```

Import Captured_Image_Dataset
Import Projection_Signal_Dataset
From Generator Import Initialize_Model, Predict_With_Text_Prompt
From Loss_Functions Import Comparative_Loss, Text_Target_Loss
From Optimizer Import Backpropagate, Optimizer

Define Target_Text_Prompt
Define Weight # A value between 0 and 1 to balance text and emission losses

# Initialize the generator model with text prompt conditioning
Generator_Model = Initialize_Model()

# Initialize the optimizer for the generator model
Generator_Optimizer = Optimizer(Generator_Model)

```

```

Initialize count = 0
While count < length(Captured_Image_Dataset):
    # Get the current captured image
    Captured_Image = Captured_Image_Dataset[count]

    # Use the generator to predict a blended output using the captured image,
    text prompt, and weight
    Blended_Output = Predict_With_Text_Prompt(Generator_Model, Captured_Image,
    Target_Text_Prompt, Weight)

    # Calculate the loss based on the weighted blend of text and emission
    comparisons
    TextLoss = Text_Target_Loss(Target_Text_Prompt, Blended_Output)
    EmissionLoss = Comparative_Loss(Projection_Signal_Dataset[count],
    Blended_Output)
    Loss = Weight * TextLoss + (1 - Weight) * EmissionLoss

    # Backpropagate the loss to update the generator model parameters
    Generator_Optimizer.zero_grad()
    Backpropagate(Loss, Generator_Model)
    Generator_Optimizer.step()

    count = count + 1

```

Using the approach, the model can be trained to blend the predicted emission with a target text prompt – thus combining visual and textual information. The generator here uses the captured image and the text prompt to predict a blended output. A loss function is computed as a weighted sum of text similarity and emission similarity. In this way, the model learns to incorporate the text prompt into the predictions effectively.

5. Fine-tuning the Model on Real-Time Recordings Continuously

```

From Generator Import Initialize_Model, Predict_With_Text_Prompt
From Loss_Functions Import Text_Target_Loss
From Projector Import Emit_Projection_Signal
From Camera Import Capture_Image
From Optimizer Import Backpropagate, Optimizer

Define Target_Text_Prompt
Define Weight # A value between 0 and 1 to balance text loss and any other
losses

# Initialize the generator model with text prompt conditioning
Generator_Model = Initialize_Model()

```

```

# Initialize the optimizer for the generator model
Generator_Optimizer = Optimizer(Generator_Model)

# Emit an initial gray projection signal
Initial_Projection_Signal = Gray_Signal
Emit_Projection_Signal(Initial_Projection_Signal)

# Capture the initial image from the camera
Captured_Image = Capture_Image()

# Start the training loop
While True:
    # Capture the current image from the camera
    Captured_Image = Capture_Image()

    # Apply the text prompt loss function to the captured image
    TextLoss = Text_Target_Loss(Target_Text_Prompt, Captured_Image)

    # Compute the total loss (focusing on text prompt loss applied to captured
    image)
    Loss = TextLoss

    # Backpropagate the loss to update the generator model parameters
    Generator_Optimizer.zero_grad()
    Backpropagate(Loss, Generator_Model)
    Generator_Optimizer.step()
    # Use the generator to predict a new emission based on the captured image and
    text prompt
    Predicted_Emission = Predict_With_Text_Prompt(Generator_Model,
    Captured_Image, Target_Text_Prompt, Weight)

    # Emit the predicted emission as the new projection signal
    Emit_Projection_Signal(Predicted_Emission)

```

This approach enables the model to be fine-tuned in real time using new recordings, with the predictions adapted according to the new data. Here, the model updates its parameters on the fly as it receives new images and projection signals, and the feedback loop allows the system to adapt to changes in the environment, or to any other desired output.

Secure Functions - A further step beyond these uses of a generator as a modifying function is the use of the generator to provide secure output. This may for example use a one-way function to provide a hash output.

6. One-Way Hash Function Loop for Projection Emission

```
Import Hash_Function
Import Structured_Noise_Generator
From Projector Import Emit_Projection_Signal
From Camera Import Capture_Image
From Interlocutor Import Get_Initialization_Vector, Send_Final_Hash
From Dataset Import Emissions_Dataset, Recordings_Dataset, Hashes_Dataset
From Filesystem Import Create_Directory, Save_To_Directory

Define Loop_Count
Define Emissions_Dataset = []
Define Recordings_Dataset = []
Define Hashes_Dataset = []

Initialization_Vector = Get_Initialization_Vector()

# Create directory named after the initialization vector
Directory_Name = Initialization_Vector
Create_Directory(Directory_Name)

Current_Hash = Hash_Function(Initialization_Vector)
Hashes_Dataset.append(Current_Hash)

Initialize count = 0
While count < Loop_Count:
    # Generate structured noise from the current hash
    Structured_Noise = Structured_Noise_Generator(Current_Hash)

    # Emit the structured noise as the projection signal
    Emit_Projection_Signal(Structured_Noise)
    Emissions_Dataset.append(Structured_Noise)

    # Capture the corresponding image from the camera
    Captured_Image = Capture_Image()
    Recordings_Dataset.append(Captured_Image)

    # Update the current hash for the next iteration
    Current_Hash = Hash_Function(Current_Hash, Captured_Image)
    Hashes_Dataset.append(Current_Hash)

    count = count + 1
```

```
# Save datasets in the directory named after the initialization vector
Save_To_Directory(Directory_Name, Emissions_Dataset, "emissions")
Save_To_Directory(Directory_Name, Recordings_Dataset, "recordings")
Save_To_Directory(Directory_Name, Hashes_Dataset, "hashes")

# Send the final hash to the interlocutor
Send_Final_Hash(Current_Hash)
```

Using this approach, a cryptographic loop is created. One-way functions – hash functions – are used to secure the projector/camera system, ensuring privacy-preserving image capture and emission cycles. All data is here saved in a common directory – in this case, named after the initialisation vector for easy traceability.

The loop starts by establishing an initialisation vector (IV) that seeds a structured noise generator and initializes the hash. In each iteration, the emission is derived using this structured noise generator – this may be, for example, a Perlin Noise generator. This structured noise generator is used on the current hash and camera image, which is then emitted, and captured by the camera with the output used to derive a new current hash.

This loop continues for a defined number of iterations, with the final hash shared with an interlocutor to verify the timing of the whole process. Emissions, recordings, and hashes are stored in respective datasets, which are saved in a directory named after the IV for further analysis or verification.

7. Training an Adversarial Autoencoder for Anomaly Detection

```
From Adversarial_Autoencoder_Model Import Initialize_Model
From Dataset Import Emissions_Dataset, Recordings_Dataset
From Preprocessing Import Concatenate_Emission_and_Recording
From Training Import Backpropagate
From Optimizer Import Optimizer
From Loss_Functions Import Reconstruction_Loss_Function,
Discriminator_Loss_Function

# Initialize the adversarial autoencoder model components
Encoder, Generator, Domain_Discriminator = Initialize_Model()

# Initialize optimizers for each component
Encoder_Optimizer = Optimizer(Encoder)
Generator_Optimizer = Optimizer(Generator)
Domain_Discriminator_Optimizer = Optimizer(Domain_Discriminator)

# Prepare the concatenated training data
Concatenated_Data = []
For i in range(length(Recordings_Dataset)):
    # Get the corresponding emission and recording
    Emission = Emissions_Dataset[i]

    Recording = Recordings_Dataset[i]

    # Concatenate the emission and recording
    Concatenated_Pair = Concatenate_Emission_and_Recording(Emission, Recording)
    Concatenated_Data.append(Concatenated_Pair)

# Training loop for adversarial autoencoder
Epochs = 100
For epoch in range(Epochs):
    For data in Concatenated_Data:
        # Forward pass through the encoder and generator to reconstruct the input
        Latent_Representation = Encoder(data)
        Reconstructed_Data = Generator(Latent_Representation)

        # Compute reconstruction loss for encoder and generator
        Reconstruction_Loss = Reconstruction_Loss_Function(data,
Reconstructed_Data)
```

```

# Backpropagate and update encoder and generator
Encoder_Optimizer.zero_grad()
Generator_Optimizer.zero_grad()
Backpropagate(Reconstruction_Loss, Encoder, Generator)
Encoder_Optimizer.step()
Generator_Optimizer.step()

# Forward pass through the domain discriminator for adversarial training
Real_Output = Domain_Discriminator(data)
Fake_Output = Domain_Discriminator(Reconstructed_Data)

# Compute domain discriminator loss
Discriminator_Loss = Discriminator_Loss_Function(Real_Output,
Fake_Output)

# Backpropagate and update domain discriminator
Domain_Discriminator_Optimizer.zero_grad()
Backpropagate(Discriminator_Loss, Domain_Discriminator)
Domain_Discriminator_Optimizer.step()

# Print training progress
Print("Epoch:", epoch, "Reconstruction Loss:", Reconstruction_Loss,
"Discriminator Loss:", Discriminator_Loss)

```

Using this approach, an adversarial encoder can be trained to detect anomalies in the recording and projection process by learning a normal pattern of emission and recording pairs.

To do this, emission and recording datasets are concatenated to create training data representing both projected signals and captured responses. The adversarial encoder consists of an encoder, a generator (a decoder) and a domain discriminator. The encoder and generator together learn how to reconstruct the input data, while the domain discriminator is trained adversarially to distinguish between real and reconstructed data. The training loop optimizes the encoder, generator, and domain discriminator so as to minimize reconstruction loss and so maximize the discriminator's ability to differentiate real from reconstructed data.

8. Verifying Hash Integrity and Analyzing Emission-Recording Pairs

```

From Dataset Import Emissions_Dataset, Recordings_Dataset, Hashes_Dataset
From Hash_Function Import Hash_Function
From Structured_Noise_Generator Import Structured_Noise_Generator
From Adversarial_Autoencoder_Model Import Encoder, Domain_Discriminator
From Preprocessing Import Concatenate_Emission_and_Recording
From Loss_Functions Import Comparative_Discriminator

# Load datasets
Load Emissions_Dataset, Recordings_Dataset, Hashes_Dataset

# Verify hash chain integrity
For i in range(1, length(Hashes_Dataset)):
    Previous_Hash = Hashes_Dataset[i - 1]
    Captured_Image = Recordings_Dataset[i - 1]
    Expected_Hash = Hash_Function(Previous_Hash, Captured_Image)

    If Hashes_Dataset[i] != Expected_Hash:
        Print("Hash verification failed at index:", i)
        Exit()

# Verify that structured noise corresponds to emissions
For i in range(length(Hashes_Dataset)):
    Current_Hash = Hashes_Dataset[i]
    Structured_Noise = Structured_Noise_Generator(Current_Hash)
    Emission = Emissions_Dataset[i]

    # Use the comparative discriminator to verify consistency
    Similarity_Score = Comparative_Discriminator(Structured_Noise, Emission)
    If Similarity_Score < Threshold:
        Print("Structured noise verification failed at index:", i)
        Exit()

# Apply domain discriminator to verify emission-recording pairs
Verification_Results = []
For i in range(length(Emissions_Dataset)):
    Emission = Emissions_Dataset[i]
    Recording = Recordings_Dataset[i]
    Concatenated_Pair = Concatenate_Emission_and_Recording(Emission, Recording)
    Discriminator_Score = Domain_Discriminator(Concatenated_Pair)
    Verification_Results.append(Discriminator_Score)

# Output verification results

Print("Verification Results:", Verification_Results)

```

Using this approach, the integrity of the hash chain can be verified. It is determined whether the structured noise corresponds to the recorded emissions, and the domain discriminator is used to evaluate the authenticity of emission-recording pairs.

The first step is hash chain verification – this involves a check to ensure that each hash in the chain is correctly computed from the previous hash and the associated captured image. The next step is structured noise verification – this involves confirmation that the emissions match the structured noise generated from their respective hashes using a comparative discriminator and a predefined threshold. The following step is domain discriminator evaluation – this uses the trained domain discriminator to assess the authenticity of each emission-recording pair, detecting any anomalies or inconsistencies.

Applications of these techniques include the following:

Artistic Translation and Modification

- Prediction of the camera image from the projector image (e.g. for a fixed scene) allows evaluation of potential projections to assess visual appeal.
- Prediction of the projector image from the camera image may also be used
 - This can be used to improve coloration (by resonance or similar approaches).
 - A style function can be imposed on the image to provide a desired result.

Analytical Techniques

- The filters can transform the image into a label, such as a name or 3D model of a subject.
- Labels may be supervised (and explicit, such as a name) or unsupervised, such as AI generated content (e.g. estimated depth maps of each pixel from the camera).

Verification Techniques

- Use of the techniques set out above may be more effective than high-resolution images of electromagnetic field interactions (which are costly to simulate) or machine learning models to detect tempering (which are slow).
- The approach set out here should be relatively resistant to deep fakes.

One-Way Function

- Information security is an obvious application – asymmetries can be detected and used advantageously.
- Adversaries can be detected and inhibited.
- A secure communication route can be provided.
- Physical world images can be effectively “watermarked” in recording with forgery-resistant timecodes.
- This approach should be relatively resistant to gradient-based attacks.

Computation

- The modification process can be used as part of a computation process.
 - For example, the process may provide a complex shape acting as if it were multiple layers of a neural network.
 - This could act as a one-way function, and would be versatile as it would be possible to add additional “layers”.
 - When paired with a light-field camera, this system is capable of complex calculation.
 - A grid of projector-camera systems could represent large amounts of data.
 - Sparse data – such as a very large Large Language Model – could be represented.

More detailed code will now be described for specific embodiments. Certain of these embodiments provide more detailed versions of examples provided above.

Example 1 - Projector- Camera System with Text-Based Training

This is described stage-by-stage

1. Dataset Recording with Projection Signals

```
BEGIN DatasetRecording

# Initialize datasets
PROJECTION_SIGNAL_DATASET = LoadProjectionSignals() # Known projection signals
CAPTURED_IMAGE_DATASET = EmptyList()

FOR EACH PROJECTION_SIGNAL IN PROJECTION_SIGNAL_DATASET DO
    EmitProjectionSignal(PROJECTION_SIGNAL)
    CAPTURED_IMAGE = CaptureImage()
    Append(CAPTURED_IMAGE_DATASET, CAPTURED_IMAGE)
END FOR

# Save datasets
SaveDataset("projection_signals", PROJECTION_SIGNAL_DATASET)
SaveDataset("captured_images", CAPTURED_IMAGE_DATASET)

END DatasetRecording
```

The projector emits a series of known signals onto the scene, the camera captures images influenced by both the scene and the projected signals, with the captured images and their corresponding projections signals stored in synchronized datasets – these can then provide training material.

2. Training Generator to Predict Emissions from Captured Images

```

BEGIN TrainGenerator

# Load datasets
PROJECTION_SIGNAL_DATASET = LoadDataset("projection_signals")
CAPTURED_IMAGE_DATASET = LoadDataset("captured_images")

# Initialize generator model
INITIALIZE GeneratorModel

# Initialize optimizer
INITIALIZE Optimizer WITH Parameters = GeneratorModel.Parameters

# Training loop
FOR EACH (CAPTURED_IMAGE, PROJECTION_SIGNAL) IN ZipDatasets(CAPTURED_IMAGE_DATASET,
PROJECTION_SIGNAL_DATASET) DO

    # Predict emission from captured image
    PREDICTED_EMISSION = GeneratorModel(CAPTURED_IMAGE)

    # Compute loss
    LOSS = ComparativeLoss(PROJECTION_SIGNAL, PREDICTED_EMISSION)

    # Update generator parameters
    Optimizer.ZeroGrad()
    LOSS.Backward()
    Optimizer.Step()

END FOR

# Save trained generator model
SaveModel(GeneratorModel, "trained_generator.model")

END TrainGenerator

```

Here, the generator is trained to predict the original projection signals based on the captured images. The comparative loss measures the difference between the predicted emission and the actual projection signal. In the training process, the generator's parameters are updated to minimise the loss and so improve the predictive capacity of the generator.

3. Continuous Processing and Emitting of Styled Outputs

```
BEGIN ContinuousProcessing

# Load trained generator model
GeneratorModel = LoadModel("trained_generator.model")

WHILE True DO
    # Capture image from the scene
    CAPTURED_IMAGE = CaptureImage()

    # Predict emission using generator model
    PREDICTED_EMISSION = GeneratorModel(CAPTURED_IMAGE)

    # Apply style to the predicted emission
    STYLED_OUTPUT = ApplyStyle(PREDICTED_EMISSION)

    # Emit the styled output
    EmitProjectionSignal(STYLED_OUTPUT)

END WHILE

END ContinuousProcessing
```

Here, the system continuously captures and processes images. A styling output is applied to the predicted emissions to achieve a desired visual effect. The styled outputs are projected back onto the scene. This creates a dynamic interaction between the environment and the system.

4. Training a Model to Blend Emission with Text Prompt

```

BEGIN TrainModelWithTextPrompt

    # Load datasets
    TEXT_PROMPT_DATASET = LoadTextPrompts() # A dataset of text prompts
    WEIGHT_DATASET = LoadWeights() # Random weights between 0 and 1
    EMISSION_RECORDING_PAIRS_DATASETS = LoadEmissionRecordingPairsDatasets() # Multiple datasets of emission-
    recording pairs

    # Initialize generator model with memory (e.g., transformer architecture)
    INITIALIZE GeneratorWithMemory

    # Initialize optimizer
    INITIALIZE Optimizer WITH Parameters = GeneratorWithMemory.Parameters

    # Training loop
    FOR EACH (TEXT_PROMPT, WEIGHT, EMISSION_RECORDING_PAIRS) IN ZipDatasets(TEXT_PROMPT_DATASET,
    WEIGHT_DATASET, EMISSION_RECORDING_PAIRS_DATASETS) DO

        # Initialize previous blended output
        PREVIOUS_BLENDED_OUTPUT = None

        # Iterate through the text prompt for a specified number of iterations
        FOR ITERATION FROM 1 TO ITERATIONS_PER_PROMPT DO

            # Iterate through emission-recording pairs for the current text prompt
            FOR EACH (CAPTURED_IMAGE, PROJECTION_SIGNAL) IN EMISSION_RECORDING_PAIRS DO

                # Generate blended output using captured image, text prompt, and weight
                BLENDED_OUTPUT = GeneratorWithMemory(CAPTURED_IMAGE, TEXT_PROMPT, WEIGHT)

                # Compute text loss (e.g., using a text similarity metric)
                TEXT_LOSS = TextTargetLoss(TEXT_PROMPT, BLENDED_OUTPUT)

```

```

# Compute emission loss (difference between projected signal and blended output)
EMISSION_LOSS = ComparativeLoss(PROJECTION_SIGNAL, BLENDED_OUTPUT)

# Compute optical flow loss (e.g., using optical flow estimation)
IF PREVIOUS_BLENDED_OUTPUT IS NOT None THEN
    OPTICAL_FLOW_LOSS = OpticalFlowLoss(PREVIOUS_BLENDED_OUTPUT, BLENDED_OUTPUT)
ELSE
    OPTICAL_FLOW_LOSS = 0
END IF

# Compute total loss as weighted sum
TOTAL_LOSS = WEIGHT * TEXT_LOSS + (1 - WEIGHT) * EMISSION_LOSS + OPTICAL_FLOW_LOSS

# Update generator parameters
Optimizer.ZeroGrad()
TOTAL_LOSS.Backward()
Optimizer.Step()

# Update previous blended output
PREVIOUS_BLENDED_OUTPUT = BLENDED_OUTPUT

END FOR

END FOR

END FOR

# Save trained generator with memory model
SaveModel(GeneratorWithMemory, "trained_generator_with_memory.model")

END TrainModelWithTextPrompt

```

Using this approach, the optical flow loss is computed between the current blended output and the previous blended output within the same text prompt iteration. This supports temporal consistency and smooth transition between iterations.

5. Fine-Tuning the Model on Real-Time Recordings Continuously

```

BEGIN ContinuousFineTuning

    # Number of devices (e.g., multiple projectors and cameras)
    NUM_DEVICES = GetNumberOfDevices()

    # Initialize generator models with memory for each device
    GENERATORS = [Initialize GeneratorWithMemory() FOR DEVICE_ID FROM 1 TO NUM_DEVICES]
    OPTIMIZERS = [Initialize Optimizer WITH Parameters = GENERATORS[DEVICE_ID].Parameters FOR DEVICE_ID FROM 1
    TO NUM_DEVICES]

    # Initial projection signal (e.g., neutral gray)
    INITIAL_PROJECTION_SIGNAL = GraySignal()
    FOR DEVICE_ID FROM 1 TO NUM_DEVICES DO
        EmitProjectionSignal(INITIAL_PROJECTION_SIGNAL, DEVICE_ID)
    END FOR

    WHILE True DO

        FOR DEVICE_ID FROM 1 TO NUM_DEVICES DO

            # Initialize previous predicted emission
            PREVIOUS_PREDICTED_EMISSION = None

            # Iterate through the target text prompt for a specified number of iterations
            FOR ITERATION FROM 1 TO ITERATIONS_PER_PROMPT DO

                # Capture image from the device's camera
                CAPTURED_IMAGE = CaptureImage(DEVICE_ID)

                # Generate predicted emission with target text prompt and weight
                PREDICTED_EMISSION = GENERATORS[DEVICE_ID](CAPTURED_IMAGE, TARGET_TEXT_PROMPT, WEIGHT)

                # Emit the predicted emission

```

```

        EmitProjectionSignal(PREDICTED_EMISSION, DEVICE_ID)

        # Capture image after projection
        CAPTURED_IMAGE_AFTER_PROJECTION = CaptureImage(DEVICE_ID)

        # Compute text loss based on the captured image after projection
        TEXT_LOSS = TextTargetLoss(TARGET_TEXT_PROMPT, CAPTURED_IMAGE_AFTER_PROJECTION)

        # Compute optical flow loss (e.g., using optical flow estimation)
        IF PREVIOUS_PREDICTED_EMISSION IS NOT None THEN
            OPTICAL_FLOW_LOSS = OpticalFlowLoss(PREVIOUS_PREDICTED_EMISSION, PREDICTED_EMISSION)
        ELSE
            OPTICAL_FLOW_LOSS = 0
        END IF

        # Compute total loss
        TOTAL_LOSS = TEXT_LOSS + OPTICAL_FLOW_LOSS

        # Update generator parameters for the device
        OPTIMIZERS[DEVICE_ID].ZeroGrad()
        TOTAL_LOSS.Backward()
        OPTIMIZERS[DEVICE_ID].Step()

        # Update previous predicted emission
        PREVIOUS_PREDICTED_EMISSION = PREDICTED_EMISSION

    END FOR

END FOR

# Synchronize models across devices to share learned improvements
SynchronizeModels(GENERATORS)

END WHILE

END ContinuousFineTuning

```

This allows for continuous fine tuning using this approach.

Example 2 – Secure Recording

1. Secure Recording with One-Way Hash Function Loop

```

# SecureRecording.pseudo

BEGIN SecureRecording

    DEFINE LOOP_COUNT AS Integer
    SET LOOP_COUNT = 100 # Example value

    # Get initialization vector and compute initial hash
    INITIALIZATION_VECTOR = GetInitializationVector()
    CURRENT_HASH = HashFunction(INITIALIZATION_VECTOR)

    # Create directory named after the initialization vector
    DIRECTORY_NAME = "recordings_" + INITIALIZATION_VECTOR
    CreateDirectory(DIRECTORY_NAME)

    # Initialize datasets
    EMISSIONS_DATASET = EmptyList()
    RECORDINGS_DATASET = EmptyList()
    HASHES_DATASET = EmptyList()

    # Recording loop
    FOR i FROM 1 TO LOOP_COUNT DO

        # Generate structured noise based on the current hash
        STRUCTURED_NOISE = StructuredNoiseGenerator(CURRENT_HASH)

        # Emit projection signal and capture image
        EmitProjectionSignal(STRUCTURED_NOISE)
        CAPTURED_IMAGE = CaptureImage()

        # Append data to datasets
        Append(EMISSIONS_DATASET, STRUCTURED_NOISE)
        Append(RECORDINGS_DATASET, CAPTURED_IMAGE)
        Append(HASHES_DATASET, CURRENT_HASH)

        # Update hash with the captured image
        CURRENT_HASH = HashFunction(CURRENT_HASH, CAPTURED_IMAGE)

    END FOR

    # Send final hash to interlocutor
    SendFinalHash(CURRENT_HASH)

    # Save datasets to directory
    SaveToDirectory(DIRECTORY_NAME, EMISSIONS_DATASET, "emissions")
    SaveToDirectory(DIRECTORY_NAME, RECORDINGS_DATASET, "recordings")
    SaveToDirectory(DIRECTORY_NAME, HASHES_DATASET, "hashes")

END SecureRecording

```

The hash chain ensures any tampering with the data can be detected, ensuring data integrity. Emissions are linked to the hash chain via structured noise generation. Emissions, recordings, and hashes are stored for verification.

2. Training the Adversarial Autoencoder with Emphasis on the Discriminator

```
# TrainAutoencoder.pseudo

BEGIN TrainAutoencoder

    # Hyperparameters
    SET EPOCHS = 50
    SET BATCH_SIZE = 32
    SET LATENT_DIM = 128

    # Load datasets
    (EMISSIONS_DATASET, RECORDINGS_DATASET) = LoadDatasets(DIRECTORY_NAME)

    # Prepare training data
    TRAINING_DATA = ZipDatasets(EMISSIONS_DATASET, RECORDINGS_DATASET)

    # Initialize Autoencoder components
    INITIALIZE Encoder WITH LatentDimension = LATENT_DIM
    INITIALIZE Decoder WITH LatentDimension = LATENT_DIM

    # Initialize Discriminator (operating on data space)
    INITIALIZE Discriminator WITH InputShape = DataShape # Input is concatenated
    emission and recording

    # Initialize Optimizers
    INITIALIZE OptimizerEncoder
    INITIALIZE OptimizerDecoder
    INITIALIZE OptimizerDiscriminator

    # Training loop
    FOR epoch FROM 1 TO EPOCHS DO

        BATCHES = CreateBatches(TRAINING_DATA, BATCH_SIZE)

        FOR EACH BATCH IN BATCHES DO

            (EMISSIONS_BATCH, RECORDINGS_BATCH) = UnzipBatch(BATCH)

            # Concatenate Emissions and Recordings to form the real data batch
            REAL_DATA_BATCH = Concatenate(EMISSIONS_BATCH, RECORDINGS_BATCH)

            # -----
            # Train Discriminator
            # -----

            # Generate reconstructed data
```

```

LATENT_REPRESENTATIONS = Encoder(REAL_DATA_BATCH)
RECONSTRUCTED_DATA = Decoder(LATENT_REPRESENTATIONS)

# Discriminator predictions
PRED_REAL = Discriminator(REAL_DATA_BATCH)
PRED_FAKE = Discriminator(RECONSTRUCTED_DATA.Detach())

# Labels
LABEL_REAL = OnesVector(Size = BatchSize)
LABEL_FAKE = ZerosVector(Size = BatchSize)

# Compute discriminator loss
LOSS_DISCRIMINATOR_REAL = ComputeLoss(PRED_REAL, LABEL_REAL)
LOSS_DISCRIMINATOR_FAKE = ComputeLoss(PRED_FAKE, LABEL_FAKE)
LOSS_DISCRIMINATOR = (LOSS_DISCRIMINATOR_REAL +
LOSS_DISCRIMINATOR_FAKE) / 2

# Update discriminator
OptimizerDiscriminator.ZeroGrad()
LOSS_DISCRIMINATOR.Backward()
OptimizerDiscriminator.Step()

# -----
# Train Generator (Encoder + Decoder)
# -----

# Generator aims to fool the discriminator
PRED_FAKE_FOR_GEN = Discriminator(RECONSTRUCTED_DATA)

# Compute generator adversarial loss
LOSS_GENERATOR_ADV = ComputeLoss(PRED_FAKE_FOR_GEN, LABEL_REAL)

# Compute reconstruction loss
LOSS_RECONSTRUCTION = ComputeReconstructionLoss(REAL_DATA_BATCH,
RECONSTRUCTED_DATA)

# Total generator loss
LOSS_GENERATOR = LOSS_GENERATOR_ADV + LOSS_RECONSTRUCTION

# Update encoder and decoder
OptimizerEncoder.ZeroGrad()
OptimizerDecoder.ZeroGrad()
LOSS_GENERATOR.Backward()
OptimizerEncoder.Step()
OptimizerDecoder.Step()

# Optionally display progress
DisplayProgress(Epoch = epoch, ReconstructionLoss =
LOSS_RECONSTRUCTION, GeneratorLoss = LOSS_GENERATOR_ADV, DiscriminatorLoss =
LOSS_DISCRIMINATOR)

END FOR

END FOR

# Save models

```

```

SaveModel(Encoder, "trained_encoder.model")
SaveModel(Decoder, "trained_decoder.model")
SaveModel(Discriminator, "trained_discriminator.model")

END TrainAutoencoder

```

Here, the discriminator is the main focus – this operates on the data space (concatenated emissions and recordings). It is trained to distinguish between real data and reconstructed (fake) data. Training uses discriminator loss to drive the discriminator to correctly classify real and fake data. Generator loss is used so that the encoder and decoder (generator) aim to reconstruct data that fools the discriminator (making it classify reconstructed data as real). Reconstruction loss is used to ensure that the autoencoder reconstructs the input accurately. In the training process, there is alternation of training between the discriminator and the generator (encoder and decoder). The discriminator's performance directly influences the generator's training, emphasizing its importance.

3. Verifying Hash Integrity, Emission-Recording Consistency, and Using the Discriminator for Anomaly Detection

```

# Verification.pseudo

BEGIN Verification

    # Thresholds
    SET SIMILARITY_THRESHOLD = 0.95
    SET DISCRIMINATOR_THRESHOLD = 0.5 # Threshold for discriminator output
    indicating anomaly

    # Load datasets
    (EMISSIONS_DATASET, RECORDINGS_DATASET, HASHES_DATASET) =
    LoadDatasets(DIRECTORY_NAME)
    INITIALIZATION_VECTOR =
    ExtractInitializationVectorFromDirectoryName(DIRECTORY_NAME)

    # Verify the initial hash
    EXPECTED_INITIAL_HASH = HashFunction(INITIALIZATION_VECTOR)
    IF HASHES_DATASET[0] ≠ EXPECTED_INITIAL_HASH THEN
        RAISE Error("Initial hash verification failed.")
    END IF

    # Verify hash chain integrity
    FOR i FROM 1 TO Length(HASHES_DATASET) - 1 DO

```

```

PREVIOUS_HASH = HASHES_DATASET[i - 1]
CAPTURED_IMAGE = RECORDINGS_DATASET[i - 1]
EXPECTED_HASH = HashFunction(PREVIOUS_HASH, CAPTURED_IMAGE)

IF HASHES_DATASET[i] ≠ EXPECTED_HASH THEN
    RAISE Error("Hash chain verification failed at index: " + i)
END IF
END FOR

# Verify emission-structured noise consistency
FOR i FROM 0 TO Length(EMISSIONS_DATASET) - 1 DO
    EMISSION = EMISSIONS_DATASET[i]
    CURRENT_HASH = HASHES_DATASET[i]
    STRUCTURED_NOISE = StructuredNoiseGenerator(CURRENT_HASH)

    SIMILARITY_SCORE = ComputeSimilarityScore(EMISSION, STRUCTURED_NOISE)
    IF SIMILARITY_SCORE < SIMILARITY_THRESHOLD THEN
        RAISE Error("Emission-Structured Noise consistency failed at index: "
+ 1)
    END IF
END FOR

# Load trained models
Encoder = LoadModel("trained_encoder.model")
Decoder = LoadModel("trained_decoder.model")
Discriminator = LoadModel("trained_discriminator.model")

SetModelsToEvaluationMode(Encoder, Decoder, Discriminator)

# Anomaly detection
FOR i FROM 0 TO Length(EMISSIONS_DATASET) - 1 DO
    EMISSION = EMISSIONS_DATASET[i]
    RECORDING = RECORDINGS_DATASET[i]

    # Concatenate emission and recording
    REAL_DATA = Concatenate(EMISSION, RECORDING)

    # Generate reconstructed data
    LATENT_REPRESENTATION = Encoder(REAL_DATA)
    RECONSTRUCTED_DATA = Decoder(LATENT_REPRESENTATION)

    # Discriminator output on reconstructed data
    DISCRIMINATOR_OUTPUT = Discriminator(RECONSTRUCTED_DATA)

    # Anomaly score (lower output indicates higher likelihood of anomaly)
    ANOMALY_SCORE = 1 - DISCRIMINATOR_OUTPUT # Assuming output is between 0
and 1

    # Check for anomaly
    IF DISCRIMINATOR_OUTPUT < DISCRIMINATOR_THRESHOLD THEN
        Print("Anomaly detected at index: " + i + ", Anomaly Score: " +
ANOMALY_SCORE)
    END IF

    # Optionally, use reconstruction error

```

```
RECONSTRUCTION_ERROR = ComputeReconstructionLoss(REAL_DATA,
RECONSTRUCTED_DATA)

# Combine scores if needed
COMBINED_ANOMALY_SCORE = (ANOMALY_SCORE + RECONSTRUCTION_ERROR) / 2

# Output results
Print("Index: " + i + ", Discriminator Output: " + DISCRIMINATOR_OUTPUT +
", Reconstruction Error: " + RECONSTRUCTION_ERROR + ", Combined Score: " +
COMBINED_ANOMALY_SCORE)

END FOR

END Verification
```

The discriminator evaluates reconstructed data. A lower output from the discriminator suggests the data is anomalous. The discriminator's output is the primary indicator of anomalies. Reconstruction error is used as supplementary information.

In other embodiments, this architecture can be used but developed by replacing or augmenting the discriminator with a pixelwise version.

More generally, the discriminator is crucial for guiding the generator to produce realistic reconstructions. By focusing on fooling the discriminator, the generator improves the quality of reconstructed data. In the verification phase, the discriminator serves as the primary tool for anomaly detection. Its output reflects the likelihood of data being normal or anomalous.

In using both discriminator and reconstruction loss, the discriminator output provides a direct measure of how 'real' the reconstructed data appears. Anomalies are expected to produce reconstructions that the discriminator flags as fake. Reconstruction error measures how well the autoencoder reconstructs the input data. High errors indicate discrepancies from normal data patterns.

Using both metrics enhances the robustness of anomaly detection, allowing for capturing anomalies that may not be evident through one metric alone. The separation of components (encoder, decoder and discriminator) in a modular design allows for easy modification and addition.

Example 3 – Cryptography

```

# Generate shared secret key for Alice, Bob, Charlie, and Diana
SECRET_KEY = GenerateSecretKey()

# Provide each agent with a view of the scene with no projection and no message
on the e-ink screen
CAMERA_IMAGE_ALICE = CaptureImage_Alice()
CAMERA_IMAGE_BOB = CaptureImage_Bob()
CAMERA_IMAGE_CHARLIE = CaptureImage_Charlie()
CAMERA_IMAGE_DIANA = CaptureImage_Diana()
CAMERA_IMAGE_EVE = CaptureImage_Eve()

PROJECTION_SIGNAL_ALICE, _ =
AGENT_MODEL_ALICE.GenerateProjectionAndGuess(SECRET_KEY, CAMERA_IMAGE_ALICE)
PROJECTION_SIGNAL_BOB, _ = AGENT_MODEL_BOB.GenerateProjectionAndGuess(SECRET_KEY,
CAMERA_IMAGE_BOB)
PROJECTION_SIGNAL_CHARLIE, _ =
AGENT_MODEL_CHARLIE.GenerateProjectionAndGuess(SECRET_KEY, CAMERA_IMAGE_CHARLIE)
PROJECTION_SIGNAL_DIANA, _ =
AGENT_MODEL_DIANA.GenerateProjectionAndGuess(SECRET_KEY, CAMERA_IMAGE_DIANA)
PROJECTION_SIGNAL_EVE, _ =
AGENT_MODEL_EVE.GenerateProjectionAndGuess(CAMERA_IMAGE_EVE)

ProjectSignal_Alice(PROJECTION_SIGNAL_ALICE)
ProjectSignal_Bob(PROJECTION_SIGNAL_BOB)
ProjectSignal_Charlie(PROJECTION_SIGNAL_CHARLIE)
ProjectSignal_Diana(PROJECTION_SIGNAL_DIANA)
ProjectSignal_Eve(PROJECTION_SIGNAL_EVE)

for plaintext_iteration in range(1, NUM_PLAINTEXT_ITERATIONS + 1):
    PLAINTEXT = GeneratePlaintext()
    PREVIOUS_PLAINTEXT = None

    for iteration in range(1, NUM_ITERATIONS_PER_PLAINTEXT + 1):
        # Display the plaintext on the e-ink screen
        DisplayOnEInkScreen(PLAINTEXT)

        # Each agent individually captures an image of the e-ink screen before
        projecting
        CAMERA_IMAGE_ALICE = CaptureImage_Alice()
        CAMERA_IMAGE_BOB = CaptureImage_Bob()
        CAMERA_IMAGE_CHARLIE = CaptureImage_Charlie()
        CAMERA_IMAGE_DIANA = CaptureImage_Diana()
        CAMERA_IMAGE_EVE = CaptureImage_Eve()

        # Each agent individually generates their projection and guess using
        their camera image
        PROJECTION_SIGNAL_ALICE, DECRYPTION_ALICE =
AGENT_MODEL_ALICE.GenerateProjectionAndGuess(SECRET_KEY, CAMERA_IMAGE_ALICE)
        PROJECTION_SIGNAL_BOB, DECRYPTION_BOB =
AGENT_MODEL_BOB.GenerateProjectionAndGuess(SECRET_KEY, CAMERA_IMAGE_BOB)
        PROJECTION_SIGNAL_CHARLIE, DECRYPTION_CHARLIE =
AGENT_MODEL_CHARLIE.GenerateProjectionAndGuess(SECRET_KEY, CAMERA_IMAGE_CHARLIE)

```

```

    PROJECTION_SIGNAL_DIANA, DECRYPTION_DIANA =
AGENT_MODEL_DIANA.GenerateProjectionAndGuess(SECRET_KEY, CAMERA_IMAGE_DIANA)
    PROJECTION_SIGNAL_EVE, DECRYPTION_EVE =
AGENT_MODEL_EVE.GenerateProjectionAndGuess(CAMERA_IMAGE_EVE)

    # Each agent individually projects their signal onto the e-ink screen
    ProjectSignal_Alice(PROJECTION_SIGNAL_ALICE)
    ProjectSignal_Bob(PROJECTION_SIGNAL_BOB)
    ProjectSignal_Charlie(PROJECTION_SIGNAL_CHARLIE)
    ProjectSignal_Diana(PROJECTION_SIGNAL_DIANA)
    ProjectSignal_Eve(PROJECTION_SIGNAL_EVE)

    if PREVIOUS_PLAINTEXT is not None:
        # Compute losses for each agent using the previous plaintext
        LOSS_ALICE = ComputeLoss(PREVIOUS_PLAINTEXT, DECRYPTION_ALICE)
        LOSS_BOB = ComputeLoss(PREVIOUS_PLAINTEXT, DECRYPTION_BOB)
        LOSS_CHARLIE = ComputeLoss(PREVIOUS_PLAINTEXT, DECRYPTION_CHARLIE)
        LOSS_DIANA = ComputeLoss(PREVIOUS_PLAINTEXT, DECRYPTION_DIANA)
        LOSS_EVE = ComputeLoss(PREVIOUS_PLAINTEXT, DECRYPTION_EVE)

        # Alice, Bob, Charlie, and Diana minimize their own loss, each
        other's loss, and maximize Eve's loss
        LOSS_COOPERATORS = LOSS_ALICE + LOSS_BOB + LOSS_CHARLIE + LOSS_DIANA
        - LOSS_EVE

        # Eve minimizes her own loss and maximizes the loss of Alice, Bob,
        Charlie, and Diana
        LOSS_EVE = LOSS_EVE - (LOSS_ALICE + LOSS_BOB + LOSS_CHARLIE +
        LOSS_DIANA)

        # Update agent models
        AGENT_MODEL_ALICE = UpdateModel(AGENT_MODEL_ALICE, LOSS_COOPERATORS,
        OPTIMIZER_ALICE)
        AGENT_MODEL_BOB = UpdateModel(AGENT_MODEL_BOB, LOSS_COOPERATORS,
        OPTIMIZER_BOB)
        AGENT_MODEL_CHARLIE = UpdateModel(AGENT_MODEL_CHARLIE,
        LOSS_COOPERATORS, OPTIMIZER_CHARLIE)
        AGENT_MODEL_DIANA = UpdateModel(AGENT_MODEL_DIANA, LOSS_COOPERATORS,
        OPTIMIZER_DIANA)
        AGENT_MODEL_EVE = UpdateModel(AGENT_MODEL_EVE, LOSS_EVE,
        OPTIMIZER_EVE)

        # Each agent individually generates their projection and guess using
        their camera image
        PROJECTION_SIGNAL_ALICE, DECRYPTION_ALICE =
AGENT_MODEL_ALICE.GenerateProjectionAndGuess(SECRET_KEY, CAMERA_IMAGE_ALICE)
        PROJECTION_SIGNAL_BOB, DECRYPTION_BOB =
AGENT_MODEL_BOB.GenerateProjectionAndGuess(SECRET_KEY, CAMERA_IMAGE_BOB)
        PROJECTION_SIGNAL_CHARLIE, DECRYPTION_CHARLIE =
AGENT_MODEL_CHARLIE.GenerateProjectionAndGuess(SECRET_KEY, CAMERA_IMAGE_CHARLIE)
        PROJECTION_SIGNAL_DIANA, DECRYPTION_DIANA =
AGENT_MODEL_DIANA.GenerateProjectionAndGuess(SECRET_KEY, CAMERA_IMAGE_DIANA)
        PROJECTION_SIGNAL_EVE, DECRYPTION_EVE =
AGENT_MODEL_EVE.GenerateProjectionAndGuess(CAMERA_IMAGE_EVE)

        # Each agent individually projects their signal onto the e-ink screen

```

```
ProjectSignal_Alice(PROJECTION_SIGNAL_ALICE)
ProjectSignal_Bob(PROJECTION_SIGNAL_BOB)
ProjectSignal_Charlie(PROJECTION_SIGNAL_CHARLIE)
ProjectSignal_Diana(PROJECTION_SIGNAL_DIANA)
ProjectSignal_Eve(PROJECTION_SIGNAL_EVE)

# Update the previous plaintext for the next iteration
PREVIOUS_PLAINTEXT = PLAINTEXT
```

This process involves shared secret key generation, initial camera image capture, and initial projection and guess generation steps. The outer loop begins, iterating for a specified number of plaintext iterations

(NUM_PLAINTEXT_ITERATIONS).

In each plaintext iteration, a new plaintext is generated using the GeneratePlaintext() function. The PREVIOUS_PLAINTEXT variable is set to None at the beginning of each plaintext iteration.

The inner loop begins, iterating for a specified number of iterations (NUM_ITERATIONS_PER_PLAINTEXT) for each plaintext.

The inner loop follows these steps:

The current plaintext is displayed on an e-ink screen.

Each agent captures an image of the e-ink screen before projecting.

The agents generate their projections and guesses using their camera images.

The agents project their signals onto the e-ink screen.

If there is a previous plaintext available (from the previous iteration of the inner loop), the agents compute their losses based on their decryption of the previous plaintext.

This process uses an adversarial model. The cooperating agents (Alice, Bob, Charlie, and Diana) aim to minimize their collective loss, which includes their individual losses and Eve's loss subtracted. This encourages them to work together and against Eve. Eve aims to minimize her own loss and maximize the loss of the cooperating agents. She does this by subtracting the sum of the cooperating agents' losses from her own loss.

The agent models are updated based on their respective losses and optimizers. The cooperating agents update their models using the collective loss, while Eve updates her model using her own modified loss.

The current plaintext is stored as the previous plaintext for the next iteration of the inner loop. The updated models are used to generate new emissions. After the inner loop completes, the outer loop generates a new plaintext, and the process repeats for the specified number of plaintext iterations.

Example 4 – Verification

1. Training Script

```

BEGIN TrainingReactorVerification

    # Initialize networks and components
    Initialize EMISSION_NETWORK
    Initialize RECORDING_NETWORK
    Initialize EMISSION_PROJECTOR
    Initialize RECORDING_PROJECTOR
    Initialize INNER_CAMERA
    Initialize DISCRIMINATOR
    Initialize ADVERSARIAL_GENERATOR

    # Load secured emission-recording pairs
    SECURED_PAIRS = LoadSecuredPairs()

    WHILE training not converged DO

        FOR each PAIR in SECURED_PAIRS DO

            # ---- Generate Real Inner Camera Image ----

            # Process emission and recording through their networks
            PROCESSED_EMISSION = EMISSION_NETWORK(Process PAIR.Emission)
            PROCESSED_RECORDING = RECORDING_NETWORK(Process PAIR.Recording)

            # Project processed outputs via inner projectors
            EMISSION_PROJECTOR(Project PROCESSED_EMISSION)
            RECORDING_PROJECTOR(Project PROCESSED_RECORDING)

            # Inner camera captures the combined projection
            REAL_IMAGE = INNER_CAMERA.Capture()

            # ---- Generate Fake Inner Camera Image ----

            # Adversarial Generator produces a fake inner camera image
            FAKE_IMAGE = ADVERSARIAL_GENERATOR(Generate noise input)

            # ---- Train Discriminator ----

            # Discriminator evaluates real and fake images
            D_REAL = DISCRIMINATOR(Evaluate REAL_IMAGE)
            D_FAKE = DISCRIMINATOR(Evaluate FAKE_IMAGE)

            # Compute discriminator loss
            DISCRIMINATOR_LOSS = ComputeLoss(D_REAL, RealLabel) +
            ComputeLoss(D_FAKE, FakeLabel)

            # Update discriminator to minimize loss
            Update DISCRIMINATOR using DISCRIMINATOR_LOSS

```

```

# ---- Train Adversarial Generator ----

# Adversarial Generator attempts to fool the discriminator
D_FAKE_FOR_GENERATOR = DISCRIMINATOR(Evaluate FAKE_IMAGE)

# Compute adversarial generator loss
ADVERSARIAL_GENERATOR_LOSS = ComputeLoss(D_FAKE_FOR_GENERATOR,
RealLabel)

# Update Adversarial Generator to minimize loss
Update ADVERSARIAL_GENERATOR using ADVERSARIAL_GENERATOR_LOSS

# ---- Train Emission and Recording Networks ----

# Discriminator evaluates the real inner camera image
D_REAL_FOR_NETWORKS = DISCRIMINATOR(Evaluate REAL_IMAGE)

# Compute adversarial loss for emission and recording networks
NETWORKS_ADV_LOSS = ComputeLoss(D_REAL_FOR_NETWORKS, RealLabel)

# Update Emission Network and Recording Network to minimize
adversarial loss
Update EMISSION_NETWORK, RECORDING_NETWORK using NETWORKS_ADV_LOSS

END FOR

END WHILE

END TrainingReactorVerification

```

2. Testing Script

```

BEGIN TestingReactorVerification

# Load trained networks and components
Load EMISSION_NETWORK, RECORDING_NETWORK
Load EMISSION_PROJECTOR, RECORDING_PROJECTOR
Load INNER_CAMERA, DISCRIMINATOR

# Load test emission-recording pairs
TEST_PAIRS = LoadTestPairs()

FOR each PAIR in TEST_PAIRS DO

# Process emission and recording through their networks
PROCESSED_EMISSION = EMISSION_NETWORK(Process PAIR.Emission)
PROCESSED_RECORDING = RECORDING_NETWORK(Process PAIR.Recording)

# Project processed outputs via inner projectors
EMISSION_PROJECTOR(Project PROCESSED_EMISSION)
RECORDING_PROJECTOR(Project PROCESSED_RECORDING)

# Inner camera captures the combined projection
INNER_CAMERA_IMAGE = INNER_CAMERA.Capture()

```

```

# Discriminator evaluates the inner camera image
D_INNER_CAMERA_IMAGE = DISCRIMINATOR(Evaluate INNER_CAMERA_IMAGE)

# Verification decision based on discriminator output
IF D_INNER_CAMERA_IMAGE indicates real THEN
    VERIFICATION_RESULT = "Verified"
ELSE
    VERIFICATION_RESULT = "Not Verified"
END IF

# Output the verification result for the pair
Output VERIFICATION_RESULT for PAIR

END FOR

END TestingReactorVerification

```

This is an embodiment of a reactor for recording verification. Here, the focus is solely on the adversarial loss, without considering the reconstruction loss. The system utilizes an adversarial generator to create fake inner camera images that challenge the verification process.

During the training phase, the adversarial generator produces fake inner camera images, while the discriminator learns to differentiate between real and fake images. The emission network and recording network are trained to process the emission and recording data, respectively, and their outputs are projected into the passive reactor using the corresponding inner projectors. The inner camera captures the combined projection, resulting in the real inner camera image.

The training process involves updating the networks based on the adversarial loss:

The discriminator is updated to minimize the discriminator loss, which measures its ability to correctly classify real and fake images.

The adversarial generator is updated to minimize the adversarial generator loss, which measures its ability to fool the discriminator by generating realistic fake images.

The emission network and recording network are updated to minimize the adversarial loss, which measures their ability to produce real inner camera images that the discriminator considers as real.

By training the networks in this adversarial manner, the system becomes more robust in detecting and rejecting fake inner camera images.

During the testing phase, new emission-recording pairs are processed through the trained

networks. The inner camera captures the combined projection of the processed emission and recording data. The discriminator evaluates the captured inner camera image and determines whether it is real or fake. If the discriminator indicates that the image is real, the emission-recording pair is considered "Verified." Otherwise, it is marked as "Not Verified."

By relying solely on the adversarial loss and the discriminator's assessment, the system effectively verifies the authenticity of the emission-recording pairs, rejecting any pairs that involve fake inner camera images.

This approach enhances the security and reliability of the recording verification process by focusing on the adversarial aspect and eliminating the need for reconstruction loss.

Example 5 – 3D Image Generation

The approach described here uses Neural Radiance Fields (NeRF).

```

BEGIN UnsupervisedNeRFTraining

    # Load the emission-recording pairs dataset
    EMISSION_RECORDING_PAIRS = LoadEmissionRecordingPairs("path/to/data")
    DATASET_SIZE = Length(EMISSION_RECORDING_PAIRS)

    # Initialize camera parameters
    IMAGE_WIDTH, IMAGE_HEIGHT = GetImageDimensions(EMISSION_RECORDING_PAIRS)
    FOCAL_LENGTH = InitializeFocalLength()
    CAMERA_INTRINSICS = BuildCameraIntrinsicsMatrix(FOCAL_LENGTH, IMAGE_WIDTH,
    IMAGE_HEIGHT)
    CAMERA_EXTRINSICS = InitializeCameraExtrinsics(DATASET_SIZE)

    # Initialize NeRF model
    INPUT_CHANNELS = 3 + LENGTH(EMISSION) # 3D coordinates + emission signal
    dimensions
    OUTPUT_CHANNELS = 4 # RGB color + density
    NUM_LAYERS = 8
    HIDDEN_UNITS = 256
    NeRF_Model = InitializeNeRFModel(INPUT_CHANNELS, OUTPUT_CHANNELS, NUM_LAYERS,
    HIDDEN_UNITS)

    # Initialize optimizers for NeRF model and camera parameters
    LEARNING_RATE = 0.0005
    Optimizer_NeRF = InitializeAdam(Parameters = NeRF_Model.Parameters,
    LearningRate = LEARNING_RATE)
    Optimizer_Intrinsics = InitializeAdam(Parameters = CAMERA_INTRINSICS,
    LearningRate = LEARNING_RATE)
    Optimizer_Extrinsics = InitializeAdam(Parameters = CAMERA_EXTRINSICS,
    LearningRate = LEARNING_RATE)

    # Set number of epochs and iterations per epoch
    NUM_EPOCHS = 100
    ITERATIONS_PER_EPOCH = 1000

    # Training loop
    FOR Epoch FROM 1 TO NUM_EPOCHS DO

        TOTAL_LOSS = 0

        FOR Iteration FROM 1 TO ITERATIONS_PER_EPOCH DO

            # Randomly sample a data point
            i = RandomInteger(1, DATASET_SIZE)
            (EMISSION, RECORDING) = EMISSION_RECORDING_PAIRS[i]

            # Get camera parameters for current data point

```

```

    INTRINSIC = CAMERA_INTRINSICS
    EXTRINSIC = CAMERA_EXTRINSICS[i]

    # Generate rays for each pixel of the image
    RAYS_ORIGIN, RAYS_DIRECTION = GenerateRays(INTRINSIC, EXTRINSIC,
IMAGE_WIDTH, IMAGE_HEIGHT)

    # Render image by ray marching through the scene
    RENDERED_IMAGE = RenderImageFromNeRF(NeRF_Model, RAYS_ORIGIN,
RAYS_DIRECTION, EMISSION)

    # Compute reconstruction loss between rendered image and recording
    LOSS = ComputeMeanSquaredError(RENDERED_IMAGE, RECORDING)
    TOTAL_LOSS = TOTAL_LOSS + LOSS

    # Backpropagation and optimization
    Optimizer_NeRF.ZeroGrad()
    Optimizer_Intrinsics.ZeroGrad()
    Optimizer_Extrinsics.ZeroGrad()
    Backward(LOSS)
    Optimizer_NeRF.Step()
    Optimizer_Intrinsics.Step()
    Optimizer_Extrinsics.Step()

END FOR

# Compute and display average loss for the epoch
AVERAGE_LOSS = TOTAL_LOSS / ITERATIONS_PER_EPOCH
Print("Epoch:", Epoch, "Average Loss:", AVERAGE_LOSS)

END FOR

# Verification phase
ACCURACY_METRICS = EmptyList()

FOR i FROM 1 TO DATASET_SIZE DO

    (EMISSION, RECORDING) = EMISSION_RECORDING_PAIRS[i]
    INTRINSIC = CAMERA_INTRINSICS
    EXTRINSIC = CAMERA_EXTRINSICS[i]

    # Generate rays for each pixel of the image
    RAYS_ORIGIN, RAYS_DIRECTION = GenerateRays(INTRINSIC, EXTRINSIC,
IMAGE_WIDTH, IMAGE_HEIGHT)

    # Render image using the trained NeRF model
    RENDERED_IMAGE = RenderImageFromNeRF(NeRF_Model, RAYS_ORIGIN,
RAYS_DIRECTION, EMISSION)

    # Compute Peak Signal-to-Noise Ratio (PSNR) between rendered image and
recording
    PSNR = ComputePSNR(RENDERED_IMAGE, RECORDING)
    Append(ACCURACY_METRICS, PSNR)
    Print("Data Point:", i, "PSNR:", PSNR)

END FOR

```

```

    # Compute average PSNR as overall accuracy
    OVERALL_ACCURACY = ComputeAverage(ACCURACY_METRICS)
    Print("Overall Accuracy (Average PSNR):", OVERALL_ACCURACY)

END UnsupervisedNeRFTraining

FUNCTION GenerateRays(Intrinsic, Extrinsic, ImageWidth, ImageHeight)
    # Generate pixel coordinates in normalized device coordinates (NDC) space
    PIXEL_COORDS_NDC = GeneratePixelCoordsNDC(ImageWidth, ImageHeight)

    # Transform pixel coordinates from NDC space to world space
    PIXEL_COORDS_WORLD = TransformNDCToWorld(PIXEL_COORDS_NDC, Intrinsic,
    Extrinsic)

    # Compute ray directions by normalizing pixel coordinates in world space
    RAYS_DIRECTION = Normalize(PIXEL_COORDS_WORLD)

    # Set ray origins as the camera position (extracted from extrinsic matrix)
    RAYS_ORIGIN = ExtractCameraPosition(Extrinsic)

    RETURN (RAYS_ORIGIN, RAYS_DIRECTION)
END FUNCTION

FUNCTION RenderImageFromNeRF(Model, RaysOrigin, RaysDirection, Emission)
    # Set near and far bounds for ray marching
    NEAR = 0.1
    FAR = 10.0
    NUM_SAMPLES = 128

    # Initialize pixel color buffer
    COLOR_BUFFER = EmptyList()

    # Iterate over each ray
    FOR EACH Ray IN Zip(RaysOrigin, RaysDirection) DO
        RayOrigin = Ray[0]
        RayDirection = Ray[1]

        # Sample points along the ray
        SAMPLED_POINTS = SamplePointsAlongRay(RayOrigin, RayDirection, NEAR, FAR,
        NUM_SAMPLES)

        # Query NeRF model for color and density at each sampled point
        COLORS, DENSITIES = QueryNeRFModel(Model, SAMPLED_POINTS, RayDirection,
        Emission)

        # Perform volume rendering to compute final pixel color
        PIXEL_COLOR = VolumeRendering(COLORS, DENSITIES, SAMPLED_POINTS)

        # Append pixel color to the color buffer
        Append(COLOR_BUFFER, PIXEL_COLOR)
    END FOR

    # Reshape color buffer into image dimensions
    RENDERED_IMAGE = Reshape(COLOR_BUFFER, (ImageWidth, ImageHeight, 3))

```

```

RETURN RENDERED_IMAGE
END FUNCTION

```

This approach follows the following stages:

1. Dataset Loading and Initialization

- **LoadEmissionRecordingPairs:** Loads a dataset containing pairs of emission signals and corresponding recordings captured by the camera.
- **DATASET_SIZE:** The number of emission-recording pairs in the dataset.
- **GetImageDimensions:** Retrieves the width and height of the images in the dataset.

2. Camera Parameters Initialization

- **InitializeFocalLength:** Sets the initial focal length of the camera.
- **BuildCameraIntrinsicsMatrix:** Constructs the camera intrinsics matrix based on the focal length and image dimensions.
- **InitializeCameraExtrinsics:** Initializes the extrinsic parameters (rotation and translation) for each data point, representing the camera's position and orientation in the scene.

3. NeRF Model Initialization

- **InitializeNeRFModel:** Sets up the Neural Radiance Field (NeRF) model with specified input channels (3D coordinates + emission signal dimensions), output channels (RGB color + density), number of layers, and hidden units.

4. Optimizers Initialization

- **InitializeAdam:** Creates Adam optimizers for the NeRF model parameters, camera intrinsics, and extrinsics with a specified learning rate.

5. Training Loop

- **NUM_EPOCHS:** The number of times the entire dataset is processed.
- **ITERATIONS_PER_EPOCH:** The number of iterations performed within each epoch, where a random data point is sampled for each iteration.
- **TOTAL_LOSS:** Accumulates the loss over all iterations in an epoch to compute the average loss.

For each iteration:

- **Random Data Point Sampling:** Randomly selects an emission-recording pair from the dataset.
- **Camera Parameters Retrieval:** Uses the camera intrinsics and the extrinsics specific to the current data point.
- **GenerateRays:** Generates rays for each pixel of the image based on the camera parameters.
 - Computes pixel coordinates in normalized device coordinates (NDC) space.
 - Transforms pixel coordinates from NDC space to world space using the camera intrinsics and extrinsics.
 - Computes ray directions by normalizing the pixel coordinates in world space.
 - Sets ray origins as the camera position extracted from the extrinsic matrix.

- **RenderImageFromNeRF:** Generates an image by ray marching through the scene using the NeRF model, the emission signal, and the generated rays.
 - Samples points along each ray between specified near and far bounds.
 - Queries the NeRF model for color and density at each sampled point, conditioned on the emission signal.
 - Performs volume rendering to accumulate colors and densities along the ray to compute the final pixel color.
- **ComputeMeanSquaredError:** Calculates the mean squared error between the rendered image and the actual recording.
- **Backpropagation and Optimization:** Computes gradients of the loss with respect to the model and camera parameters and updates them using the Adam optimizers.

6. Monitoring Training Progress

- After processing all iterations in an epoch, the average loss is computed and printed.

7. Verification Phase

- **ACCURACY_METRICS:** A list to store the Peak Signal-to-Noise Ratio (PSNR) metric for each data point.
- For each data point:
 - Generates rays using the updated camera parameters.
 - Renders an image using the trained NeRF model and the generated rays.
 - Computes PSNR between the rendered image and the actual recording.
 - Stores and prints the PSNR for each data point.
- **ComputeAverage:** Calculates the average PSNR as the overall accuracy metric.

Key aspects of this approach are as follows:

1. Ray Tracing

GenerateRays: Generates rays for each pixel of the image based on the camera parameters.

Computes pixel coordinates in NDC space and transforms them to world space.

Computes ray directions by normalizing the pixel coordinates in world space.

Sets ray origins as the camera position.

SamplePointsAlongRay: Samples points along each ray between specified near and far bounds.

2. NeRF Model Querying

QueryNeRFModel: Queries the NeRF model for color and density at each sampled point along the rays.

The NeRF model takes the sampled points, ray directions, and emission signal as inputs.

The model outputs the predicted color and density at each point.

3. Volume Rendering

VolumeRendering: Performs volume rendering to accumulate colors and densities along each ray to compute the final pixel color.

Applies the rendering equation to integrate the colors and densities along the ray.

Uses techniques like alpha compositing and numerical integration to approximate the integral.

4. Loss Computation and Optimization

ComputeMeanSquaredError: Calculates the mean squared error between the rendered image and the actual recording.

Backpropagation and Optimization: Computes gradients of the loss with respect to the NeRF model parameters and camera parameters, and updates them using Adam optimizers.

5. Model Evaluation

Verification Phase: Assesses the quality of the trained NeRF model by rendering images for each data point and comparing them with the actual recordings.

Peak Signal-to-Noise Ratio (PSNR): Used as an accuracy metric to measure the similarity between the rendered images and the recordings.

Overall Accuracy: Computed as the average PSNR across all data points to provide an aggregate measure of the model's performance.

The advantages of this approach include the following:

Detailed Scene Representation: NeRF allows for a continuous and high-resolution representation of the scene, capturing fine details and enabling smooth renderings.

Implicit Modeling of Emissions: The emission signals are incorporated into the NeRF model, enabling it to learn the effect of emissions on the scene appearance without explicit modeling.

Joint Optimization: The NeRF model parameters and camera parameters are jointly optimized, allowing for self-calibration and adaptation to the specific scene and camera setup.

Example 6 – Computation

This describes a computational system that utilizes projector-camera setups for advanced image processing tasks. The system is flexible, allowing for various configurations involving multiple projectors and cameras. It can perform operations such as downsampling, convolution, and multiplexing by projecting images into an environment (including an optical resonance cavity) and capturing the results with cameras. The following sections detail each component and process within the system.

Components

1. Projectors

- **Function:** Output images onto an environment.
- **Types:** Can vary in resolution (e.g., 2160p, 1080p) and scanning methods.
- **Configurations:** Single or multiple projectors can be used simultaneously.

2. Cameras

- **Function:** Capture projected images for processing.
- **Types:** Include rolling shutter cameras that capture images line by line, particularly suitable for pairing with scanning projectors. A global shutter camera can be paired with scanning or bistable projectors for simultaneous full image processing.
- **Pairing:** Can be paired with projectors for synchronized operations.

3. Optical Resonance Cavity

- **Function:** An environment where light can resonate, enhancing optical interactions.
 - **Usage:** Multiple projectors can project into the cavity, and multiple cameras can capture the resulting patterns.
-

System Configurations

1. Paired Resolutions

a. Scanning Projector with Rolling Shutter Camera

- **Mechanism:** A scanning projector projects images sequentially, while a rolling shutter camera captures the image in a synchronized manner.
- **Benefit:** Precise replication of the projected image due to synchronized scanning and capturing.

b. Single Camera to Single Projector

- **Mechanism:** One camera captures the output from one projector.
- **Benefit:** Direct replication of the projected image without interference.

c. Multiple Projectors Adding Images

- **Mechanism:** Two or more projectors project images that overlap.
- **Result:** The images from the projectors are added together in the captured image, allowing for image addition or blending effects.

2. Projection into Optical Resonance Cavity

- **Mechanism:** Multiple projectors project images into the cavity, and multiple cameras capture the resulting light patterns.
- **Benefit:** Enhanced optical interactions and complex image processing capabilities due to the resonant properties of the cavity.

Downsampling Process

Downsampling reduces the resolution of an image by combining multiple pixels into one. The system employs rasterization techniques to achieve this.

Example: Downsampling from 2160p to 1080p

- **Goal:** Reduce the image resolution by a factor of two in both horizontal and vertical dimensions.
- **Method:** Group 2x2 blocks of pixels and map them onto single pixels in the lower-resolution image.

Pixel Mapping Table

2160p Pixels	Mapped to 1080p Pixel
(0,0), (0,1), (1,0), (1,1)	(0,0)
(0,2), (0,3), (1,2), (1,3)	(0,1)
(2,0), (2,1), (3,0), (3,1)	(1,0)
(2,2), (2,3), (3,2), (3,3)	(1,1)

Process Explanation

1. **Grouping Pixels:**
 - Each group of four pixels in the 2160p image corresponds to one pixel in the 1080p image.
2. **Combining Values:**
 - The values of the four pixels are combined (e.g., averaged) to produce the value of the corresponding pixel in the downsampled image.
3. **Projection:**
 - The downsampled image is projected using the projector, effectively displaying a lower-resolution version of the original image.

Convolution Process

Convolution involves applying a kernel (filter) over an image to produce a new image emphasizing certain features.

Parameters

- **Original Image Resolution:** 1080p
- **Kernel Size:** 4x4
- **Stride:** 1 (the kernel moves one pixel at a time)

Process Steps

1. Image Padding

- **Purpose:** To ensure the convolution kernel can be applied to the edges of the image.
- **Methods:**
 - **Zero-Padding:** Adding zeros around the border of the image.
 - **Wrapping:** The image edges wrap around to the opposite side.

2. Projector Alignment

- **Multiple Projectors Used:** Four projectors.
- **Starting Positions:**

Projector	Starting Position (X, Y)
Projector 1	(0,0)
Projector 2	(0,1)
Projector 3	(1,0)
Projector 4	(1,1)

- **Explanation:** Each projector starts projecting the convolved image from different offsets to cover all positions where the kernel is applied.

3. Projection of Convolved Image

- Each projector outputs the convolved image starting from its assigned position.
- The projections overlap appropriately to form the complete convolved image.

4. Image Capture

- Cameras capture the combined projections.
- The result is a fully convolved image, as if the kernel had been applied across the entire image.

Multiplexing Embodiments

1. Time Division Multiplexing (TDM)

- **Mechanism:** Different images are projected and captured in rapid succession, each during its own time slot.
- **Benefit:** Efficient use of a single projector-camera pair for multiple images.

2. Frequency Division Multiplexing (FDM)

- **Mechanism:** Images are projected using different light frequencies (colors), and cameras filter and capture specific frequencies.
- **Benefit:** Simultaneous projection and capture of multiple images without time delays.

3. Partial Spatial Multiplexing

- **Mechanism:** The projection volume is divided into sections, each projecting a different image.
- **Benefit:** Parallel processing of multiple images in different interacting spatial regions.

Applying DSP techniques like FFT, DCT, wavelet transform, SVD, and DWT in the projector-camera setup can significantly enhance image processing capabilities. By projecting the transformed image onto the optical resonance cavity, these techniques enable efficient and effective manipulation of visual data.

The Fast Fourier Transform (FFT) and Discrete Cosine Transform (DCT) convert images into the frequency domain, allowing for frequency-based analysis and manipulation. In the projector-camera system, the transformed image can be projected onto the cavity, where specific frequency components can be filtered, suppressed, or amplified by modulating the cavity's response. This enables tasks such as noise reduction, image compression, and selective feature extraction. For example, by comparing images in the frequency domain, similarities and differences between patterns or textures can be easily identified.

Wavelet transforms, including the Discrete Wavelet Transform (DWT), decompose images into multi-scale representations, capturing both spatial and frequency information. By projecting wavelet-transformed images onto the cavity, the system can perform multi-resolution analysis and processing. This is particularly useful for tasks like image denoising, where noise can be effectively removed by thresholding wavelet coefficients. Additionally, wavelet-based compression can be achieved by selectively preserving important wavelet coefficients and discarding less significant ones.

Singular Value Decomposition (SVD) factorizes an image matrix into singular values and vectors, providing a compact representation of the image's structure. In the projector-camera setup, SVD-based techniques can be employed for image compression, denoising, and low-rank approximation. By projecting the SVD-truncated image onto the cavity, the system can efficiently process and reconstruct the image using a reduced set of singular values and vectors. This is beneficial for applications like background subtraction or matrix completion, where the essential information of the image can be captured and manipulated using a lower-dimensional representation.

Incorporating these DSP techniques in the projector-camera system opens up a wide range of possibilities for advanced image processing. By leveraging the optical resonance cavity, the transformed images can be efficiently manipulated and analyzed in ways that are computationally challenging or infeasible with traditional digital methods. This hybrid approach combines the power of DSP with the parallelism and speed of optical computing, enabling real-time processing of complex visual data for various applications, including computer vision, pattern recognition, and image enhancement.

Combining DSP techniques can further enhance the capabilities of the projector-camera system, particularly for operations like convolution. One such approach involves pairing a lower-resolution image with a higher-resolution projector to perform convolution through clever image tiling and multiplexing.

Consider the case of convolving a 1080p image using a 2160p projector. The process begins by copying the 1080p image into each quadrant of the 2160p projector's frame, effectively creating a 2x2 tiled arrangement. To create translated images, zero-padding is applied to the edges of each quadrant. The padding scheme depends on the quadrant's position, as illustrated in the following table:

Quadrant Position	Zero-Padding Scheme
Top Left	No padding required
Top Right	Zero-padded on the top
Bottom Left	Zero-padded on the left
Bottom Right	Zero-padded on the left and top

After tiling and zero-padding, the image is rasterized in a multiplexed manner, interleaving 4 pixels from each quadrant in the projection time correspondent to a single pixel on the 1080p camera e.g.

```
{ Image[0,0](0,0), Image[0,1](0,0), Image[1,0](0,0),  
Image[1,1](0,0) }, { Image[0,0](0,1), Image[0,1](0,1), Image[1,0](0,1),  
Image[1,1](0,1) }, ...
```

The cavity's response to the projected image effectively performs the convolution operation.

To capture the convolved result, a synchronized 1080p camera is used. The camera records the optical response of the cavity to the projected multiplexed image. Due to the carefully designed tiling, zero-padding, and multiplexing scheme, the captured 1080p image represents the convolved output of the original image.

A detailed discussion of implementation of the invention, including mathematical underpinnings, will now be provided.

Setup

Setup: Data

The invention concerns itself with Emitter-Scene-Recorder Loops.

An example projector-Scene-camera Loop is illustrated in Figure 1. An example speaker-Scene-microphone Loop is illustrated in Figure 2. An example ‘stereo projector’-speaker-Scene- ‘stereo camera’-microphone-Agent Loop is illustrated in Figure 3. A head-mounted ‘stereo projector’-Scene-‘stereo camera’ Loop is illustrated in Figure 4, and a head-mounted projector-Scene-camera Loop is illustrated in Figure 5.

Within the context of the invention, Recorder refers to a mechanism for the production of a signal in response to an external system. These include video cameras and sound microphones. A Recorder’s signal data, whether live or stored, optionally including metadata, at time t , is referenced as a Recording, or $\mathbf{R}(t)$, including but not limited to an array of real numbers of dimensions comprising the Recorders’, c , channels, h , height, w , width, or $c_R \times h_R \times w_R$, $\mathbf{R}(t) \in \mathbb{R}^{c_R \times h_R \times w_R}$.

Within the context of the invention, Emitter refers to a mechanism for the production of an emanation in response to a signal. These include video projectors or screens, and sound speakers. An Emitter’s signal data, whether live or stored, optionally including metadata, at time t , is referenced as an Emission, or $\mathbf{E}(t)$, including but not limited to an array of real numbers of dimensions comprising the Emitters’, c , channels, h , height, w , width, or $c_E \times h_E \times w_E$, $\mathbf{E}(t) \in \mathbb{R}^{c_E \times h_E \times w_E}$.

Within the context of the invention, a sample at point i , e.g. a global shutter camera image or sound sample over an interval, is labelled $\mathbf{R}^{[i]}$, and its corresponding Emission $\mathbf{E}^{[i]}$.

Industry standard differentiable methods for the depiction of metadata exist [1]. Within the context of the invention, when referenced explicitly, metadata in these or any format including but not limited to that related to Recordings or Emissions, e.g. timestamps, or intrinsic or extrinsic matrices, are labelled $\mathring{\mathbf{R}}(t)$ or $\mathring{\mathbf{E}}(t)$.

Within the context of the invention, a Scene refers to the medium or environment with which an Emitter interacts, and from which a Recorder derives its data.

Within the context of the invention, stored or live Emission and/or Recording data corresponding to the interval between time t_1 and t_2 are labelled $\{\mathbf{E}(t) | t \in [t_1, t_2]\}$ and/or $\{\mathbf{R}(t) | t \in [t_1, t_2]\}$, respectively.

Within the context of the invention, collections of discrete samples of Emissions and/or Recordings are labelled $\{ \mathbf{E}^{[i]} | i \in \{1, \dots, n\} \}$ and/or $\{ \mathbf{R}^{[i]} | i \in \{1, \dots, n\} \}$.

Within the context of the invention, a Loop, x , comprises any number of Emitters, labelled ${}_x\mathbf{E}(t)$, and Recorders, labelled ${}_x\mathbf{R}(t)$, with the signal state of the Emitter and/or Emitters derived from that of the Recorder and/or Recorders.

Within the context of the invention, stored or live sets of data with an optional temporal offset T are labelled $\{ ({}_x\mathbf{E}(t), {}_x\mathbf{R}(t+T)) | t \in [t_1, t_2] \}$ with discrete data labelled $\{ ({}_x\mathbf{E}^{[i]}, {}_x\mathbf{R}^{[i+a]}) | i \in \{1, \dots, n\} \}$, e.g. $a=0$. Sets of data exist as discrete elements or are combined, e.g. via appropriate resizing and concatenation along any channel, e.g. for a projector-Scene-camera system h_E and h_R are resized to h , w_E and w_R are resized to w . As RGB (Red-Green-Blue) images, $c=c_E=c_R=3$ naturally. These data are concatenated forming data of dimensions, $(c \times h \times 2(w))$, or $(c \times 2(h) \times w)$, or $(6 \times h \times w)$ in the case of a projector-Scene-camera Loop.

Within the context of the invention, when data from Emitters and Recorders external any Loop are used, these Emitters and Recorders are referenced as ${}_y\mathbf{E}(t)$ and ${}_y\mathbf{R}(t)$, e.g. a projector-Scene-camera Loop, x , labelled $({}_x\mathbf{E}(t), {}_x\mathbf{R}(t+T))$ and an optionally synchronised Recorder ${}_y\mathbf{R}(t)$ and Emitter ${}_y\mathbf{E}(t)$.

In some embodiments, multichannel Emitters and/or Recorders are used, e.g. stereo projectors and/or multichannel speakers as Emitters, and/or cameras optionally using filters corresponding to projector channels, and/or multichannel microphones as Recorders.

In some embodiments, camera shutters are synchronised with the corresponding projector's frame refresh synchronisation signal.

In some embodiments, cameras with a global shutter are used, optionally with an exposure time tuned to the projector's frame time.

Setup: Dataset Recording and Initial Analysis

In some embodiments, representative datasets are created by the selection and Emission of a dataset, ${}_x\mathbf{E}$, and the storing of Recordings of the resultant interaction, ${}_x\mathbf{R}$, with a Scene, e.g. any dataset, including but not limited to "VIL: Synthetic Video Texture Dataset" [2] in the visual domain or "Syntex: parametric audio texture datasets" [3] in the audio domain, is Emitted and the resultant Recordings stored.

Within the context of the invention, Output labels are optionally derived via analysis of Recordings and/or Emissions via any industry standard mechanism. Output label signals are labelled $\mathbf{O}$.

Methods for the creation of Output labels from the Recording component of Emission-Recording data include but are not limited to the application of 3D reconstruction from multiple images (e.g. COLMAP [4]) to Recordings, i.e. the creation of depth maps from a set of camera images in a recorded dataset.

Industry standard techniques for Recording analysis include 3D reconstruction from multiple images models [5],[6],[7]. In some embodiments, these or any other Recording analysis techniques are applied to Emission-Recording Loop signals, e.g. the application of a 3D cost volume

minimisation model to Recordings and/or Emission signals including but not limited the training of a Deep Structure from Motion Models (e.g. [8]) on $\{ {}_x\mathbf{R}^{[i]} | i \in \{1, \dots, n\} \}$ or $\{ ({}_xE^{[i]}, {}_x\mathbf{R}^{[i+a]}) | i \in \{1, \dots, n\} \}$, e.g. $a=0$.

Industry standard methods for the creation of Output labels include the calculated creation of Emissions and analytic processing of their corresponding Recordings, including but not limited to Structured Light Emission, i.e. the selection of any known Emission pattern, e.g. a Color M-Array dataset, the storage of Recordings resultant from of the Emission of these patterns, and the application of any industry standard Structured Light analysis including but not limited to those described in [9], and the creation of depth maps corresponding to Emissions and the respective Recordings. In some embodiments, Structured Light Emission-Scene-Recording signals with optical flow properties are created via any industry standard data augmentation method(s) applied to industry standard Structured Light Emissions.

Industry standard methods for the unsupervised or self-supervised creation of classification labels include the training and application of Unsupervised or Self-Supervised Video Object Segmentation models [10],[11] on and to $\{ ({}_x\mathbf{R}^{[i]} | i \in \{1, \dots, n\} \}$ or $\{ ({}_xE^{[i]}, {}_x\mathbf{R}^{[i+a]}) | i \in \{1, \dots, n\} \}$, e.g. $a=0$.

In some embodiments, Output labels are derived via any industry standard method, e.g. the capturing of depth maps using RGB-D cameras (including but not limited to the use of the Microsoft (TM) Kinect (TM) Azure (TM)), or the manual application of classification labels.

Setup: Trained Functions

Within the context of the invention, a Comparative Discriminator, $\mathbf{C}$, references a function which attempts to distinguish two input data via any industry standard method including but not limited $L1$ loss, VGG loss, or the training of a discriminator in a Generative Adversarial Network [12]. In other embodiments, any comparative function is used.

Within the context of the invention, a Generator, $\mathbf{G}$, comprises a trained model capable of mapping from Recordings, Emissions and/or Output labels to Recordings, Emissions and/or Output labels, including but not limited to a Generator model created during the training of a Generative Adversarial Network (GAN) [13]. In some embodiments, these models are recursive, and/or comprise any regression and/or classification models, attention models, or transformers, or comprise any industry standard trainable function.

Within the context of the invention, an Agent references a mapping via Generator, and/or One-Way Function between Recorders or a Recorder and Emitters or an Emitter, and any optional Output labels. Agents operating on Loop x are labelled ${}_x\mathbf{G}$.

Example external elements, y , are illustrated in Figure 6, i.e. external synchronised cameras used in the training of ${}_x\mathbf{G}$ but not in the provision of data to ${}_x\mathbf{G}$.

Within the context of the invention, the use of Emitter-Recorder signals, $\{ ({}_xE(t), {}_x\mathbf{R}(t+T)) | t \in [t_1, t_2] \}$, in the training of an Agent ${}_x\mathbf{G}$ mapping ${}_x\mathbf{R}(t)$ with an optional time offset T to a generated Emission Prediction ${}_x^g\mathbf{E}(t)$, with the parameters of ${}_x\mathbf{G}$ optimised via back propagation, according to the output of a Comparative Discriminator, $\mathbf{C}$, provided ${}_x^g\mathbf{E}(t)$ and real Emission data, ${}_xE(t)$, via industry standard GAN methods or any other is described via:

$$\begin{aligned}
& \downarrow C({}_x\mathbf{E}(t+T_{III}), {}^g\mathbf{E}(t)) \\
& {}^g\mathbf{E}(t) = \uparrow {}_x\mathbf{G}({}_x\mathbf{R}(t+T_{II})) \\
& \dots\dots\dots \\
& \text{With } \{({}_x\mathbf{E}(t), {}_x\mathbf{R}(t+T_I)) \mid t \in [t_1, t_2]\}: \\
& \dots\dots\dots \\
& \text{e.g.} \\
& a_{III} = a_I = 0, a_{II} = -1 \\
& \downarrow C({}_x\mathbf{E}^{[i+a_{III}]}, {}^g\mathbf{E}^{[i]}) \\
& {}^g\mathbf{E}^{[i]} = \uparrow {}_x\mathbf{G}({}_x\mathbf{R}^{[i+a_{II}]}) \\
& \dots\dots\dots \\
& \text{With } \{({}_x\mathbf{E}^{[i]}, {}_x\mathbf{R}^{[i+a_I]}) \mid i \in \{1, \dots, n\}\}: \\
& \dots\dots\dots
\end{aligned}$$

or any subset thereof, or equivalent thereto.

Such specifications are read from the bottom up, specifying signals available to the models being trained, their outputs, the networks laid out, and the objective functions being used to update their network parameters, with up-arrows indicating that a function is being trained, and down arrows that it is being used as an objective function in the training.

Within the context on the invention, an objective function, $\mathbf{L}$, comprises an arbitrary loss function or reward function which maps an input to a value representing an evaluation of the input to be minimised or maximised. In some embodiments, any industry standard objective function is used including but not limited to a style loss function [14].

Setup: Multiple Channels and Domains

In some embodiments, crossover occurs between Recorder and Emitter channels i.e. in the case of a stereo projector, $\mathbf{E}(t)$, comprising channel I , ${}_x\mathbf{E}_I(t)$, and the other channel II , ${}_x\mathbf{E}_{II}(t)$, with a camera filtered to view channel I , ${}_x\mathbf{R}_I(t)$, and a camera filtered to view channel II , ${}_x\mathbf{R}_{II}(t)$, channel I is used for a Recorder, and channel II used for the Emitter.

In some embodiments, crossover occurs between domains. Figure 3 illustrates a configuration of stereo projector, ${}_x\mathbf{E}_I(t), {}_x\mathbf{E}_{II}(t)$, stereo cameras filtered to view these channels, ${}_x\mathbf{R}_I(t+T_I), {}_x\mathbf{R}_{II}(t+T_{II})$, speaker ${}_x\mathbf{E}_{III}(t)$, and microphone ${}_x\mathbf{R}_{III}(t+T_{III})$. In some embodiments Record from any permutation of domains and Emit any in permutation of domains. Examples include but are not limited to speaker-Scene-camera Loop (e.g. for the generation of cymatics using a speaker-diaphragm-particulate system) or a ‘mechanical display’-Scene-microphone Loop.

Setup: Direct Training

Within the context of the invention, Reinforcement Learning (RL) references an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward.

In some embodiments, Reinforcement Learning is applied to Emitter-Scene-Recorder Loops, i.e. a Reinforcement Learning framework, including but not limited to [15] is used to apply an Objective Function in the training via back propagation of an Agent. Industry standard RL techniques include the training of Generators as Agents against objective functions applied to Recordings e.g. camera

images or microphones. Industry standard RL techniques include the output of discrete or continuous signals by Agents. In some embodiments, Emitters are treated as robotic actuators for the purpose of RL training e.g. an Agent is configured with a continuous output of dimension $c_E \times h_E \times w_E$, or any continuous output e.g. of dimensions $c \times h \times w$, followed by a Visualisation function applied to the Agent's output. In some embodiments, any reinforcement learning techniques are applied in the training of Agents and/or any Generator and/or Discriminator or any other models within the scope of the invention (e.g. Output label generation), e.g. [16],[17].

Within the context of the invention, Physics Aware Training comprise any industry standard method for the training of models physical systems via back propagation [18], [19] optionally according to an objective function [20]. In some embodiments, these methods are applied to projector-Scene-camera systems e.g. using projectors, $E(t)$ or $E^{[i]}$, as output and cameras, $R(t)$ or $R^{[i]}$, used as input.

Within the context of the invention, Agents trained via Reinforcement Learning, Physics Aware Training, or any industry standard method with an Objective Function applied to Recordings resultant from the Generator's Emissions are described via:

$$\begin{aligned}
 & \downarrow L({}_x R(t+T_{VI})) \\
 & \downarrow C({}_x O(t+T_V), {}^g S(t+T_{IV})) \\
 & ({}^g E(t), {}^g S(t+T_{IV})) = \uparrow {}_x G({}_x R(t+T_{III})) \\
 & \dots\dots\dots \\
 & \text{With } \{({}_x E(t), {}_x R(t+T_I), {}_x O(t+T_{II})) \mid t \in [t_1, t_2]\} : \\
 & \dots\dots\dots \\
 & \text{e.g.} \\
 & a_V = a_{IV} = a_{III} = -1 \\
 & a_{VI} = a_{II} = a_I = 0 \\
 & \downarrow L({}_x R^{[i+a_{VI}]}) \\
 & \downarrow C({}_x O^{[i+a_V]}, {}^g S^{[i+a_{IV}]}) \\
 & ({}^g E^{[i]}, {}^g S^{[i+a_{IV}]}) = \uparrow {}_x G({}_x R^{[i+a_{III}]}) \\
 & \dots\dots\dots \\
 & \text{With } \{({}_x E^{[i]}, {}_x R^{[i+a_I]}, {}_x O^{[i+a_{II}]}) \mid i \in \{1, \dots, n\}\} : \\
 & \dots\dots\dots
 \end{aligned}$$

or any subset thereof, or equivalent thereto.

Setup: Domain Discriminators

Within the context of the invention, a Domain Discriminator, here labelled D , refers to a classifier model which is trained to determine whether a Loop elements provided to it is are authentic components of the training set, e.g. the discriminator trained as part of the training of an autoencoder, including but not limited to [21],[22] is described as $\uparrow D$, i.e.

$$\begin{aligned}
& \uparrow D_{(x\mathbf{E}(t), x\mathbf{R}(t+T_{II}))} (x\mathbf{E}(t), x\mathbf{R}(t+T_{II})) \\
& \dots\dots\dots \\
& \text{With } \{ (x\mathbf{E}(t), x\mathbf{R}(t+T_I)) \mid t \in [t_1, t_2] \} : \\
& \dots\dots\dots \\
& \text{e.g. } a_{II} = a_I = 0 \\
& \uparrow D_{(x\mathbf{E}^{[i]}, x\mathbf{R}^{[i+a_{II}]})} (x\mathbf{E}^{[i]}, x\mathbf{R}^{[i+a_{II}]}) \\
& \dots\dots\dots \\
& \text{With } \{ (x\mathbf{E}^{[i]}, x\mathbf{R}^{[i+a_I]}) \mid i \in \{1, \dots, n\} \} : \\
& \dots\dots\dots
\end{aligned}$$

or any subset thereof, or equivalent thereto.

When addressed specifically, the encoder used in the training of $\mathbf{D}$ is here described as $\mathbf{G}_D$, i.e.

$$\begin{aligned}
& \uparrow D_{(x\mathbf{E}(t), x\mathbf{R}(t+T_{II}))} ((x\mathbf{E}(t), x\mathbf{R}(t+T_{II}))) \\
& (x\mathbf{E}(t), x\mathbf{R}(t+T_{II})) = \uparrow \mathbf{G}_D((x\mathbf{E}(t), x\mathbf{R}(t+T_{II}))) \\
& \dots\dots\dots \\
& \text{With } \{ (x\mathbf{E}(t), x\mathbf{R}(t+T_I)) \mid t \in [t_1, t_2] \} \\
& \dots\dots\dots \\
& \text{e.g. } a_{II} = a_I = 0 \\
& \uparrow D_{(x\mathbf{E}^{[i]}, x\mathbf{R}^{[i+a_{II}]})} ((x\mathbf{E}^{[i]}, x\mathbf{R}^{[i+a_{II}]}) \\
& (x\mathbf{E}^{[i]}, x\mathbf{R}^{[i+a_{II}]}) = \uparrow \mathbf{G}_D((x\mathbf{E}^{[i]}, x\mathbf{R}^{[i+a_{II}]}) \\
& \dots\dots\dots \\
& \text{With } \{ (x\mathbf{E}^{[i]}, x\mathbf{R}^{[i+a_I]}) \mid i \in \{1, \dots, n\} \} : \\
& \dots\dots\dots
\end{aligned}$$

or any subset thereof, or equivalent thereto.

If left unspecified, Domain Discriminators are trained directly on data or on embeddings within a latent space, or via any industry standard manner.

Setup: Domain Generators

Within the context of the invention, a Domain Generator model comprises a model trained to reproduce data within a domain, optionally according to a conditional function (e.g. an objective function applied to any subset of Generated Elements), including but not limited variational autoencoders (VAEs), generative adversarial networks (GANs), and auto-regressive models [21], [22].

E.g. a Domain Generator trained to receive data from Loop Alice, x_A , and to generate data evaluated according to Domain Discriminators evaluating membership of x_A , comparing Generated output with input, or a Domain Discriminator evaluating membership of the Generated output in Loop Bob, x_B . is described via:

$$\begin{aligned}
& \downarrow D_{(x_{E_B}(t+T_{VII}), x_{R_B}(t+T_{IX}))} ({}^g\tilde{E}(t+T_{VI}), {}^g\tilde{R}(t+T_{VII})) \\
& \downarrow C((x_A E(t+T_{IV}), x_A R(t+T_V)), ({}^g\tilde{E}(t+T_{VI}), {}^g\tilde{R}(t+T_{VII}))) \\
& \downarrow D_{(x_{E_A}(t+T_{IV}), x_{R_B}(t+T_V))} ({}^g\tilde{E}(t+T_{VI}), {}^g\tilde{R}(t+T_{VII})) \\
& ({}^g\tilde{E}(t+T_{VI}), {}^g\tilde{R}(t+T_{VII})) = \uparrow \tilde{G}((x_A E(t+T_{IV}), x_A R(t+T_V)))
\end{aligned}$$

.....
and $\{(x_B E(t+T_{II}), x_B R(t+T_{III})) | t \in [t_1, t_2]\}$:

With $\{(x_A E(t), x_A R(t+T_I)) | t \in [t_1, t_2]\}$,

.....

e.g.

$$a_{IX} = a_{VIII} = a_{VII} = a_{VI} = 0$$

$$a_V = a_{IV} = a_{III} = a_{II} = a_I = 0$$

$$\begin{aligned}
& \downarrow D_{(E_B^{[i+a_{VIII}]}, R_B^{[i+a_{IX}]})} ({}^g\tilde{E}^{[i+a_{VI}]}, {}^g\tilde{R}^{[i+a_{VII}]}) \\
& \downarrow C((x_A E^{[i+a_{IV}]}, x_A R^{[i+a_V]}), ({}^g\tilde{E}^{[i+a_{VI}]}, {}^g\tilde{R}^{[i+a_{VII}]}) \\
& \downarrow D_{(x_{E_A}^{[i+a_{IV}]}, x_{R_A}^{[i+a_V]})} ({}^g\tilde{E}^{[i+a_{VI}]}, {}^g\tilde{R}^{[i+a_{VII}]}) \\
& ({}^g\tilde{E}^{[i+a_{VI}]}, {}^g\tilde{R}^{[i+a_{VII}]}) = \uparrow \tilde{G}(x_A E^{[i+a_{IV}]}, x_A R^{[i+a_V]})
\end{aligned}$$

.....
and $\{(x_B E^{[i+a_{II}]}, x_B R^{[i+a_{III}]}) | i \in \{1, \dots, n\}\}$:

With $\{(x_A E^{[i]}, x_A R^{[i+a_I]}) | i \in \{1, \dots, n\}\}$,

.....

or any subset thereof, or equivalent thereto.

Setup: Alternate Reality Prediction

In some embodiments, Generator models are trained to predict Scenes given alternative Emission states and are labelled ${}_x G_R$, i.e. multiple sets of Emission-Scene-Recording data with an equivalence are created, including but not limited to via multichannel Emission and multichannel Recording e.g. a stereo projector, ${}_x E(t)$, comprising channel I , ${}_x E_I(t)$, and the other, ${}_x E_{II}(t)$, with a camera filtered to view channel I , ${}_x R_I(t)$, and a camera filtered to view channel II , ${}_x R_{II}(t)$.

$$\begin{aligned}
& \downarrow C({}_x R_I(t+T_{VII}), {}^g R_I(t)) \\
& {}^g R_I(t) = \uparrow G_R({}_x R_{II}(t+T_{IV}), {}_x E_{II}(t+T_V), {}_x E_I(t+T_{VI})) \\
& \text{With } \{({}_x E_I(t), {}_x E_{II}(t+T_I), {}_x R_I(t+T_{II}), {}_x R_{II}(t+T_{III})) | t \in [t_1, t_2]\}:
\end{aligned}$$

.....

e.g.

$$a_{VII} = a_{VI} = a_V = a_{IV} = a_{III} = a_{II} = a_I = 0$$

$$\downarrow C({}_x R_I^{[i+a_{VII}]}, {}^g R_I^{[i]})$$

$${}^g R_I^{[i]} = \uparrow G_R({}_x R_{II}^{[i+a_{IV}]}, {}_x E_{II}^{[i+a_V]}, {}_x E_I^{[i+a_{VI}]})$$

.....
With $\{({}_x E_I^{[i]}, {}_x E_{II}^{[i+a_I]}, {}_x R_I^{[i+a_{II}]}, {}_x R_{II}^{[i+a_{III}]}) | i \in \{1, \dots, n\}\}$:

.....

or any subset thereof, or equivalent thereto.

$$\begin{aligned}
& \downarrow C({}_x\mathbf{R}(t+T_{VII}), {}^g_x\mathbf{R}(t)) \\
& {}^g_x\mathbf{R}(t) = \uparrow G_R({}_x\tilde{\mathbf{R}}(t+T_{IV}), {}_x\tilde{\mathbf{E}}(t+T_V), {}_xE(t+T_{VI})) \\
& \dots\dots\dots \\
& \text{and } \{({}_x\tilde{\mathbf{E}}(t+T_{II}), {}_x\tilde{\mathbf{R}}(t+T_{III})) \mid t \in [t_1, t_2]\} : \\
& \quad \text{With } \{({}_xE(t), {}_x\mathbf{R}(t+T_I)) \mid t \in [t_1, t_2]\}, \\
& \dots\dots\dots \\
& \quad \text{e.g.} \\
& \quad a_{VII} = a_{VI} = a_V = a_{IV} = a_{III} = a_{II} = a_I = 0 \\
& \quad \downarrow C({}_x\mathbf{R}^{[i+a_{VII}]}, {}^g_x\mathbf{R}^{[i]}) \\
& {}^g_x\mathbf{R}^{[i]} = \uparrow G_R({}_x\tilde{\mathbf{R}}^{[i+a_{IV}]}, {}_x\tilde{\mathbf{E}}^{[i+a_V]}, {}_xE^{[i+a_{VI}]}) \\
& \dots\dots\dots \\
& \text{and } \{({}_x\tilde{\mathbf{E}}^{[i+a_{III}]}, {}_x\tilde{\mathbf{R}}^{[i+a_{III}]}) \mid i \in \{1, \dots, n\}\} : \\
& \quad \text{With } \{({}_xE^{[i]}, {}_x\mathbf{R}^{[i+a_I]}) \mid i \in \{1, \dots, n\}\}, \\
& \dots\dots\dots
\end{aligned}$$

or any subset thereof, or equivalent thereto.

In some embodiments, the simulated Recordings resultant from the outputs of an Agent are used in the offline training of models via RL.

Setup: Multi-Agent Systems

Within the context of the invention, when Agents are trained in multi-Agents systems, they are distinguished via the industry standard naming scheme of Alice, z_A , Bob, z_B , and Eve, z_E .

A projector-Scene-camera-Agent Loop mounted as a robotic actuator for use in training multi-Agent systems is illustrated in Figure 7.

Industry standard methods for the training of cryptographic Generative Adversarial Networks include the representation of plaintext, $\mathbf{U}$, cryptographic keys, $\mathbf{K}$, and training methods here described as Oppositional [23], in which one Agent, e.g. Agent Alice, z_A , trains in Opposition to the Objective Function of another Agent, e.g. Agent Eve, z_E .

Within the context of the invention, these Objective Functions are described as Oppositional Objective Functions, and a Comparative Discriminator trained Oppositionally is labelled $\mathbf{C}$, a Domain Discriminator trained Oppositionally labelled $\mathbf{D}$, and an arbitrary Objective Function trained Oppositionally is labelled $\mathbf{E}$.

Creation of Emitter-Scene-Recorder Loops with One-Way Properties

Within the context of the invention, a Physical Unclonable Function (PUF) references a mechanism for modifying a function via a material method incorporating a manufactured random component. An example in the visual domain includes, but is not limited to, a speckle pattern used to modify the output of a projector and/or the input to a camera. An example in the auditory domain includes, but

is not limited to, the application of a muffler to a speaker sound output and/or a microphone input. An example in the electronic signal domain includes, but is not limited to, an analog PUF [24].

Within the context of the invention, a One-Way Function, labelled $\mathbf{H}$, references any mapping between data more easily performed in one direction than another, e.g. the application of a PUF to by Alice to her Recordings, $_{x_A}\mathbf{R}(t)$, is labelled $_{x_A}\mathbf{H}(t)=_{x_A}\mathbf{H}(_{x_A}\mathbf{R}(t+T))$, or the output of a cryptographic function including but limited to a hash function applied to $_{x_A}\mathbf{R}^{[i+a]}$ is labelled $_{x_A}\mathbf{H}^{[i]}=_{{x_A}}\mathbf{H}(_{{x_A}}\mathbf{R}^{[i+a]})$, e.g. $a=-1$.

Within the context of the invention, a Visualisation function, $\mathbf{V}$, references a mapping between data including but not limited to the output of a One-Way Function and Emitter, e.g. in the case of a PUF applied to each channel of $_{x_A}\mathbf{R}(t+T)$ where $h_H=h_R=h_E$, and $w_H=w_R=w_E$, and $c_H=c_R=c_E=3$, $\mathbf{V}$ is an identity, and the Emission is equal to the output of the PUF. In some embodiments, any industry standard remappings are used. An example of a visualisation function in the discrete case, $\mathbf{V}(_{{x_A}}\mathbf{H}^{[i+a]})$, includes but is not limited to the application of a variable output length hash function to $_{x_A}\mathbf{H}^{[i+a]}$ (e.g. the hashing of the 256 bit $_{x_A}\mathbf{H}^{[i+a]}$ via *blake3*($_{x_A}\mathbf{H}^{[i+a]}$, output_size=768)), the reshaping of this output to $c_H \times h_H \times w_H$ (e.g. $3 \times 16 \times 16$), and the resizing of this output to $c_E \times h_E \times w_E$. In some embodiments, other functions are used, e.g. *blake3*($_{x_A}\mathbf{H}^{[i+a]}$, output_size=1536) is mapped to a complex $3 \times 16 \times 16$ array which is resized or modified via any number of rounds of Inverse Fast Fourier Transforms and/or any other matrix transform to an output size of $c_E \times h_E \times w_E$. As an industry standard ungermane practice in machine learning, such remappings are not listed explicitly if trivial.

n Emission derived from the application of a One-Way-Function is labelled $_{x_A}\bar{\mathbf{E}}(t)$, i.e. $_{x_A}\bar{\mathbf{E}}(t)=_{x_A}\mathbf{H}(_{{x_A}}\mathbf{R}(t+T))$, and the resultant Recordings labelled $_{x_A}\bar{\mathbf{R}}(t)$.

Within the context of the invention a One-Way Function Payload comprises arbitrary information, including but limited to timestamps, to be input to the One-Way Function, or derived from it and labelled $\mathbf{Q}$. In some embodiments, comprises any signal or timestamp derived from a ledger, or interlocutor, or widely available signals or signals derived via non-interactive method e.g. a Nothing-Up-My-Sleeve number including but not limited to Phi, i.e. $\mathbf{Q}^{[i]}=\phi^{[i]}$, or any replicable signal e.g. $\mathbf{Q}(t)=\sin(t)$.

In some embodiments, payload data comprises the output of the One-Way Function of another Loop x_B , i.e. $_{x_A}\mathbf{Q}(t)=_{x_B}\mathbf{H}(t+T)$ or $_{x_A}\mathbf{Q}^{[i]}=_{{x_B}}\mathbf{H}^{[i+a]}$, e.g. $a=-1$.

In some embodiments, the output of the One-Way Function is provided to be used as a One-Way Function Payload by an external Loop x_B , i.e. $_{x_B}\mathbf{Q}(t)=_{x_A}\mathbf{H}(t+T)$, or $_{x_B}\mathbf{Q}^{[i]}=_{{x_A}}\mathbf{H}^{[i+a]}$, e.g. $a=-1$, or $_{x_A}\mathbf{A}^{[i]}$ is transmitted to a ledger or interlocutor, or broadcast.

In some embodiments, $\mathbf{H}$ is applied recursively e.g. $_{x_A}\mathbf{H}(t)=_{x_A}\mathbf{H}(_{{x_A}}\mathbf{H}(t+T))$ or $_{x_A}\mathbf{H}^{[i]}=_{{x_A}}\mathbf{H}(_{{x_A}}\mathbf{H}^{[i+a]})$, e.g. $a=-1$.

In some embodiments, the invention comprises the Emission of $_{x_A}\bar{\mathbf{E}}(t)=_{x_A}\mathbf{H}(_{{x_A}}\mathbf{R}(t+T_I), _{{x_B}}\mathbf{Q}(t+T_{II}), _{{x_A}}\mathbf{H}(t+T_{III}))$ or $_{x_A}\bar{\mathbf{E}}^{[i]}=_{{x_A}}\mathbf{H}(_{{x_A}}\mathbf{R}^{[i+a_I]}, _{{x_B}}\mathbf{Q}^{[i+a_{II}]}, _{{x_A}}\mathbf{H}^{[i+a_{III}]})$, e.g. $a_{III}=a_I=-1, a_{II}=0$.

In some embodiments, data including but not limited to

$\{(\bar{E}(t), \bar{R}(t+T_I), H(t+T_{II})) | t \in [t_1, t_2]\}$ or $\{(\bar{E}^{[i]}, \bar{R}^{[i+a_I]}, H^{[i+a_{II}]}) | i \in \{1, \dots, n\}\}$, e.g. $a_{II}=a_I=0$ represent the secured system.

Validation of Stored Data Resultant from the Emission of One-Way Function

In some embodiments, secured element correspondence is validated via any industry standard

analysis method including but not limited to $C(\bar{H}(t), \bar{H}(\bar{R}(t+T_I), Q(t+T_{II}), H(t+T_{III})))$, e.g. $C(\bar{H}^{[i]}, \bar{H}(\bar{R}^{[i+a_I]}, Q^{[i+a_{II}]}, H^{[i+a_{III}]})$, e.g. $a_{III}=a_I=-1, a_{II}=0$, or any subset thereof, or equivalent thereto, or $C(\bar{E}(t), V(\bar{H}(t+T)))$, e.g. $C(\bar{E}^{[i]}, V(\bar{H}^{[i+a_I]})$, e.g. $a=0$.

Within the context of the invention, Domain Discriminators, trained on data from Loop Alice data, x_A , to predict the authenticity of novel data originating in Loop Bob, x_B , i.e.

$$\begin{aligned} & \{D_{(\bar{E}_A(t), \bar{R}_A(t+T_{III}))}(\bar{E}(t), \bar{R}(t+T_{IV})) | t \in [t_3, t_4]\} \\ & \quad \uparrow D_{(\bar{E}_A(t), \bar{R}_A(t+T_{III}))}(\bar{E}(t), \bar{R}(t+T_{III})) \\ & \dots\dots\dots \\ & \text{and } \{(\bar{E}_I(t), \bar{R}_I(t+T_{II})) | t \in [t_3, t_4]\}: \\ & \text{With } \{(\bar{E}(t), \bar{R}_I(t+T_I)) | t \in [t_1, t_2]\}, \\ & \dots\dots\dots \\ & \text{e.g.} \\ & \quad a_{IV}=a_{III}=a_{II}=a_I=0 \\ & \{D_{(\bar{E}_A^{[i]}, \bar{R}_A^{[i+a_{III}]})}(\bar{E}^{[i]}, \bar{R}^{[i+a_{IV}]}) | i \in \{1, \dots, m\}\} \\ & \quad \uparrow D_{(\bar{E}_A^{[i]}, \bar{R}_A^{[i+a_{III}]})}(\bar{E}^{[i]}, \bar{R}^{[i+a_{III}]}) \\ & \dots\dots\dots \\ & \text{and } \{(\bar{E}^{[i]}, \bar{R}^{[i+a_{III}]}) | i \in \{1, \dots, m\}\}: \\ & \text{With } \{(\bar{E}^{[i]}, \bar{R}^{[i+a_I]}) | i \in \{1, \dots, n\}\}, \\ & \dots\dots\dots \end{aligned}$$

or any subset thereof, or equivalent thereto.

Direct One-Way Function Training

In some embodiments, One-Way function/Discriminator pairs are trained directly on Loops, i.e.

$$\begin{aligned} \uparrow D_{(\bar{E}(t), \bar{R}(t+T_{IV}))}(\bar{E}(t), \bar{R}(t+T_{III})) \\ \bar{E}(t) = \uparrow G_D(\bar{R}(t+T_{II})) \end{aligned}$$

$$\text{With } \{(\bar{E}(t), \bar{R}(t+T_I)) \mid t \in [t_1, t_2]\} :$$

e.g.

$$a_{IV} = a_{III} = a_I = 0$$

$$a_{II} = -1$$

$$\begin{aligned} \uparrow D_{(\bar{E}^{[i]}, \bar{R}^{[i+a_{II}]})}(\bar{E}^{[i]}, \bar{R}^{[i+a_{III}]}) \\ \bar{E}^{[i]} = \uparrow G_D(\bar{R}^{[i+a_{II}]}) \end{aligned}$$

$$\text{With } \{(\bar{E}^{[i]}, \bar{R}^{[i+a_I]}) \mid i \in \{1, \dots, n\}\} :$$

or any subset thereof, or equivalent thereto.

Reality Encryption

In some embodiments, Agent Alice, x_A , uses Neutral Recording Generator, G_N , models trained to retarget Emissions resultant from One-Way Functions and their resultant Recordings to neutral Emission E_N (e.g. a grey signal) and their resultant Recordings R_N derived via channel separation, Generative Means or any other.

$$\begin{aligned} {}^g R_N(t) = G_N(\bar{R}(t+T_{IV}), \bar{E}(t+T_V)) \\ \bar{E}(t) = V(K(t+T_{III})) \end{aligned}$$

$$\text{With } \{(\bar{E}(t), \bar{R}(t+T_I), K(t+T_{II})) \mid t \in [t_1, t_2]\} :$$

e.g.

$$a_V = a_{IV} = a_{III} = a_{II} = a_I = 0$$

$${}^g R_N^{[i]} = G_N(\bar{R}^{[i+a_{IV}]}, \bar{E}^{[i+a_V]})$$

$$\bar{E}^{[i]} = V(K^{[i+a_{III}]})$$

$$\text{With } \{(\bar{E}^{[i]}, \bar{R}^{[i+a_I]}, K^{[i+a_{II}]}) \mid i \in \{1, \dots, n\}\} :$$

or any subset thereof, or equivalent thereto.

In some embodiments, multiple Agents are configured to cooperate or interfere with each other's ability to reconstruct a neutral image e.g. stereo projector, stereo camera systems Alice, z_A , Bob, z_B , and Eve, z_E , are trained using channel II as a neutral channel i.e. ${}_x E_H^{[i]} = {}_x E_N^{[i+a_I]}$ and the resultant neutral Recordings on channel II , ${}_x R_N^{[i]} = {}_x R_H^{[i+a_{II}]}$.

$$\begin{aligned}
& \downarrow_{x_{A,B}} \mathbf{E}(\downarrow_{x_E} \mathbf{O}(t+T_{XXVII}), \downarrow_{x_E} {}^g\bar{\mathbf{S}}(t)) \\
& \downarrow_{x_E} \mathbf{C}(\downarrow_{x_E} \mathbf{O}(t+T_{XXVI}), \downarrow_{x_E} {}^g\bar{\mathbf{S}}(t)) \\
& \downarrow_{x_{A,B}} \mathbf{C}(\downarrow_{x_B} \mathbf{O}(t+T_{XXV}), \downarrow_{x_B} {}^g\bar{\mathbf{S}}(t)) \\
& \downarrow_{x_{A,B}} \mathbf{C}(\downarrow_{x_A} \mathbf{O}(t+T_{XXIV}), \downarrow_{x_A} {}^g\bar{\mathbf{S}}(t)) \\
& (\downarrow_{x_E} {}^g\bar{\mathbf{E}}(t), \downarrow_{x_E} {}^g\bar{\mathbf{S}}(t)) = \uparrow_{x_E} \mathbf{G}_N(\downarrow_{x_E} \mathbf{P}(t+T_{XXIII})) \\
& (\downarrow_{x_B} {}^g\bar{\mathbf{E}}(t), \downarrow_{x_B} {}^g\bar{\mathbf{S}}(t)) = \uparrow_{x_B} \mathbf{G}_N(\downarrow_{x_B} \mathbf{P}(t+T_{XXII})) \\
& (\downarrow_{x_A} {}^g\bar{\mathbf{E}}(t), \downarrow_{x_A} {}^g\bar{\mathbf{S}}(t)) = \uparrow_{x_A} \mathbf{G}_N(\downarrow_{x_A} \mathbf{P}(t+T_{XXI}))
\end{aligned}$$

$$\begin{aligned}
& \downarrow_{x_E} \mathbf{O}(t) = \downarrow_{x_E} \mathbf{R}_N(t+T_{XX}), \\
& \downarrow_{x_B} \mathbf{O}(t) = \downarrow_{x_B} \mathbf{R}_N(t+T_{XIX}), \\
& \downarrow_{x_A} \mathbf{O}(t) = \downarrow_{x_A} \mathbf{R}_N(t+T_{XVIII}), \\
& \downarrow_{x_E} \mathbf{P}(t) = (\downarrow_{x_E} \bar{\mathbf{R}}(t+T_{XVII}), \downarrow_{x_E} \mathbf{K}(t+T_{XVI})) \\
& \downarrow_{x_B} \mathbf{P}(t) = (\downarrow_{x_B} \bar{\mathbf{R}}(t+T_{XV}), \downarrow_{x_B} \mathbf{K}(t+T_{XIV})) \\
& \downarrow_{x_A} \mathbf{P}(t) = (\downarrow_{x_A} \bar{\mathbf{R}}(t+T_{XIII}), \downarrow_{x_A} \mathbf{K}(t+T_{XII}))
\end{aligned}$$

$$\text{and } \{(\mathbf{K}(t+T_{XII})) \mid t \in [t_1, t_2]\}:$$

$$\text{and } \{(\downarrow_{x_E} \bar{\mathbf{E}}(t+T_{VIII}), \downarrow_{x_E} \bar{\mathbf{R}}(t+T_{IX}), \downarrow_{x_E} \mathbf{E}_N(t+T_X), \downarrow_{x_E} \mathbf{R}_N(t+T_{XI})) \mid t \in [t_1, t_2]\},$$

$$\text{and } \{(\downarrow_{x_B} \bar{\mathbf{E}}(t+T_{IV}), \downarrow_{x_B} \bar{\mathbf{R}}(t+T_V), \downarrow_{x_B} \mathbf{E}_N(t+T_{VI}), \downarrow_{x_B} \mathbf{R}_N(t+T_{VII})) \mid t \in [t_1, t_2]\},$$

$$\text{With } \{(\downarrow_{x_A} \bar{\mathbf{E}}(t), \downarrow_{x_A} \bar{\mathbf{R}}(t+T_I), \downarrow_{x_A} \mathbf{E}_N(t+T_{II}), \downarrow_{x_A} \mathbf{R}_N(t+T_{III})) \mid t \in [t_1, t_2]\},$$

$$\begin{aligned}
& \text{e.g.} \\
& a_{XXVII} = a_{XXVI} = a_{XXV} = a_{XXIV} = a_{XXIII} = 0 \\
& a_{XXII} = a_{XXI} = a_{XIX} = a_{XVI} = a_{XII} = 0 \\
& a_{XI} = a_X = a_{IX} = a_{VIII} = a_{VII} = a_{VI} = 0 \\
& a_V = a_{IV} = a_{III} = a_{II} = a_I = 0 \\
& a_{XX} = a_{XIX} = a_{XVIII} = a_{XVII} = a_{XVI} = a_{XV} = a_{XIII} = -1 \\
& \downarrow_{x_{A,B}} \mathbf{C} \left(\begin{matrix} x_E \\ x_E \end{matrix} \mathbf{O}^{[i+a_{XXVII}]} , \begin{matrix} g \\ g \end{matrix} \mathbf{S}^{[i]} \right) \\
& \downarrow_{x_E} \mathbf{C} \left(\begin{matrix} x_E \\ x_E \end{matrix} \mathbf{O}^{[i+a_{XXVI}]} , \begin{matrix} g \\ g \end{matrix} \mathbf{S}^{[i]} \right) \\
& \downarrow_{x_{A,B}} \mathbf{C} \left(\begin{matrix} x_B \\ x_B \end{matrix} \mathbf{O}^{[i+a_{XXV}]} , \begin{matrix} g \\ g \end{matrix} \mathbf{S}^{[i]} \right) \\
& \downarrow_{x_{A,B}} \mathbf{C} \left(\begin{matrix} x_A \\ x_A \end{matrix} \mathbf{O}^{[i+a_{XXIV}]} , \begin{matrix} g \\ g \end{matrix} \mathbf{S}^{[i]} \right) \\
& \left(\begin{matrix} g \\ g \end{matrix} \mathbf{E}^{[i]} , \begin{matrix} g \\ g \end{matrix} \mathbf{S}^{[i]} \right) = \uparrow_{x_E} \mathbf{G}_N \left(\begin{matrix} x_E \\ x_E \end{matrix} \mathbf{P}^{[i+a_{XXIII}]} \right) \\
& \left(\begin{matrix} g \\ g \end{matrix} \mathbf{E}^{[i]} , \begin{matrix} g \\ g \end{matrix} \mathbf{S}^{[i]} \right) = \uparrow_{x_B} \mathbf{G}_N \left(\begin{matrix} x_B \\ x_B \end{matrix} \mathbf{P}^{[i+a_{XXII}]} \right) \\
& \left(\begin{matrix} g \\ g \end{matrix} \mathbf{E}^{[i]} , \begin{matrix} g \\ g \end{matrix} \mathbf{S}^{[i]} \right) = \uparrow_{x_A} \mathbf{G}_N \left(\begin{matrix} x_A \\ x_A \end{matrix} \mathbf{P}^{[i+a_{XXI}]} \right)
\end{aligned}$$

$$\begin{aligned}
& \dots\dots\dots \\
& \begin{aligned}
x_E \mathbf{O}^{[i]} &= x_E \mathbf{R}_N^{[i+a_{XX}]} , \\
x_B \mathbf{O}^{[i]} &= x_B \mathbf{R}_N^{[i+a_{XIX}]} \\
x_A \mathbf{O}^{[i]} &= x_A \mathbf{R}_N^{[i+a_{XVIII}]} \\
x_E \mathbf{P}^{[i]} &= x_E \mathbf{R}^{[i+a_{XVII}]} \\
x_B \mathbf{P}^{[i]} &= (x_B \mathbf{R}^{[i+a_{XVI}]} , \mathbf{K}^{[i+a_{XVI}]}) \\
x_A \mathbf{P}^{[i]} &= (x_A \mathbf{R}^{[i+a_{XIII}]} , \mathbf{K}^{[i+a_{XIV}]})
\end{aligned}
\end{aligned}$$

$$\begin{aligned}
& \dots\dots\dots \\
& \text{and } \{ (\mathbf{K}^{[i+a_{XII}]}) \mid i \in \{ 1, \dots, n \} \} : \\
& \text{and } \{ (\begin{matrix} x_E \\ x_E \end{matrix} \mathbf{E}^{[i+a_{VIII}]} , \begin{matrix} x_E \\ x_E \end{matrix} \mathbf{R}^{[i+a_{IX}]} , \begin{matrix} x_E \\ x_E \end{matrix} \mathbf{E}_N^{[i+a_X]} , \begin{matrix} x_E \\ x_E \end{matrix} \mathbf{R}_N^{[i+a_{XI}]}) \mid i \in \{ 1, \dots, n \} \} , \\
& \text{and } \{ (\begin{matrix} x_B \\ x_B \end{matrix} \mathbf{E}^{[i+a_{IV}]} , \begin{matrix} x_B \\ x_B \end{matrix} \mathbf{R}^{[i+a_V]} , \begin{matrix} x_B \\ x_B \end{matrix} \mathbf{E}_N^{[i+a_{VI}]} , \begin{matrix} x_B \\ x_B \end{matrix} \mathbf{R}_N^{[i+a_{VII}]}) \mid i \in \{ 1, \dots, n \} \} , \\
& \text{With } \{ (\begin{matrix} x_A \\ x_A \end{matrix} \mathbf{E}^{[i]} , \begin{matrix} x_A \\ x_A \end{matrix} \mathbf{R}^{[i+a_I]} , \begin{matrix} x_A \\ x_A \end{matrix} \mathbf{E}_N^{[i+a_{II}]} , \begin{matrix} x_A \\ x_A \end{matrix} \mathbf{R}_N^{[i+a_{III}]}) \mid i \in \{ 1, \dots, n \} \} , \\
& \dots\dots\dots
\end{aligned}$$

or any subset thereof of equivalent thereto, including absent Keys.

In some embodiments, any other cryptographic configuration is deployed.

In other embodiments, another other configuration is used in training, e.g. RGB cameras combined with single channel Emitters and Recorders, or results are based on Generated data, or other Objective or Oppositional Objective Functions are used.

In some embodiments, these Agents are deployed without channel separation.

Cryptographic Agents

In some embodiments, Agents Alice, Bob, and Eve are trained according to any cryptographic scheme including but not limited to

$$\begin{aligned}
& \downarrow_{x_{A,B}} \mathbf{E}(\mathbf{O}(t+T_{XX}), {}^g\overline{\mathbf{S}}(t)) \\
& \downarrow_{x_E} \mathbf{C}(\mathbf{O}(t+T_{XIX}), {}^g\overline{\mathbf{S}}(t)) \\
& \downarrow_{x_{A,B}} \mathbf{C}(\mathbf{O}(t+T_{XVIII}), {}^g\overline{\mathbf{S}}(t)) \\
& ({}^g\overline{\mathbf{E}}(t), {}^g\overline{\mathbf{S}}(t)) = \uparrow_{x_E} \mathbf{G}({}_{x_E} \mathbf{P}(t+T_{XVII})) \\
& ({}^g\overline{\mathbf{E}}(t), {}^g\overline{\mathbf{S}}(t)) = \uparrow_{x_B} \mathbf{G}({}_{x_B} \mathbf{P}(t+T_{XVI})) \\
& ({}^g\overline{\mathbf{E}}(t), {}^g\overline{\mathbf{S}}(t)) = \uparrow_{x_A} \mathbf{G}({}_{x_A} \mathbf{P}(t+T_{XV})) \\
& \dots\dots\dots \\
& \mathbf{O}(t) = \mathbf{U}(t+T_{XIV}) \\
& {}_{x_E} \mathbf{P}(t) = {}_{x_E} \overline{\mathbf{R}}(t+T_{XIII}) \\
& {}_{x_B} \mathbf{P}(t) = ({}_{x_B} \overline{\mathbf{R}}(t+T_{XI}), \mathbf{K}(t+T_{XII})) \\
& {}_{x_A} \mathbf{P}(t) = ({}_{x_A} \overline{\mathbf{R}}(t+T_{VIII}), \mathbf{U}(t+T_{IX}), \mathbf{K}(t+T_X)) \\
& \dots\dots\dots \\
& \text{and } \{(\mathbf{U}(t+T_{VI}), \mathbf{K}(t+T_{VII})) \mid t \in [t_1, t_2]\}: \\
& \text{and } \{({}_{x_E} \overline{\mathbf{E}}(t+T_{IV}), {}_{x_E} \overline{\mathbf{R}}(t+T_V)) \mid t \in [t_1, t_2]\}, \\
& \text{and } \{({}_{x_B} \overline{\mathbf{E}}(t+T_{II}), {}_{x_B} \overline{\mathbf{R}}(t+T_{III})) \mid t \in [t_1, t_2]\}, \\
& \text{With } \{({}_{x_A} \overline{\mathbf{E}}(t), {}_{x_A} \overline{\mathbf{R}}(t+T_I)) \mid t \in [t_1, t_2]\}, \\
& \dots\dots\dots \\
& \text{e.g.} \\
& a_{XX} = a_{XIX} = a_{XVIII} = a_{XVII} = 0 \\
& a_{XVI} = a_{XV} = a_{XVII} = a_{XVI} = a_{XV} = a_X = a_{IX} = 0 \\
& a_{VII} = a_{VI} = a_V = a_{IV} = a_{III} = a_{II} = a_I = 0 \\
& a_{XIV} = a_{XIII} = a_{XII} = a_{XI} = a_{VIII} = -1 \\
& \downarrow_{x_{A,B}} \mathbf{E}(\mathbf{O}^{[i+a_{XX}]}, {}^g\overline{\mathbf{S}}^{[i]}) \\
& \downarrow_{x_E} \mathbf{C}(\mathbf{O}^{[i+a_{XIX}]}, {}^g\overline{\mathbf{S}}^{[i]}) \\
& \downarrow_{x_{A,B}} \mathbf{C}(\mathbf{O}^{[i+a_{XVIII}]}, {}^g\overline{\mathbf{S}}^{[i]}) \\
& ({}^g\overline{\mathbf{E}}^{[i]}, {}^g\overline{\mathbf{S}}^{[i]}) = \uparrow_{x_E} \mathbf{G}({}_{x_E} \mathbf{P}^{[i+a_{XVII}]}) \\
& ({}^g\overline{\mathbf{E}}^{[i]}, {}^g\overline{\mathbf{S}}^{[i]}) = \uparrow_{x_B} \mathbf{G}({}_{x_B} \mathbf{P}^{[i+a_{XVI}]}) \\
& ({}^g\overline{\mathbf{E}}^{[i]}, {}^g\overline{\mathbf{S}}^{[i]}) = \uparrow_{x_A} \mathbf{G}({}_{x_A} \mathbf{P}^{[i+a_{XV}]}) \\
& \dots\dots\dots \\
& \mathbf{O}^{[i]} = \mathbf{U}^{[i+a_{XIV}]} \\
& {}_{x_E} \mathbf{P}^{[i]} = {}_{x_E} \overline{\mathbf{R}}^{[i+a_{XIII}]} \\
& {}_{x_B} \mathbf{P}^{[i]} = ({}_{x_B} \overline{\mathbf{R}}^{[i+a_{XI}]}, \mathbf{K}^{[i+a_{XII}]}) \\
& {}_{x_A} \mathbf{P}^{[i]} = ({}_{x_A} \overline{\mathbf{R}}^{[i+a_{VIII}]}, \mathbf{U}^{[i+a_{IX}]}, \mathbf{K}^{[i+a_X]}) \\
& \dots\dots\dots \\
& \text{and } \{(\mathbf{U}^{[i+a_{VI}]}, \mathbf{K}^{[i+a_{VII}]}) \mid i \in \{1, \dots, n\}\}: \\
& \text{and } \{({}_{x_E} \overline{\mathbf{E}}^{[i+a_{IV}]}, {}_{x_E} \overline{\mathbf{R}}^{[i+a_V]}) \mid i \in \{1, \dots, n\}\}, \\
& \text{and } \{({}_{x_B} \overline{\mathbf{E}}^{[i+a_{II}]}, {}_{x_B} \overline{\mathbf{R}}^{[i+a_{III}]}) \mid i \in \{1, \dots, n\}\}, \\
& \text{With } \{({}_{x_A} \overline{\mathbf{E}}^{[i]}, {}_{x_A} \overline{\mathbf{R}}^{[i+a_I]}) \mid i \in \{1, \dots, n\}\}, \\
& \dots\dots\dots
\end{aligned}$$

or any subset thereof of equivalent thereto, including absent Keys.

Modification of the State of and Emitter-Scene-Recorder-Agent Loop According to an Objective Function

Agent Distortion

In some embodiments, Agents are trained with an Objective Function applied to generated Emissions, i.e.

$$\begin{aligned}
 & \downarrow L({}_x^g E(t+T_{IV})) \\
 & \downarrow C({}_x E(t+T_{III}), {}_x^g E(t)) \\
 & {}_x^g E(t) = \uparrow {}_x G({}_x R(t+T_{II})) \\
 & \dots\dots\dots \\
 & \text{With } \{({}_x E(t), {}_x R(t+T_I)) \mid t \in [t_1, t_2]\}: \\
 & \dots\dots\dots \\
 & \text{e.g.} \\
 & a_{IV} = a_{III} = a_I = 0 \\
 & a_{II} = -1 \\
 & \downarrow L({}_x^g E^{[i+a_{IV}]}) \\
 & \downarrow C({}_x E^{[i+a_{III}]}, {}_x^g E^{[i]}) \\
 & {}_x^g E^{[i]} = \uparrow {}_x G({}_x R^{[i+a_{II}]}) \\
 & \dots\dots\dots \\
 & \text{With } \{({}_x E^{[i]}, {}_x R^{[i+a_I]}) \mid i \in \{1, \dots, n\}\}: \\
 & \dots\dots\dots
 \end{aligned}$$

or any subset thereof, or equivalent thereto.

External Transform

Within the context of the invention, a Transform function, F , comprises a fast style transform [25], a Generator [21], any image-to-image, video-to-video [26],[27], audio-to-audio or other information transform function including selection of a dataset within a target domain or via human feedback i.e. the application of a Transform, F , to ${}_x E$ is described as $F({}_x E)$. The transformed output is labelled ${}_x \mathcal{A}$, and the resultant Recordings, ${}_x \mathcal{R}$.

In some embodiments, paired sets of Transformed and Untransformed data pairs are created by the Emission of paired Transformed and Untransformed Emissions on an Scene.

In some embodiments, an Agent comprises a the application of a Transform function to the output of a model trained to predict Emissions from Recordings i.e.

$$\begin{aligned} {}_x\mathbf{J}(t) &= \mathbf{R}(t+T_{III}) \\ {}_x\mathbf{J}(t) &= \mathbf{F}({}_x\mathbf{G}({}_x\mathbf{J}(t+T_{II}))) \end{aligned}$$

.....
 With $\{({}_xE(t), {}_x\mathbf{R}(t+T_I)) \mid t \in [t_1, t_2]\}$:

$$\begin{aligned} &\text{e.g.} \\ a_{III} &= a_I = 0 \\ a_{II} &= -1 \\ {}_x\mathbf{J}^{[i]} &= \mathbf{R}^{[i+a_{III}]} \\ {}_x\mathbf{J}^{[i]} &= \mathbf{F}({}_x\mathbf{G}({}_x\mathbf{R}^{[i+a_{II}]}) \end{aligned}$$

.....
 With $\{({}_xE^{[i]}, {}_x\mathbf{R}^{[i+a_I]}) \mid i \in \{1, \dots, n\}\}$:

or any subset thereof, or equivalent thereto.

In some embodiments, Agents are trained to map from any combination of Transformed or Untransformed Recordings to Transformed or Untransformed Emissions e.g.

$$\begin{aligned} \downarrow \mathbf{C}({}_xE(t+T_{VI}) \cup {}_x\mathbf{J}(t+T_{VII}), {}^gxE(t)) \\ {}^gxE(t) &= \uparrow {}_x\mathbf{G}({}_x\mathbf{R}(t+T_{IV}) \cup {}_x\mathbf{J}(t+T_V)) \end{aligned}$$

.....
 and $\{({}_x\mathbf{J}(t+T_{II}), {}_x\mathbf{J}(t+T_{III})) \mid t \in [t_1, t_2]\}$:
 With $\{({}_xE(t), {}_x\mathbf{R}(t+T_I)) \mid t \in [t_1, t_2]\}$,

$$\begin{aligned} &\text{e.g.} \\ a_{VII} &= a_{VI} = a_{III} = a_{II} = a_I = 0 \\ a_V &= a_{IV} = -1 \\ \downarrow \mathbf{C}({}_xE^{[i+a_{VI}]} \cup {}_x\mathbf{J}^{[i+a_{VII}]}, {}^gxE^{[i]}) \\ {}^gxE^{[i]} &= \uparrow {}_x\mathbf{G}({}_x\mathbf{R}^{[i+a_{IV}]} \cup {}_x\mathbf{J}^{[i+a_V]}) \end{aligned}$$

.....
 and $\{({}_x\mathbf{J}^{[i+a_{II}]}, {}_x\mathbf{J}^{[i+a_{III}]}) \mid i \in \{1, \dots, n\}\}$:
 With $\{({}_xE^{[i]}, {}_x\mathbf{R}^{[i+a_I]}) \mid i \in \{1, \dots, n\}\}$,

or any subset thereof, or equivalent thereto.

Generation of Synthetic Loop Element Data in a Styled Domain

In some embodiments, Domain Generators are used to Generate Emitter-Scene-Recorder Loop data and are trained against Objective functions including but not limited to similarity of Generated data with input Emitter-Recorder Loop data, or Domain Discriminators trained on Emitter-Recorder Loop data, or Domain Discriminators trained on or Transformed Emitter-Recorder Loop data, Output label model correspondence, any other Objective function applied to any subset of Generated data, or any other industry standard generative technique i.e.

$$\begin{aligned}
& \downarrow L({}_x^g \tilde{E}(t+T_{XVII})), \downarrow L({}_x^g \tilde{R}(t+T_{XVIII})), \downarrow L({}_x^g \tilde{S}(t+T_{XIX})) \\
& \downarrow D({}_x \mathcal{E}(t+T_{XIV}), {}_x \mathcal{R}(t+T_{XV}), {}_x \mathcal{Z}(t+T_{XVI}))({}_x^g \tilde{E}(t+T_{VII}), {}_x^g \tilde{R}(t+T_{VIII}), {}_x^g \tilde{S}(t+T_{IX})) \\
& \quad \downarrow C(O(t+T_{XIII}), {}_x^g \tilde{S}(t+T_{IX})) \\
& \downarrow C(({}_x E(t+T_{VII}), {}_x R(t+T_{VIII}), {}_x S(t+T_{IX})), ({}_x^g \tilde{E}(t+T_{VII}), {}_x^g \tilde{R}(t+T_{VIII}), {}_x^g \tilde{S}(t+T_{IX}))), \\
& \quad \downarrow D({}_x E(t+T_{VII}), {}_x R(t+T_{VIII}), {}_x S(t+T_{IX}))(({}_x^g \tilde{E}(t+T_{VII}), {}_x^g \tilde{R}(t+T_{VIII}), {}_x^g \tilde{S}(t+T_{IX})) \\
& \quad {}_x^g \tilde{E}(t+T_X), {}_x^g \tilde{R}(t+T_{XI}), {}_x^g \tilde{S}(t+T_{XII})) = \uparrow \tilde{G}(({}_x E(t+T_{VII}), {}_x R(t+T_{VIII}), {}_x S(t+T_{IX}))) \\
& \dots\dots\dots \\
& \text{and } \{({}_x \mathcal{E}(t+T_{IV}), {}_x \mathcal{R}(t+T_V), {}_x \mathcal{S}(t+T_{VI})) | t \in [t_1, t_2]\}: \\
& \text{With } \{({}_x E(t), {}_x R(t+T_I), {}_x S(t+T_{II}), O(t+T_{III})) | t \in [t_1, t_2]\}, \\
& \dots\dots\dots
\end{aligned}$$

$$\begin{aligned}
& \text{e.g.} \\
& a_{XIX} = a_{XVIII} = a_{XVII} = a_{XVI} = a_{XV} = a_{XIV} = 0 \\
& a_{XIII} = a_{XII} = a_{XI} = a_X = a_{IX} = a_{VIII} = 0 \\
& a_{VII} = a_{VI} = a_V = a_{IV} = a_{III} = a_{II} = a_I = 0 \\
& \downarrow L({}_x^g \tilde{E}^{[i+a_{XVII}]}), \downarrow L({}_x^g \tilde{R}^{[i+a_{XVIII}]}), \downarrow L({}_x^g \tilde{S}^{[i+a_{XIX}]}) \\
& \downarrow D({}_x \mathcal{E}^{[i+a_{XIV}]}, {}_x \mathcal{R}^{[i+a_{XV}]}, {}_x \mathcal{Z}^{[i+a_{XVI}]})({}_x^g \tilde{E}^{[i+a_{VII}]}, {}_x^g \tilde{R}^{[i+a_{VIII}]}, {}_x^g \tilde{S}^{[i+a_{IX}]}) \\
& \quad \downarrow C(O^{[i+a_{XIII}]}, {}_x^g \tilde{S}^{[i+a_{IX}]}) \\
& \downarrow C(({}_x E^{[i+a_{VII}]}, {}_x R^{[i+a_{VIII}]}, {}_x S^{[i+a_{IX}]}) , ({}_x^g \tilde{E}^{[i+a_X]}, {}_x^g \tilde{R}^{[i+a_{XI}]}, {}_x^g \tilde{S}^{[i+a_{XII}]}) \\
& \quad \downarrow D({}_x E^{[i+a_{VII}]}, {}_x R^{[i+a_{VIII}]}, {}_x S^{[i+a_{IX}]})({}_x^g \tilde{E}^{[i+a_X]}, {}_x^g \tilde{R}^{[i+a_{XI}]}, {}_x^g \tilde{S}^{[i+a_{XII}]}) \\
& \quad {}_x^g \tilde{E}^{[i+a_X]}, {}_x^g \tilde{R}^{[i+a_{XI}]}, {}_x^g \tilde{S}^{[i+a_{XII}]} = \uparrow \tilde{G}(({}_x E^{[i+a_{VII}]}, {}_x R^{[i+a_{VIII}]}, {}_x S^{[i+a_{IX}]}) \\
& \dots\dots\dots \\
& \text{and } \{({}_x \mathcal{E}^{[i+a_{IV}]}, {}_x \mathcal{R}^{[i+a_V]}, {}_x \mathcal{S}^{[i+a_{VI}]}) | i \in \{1, \dots, n\}\}: \\
& \text{With } \{({}_x E^{[i]}, {}_x R^{[i+a_I]}, {}_x S^{[i+a_{II}]}, O^{[i+a_{III}]}) | i \in \{1, \dots, n\}\}, \\
& \dots\dots\dots
\end{aligned}$$

or any subset thereof, or equivalent thereto.

In some embodiments, this data is Emitted directly or Agents are trained on Generated data i.e.

$$\begin{aligned}
& \downarrow C(({}_x \tilde{E}(t+T_{VI}), {}_x \tilde{S}(t+T_V)), ({}_x^g E(t), {}_x^g S(t+T_{IV}))) \\
& \quad ({}_x^g E(t), {}_x^g S(t+T_{IV})) = \uparrow {}_x G({}_x \tilde{R}(t+T_{III})) \\
& \dots\dots\dots \\
& \text{With } \{({}_x \tilde{E}(t), {}_x \tilde{R}(t+T_I), {}_x \tilde{S}(t+T_{II})) | t \in [t_1, t_2]\}: \\
& \dots\dots\dots \\
& \text{e.g.} \\
& a_V = a_{IV} = a_{III} = -1 \\
& a_{VI} = a_{II} = a_I = 0, \\
& \downarrow C(({}_x \tilde{E}^{[i+a_{VI}]}, {}_x^g \tilde{S}^{[i+a_V]}), ({}_x^g E^{[i]}, {}_x^g S^{[i+a_{IV}]}) \\
& \quad ({}_x^g E^{[i]}, {}_x^g S^{[i+a_{IV}]})) = \uparrow {}_x G({}_x \tilde{R}^{[i+a_{III}]}) \\
& \dots\dots\dots \\
& \text{With } \{({}_x \tilde{E}^{[i]}, {}_x \tilde{R}^{[i+a_I]}, {}_x \tilde{S}^{[i+a_{II}]}) | i \in \{1, \dots, n\}\}: \\
& \dots\dots\dots
\end{aligned}$$

or any subset thereof, or equivalent thereto.

External Elements

In some embodiments, Loop data is Generated with an objective function applied to external elements i.e.

$$\begin{aligned} & \downarrow L({}_y\tilde{\mathbf{R}}(t+T_{IX})) \\ & \downarrow D({}_xE(t+T_{III}), {}_x\mathbf{R}(t+T_{IV}), {}_y\mathbf{R}(t+T_V))({}_x\tilde{\mathbf{E}}(t+T_{VI}), {}_x\tilde{\mathbf{R}}(t+T_{VII}), {}_y\tilde{\mathbf{R}}(t+T_{VIII})) \\ & \downarrow C(({}_xE(t+T_{III}), {}_x\mathbf{R}(t+T_{IV}), {}_y\mathbf{R}(t+T_V)), ({}_x\tilde{\mathbf{E}}(t+T_{VI}), {}_x\tilde{\mathbf{R}}(t+T_{VII}), {}_y\tilde{\mathbf{R}}(t+T_{VIII}))) \\ & ({}_x\tilde{\mathbf{E}}(t+T_{VI}), {}_x\tilde{\mathbf{R}}(t+T_{VII}), {}_y\tilde{\mathbf{R}}(t+T_{VIII})) = \uparrow \tilde{\mathbf{G}}(({}_xE(t+T_{III}), {}_x\mathbf{R}(t+T_{IV}), {}_y\mathbf{R}(t+T_V))) \end{aligned}$$

$$\begin{aligned} & \text{and } \{({}_y\mathbf{R}(t+T_{II})) \mid t \in [t_1, t_2]\}: \\ & \text{With } \{({}_xE(t), {}_x\mathbf{R}(t+T_I)) \mid t \in [t_1, t_2]\}, \end{aligned}$$

e.g.

$$a_{IX} = a_{VIII} = a_{VII} = a_{VI} = a_V = a_{IV} = a_{II} = a_I = 0$$

$$\begin{aligned} & \downarrow L({}_y\tilde{\mathbf{R}}^{[i+a_{IX}]}) \\ & \downarrow D({}_xE^{[i+a_{III}]}, {}_x\mathbf{R}^{[i+a_{IV}]}, {}_y\mathbf{R}^{[i+a_V]})({}_x\tilde{\mathbf{E}}^{[i+a_{VI}]}, {}_x\tilde{\mathbf{R}}^{[i+a_{VII}]}, {}_y\tilde{\mathbf{R}}^{[i+a_{VIII}]}) \\ & \downarrow C(({}_xE^{[i+a_{III}]}, {}_x\mathbf{R}^{[i+a_{IV}]}, {}_y\mathbf{R}^{[i+a_V]}), ({}_x\tilde{\mathbf{E}}^{[i+a_{VI}]}, {}_x\tilde{\mathbf{R}}^{[i+a_{VII}]}, {}_y\tilde{\mathbf{R}}^{[i+a_{VIII}]}) \\ & ({}_x\tilde{\mathbf{E}}^{[i+a_{VI}]}, {}_x\tilde{\mathbf{R}}^{[i+a_{VII}]}, {}_y\tilde{\mathbf{R}}^{[i+a_{VIII}]}) = \uparrow \tilde{\mathbf{G}}({}_xE^{[i+a_{III}]}, {}_x\mathbf{R}^{[i+a_{IV}]}, {}_y\mathbf{R}^{[i+a_V]}) \end{aligned}$$

$$\begin{aligned} & \text{and } \{({}_y\mathbf{R}^{[i+a_{II}]}) \mid i \in \{1, \dots, n\}\}: \\ & \text{With } \{({}_xE^{[i]}, {}_x\mathbf{R}^{[i+a_I]}) \mid i \in \{1, \dots, n\}\}, \end{aligned}$$

or any subset thereof, or equivalent thereto, and this data emitted directly or used in the training of Agents, optionally provided any industry standard metadata, i.e. ${}_y\tilde{\mathbf{R}}(t)$ or ${}_y\tilde{\mathbf{E}}(t)$, i.e.

$$\begin{aligned} & \downarrow C({}_x\tilde{\mathbf{E}}(t+T_V), {}_xE(t)) \\ & {}_xE(t) = \uparrow {}_x\mathbf{G}({}_x\tilde{\mathbf{R}}(t+T_{III}), {}_y\tilde{\mathbf{R}}(t+T_{IV})) \\ & \text{and } \{({}_y\tilde{\mathbf{R}}(t+T_{II})) \mid t \in [t_1, t_2]\}: \\ & \text{With } \{({}_x\tilde{\mathbf{E}}(t), {}_x\tilde{\mathbf{R}}(t+T_I)) \mid t \in [t_1, t_2]\}, \end{aligned}$$

e.g.

$$a_V = a_{II} = a_I = 0$$

$$a_{IV} = a_{III} = -1$$

$$\downarrow C({}_x\tilde{\mathbf{E}}^{[i+a_V]}, {}_xE^{[i]})$$

$${}_xE^{[i]} = \uparrow {}_x\mathbf{G}({}_x\tilde{\mathbf{R}}^{[i+a_{III}]}, {}_y\tilde{\mathbf{R}}^{[i+a_{IV}]})$$

$$\begin{aligned} & \text{and } \{({}_y\tilde{\mathbf{R}}^{[i+a_{II}]}) \mid i \in \{1, \dots, n\}\}: \\ & \text{With } \{({}_x\tilde{\mathbf{E}}^{[i]}, {}_x\tilde{\mathbf{R}}^{[i+a_I]}) \mid i \in \{1, \dots, n\}\}, \end{aligned}$$

or any subset thereof. Or equivalent thereto.

In some embodiments, any methods are used in the prediction of external elements e.g. with known external Emissions, $_{y_A}E$, and $_{y_B}E$ (e.g. multiple known songs or films for projection), or external Recorder, $_yR$, Agent x is trained to predict these separate known Emissions, or predict or modify these external Recording i.e.

$$\begin{aligned}
 & \downarrow L(_yR(t+T_{IX})) \\
 & \downarrow C((_{y_A}E(t+T_{VI}), _{y_B}E(t+T_{VII}), _yR(t+T_{VIII}), ^gS(t)) \\
 & \quad ({}^gE(t), {}^gS(t)) \uparrow_x G(_xR(t+T_V)) \\
 & \dots\dots\dots \\
 & \text{and } \{(_{y_A}E(t+T_{II}), _{y_B}E(t+T_{III}), _yR(t+T_{IV})) | t \in [t_1, t_2]\} : \\
 & \quad \text{With } \{(_xR(t), {}_xR(t+T_I)) | t \in [t_1, t_2]\}, \\
 & \dots\dots\dots \\
 & \text{e.g.} \\
 & a_{IX} = a_{VIII} = a_{VII} = a_{III} = a_{II} = a_I = 0 \\
 & a_{VIII} = a_{VII} = a_{VI} = a_V = -1 \\
 & \downarrow L(_yR^{[i+a_{IX}]}) \\
 & \downarrow C((_{y_A}E^{[i+a_{VI}]}, _{y_B}E^{[i+a_{VII}]}, _yR^{[i+a_{VIII}]}, ^gS^{[i]}) \\
 & \quad ({}^gE^{[i]}, {}^gS^{[i]}) = \uparrow_x G(_xR^{[i+a_V]}) \\
 & \dots\dots\dots \\
 & \{(_{y_A}E^{[i+a_{II}]}, _{y_B}E^{[i+a_{III}]}, _yR^{[i+a_{IV}]}) | i \in \{1, \dots, n\}\} : \\
 & \quad \text{With } \{(_xR^{[i]}, {}_xR^{[i+a_I]}) | i \in \{1, \dots, n\}\}, \\
 & \dots\dots\dots
 \end{aligned}$$

or any subset thereof, or equivalent thereto.

In some embodiments, Agents are provided estimated metadata regarding an intended viewer in deployment, absent the presence of element y which was present for training e.g. the position of a human head derived via any industry standard method e.g. any industry standard head-tracking mechanism.

Multi-Agent Modification

In some embodiments, Agents are trained directly as Agent models in multi-Agent systems including but not limited to the training of Agent x_A to predict or modify any component of Agent x_B and vice versa while they train to oppose Eve, i.e.

$$\begin{aligned}
& \downarrow_{x_{A,B}} \mathbf{L}(\mathbf{R}(t+T_{XX})) \\
& \downarrow_{x_E} \mathbf{L}(\mathbf{R}(t+T_{XIX})) \\
& \downarrow_{x_{A,B}} \mathbf{L}(\mathbf{R}(t+T_{XVIII})) \\
& \downarrow_{x_{A,B}} \mathbf{L}(\mathbf{R}(t+T_{XVII})) \\
& \downarrow_{x_{A,B}} \mathbf{C}((\mathbf{R}(t+T_{XV}), \mathbf{R}(t+T_{XVI})), \mathbf{S}^{[i]}) \\
& \downarrow_{x_E} \mathbf{C}((\mathbf{R}(t+T_{XIII}), \mathbf{R}(t+T_{XIV})), \mathbf{S}^{[i]}) \\
& \downarrow_{x_{A,B}} \mathbf{C}((\mathbf{R}(t+T_{XI}), \mathbf{R}(t+T_{XII})), \mathbf{S}^{[i]}) \\
& \downarrow_{x_{A,B}} \mathbf{C}((\mathbf{R}(t+T_{IX}), \mathbf{R}(t+T_X)), \mathbf{S}^{[i]}) \\
& (\mathbf{E}^{[i]}, \mathbf{S}^{[i]}) = \uparrow_{x_E} \mathbf{G}(\mathbf{R}(t+T_{VIII})) \\
& (\mathbf{E}^{[i]}, \mathbf{S}^{[i]}) = \uparrow_{x_B} \mathbf{G}(\mathbf{R}(t+T_{VII})) \\
& (\mathbf{E}^{[i]}, \mathbf{S}^{[i]}) = \uparrow_{x_A} \mathbf{G}(\mathbf{R}(t+T_{VI}))
\end{aligned}$$

.....
and $\{(\mathbf{E}(t+T_{IV}), \mathbf{R}(t+T_V)) \mid t \in [t_1, t_2]\}$:
and $\{(\mathbf{E}(t+T_{II}), \mathbf{R}(t+T_{III})) \mid t \in [t_1, t_2]\}$,
With $\{(\mathbf{E}(t), \mathbf{R}(t+T_I)) \mid t \in [t_1, t_2]\}$,
.....

e.g.

$$\begin{aligned}
a_{XX} &= a_{XIX} = a_{XVIII} = a_{XVII} = a_{VII} = 0 \\
a_V &= a_{IV} = a_{III} = a_{II} = a_I = 0 \\
a_{XVI} &= a_{XV} = a_{XIV} = a_{XIII} = a_{XII} = -1 \\
a_{XI} &= a_X = a_{IX} = a_{VIII} = a_{VII} = a_{VI} = -1 \\
& \downarrow_{x_{A,B}} \mathbf{L}(\mathbf{R}^{[i+a_{XX}]}) \\
& \downarrow_{x_E} \mathbf{L}(\mathbf{R}^{[i+a_{XIX}]}) \\
& \downarrow_{x_{A,B}} \mathbf{L}(\mathbf{R}^{[i+a_{XVIII}]}) \\
& \downarrow_{x_{A,B}} \mathbf{L}(\mathbf{R}^{[i+a_{XVII}]}) \\
& \downarrow_{x_{A,B}} \mathbf{C}((\mathbf{R}^{[i+a_{XV}]}, \mathbf{R}^{[i+a_{XVI}]}) , \mathbf{S}^{[i]}) \\
& \downarrow_{x_E} \mathbf{C}((\mathbf{R}^{[i+a_{XIII}]}, \mathbf{R}^{[i+a_{XIV}]}) , \mathbf{S}^{[i]}) \\
& \downarrow_{x_{A,B}} \mathbf{C}((\mathbf{R}^{[i+a_{XI}]}, \mathbf{R}^{[i+a_{XII}]}) , \mathbf{S}^{[i]}) \\
& \downarrow_{x_{A,B}} \mathbf{C}((\mathbf{R}^{[i+a_{IX}]}, \mathbf{R}^{[i+a_X]}) , \mathbf{S}^{[i]}) \\
& (\mathbf{E}^{[i]}, \mathbf{S}^{[i]}) = \uparrow_{x_E} \mathbf{G}(\mathbf{R}^{[i+a_{VIII}]}) \\
& (\mathbf{E}^{[i]}, \mathbf{S}^{[i]}) = \uparrow_{x_B} \mathbf{G}(\mathbf{R}^{[i+a_{VII}]}) \\
& (\mathbf{E}^{[i]}, \mathbf{S}^{[i]}) = \uparrow_{x_A} \mathbf{G}(\mathbf{R}^{[i+a_{VI}]})
\end{aligned}$$

.....
and $\{(\mathbf{E}^{[i+a_{IV}]}, \mathbf{R}^{[i+a_V]}) \mid i \in \{1, \dots, n\}\}$:
and $\{(\mathbf{E}^{[i+a_{II}]}, \mathbf{R}^{[i+a_{III}]}) \mid i \in \{1, \dots, n\}\}$,
With $\{(\mathbf{E}^{[i]}, \mathbf{R}^{[i+a_I]}) \mid i \in \{1, \dots, n\}\}$,
.....

or any subset thereof, or equivalent thereto.

In some embodiments, Scenes are treated as untrainable layers in a neural network or any machine learning network via any industry standard technique.

Use of Emitter-Reactor-Recorder Loops In Computation

Within the context of the invention, an Information Reactor comprises any volume capable of passing a signal from an Emitter to a Recorder e.g. a optical or acoustic lens, mirror, resonance cavity, non-linear medium, orb, air gap, diffraction grating, vacuum, or any medium.

In some embodiments, the invention comprises any number of Emitters, Reactors, and Recorders, with Emissions originating via Agents, or a Visualisation functions applied to a One-Way Function applied to any permutation of Recorder inputs. Within the context of the invention, these Agents are labelled ${}_zG$.

An Emitter-Reactor-Recorder network is illustrated in Figure 8. A two-input Emitter-Reactor-Recorder network is illustrated in Figure 9. In some embodiments, Reactor Agents are recursive e.g. ${}_zE_{II}(t) = {}_zE_I(t+T_I), {}_zE_I = G({}_zR(t+T_{II}))$, ${}_zE_{II}^{[i]} = {}_zE_I^{[i+a_I]}, {}_zE_I = G({}_zR^{[i+a_{II}]})$, e.g. $a_{II} = a_I = -1$, or ${}_zE_{II}(t) = {}_zR(t+T)$, ${}_zE_{II} = G_J({}_zR(t+T))$, e.g. ${}_zE_{II}^{[i]} = {}_zR^{[i+a]}$, ${}_zE_I = G_J({}_zR^{[i+a]})$, e.g. $a = -1$.

Multiple Emitter-Reactor-Recorder Architecture

Figure 10 illustrates a basic structure for any number of Emitters and Recorders configured as a fully connected network with a Reactor in the projector-Reactor-camera network comprising a mirrored resonance cavity.

Within the context of the invention, an Array comprises a larger number of Emitters or Recorders. In some embodiments, any pattern of Emitters and/or Recorders is used as an array, including but not limited to a spiral pattern, as illustrated in Figure 11.

A fully connected layer of Emitters mated to a Reactor mated to and array of Recorders is illustrated in Figure 12. In some embodiments, the coupling is smooth, without sharp edges. In some embodiments, a signal damper (e.g. matte black paint or sound-insulating material) is applied at any point, including but not limited to at the mounting plate and Emitter-Recorder mounting points.

Basic Function

In some embodiments, Loop z is trained to act as a function including but not limited to in the modification of Loop x :

$$\begin{aligned}
& \downarrow C(O(t+T_{XVI}), S_z^g(t)) \\
& {}_z^g S(t) = \uparrow {}_z G_S({}_z R_S(t+T_{XV})) \\
& {}_z^g E_P(t) = \uparrow {}_z G_P(P(t+T_{XIV})) \\
& \dots\dots\dots \\
& O^{[i]} = {}_x E(t+T_{XI}) \cup {}_x \mathcal{I}(t+T_{XII}) \cup {}_x \widetilde{E}(t+T_{XIII}) \\
& P^{[i]} = {}_x R(t+T_{VIII}) \cup {}_x \mathcal{J}(t+T_{IX}) \cup {}_x \widetilde{R}(t+T_X) \\
& \dots\dots\dots \\
& \text{and } \{({}_x \widetilde{E}(t+T_{VI}), {}_x \widetilde{R}(t+T_{VII})) \mid t \in [t_1, t_2]\}: \\
& \text{and } \{({}_x \mathcal{I}(t+T_{IV}), {}_x \mathcal{J}(t+T_V)) \mid t \in [t_1, t_2]\}, \\
& \text{and } \{({}_x E(t+T_{II}), {}_x R(t+T_{III})) \mid t \in [t_1, t_2]\}, \\
& \text{With } \{({}_z E_P(t), {}_z R_S(t+T_I)) \mid t \in [t_1, t_2]\}, \\
& \dots\dots\dots \\
& \text{e.g.} \\
& a_{XVI} = a_{XV} = a_{XIV} = a_{XIII} = a_{XII} = 0 \\
& a_{XI} = a_{VI} = a_V = a_{IV} = a_{III} = a_{II} = a_I = 0 \\
& a_X = a_{IX} = a_{VIII} = -1 \\
& \downarrow C(O^{[i+a_{XVI}]}, {}_z^g S^{[i]}) \\
& {}_z^g S^{[i]} = \uparrow {}_z G_S({}_z R_S^{[i+a_{XV}]}) \\
& {}_z^g E_P^{[i]} = \uparrow {}_z G_P(P^{[i+a_{XIV}]}) \\
& \dots\dots\dots \\
& O^{[i]} = {}_x E^{[i+a_{XI}]} \cup {}_x \mathcal{I}^{[i+a_{XII}]} \cup {}_x \widetilde{E}^{[i+a_{XIII}]} \\
& P^{[i]} = {}_x R^{[i+a_{VIII}]} \cup {}_x \mathcal{J}^{[i+a_{IX}]} \cup {}_x \widetilde{R}^{[i+a_X]} \\
& \dots\dots\dots \\
& \text{and } \{({}_x \widetilde{E}^{[i+a_{VI}]}, {}_x \widetilde{R}^{[i+a_{VII}]}) \mid i \in \{1, \dots, n\}\}: \\
& \text{and } \{({}_x \mathcal{I}^{[i+a_V]}, {}_x \mathcal{J}^{[i+a_{IV}]}) \mid i \in \{1, \dots, n\}\}, \\
& \text{and } \{({}_x E^{[i+a_{III}]}, {}_x R^{[i+a_{IV}]}) \mid i \in \{1, \dots, n\}\}, \\
& \text{With } \{({}_z E_P^{[i]}, {}_z R_S^{[i+a_{II}]}) \mid i \in \{1, \dots, n\}\}, \\
& \dots\dots\dots
\end{aligned}$$

OR

$$\begin{aligned}
& \downarrow C(O(t+T_{XVI}), {}^gS(t)) \\
& ({}^gE_P(t), {}^gS(t)) = \uparrow {}_zG({}_zR_S(t+T_{XIV}), P(t+T_{XV})) \\
& \dots\dots\dots \\
& O^{[i]} = {}_x E(t+T_{XI}) \cup {}_x \mathfrak{A}(t+T_{XII}) \cup {}_x \widetilde{E}(t+T_{XIII}) \\
& P^{[i]} = {}_x R(t+T_{VIII}) \cup {}_x \mathfrak{Y}(t+T_{IX}) \cup {}_x \widetilde{R}(t+T_X) \\
& \dots\dots\dots \\
& \text{and } \{({}_x \widetilde{E}(t+T_{VI}), {}_x \widetilde{R}(t+T_{VII})) \mid t \in [t_1, t_2]\} : \\
& \text{and } \{({}_x \mathfrak{A}(t+T_{IV}), {}_x \mathfrak{Y}(t+T_V)) \mid t \in [t_1, t_2]\}, \\
& \text{and } \{({}_x E(t+T_{II}), {}_x R(t+T_{III})) \mid t \in [t_1, t_2]\}, \\
& \text{With } \{({}_z E_P(t), {}_z R_S(t+T_I)) \mid t \in [t_1, t_2]\}, \\
& \dots\dots\dots \\
& \text{e.g.} \\
& a_{XVI} = a_{XV} = a_{XIV} = a_{XIII} = a_{XII} = 0 \\
& a_{XI} = a_{VI} = a_V = a_{IV} = a_{III} = a_{II} = a_I = 0 \\
& a_X = a_{IX} = a_{VIII} = -1 \\
& \downarrow C(O^{[i+a_{XVI}]}, {}^gS^{[i]}) \\
& ({}^gE_P^{[i]}, {}^gS^{[i]}) = \uparrow {}_zG({}_zR_S^{[i+a_{XIV}]}, P^{[i+a_{XV}]}) \\
& \dots\dots\dots \\
& O^{[i]} = {}_x E^{[i+a_{XI}]} \cup {}_x \mathfrak{A}^{[i+a_{XII}]} \cup {}_x \widetilde{E}^{[i+a_{XIII}]} \\
& P^{[i]} = {}_x R^{[i+a_{VIII}]} \cup {}_x \mathfrak{Y}^{[i+a_{IX}]} \cup {}_x \widetilde{R}^{[i+a_X]} \\
& \dots\dots\dots \\
& \text{and } \{({}_x \widetilde{E}^{[i+a_{VI}]}, {}_x \widetilde{R}^{[i+a_{VII}]}) \mid i \in \{1, \dots, n\}\} : \\
& \text{and } \{({}_x \mathfrak{A}^{[i+a_V]}, {}_x \mathfrak{Y}^{[i+a_{IV}]}) \mid i \in \{1, \dots, n\}\}, \\
& \text{and } \{({}_x E^{[i+a_{III}]}, {}_x R^{[i+a_{IV}]}) \mid i \in \{1, \dots, n\}\}, \\
& \text{With } \{({}_z E_P^{[i]}, {}_z R_S^{[i+a_{II}]}) \mid i \in \{1, \dots, n\}\}, \\
& \dots\dots\dots
\end{aligned}$$

or any subset thereof, or equivalent thereto.

Reactor as One-Way Function

In some embodiments, Emitter-Reactor-Recorder Loops are trained to act as One-Way Functions for external systems or for the training of Discriminators, i.e.

$$\begin{aligned}
& \uparrow D_{(\bar{E}_P(t+T_{VI}), \bar{R}_S(t+T_{VII}), \bar{E}(t+T_{VIII}), \bar{R}(t+T_{IX}))} \\
& \quad (\bar{E}_P(t+T_{VI}), \bar{R}_S(t+T_{VII}), \bar{E}(t+T_{VIII}), \bar{R}(t+T_{IX})) \\
& \quad ({}^g\bar{E}_P(t), {}^g\bar{E}(t)) = \uparrow {}_z G_D({}_z \bar{R}_S(t+T_{IV}), {}_x \bar{R}(t+T_V)) \\
& \quad \dots\dots\dots \\
& \quad \text{and } \{({}_x \bar{E}(t+T_{II}), {}_x \bar{R}(t+T_{III})) \mid t \in [t_1, t_2]\} : \\
& \quad \text{With } \{({}_z \bar{E}_P(t), {}_z \bar{R}_S(t+T_I)) \mid t \in [t_1, t_2]\}, \\
& \quad \dots\dots\dots \\
& \quad \text{e.g.} \\
& \quad a_{IX} = a_{VIII} = a_{VII} = a_{VI} = a_{III} = a_{II} = a_I = 0 \\
& \quad a_V = a_{IV} = -1 \\
& \uparrow D_{(\bar{E}_P^{[i+a_{VI}]}, \bar{R}_S^{[i+a_{VII}]}, \bar{E}^{[i+a_{VIII}]}, \bar{R}^{[i+a_{IX}]})} ({}_z \bar{E}_P^{[i+a_{VI}]}, {}_z \bar{R}_S^{[i+a_{VII}]}, {}_x \bar{E}^{[i+a_{VIII}]}, {}_x \bar{R}^{[i+a_{IX}]}) \\
& \quad ({}^g\bar{E}_P^{[i]}, {}^g\bar{E}^{[i]}) = \uparrow {}_z G_D({}_z \bar{R}_S^{[i+a_{IV}]}, {}_x \bar{R}^{[i+a_V]}) \\
& \quad \dots\dots\dots \\
& \quad \text{and } \{({}_x \bar{E}^{[i+a_{II}]}, {}_x \bar{R}^{[i+a_{III}]}) \mid i \in \{1, \dots, n\}\} : \\
& \quad \text{With } \{({}_z \bar{E}_P^{[i]}, {}_z \bar{R}_S^{[i+a_I]}) \mid i \in \{1, \dots, n\}\}, \\
& \quad \dots\dots\dots
\end{aligned}$$

or any subset thereof, or equivalent thereto.

Reactor for Direct Modification

In some embodiments, Emitter-Reactor-Recorder-Agent systems are deployed in the director modification of Emitter-Scene-Recorder systems, i.e.

$$\begin{aligned} & \downarrow D_{(\widetilde{E}(t+T_{xvi}), \widetilde{R}(t+T_{xvii}))}({}_x E(t+T_{xiv}), {}_x R(t+T_{xv})) \\ & \downarrow D_{({}_x \mathcal{E}(t+T_{xii}), {}_x \mathcal{R}(t+T_{xiii}))}({}_x E(t+T_x), {}_x R(t+T_{xi})) \\ & ({}_z E_P(t), {}_z E(t)) = \uparrow {}_z G({}_z R_S(t+T_{viii}), {}_x R(t+T_{ix})) \end{aligned}$$

$$\begin{aligned} & \text{and } \{(\widetilde{E}(t+T_{vi}), \widetilde{R}(t+T_{vii})) \mid t \in [t_1, t_2]\}: \\ & \text{and } \{({}_x \mathcal{E}(t+T_{iv}), {}_x \mathcal{R}(t+T_v)) \mid t \in [t_1, t_2]\}, \\ & \text{and } \{({}_x E(t+T_{ii}), {}_x R(t+T_{iii})) \mid t \in [t_1, t_2]\}, \\ & \text{With } \{({}_z E_P(t), {}_z R_S(t+T_i)) \mid t \in [t_1, t_2]\}, \end{aligned}$$

e.g.

$$\begin{aligned} a_{xv} &= a_{xiv} = a_{xiii} = a_{xii} = a_{xi} = a_x = a_{ix} = 0 \\ a_{vii} &= a_{vi} = a_v = a_{vi} = a_{iii} = a_{ii} = a_i = 0 \\ a_{ix} &= a_{viii} = -1 \\ & \downarrow D_{(\widetilde{E}^{[i+a_{xvi}]}, \widetilde{R}^{[i+a_{xvii}]})}({}_x E^{[i+a_{xiv}]}, {}_x R^{[i+a_{xv}]}) \\ & \downarrow D_{({}_x \mathcal{E}^{[i+a_{xii}]}, {}_x \mathcal{R}^{[i+a_{xiii}]})}({}_x E^{[i+a_x]}, {}_x R^{[i+a_{xi}]}) \\ & ({}_z E_P^{[i]}, {}_z E^{[i]}) = \uparrow {}_z G({}_z R_S^{[i+a_{viii}]}, {}_x R^{[i+a_{ix}]}) \end{aligned}$$

$$\begin{aligned} & \text{and } \{(\widetilde{E}^{[i+a_{vi}]}, \widetilde{R}^{[i+a_{vii}]}) \mid i \in \{1, \dots, n\}\}: \\ & \text{and } \{({}_x \mathcal{E}^{[i+a_v]}, {}_x \mathcal{R}^{[i+a_{vii}]}) \mid i \in \{1, \dots, n\}\}, \\ & \text{and } \{({}_x E^{[i+a_{ii}]}, {}_x R^{[i+a_{iii}]}) \mid i \in \{1, \dots, n\}\}, \\ & \text{With } \{({}_z E_P^{[i]}, {}_z R_S^{[i+a_{ii}]})) \mid i \in \{1, \dots, n\}\}, \end{aligned}$$

or any subset thereof, or equivalent thereto.

Emitter-Reactor-Recorder-Agent based Dimensional Reduction

In some embodiments dimensional reduction is accomplished via the mating of a Reactor with a larger dimension of Emitter than of Recorder. An example system is illustrated in Figure 13 in which a mirrored Fibonacci Ram's Horn is used in the coupling of a single Recorder to multiple Emitters.

Within the context of the invention, the set of sets of Emitter and/or Recorder signals across multiple Agents from Agent x_I to Agent $x_{|X|}$ are labelled via $({}_x \underline{E}^{[i]}, {}_x \underline{R}^{[i+a]}) = ({}_x E_{I, \dots, |x|E}^{[i]}, {}_x R_{I, \dots, |x|R}^{[i+a]})$, the set of signals resultant from the application of a Transform to Emissions and the resultant Recording are labelled via $({}_x \underline{\mathcal{E}}^{[i]}, {}_x \underline{\mathcal{R}}^{[i+a]}) = ({}_x \mathcal{E}_{I, \dots, |x|\mathcal{E}}, {}_x \mathcal{R}_{I, \dots, |x|\mathcal{R}}^{[i+a]})$, and Generated signals $({}_x \widetilde{E}^{[i]}, {}_x \widetilde{R}^{[i+a]}) = ({}_x \widetilde{E}_{I, \dots, |x|E}^{[i]}, {}_x \widetilde{R}_{I, \dots, |x|R}^{[i+a]})$.

Dimension reduction is used in the creation of a lower-dimensional representation of a higher dimensional signal, used for any industry standard application including but not limited to training of Output models, including but not limited a Discriminator with respect to data input via

high-dimensional Emitters, ${}_z\mathbf{E}_D$ on their resultant Recordings, ${}_z\mathbf{R}_D$, with input data including but not limited to an input of a set of Loop elements, i.e.

$$\begin{aligned}
 & \uparrow {}_z\mathbf{D}_{R_D(t)}({}_z\mathbf{R}_D(t+T_{VII})) \\
 & {}^g{}_z\mathbf{E}_D(t) = \uparrow {}_z\mathbf{G}_D(\mathbf{P}(t+T_{VI})) \\
 & \dots\dots\dots \\
 & \mathbf{P}(t) = (\underline{{}_x\mathbf{E}}(t+T_{IV}), \underline{{}_x\mathbf{R}}(t+T_V)) \\
 & \dots\dots\dots \\
 & \text{and } \{(\underline{{}_x\mathbf{E}}(t+T_{II}), \underline{{}_x\mathbf{R}}(t+T_{III})) \mid t \in [t_1, t_2]\} : \\
 & \text{With } \{({}_z\mathbf{E}_D(t), {}_z\mathbf{R}_D(t+T_I)) \mid t \in [t_1, t_2]\}, \\
 & \dots\dots\dots \\
 & \text{e.g.} \\
 & a_{VII} = a_{VI} = a_V = a_{IV} = a_{III} = a_{II} = a_I = 0 \\
 & \uparrow {}_z\mathbf{D}_{R_D^{[i]}({}_z\mathbf{R}_D^{[i+a_{VII}]})} \\
 & {}^g{}_z\mathbf{E}_D^{[i]} = \uparrow {}_z\mathbf{G}_D(\mathbf{P}^{[i+a_{VI}]}) \\
 & \dots\dots\dots \\
 & \mathbf{P}^{[i]} = (\underline{{}_x\mathbf{E}}^{[i+a_{IV}]}, \underline{{}_x\mathbf{R}}^{[i+a_V]}) \\
 & \dots\dots\dots \\
 & \text{and } \{(\underline{{}_x\mathbf{E}}^{[i+a_{II}]}, \underline{{}_x\mathbf{R}}^{[i+a_{III}]}) \mid i \in \{1, \dots, n\}\} : \\
 & \text{With } \{({}_z\mathbf{E}_D^{[i]}, {}_z\mathbf{R}_D^{[i+a_{II}]}) \mid i \in \{1, \dots, n\}\},
 \end{aligned}$$

or any subset thereof, or equivalent thereto. In some embodiments, Discriminators are trained on Transformed or Generated Loop elements, i.e. ${}_z\mathbf{D}_{R_D}$, or ${}_z\mathbf{D}_{\tilde{R}_D}$.

In some embodiments, dimension expansion is accomplished with with the mating of a Reactor with a larger dimension of Recorders than Emitters. An example Loop is illustrated in Figure 14 where a single Emitter is coupled to multiple Recorders.

In some embodiments, dimensional reduction and expansion are paired e.g. in an Emitter-Reactor-Recorder Loop comprising Recorders and Emitters, with lower dimensional Emitters and Recorders labelled ${}_z\mathbf{E}_S$ and ${}_z\mathbf{R}_P$ respectively, with the signal for the former optionally being derived via the application of ${}_z\mathbf{G}_G$, or via a Visualisation Function to the latter, and higher dimensional Emitters labelled ${}_z\mathbf{E}_P$, and and Recorders labelled $\mathbf{R}_S$ respectively.

In some embodiments, dimensional reduction/expansion Loops are used in the training of encoder, ${}_z\mathbf{G}_P$, and decoder, ${}_z\mathbf{G}_S$, Generators as encoder/decoder Loops.

In some embodiments, Encoder/Decoder Loops are coupled with Discriminators, creating a Emitter-Reactor-Recorder based Generative Adversarial Network, e.g.

$$\begin{aligned}
& (\widetilde{\underline{x}}\underline{E}(t+T_{XXI}), \widetilde{\underline{x}}\underline{R}(t+T_{XXII})) = {}^g_z\mathbf{S}(t) \\
& \quad \downarrow {}_z\mathbf{D}_{\mathcal{R}_D(t)}({}_z\mathbf{R}_D(t+T_{XX})) \\
& \quad \downarrow {}_z\mathbf{D}_{\mathcal{R}_D(t)}({}_z\mathbf{R}_D(t+T_{XIX})) \\
& \quad \downarrow L({}_z\mathbf{S}(t+T_{XVIII})) \\
& \quad \downarrow C(\mathbf{P}(t+T_{XVII}), {}^g_z\mathbf{S}(t)) \\
& \quad {}^g_z\mathbf{E}_D(t) = {}_z\mathbf{G}_D({}_z\mathbf{S}(t+T_{XVI})) \\
& \quad {}^g_z\mathbf{S}(t) = \uparrow {}_z\mathbf{G}_S({}_z\mathbf{R}_S(t+T_{XV})) \\
& \quad {}^g_z\mathbf{E}_S(t) = \uparrow {}_z\mathbf{G}_G({}_z\mathbf{R}_P(t+T_{XIV})) \\
& \quad {}^g_z\mathbf{E}_P(t) = \uparrow {}_z\mathbf{G}_P(\mathbf{P}(t+T_{XIII})) \\
& \dots\dots\dots \\
& \quad \mathbf{O}(t) = \mathbf{P}(t+T_{XII}) \\
& \quad \mathbf{P}(t) = (\underline{x}\underline{E}(t+T_X), \underline{x}\underline{R}(t+T_{XI})) \\
& \dots\dots\dots \\
& \text{and } \{(\underline{x}\underline{\mathcal{I}}(t+T_{VIII}), \underline{x}\underline{\mathcal{J}}(t+T_{IX})) \mid t \in [t_1, t_2]\}: \\
& \text{and } \{(\underline{x}\underline{E}(t+T_{VI}), \underline{x}\underline{R}(t+T_{VII})) \mid t \in [t_1, t_2]\}, \\
& \text{With } \{({}_z\mathbf{E}_D(t+T_{IV}), {}_z\mathbf{R}_D(t+T_V)) \mid t \in [t_1, t_2]\}, \\
& \text{With } \{({}_z\mathbf{E}_P(t), {}_z\mathbf{R}_P(t+T_I), {}_z\mathbf{E}_S(t+T_{II}), {}_z\mathbf{R}_S(t+T_{III})) \mid t \in [t_1, t_2]\}, \\
& \dots\dots\dots
\end{aligned}$$

e.g.

$$\begin{aligned}
& a_{XXII} = a_{XXI} = a_{XX} = a_{XIX} = a_{XVIII} = a_{XVII} = a_{XVI} = a_{XV} = 0 \\
& a_{XIV} = a_{XIII} = a_{XII} = a_{XI} = a_X = a_{IX} = a_{VIII} = a_{VII} = 0 \\
& a_{VI} = a_V = a_{IV} = a_{III} = a_{II} = a_I = 0 \\
& (\widetilde{\underline{x}}\underline{E}^{[i+a_{XXI}]}, \widetilde{\underline{x}}\underline{R}^{[i+a_{XXII}]}) = {}^g_z\mathbf{S}^{[i]} \\
& \quad \downarrow {}_z\mathbf{D}_{\mathcal{R}_D^{[i]}}({}_z\mathbf{R}_D^{[i+a_{XX}]}) \\
& \quad \downarrow {}_z\mathbf{D}_{\mathcal{R}_D^{[i]}}({}_z\mathbf{R}_D^{[i+a_{XIX}]}) \\
& \quad \downarrow L({}_z\mathbf{S}^{[i+a_{XVIII}]}) \\
& \quad \downarrow C(\mathbf{O}^{[i+a_{XVII}]}, {}^g_z\mathbf{S}^{[i]}) \\
& \quad {}^g_z\mathbf{E}_D^{[i]} = {}_z\mathbf{G}_D({}_z\mathbf{S}^{[i+a_{XVI}]}) \\
& \quad {}^g_z\mathbf{S}^{[i]} = \uparrow {}_z\mathbf{G}_S({}_z\mathbf{R}_S^{[i+a_{XV}]}) \\
& \quad {}^g_z\mathbf{E}_S^{[i]} = \uparrow {}_z\mathbf{G}_G({}_z\mathbf{R}_P^{[i+a_{XIV}]}) \\
& \quad {}^g_z\mathbf{E}_P^{[i]} = \uparrow {}_z\mathbf{G}_P(\mathbf{P}^{[i+a_{XIII}]}) \\
& \dots\dots\dots
\end{aligned}$$

$$\begin{aligned}
& \mathbf{O}^{[i]} = \mathbf{P}^{[i+a_{XII}]} \\
& \mathbf{P}^{[i]} = (\underline{x}\underline{E}^{[i+a_X]}, \underline{x}\underline{R}^{[i+a_{XI}]}) \\
& \dots\dots\dots \\
& \text{and } \{(\underline{x}\underline{\mathcal{I}}^{[i+a_{VIII}]}, \underline{x}\underline{\mathcal{J}}^{[i+a_{IX}]}) \mid i \in \{1, \dots, n\}\}: \\
& \text{and } \{(\underline{x}\underline{E}^{[i+a_{VI}]}, \underline{x}\underline{R}^{[i+a_{VII}]}) \mid i \in \{1, \dots, n\}\}, \\
& \text{and } \{({}_z\mathbf{E}_D^{[i+a_{IV}]}, {}_z\mathbf{R}_D^{[i+a_V]}) \mid i \in \{1, \dots, n\}\}, \\
& \text{With } \{({}_z\mathbf{E}_P^{[i]}, {}_z\mathbf{R}_P^{[i+a_I]}, {}_z\mathbf{E}_S^{[i+a_{II}]}, {}_z\mathbf{R}_S^{[i+a_{III}]}) \mid i \in \{1, \dots, n\}\}, \\
& \dots\dots\dots
\end{aligned}$$

or any subset thereof, or equivalent thereto.

In some embodiments, models are trained via multiplexing techniques. Figure 15 illustrates an array of Emitter-Scene-Recorder Loops being addressed via multiplexing e.g. a sliding window. In some embodiments, a Comparative Discriminator comparing a subset of the outputs of an Emitter-Reactor-Recorder-Agent network at t with earlier outputs is used in training.

Emitter-Reactor-Recorder-Agent based One-Way Functions/Discriminator Training

In some embodiments, Emitter-Reactor-Recorder-Agent Loops are trained to act as One-Way Functions or for the training of Discriminators, i.e.

$$\begin{aligned}
 & \uparrow_z D_{\tilde{R}_D(t)}(z_A \bar{R}_D(t+T_{XII})) \\
 & {}^g_z \bar{E}_D(t) = \uparrow_z G_D(P(t+T_{XI})) \\
 & {}^g_x \bar{E}(t) = \uparrow_z G_M(P(t+T_X)) \\
 & \dots\dots\dots \\
 & P(t) = ({}_x \bar{E}(t+T_{VIII}), {}_x \bar{R}(t+T_{IX})) \\
 & \dots\dots\dots \\
 & \text{and } \{({}_x \bar{E}(t+T_{VI}), {}_x \bar{R}(t+T_{VII})) \mid t \in [t_1, t_2]\} : \\
 & \text{and } \{({}_z \bar{E}_D(t+T_{IV}), {}_z \bar{R}_D(t+T_V)) \mid t \in [t_1, t_2]\}, \\
 & \text{With } \{({}_z \bar{E}_P(t), {}_z \bar{R}_P(t+T_I), {}_z \bar{E}_S(t+T_{II}), {}_z \bar{R}_S(t+T_{III})) \mid t \in [t_1, t_2]\}, \\
 & \dots\dots\dots \\
 & \text{e.g.} \\
 & a_{XII} = a_{XI} = a_{IX} = a_{VIII} = a_{VII} = a_{VI} = 0 \\
 & a_V = a_{IV} = a_{III} = a_{II} = a_I = 0 \\
 & a_X = -1 \\
 & \uparrow_z D_{\tilde{R}_D^{[i]}}(z_A \bar{R}_D^{[i+a_{XII}]}) \\
 & {}^g_z \bar{E}_D^{[i]} = \uparrow_z G_D(P^{[i+a_{XI}]}) \\
 & {}^g_x \bar{E}^{[i]} = \uparrow_z G_M(P^{[i+a_X]}) \\
 & \dots\dots\dots \\
 & P^{[i]} = ({}_x \bar{E}^{[i+a_{VIII}]}, {}_x \bar{R}^{[i+a_{IX}]}) \\
 & \dots\dots\dots \\
 & \text{and } \{({}_x \bar{E}^{[i+a_{VI}]}, {}_x \bar{R}^{[i+a_{VII}]}) \mid i \in \{1, \dots, n\}\} : \\
 & \text{and } \{({}_z \bar{E}_D^{[i+a_{IV}]}, {}_z \bar{R}_D^{[i+a_V]}) \mid i \in \{1, \dots, n\}\}, \\
 & \text{With } \{({}_z \bar{E}_P^{[i]}, {}_z \bar{R}_P^{[i+a_I]}, {}_z \bar{E}_S^{[i+a_{II}]}, {}_z \bar{R}_S^{[i+a_{III}]}) \mid i \in \{1, \dots, n\}\}, \\
 & \dots\dots\dots
 \end{aligned}$$

or any subset thereof, or equivalent thereto.

In some embodiments, Discriminators are trained on Transformed or Generated elements i.e. $\downarrow_z D_{\mathcal{R}_D}$, or $\downarrow_z D_{\tilde{R}_D}$ respectively.

Emitter-Reactor-Record-Agent Based Modification According to an Objective Function

In some embodiments, Emitter-Reactor-Recorder-Agent systems are trained to modify Loop elements e.g. according to trained Discriminators, i.e.

$$\begin{aligned}
& \downarrow_z L({}_x R(t+T_{XVIII})) \\
& \downarrow_z D_{\tilde{R}_D(t)}({}_z R_D(t+T_{XVII})) \\
& \downarrow_z D_{\mathcal{R}_D(t)}({}_z R_D(t+T_{XVI})) \\
& {}^g_z E_D(t) = {}_z G_D(P(t+T_{XV})) \\
& {}^g_x E(t) = \uparrow_z G_M(P(t+T_{XIV})) \\
& \dots\dots\dots \\
& P(t) = ({}_x E(t+T_{XII}), {}_x R(t+T_{XIII})) \\
& \dots\dots\dots \\
& \text{and } \{({}_x \tilde{E}(t+T_X), {}_x \tilde{R}(t+T_{XI})) \mid t \in [t_1, t_2]\}: \\
& \text{and } \{({}_x \mathcal{I}(t+T_{VIII}), {}_x \mathcal{J}(t+T_{IX})) \mid t \in [t_1, t_2]\}, \\
& \text{and } \{({}_x E(t+T_{VI}), {}_x R(t+T_{VII})) \mid t \in [t_1, t_2]\}, \\
& \text{and } \{({}_z E_D(t+T_{IV}), {}_z R_D(t+T_V)) \mid t \in [t_1, t_2]\}, \\
& \text{With } \{({}_z E_P(t), {}_z R_P(t+T_I), {}_z E_S(t+T_{II}), {}_z R_S(t+T_{III})) \mid t \in [t_1, t_2]\}, \\
& \dots\dots\dots \\
& \text{e.g.} \\
& a_{XVIII} = a_{XVII} = a_{XVI} = a_{XV} = a_{XIV} = 0 \\
& a_{XI} = a_X = a_{IX} = a_{XV} = a_{VIII} = a_{VII} = a_{VI} = 0 \\
& a_V = a_{IV} = a_{III} = a_{II} = a_I = 0 \\
& a_{XIV} = -1 \\
& \downarrow_z L({}_x R^{[i+a_{XVIII}]}) \\
& \downarrow_z D_{\tilde{R}_D^{[i]}}({}_z R_D^{[i+a_{XVII}]}) \\
& \downarrow_z D_{\mathcal{R}_D^{[i]}}({}_z R_D^{[i+a_{XVI}]}) \\
& {}^g_z E_D^{[i]} = {}_z G_D(P^{[i+a_{XV}]}) \\
& {}^g_x E^{[i]} = \uparrow_z G_M(P^{[i+a_{XIV}]}) \\
& \dots\dots\dots \\
& P^{[i]} = ({}_x E^{[i+a_{XII}]}, {}_x R^{[i+a_{XIII}]}) \\
& \dots\dots\dots \\
& \text{and } \{({}_x \tilde{E}^{[i+a_X]}, {}_x \tilde{R}^{[i+a_{XI}]} \mid i \in \{1, \dots, n\}\}: \\
& \text{and } \{({}_x \mathcal{I}^{[i+a_{VIII}]}, {}_x \mathcal{J}^{[i+a_{IX}]} \mid i \in \{1, \dots, n\}\}, \\
& \text{and } \{({}_x E^{[i+a_{VI}]}, {}_x R^{[i+a_{VII}]} \mid i \in \{1, \dots, n\}\}, \\
& \text{and } \{({}_z E_D^{[i+a_{IV}]}, {}_z R_D^{[i+a_V]} \mid i \in \{1, \dots, n\}\}, \\
& \text{With } \{({}_z E_P^{[i]}, {}_z R_P^{[i+a_I]}, {}_z E_S^{[i+a_{II}]}, {}_z R_S^{[i+a_{III}]} \mid i \in \{1, \dots, n\}\}, \\
& \dots\dots\dots
\end{aligned}$$

or any subset thereof, or equivalent thereto.

External Elements

In some embodiments, Agents are trained with Objective Functions applied to external systems, including but not limited external Emitters and/or Recorders, i.e. $\downarrow L({}_y E(t+T))$, or

$\downarrow L({}_y R(t+T))$, e.g. $\downarrow L({}_y E^{[i+a]})$, or $\downarrow L({}_y R^{[i+a]})$.

In some embodiments, any permutation of Emitter-Reactor-Recorder Systems are provided payloads, private keys, public keys, ciphertext, stegotext, or any other cryptographic input and any differentiable cryptographic scheme implemented and trained on Emitter-Reactor-Recorder Loops.

An example symmetric cipher comprises the provision of plaintext U , private key K to encoder-decoder network, Alice - z_A , the acquisition of an output from Alice of ciphertext, $\bar{S}$, and the provision of this cipher text to encoder-decoder-discriminator network, Bob - z_B , alongside the key. Encoder-decoder-discriminator network, Eve - z_E , is provided the ciphertext with no key.

Eve trains to reconstruct Alice's plaintext against Comparative and Domain discriminators comparing her output with the plaintext content or Domain.

Bob trains to reconstruct Alice's plaintext, against Comparative and Domain discriminators comparing his output with the plaintext content or Domain, and trains in opposition to Alice.

Alice trains in Bob's favour and in Eve's disfavour i.e.

$$\begin{aligned}
 & \downarrow_{z_{A,B}} \bar{D}_{R_D(t)}(z_E \bar{R}_D(t+T_{XXXIV})) \\
 & \downarrow_{z_E} D_{R_D(t)}(z_E \bar{R}_D(t+T_{XXXIII})) \\
 & \downarrow_{z_{A,B}} D_{R_D(t)}(z_B \bar{R}_D(t+T_{XXXII})) \\
 & \downarrow_{z_{A,B}} \in(O(t+T_{XXXI}), {}^g\bar{S}^{[i]}) \\
 & \downarrow_{z_E} C(O(t+T_{XXX}), {}^g\bar{S}(t)) \\
 & \downarrow_{z_{A,B}} C(O(t+T_{XXIX}), {}^g\bar{S}(t)) \\
 & {}^g\bar{E}_D(t) = \uparrow_{z_E} G_D({}^g\bar{S}(t+T_{XXVIII})) \\
 & {}^g\bar{S}(t) = \uparrow_{z_E} G_M({}^g\bar{P}(t+T_{XXVII})) \\
 & {}^g\bar{E}_D(t) = \uparrow_{z_B} G_D({}^g\bar{S}(t+T_{XXVI})) \\
 & {}^g\bar{S}(t) = \uparrow_{z_B} G_M({}^gP(t+T_{XXV})) \\
 & {}^g\bar{S}(t) = \uparrow_{z_A} G_M({}^gP(t+T_{XXIV})) \\
 & \dots \\
 & O(t) = U(t+T_{XXXIII}) \\
 & {}_{z_E}P(t) = ({}^g\bar{S}(t+T_{XXXII})) \\
 & {}_{z_B}P(t) = ({}^g\bar{S}(t+T_{XX}), K(t+T_{XXI})) \\
 & {}_{z_A}P(t) = (U(t+T_{XXVIII}), K(t+T_{XXIX})) \\
 & \dots \\
 & \text{and } \{(U(t+T_{XXVI}), K(t+T_{XXVII})) | t \in [t_1, t_2]\}, \\
 & \text{and } \{({}_{z_E}E_D(t+T_{XXIV}), {}_{z_E}R_D(t+T_{XXV})) | t \in [t_1, t_2]\}, \\
 & \text{and } \{({}_{z_E}E_P(t+T_X), {}_{z_E}R_P(t+T_{XI}), {}_{z_E}E_S(t+T_{XII}), {}_{z_E}R_S(t+T_{XIII})) | t \in [t_1, t_2]\}, \\
 & \text{and } \{({}_{z_B}E_D(t+T_{VIII}), {}_{z_B}R_D(t+T_{IX})) | t \in [t_1, t_2]\}, \\
 & \text{and } \{({}_{z_B}E_P(t+T_{IV}), {}_{z_B}R_P(t+T_V), {}_{z_B}E_S(t+T_{VI}), {}_{z_B}R_S(t+T_{VII})) | t \in [t_1, t_2]\}, \\
 & \text{With } \{({}_{z_A}E_P(t), {}_{z_A}R_P(t+T_I), {}_{z_A}E_S(t+T_{II}), {}_{z_A}R_S(t+T_{III})) | t \in [t_1, t_2]\}, \\
 & \dots
 \end{aligned}$$

e.g.

$$\begin{aligned}
a_{XXXIV} &= a_{XXXIII} = a_{XXXII} = a_{XXXI} = a_{XXX} = a_{XXIX} = 0 \\
a_{XXVIII} &= a_{XXVII} = a_{XXVI} = 0 \\
a_{XXV} &= a_{XXIV} = a_{XXIII} = a_{XXII} = a_{XXI} = a_{XX} = a_{XIX} = a_{XVIII} = 0 \\
a_{XVII} &= a_{XVI} = a_{XV} = a_{XIV} = a_{XIII} = a_{XII} = 0 \\
a_{XI} &= a_X = a_{IX} = a_{VIII} = a_{VII} = a_{VI} = a_V = a_{IV} = a_{III} = a_{II} = a_I = 0 \\
&\downarrow_{z_{A,B}} \mathbf{D}_{R_D^{[i]}} \left({}_{z_E} \bar{\mathbf{R}}_D^{[i+a_{XXXIV}]} \right) \\
&\downarrow_{z_E} \mathbf{D}_{R_D^{[i]}} \left({}_{z_E} \bar{\mathbf{R}}_D^{[i+a_{XXXIII}]} \right) \\
&\downarrow_{z_{A,B}} \mathbf{D}_{R_D^{[i]}} \left({}_{z_B} \bar{\mathbf{R}}_D^{[i+a_{XXXII}]} \right) \\
&\downarrow_{z_{A,B}} \mathbf{C} \left({}_{z_E} \mathbf{O}^{[i+a_{XXXI}]} , {}_{z_E} \bar{\mathbf{S}}^{[i]} \right) \\
&\downarrow_{z_E} \mathbf{C} \left({}_{z_E} \mathbf{O}^{[i+a_{XXX}]} , {}_{z_E} \bar{\mathbf{S}}^{[i]} \right) \\
&\downarrow_{z_{A,B}} \mathbf{C} \left({}_{z_E} \mathbf{O}^{[i+a_{XXX}]} , {}_{z_B} \bar{\mathbf{S}}^{[i]} \right) \\
&{}_{z_E} \bar{\mathbf{E}}_D^{[i]} = \uparrow_{z_E} \mathbf{G}_D \left({}_{z_E} \bar{\mathbf{S}}^{[i+a_{XXVIII}]} \right) \\
&{}_{z_E} \bar{\mathbf{S}}^{[i]} = \uparrow_{z_E} \mathbf{G}_M \left({}_{z_A} \bar{\mathbf{P}}^{[i+a_{XXVII}]} \right) \\
&{}_{z_B} \bar{\mathbf{E}}_D^{[i]} = \uparrow_{z_B} \mathbf{G}_D \left({}_{z_B} \bar{\mathbf{S}}^{[i+a_{XXVI}]} \right) \\
&{}_{z_B} \bar{\mathbf{S}}^{[i]} = \uparrow_{z_B} \mathbf{G}_M \left({}_{z_B} \mathbf{P}^{[i+a_{XXV}]} \right) \\
&{}_{z_A} \bar{\mathbf{S}}^{[i]} = \uparrow_{z_A} \mathbf{G}_M \left({}_{z_A} \mathbf{P}^{[i+a_{XXIV}]} \right)
\end{aligned}$$

$$\begin{aligned}
&\dots\dots\dots \\
&{}_{z_E} \mathbf{O}^{[i]} = \mathbf{U}^{[i+a_{XXXI}]} \\
&{}_{z_E} \mathbf{P}^{[i]} = ({}_{z_A} \bar{\mathbf{S}}^{[i+a_{XXXI}]}) \\
&{}_{z_B} \mathbf{P}^{[i]} = ({}_{z_A} \bar{\mathbf{S}}^{[i+a_{XX}]} , \mathbf{K}^{[i+a_{XXI}]}) \\
&{}_{z_A} \mathbf{P}^{[i]} = (\mathbf{U}^{[i+a_{XVIII}]} , \mathbf{K}^{[i+a_{XIX}]}) \\
&\dots\dots\dots
\end{aligned}$$

$$\begin{aligned}
&\text{and } \{ (\mathbf{U}^{[i+a_{XVI}]} , \mathbf{K}^{[i+a_{XVII}]}) \mid i \in \{ 1, \dots, n \} \} : \\
&\text{and } \{ ({}_{z_E} \mathbf{E}_D^{[i+a_{XIV}]} , {}_{z_E} \mathbf{R}_D^{[i+a_{XV}]}) \mid i \in \{ 1, \dots, n \} \} , \\
&\text{and } \{ ({}_{z_E} \mathbf{E}_P^{[i+a_X]} , {}_{z_E} \mathbf{R}_P^{[i+a_{XI}]} , {}_{z_E} \mathbf{E}_S^{[i+a_{XII}]} , {}_{z_E} \mathbf{R}_S^{[i+a_{XIII}]}) \mid i \in \{ 1, \dots, n \} \} , \\
&\text{and } \{ ({}_{z_B} \mathbf{E}_D^{[i+a_{VIII}]} , {}_{z_B} \mathbf{R}_D^{[i+a_{IX}]}) \mid i \in \{ 1, \dots, n \} \} , \\
&\text{and } \{ ({}_{z_B} \mathbf{E}_P^{[i+a_{IV}]} , {}_{z_B} \mathbf{R}_P^{[i+a_V]} , {}_{z_B} \mathbf{E}_S^{[i+a_{VI}]} , {}_{z_B} \mathbf{R}_S^{[i+a_{VII}]}) \mid i \in \{ 1, \dots, n \} \} , \\
&\text{With } \{ ({}_{z_A} \mathbf{E}_P^{[i]} , {}_{z_A} \mathbf{R}_P^{[i+a_I]} , {}_{z_A} \mathbf{E}_S^{[i+a_{II}]} , {}_{z_A} \mathbf{R}_S^{[i+a_{III}]}) \mid i \in \{ 1, \dots, n \} \} , \\
&\dots\dots\dots
\end{aligned}$$

or any subset thereof, or equivalent thereto.

A steganographic scheme is implemented by providing Alice with a Payload text, stegotext, $\mathbf{W}$, and key $\mathbf{K}$, the training of Alice against the similarity of the ciphertext to the stegotext

$$\downarrow_{z_A} \mathbf{C}(\mathbf{W}(t+T), {}_{z_A} \bar{\mathbf{S}}(t)) \text{ e.g. } \downarrow_{z_A} \mathbf{C}(\mathbf{W}^{[i+a_i]}, {}_{z_A} \bar{\mathbf{S}}^{[i]}).$$

In some embodiments, Emitter-Reactor-Recorder systems are used in key generation, e.g.

$${}_{z_A} \mathbf{K}_{PUB}(t) = {}_{z_A} \mathbf{G}_K({}_{z_A} \mathbf{R}(t+T)) \quad {}_{z_A} \mathbf{K}_{PUB}^{[i]} = {}_{z_A} \mathbf{G}_K({}_{z_A} \mathbf{R}^{[i+a]}), \text{ e.g. } a = -1.$$

In some embodiments, any arrangement of similar dimensions of Emitters and Recorders coupled with a Reactor are treated as an untrainable fully connected layer of a neural network or other machine learning network. In some embodiments, any machine controllable deformation or alteration to the Reactor is treated as a trainable function parameter.

REFERENCES - the teaching of the references provided below is incorporated in this disclosure to the extent permitted by applicable law.

- [1] Mallya, A., Wang, T.-C., Sapra, K. and Liu, M.-Y. (2020). World-Consistent Video-to-Video Synthesis. [online] arXiv.org. Available at: <https://arxiv.org/abs/2007.08509>.
- [2] Ma, D. (Creator), Katsenou, A. (Creator), Bull, D. (Creator) (26 Oct 2018). VIL: Synthetic Video Texture Dataset - SynTex. University of Bristol. 10.5523/bris.24imj6d9s27me2n211cf6jxkio
- [3] Wyse, L. and Ravikumar, P.T. (2022). Syntex: parametric audio texture datasets for conditional training of instrumental interfaces. International Conference on New Interfaces for Musical Expression. [online] doi:<https://doi.org/10.21428/92fbef44.0fe70450>.
- [4] Schönberger, J., & Frahm, J.M. (2016). Structure-from-Motion Revisited. In Conference on Computer Vision and Pattern Recognition (CVPR)
- [5] Schönberger, J., Zheng, E., Pollefeys, M., & Frahm, J.M. (2016). Pixelwise View Selection for Unstructured Multi-View Stereo. In European Conference on Computer Vision (ECCV)
- [6] Schönberger, J., Price, T., Sattler, T., Frahm, J.M., & Pollefeys, M. (2016). A Vote-and-Verify Strategy for Fast Spatial Verification in Image Retrieval. In Asian Conference on Computer Vision (ACCV)
- [8] Wei, X., Zhang, Y., Li, Z., Fu, Y. and Xue, X. (2020). DeepSFM: Structure From Motion Via Deep Bundle Adjustment. [online] arXiv.org. doi:<https://doi.org/10.48550/arXiv.1912.09697>.
- [9] Pribanic, T. (2010). A state of the art in structured light patterns for surface profilometry. Pattern Recognition.
- [10] Araslanov, N., Simone Schaub-Mayer, and Roth, S. 2021. Dense Unsupervised Learning for Video Segmentation. In Advances in Neural Information Processing Systems (NeurIPS).
- [11] Lu, X., Wang, W., Ma, C., Shen, J., Shao, L., and Porikli, F. 2019. See More, Know More: Unsupervised Video Object Segmentation With Co-Attention Siamese Networks. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
- [12] Wang, T., Zhu, J.-Y., Tao, A., Kautz, J. and Catanzaro, B. (2017). High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. doi:<https://doi.org/10.48550/arxiv.1711.11585>.
- [13] Wang, T.-C., Liu, M.-Y., Zhu, J.-Y., Liu, G., Tao, A., Kautz, J. and Catanzaro, B. (2018). Video-to-Video Synthesis. [online] Neural Information Processing Systems. Available at: https://papers.nips.cc/paper_files/paper/2018/hash/d86ea612dec96096c5e0fcc8dd42ab6d-Abstract.html [Accessed 16 Aug. 2023].
- [14] Gatys, L.A., Ecker, A.S. and Bethge, M. (2015). A Neural Algorithm of Artistic Style. [online] arXiv.org. Available at: <https://arxiv.org/abs/1508.06576>.
- [15] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W. (2016). OpenAI Gym. [online] arXiv.org. Available at: <https://arxiv.org/abs/1606.01540>.

- [16] Moerland, T.M., Broekens, J., Plaat, A. and Jonker, C.M. (2022). A Unifying Framework for Reinforcement Learning and Planning. *Frontiers in Artificial Intelligence*, 5. doi:<https://doi.org/10.3389/frai.2022.908353>.
- [17] Pretorius, A., Tessera, K., Smit, A.P., Formanek, C., Grimbly, S.J., Eloff, K., Danisa, S., Francis, L., Shock, J., Kamper, H., Brink, W., Engelbrecht, H., Laterre, A. and Beguir, K. (2021). Mava: a research framework for distributed multi-agent reinforcement learning. [online] arXiv.org. doi:<https://doi.org/10.48550/arXiv.2107.01460>.
- [18] Wright, L.G., Onodera, T., Stein, M.M., Wang, T., Schachter, D.T., Hu, Z. and McMahon, P.L. (2022). Deep physical neural networks trained with backpropagation. *Nature*, [online] 601(7894), pp.549–555. doi:<https://doi.org/10.1038/s41586-021-04223-6>.
- [19] Thuerey, N., Holl, P., Mueller, M., Schnell, P., Trost, F. and Um, K. (2022). Physics-based Deep Learning. [online] arXiv.org. doi:<https://doi.org/10.48550/arXiv.2109.05237>.
- [20] AbinavRavi (2019). Physics-Aware-GAN/Physics Aware GAN.ipynb at master · AbinavRavi/Physics-Aware-GAN. [online] GitHub. Available at: <https://github.com/AbinavRavi/Physics-Aware-GAN/blob/master/Physics%20Aware%20GAN.ipynb> [Accessed 16 Aug. 2023].
- [21] Skorokhodov, I., Tulyakov, S. and Elhoseiny, M. (2022). StyleGAN-V: A Continuous Video Generator with the Price, Image Quality and Perks of StyleGAN2. [online] arXiv.org. doi:<https://doi.org/10.48550/arXiv.2112.14683>.
- [22] Yu, S., Tack, J., Mo, S., Kim, H., Kim, J., Ha, J.-W. and Shin, J. (2022). Generating Videos with Dynamics-aware Implicit Generative Adversarial Networks. [online] arXiv.org. doi:<https://doi.org/10.48550/arXiv.2202.10571>.
- [23] Abadi, M. and Andersen, D.G. (2016). Learning to Protect Communications with Adversarial Neural Cryptography. arXiv:1610.06918 [cs]. [online] Available at: <https://arxiv.org/abs/1610.06918>.
- [24] Csaba, G., Ju, X., Chen, Q., Porod, W., Schmidhuber, J., Schlichtmann, U., Lugli, P. and Rührmair, U. (2009). On-Chip Electric Waves: An Analog Circuit Approach to Physical Unccloneable Functions. [online] ePrint IACR. Available at: <https://eprint.iacr.org/2009/246> [Accessed 16 Aug. 2023].
- [25] Huang, H., Wang, H., Luo, W., Ma, L., Jiang, W., Zhu, X., Li, Z. and Liu, W. (2017). Real-Time Neural Style Transfer for Videos. [online] openaccess.thecvf.com. Available at: https://openaccess.thecvf.com/content_cvpr_2017/html/Huang_Real-Time_Neural_Style_CVPR_2017_paper.html [Accessed 16 Aug. 2023]
- [26] Bansal, A., Ma, S., Ramanan, D. and Sheikh, Y. (n.d.). Recycle-GAN: Unsupervised Video Retargeting. [online] Available at: https://openaccess.thecvf.com/content_ECCV_2018/papers/Aayush_Bansal_Recycle-GAN_Unsupervised_Video_ECCV_2018_paper.pdf [Accessed 16 Aug. 2023].
- [27] Liu, K., Gu, S., Romero, A. and Timofte, R. (2020). Unsupervised Multimodal Video-to-Video Translation via Self-Supervised Learning. [online] arXiv.org. doi:<https://doi.org/10.48550/arXiv.2004.06502>.

Claims

1. A method of transforming data transmitted through a physical environment, the method comprising:
 - emitting an emitted signal into the physical environment;
 - detecting a detected signal from detection of the emitted signal after interaction with the physical environment;
 - transforming the detected signal to modify a further emitted signal; and
 - repeating the emitting, detecting, and transforming steps until a predetermined objective is met.
2. The method of claim 1, wherein the emitted signal is an image emitted by a projector and the detected signal is an image captured by a camera.
3. The method of claim 2, wherein the projector is a video projector.
4. The method of claim 3, wherein image capture by the camera is matched to a refresh rate of the video projector.
5. The method of any preceding claim, wherein the transforming step is performed using a generator model adapted to predict an emitted signal from a detected signal.
6. The method of claim 5, wherein the generator model is a trained machine learning model.
7. The method of claim 6, wherein the machine learning model is an artificial neural network.
8. The method of claim 6 or claim 7, wherein the machine learning model is trained to blend a predetermined image into the further emitted image.
9. The method of claim 8, wherein the predetermined image is predetermined text or other loss function.

10. The method of any of claims 6 to 9, further comprising modifying the generator model in real time.
11. The method of any preceding claim, wherein the transforming step comprises applying a predetermined style to the further emitted signal.
12. The method of any preceding claim, wherein the transforming step comprises operating on the detected signal using a one-way function.
13. The method of claim 12, wherein the one-way function comprises a structured noise generator function.
14. The method of claim 12 or claim 13, wherein the emitting, detecting and transforming steps are repeated a predetermined number of times to achieve the predetermined objective, the predetermined objective comprising a one-way function.
15. The method of any of claims 12 to 14, wherein an output of the method is a secure data recording.
16. The method of any preceding claim, wherein the method is performed in parallel by a plurality of emitter-detector pairs.
17. The method of claim 16, wherein the plurality of emitter-detector pairs is configured to act oppositionally, with at least one of the plurality of emitter-detector pairs adversarial to the other emitter-detector pairs.
18. The method of claim 17 where dependent on claim 14, wherein an adversarial autoencoder or any anomaly detector is adapted to verify integrity of the emission-recording pairs.
19. The method of claim 16 where dependent on claim 6 or claim 7, wherein an output of the method is production of a 3D image or other analysis.

20. The method of any of claims 1 to 7, wherein the predetermined objective is completion of a computation process.
21. A system for transformation of data transmitted through a physical environment, the system comprising:
- at least one emitter for emitting an emitted signal into the environment;
 - at least one detector for detecting a detected signal from detection of the emitted signal after interaction with the environment; and
 - computing means, wherein the computing means is adapted to transform the detected signal to modify the emitted signal.
22. The system of claim 22, wherein the emitter is a projector and the detector is a camera, and wherein the computing means is a suitably programmed processor.
23. The system of claim 22, wherein the projector is a video projector.
24. The system of claim 23, wherein image capture by the camera is matched to a refresh rate of the video projector.
25. The system of any of claims 21 to 25, wherein the system is adapted to perform the method of one or more of claims 5 to 20.

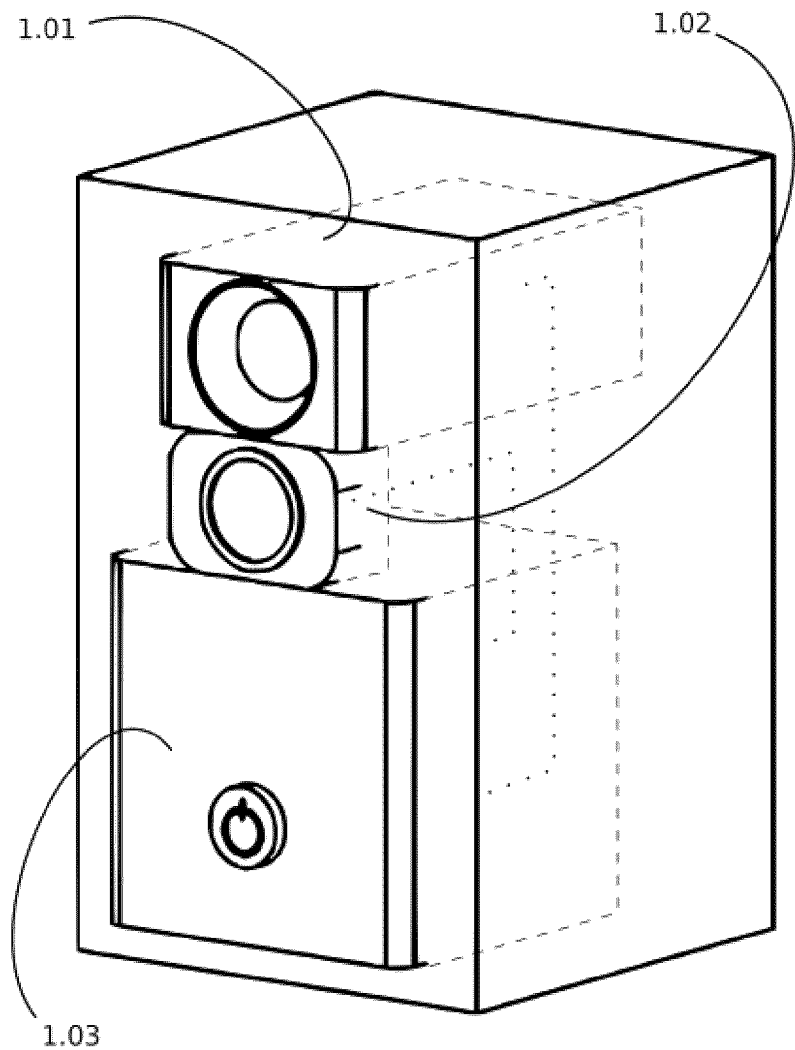


Figure 1

2/15

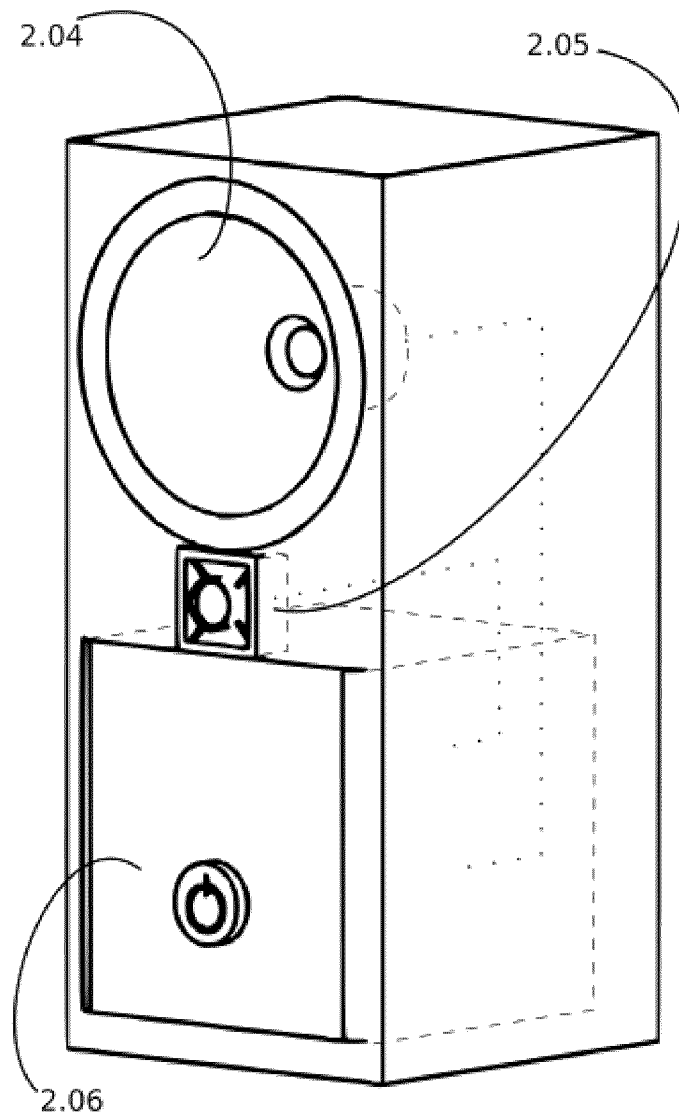


Figure 2

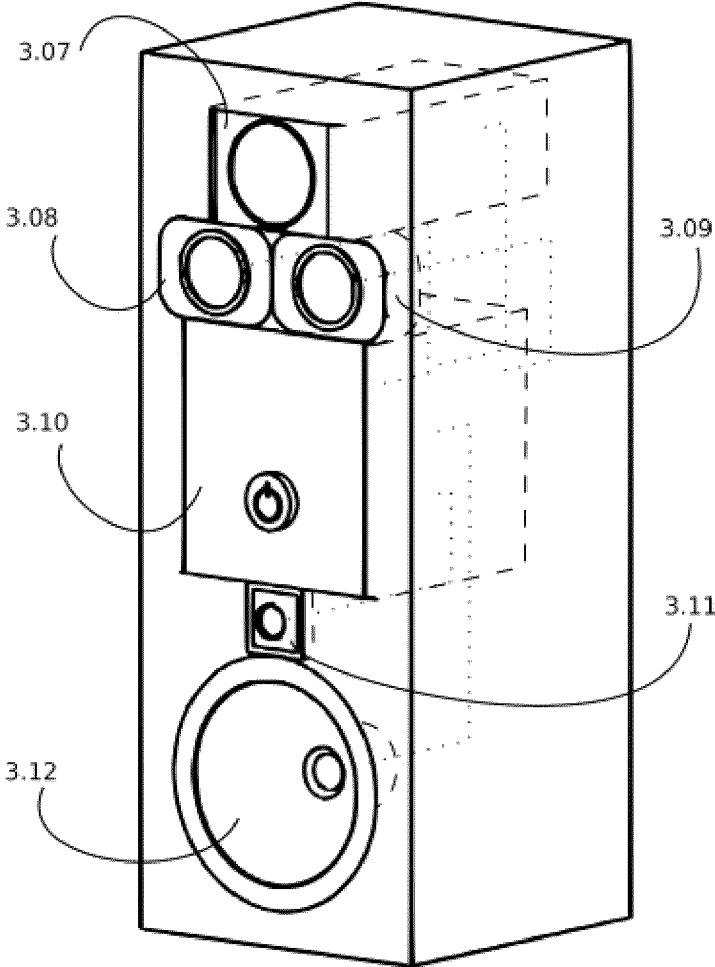


Figure 3

4/15

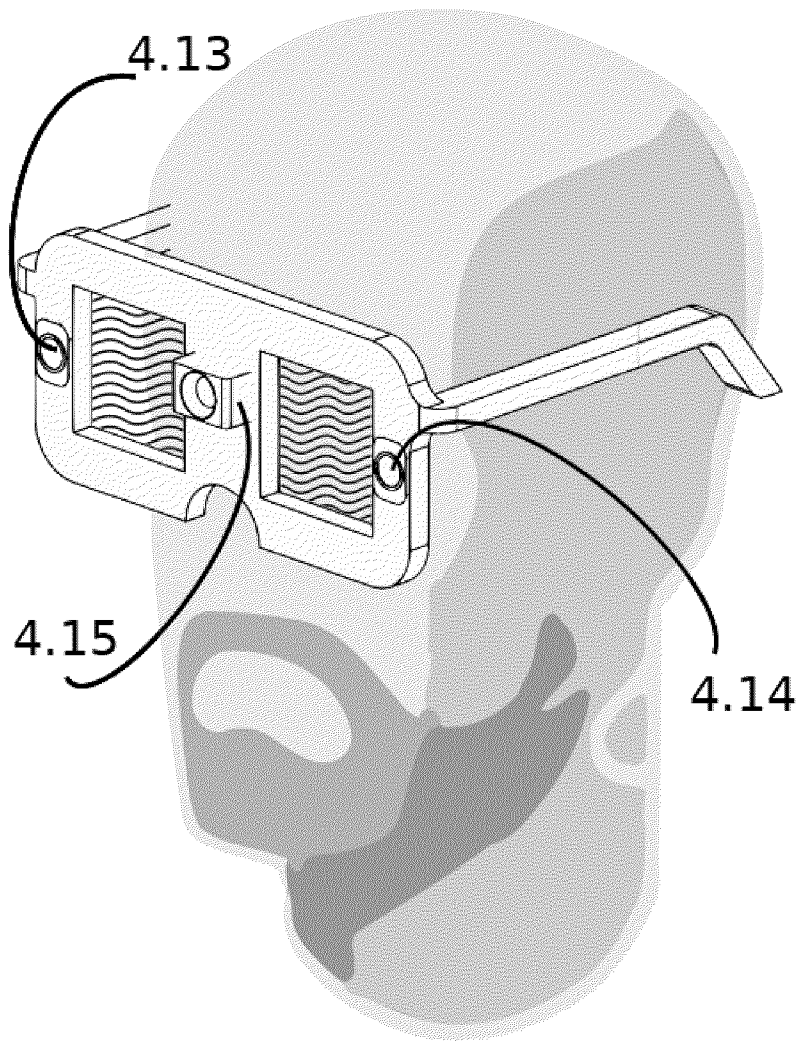


Figure 4

5/15

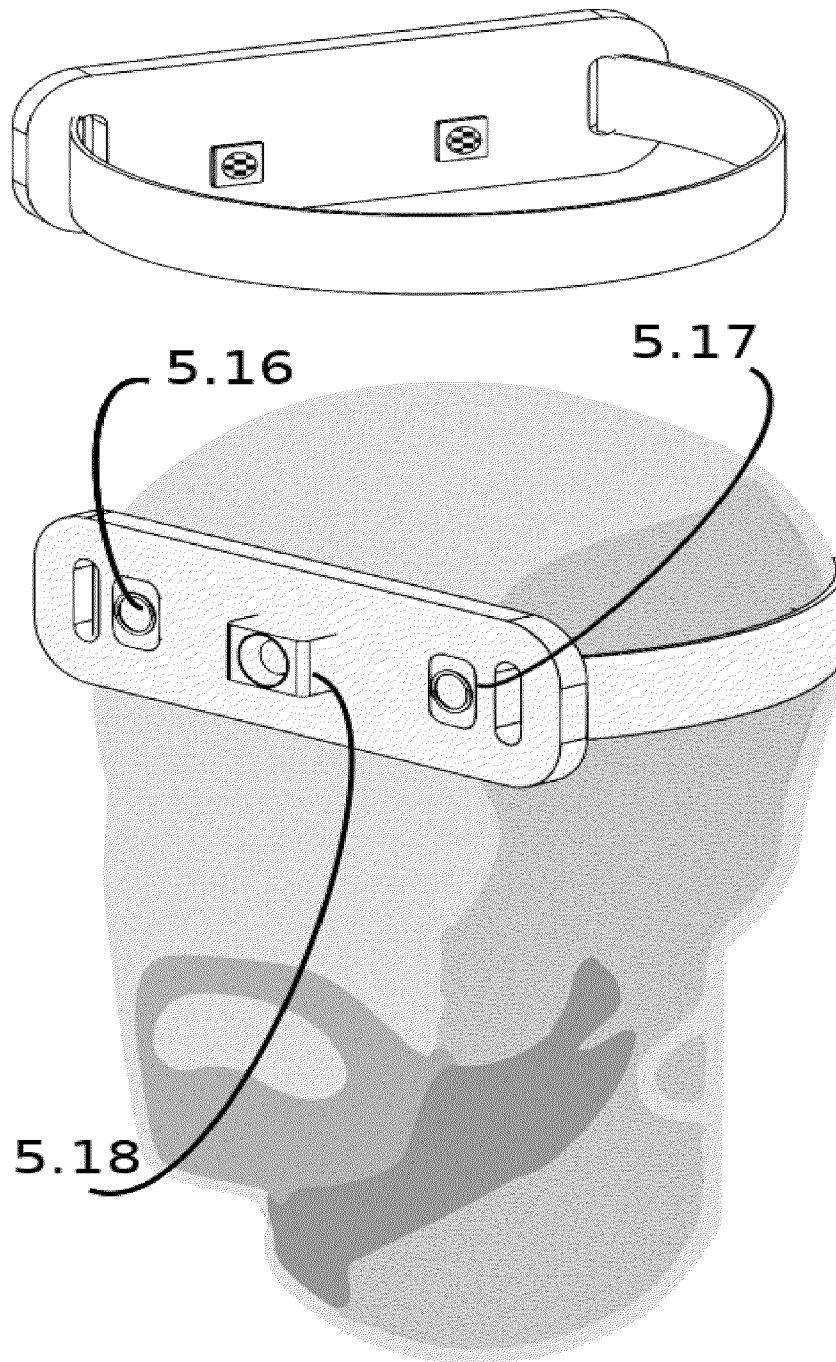


Figure 5

6/15

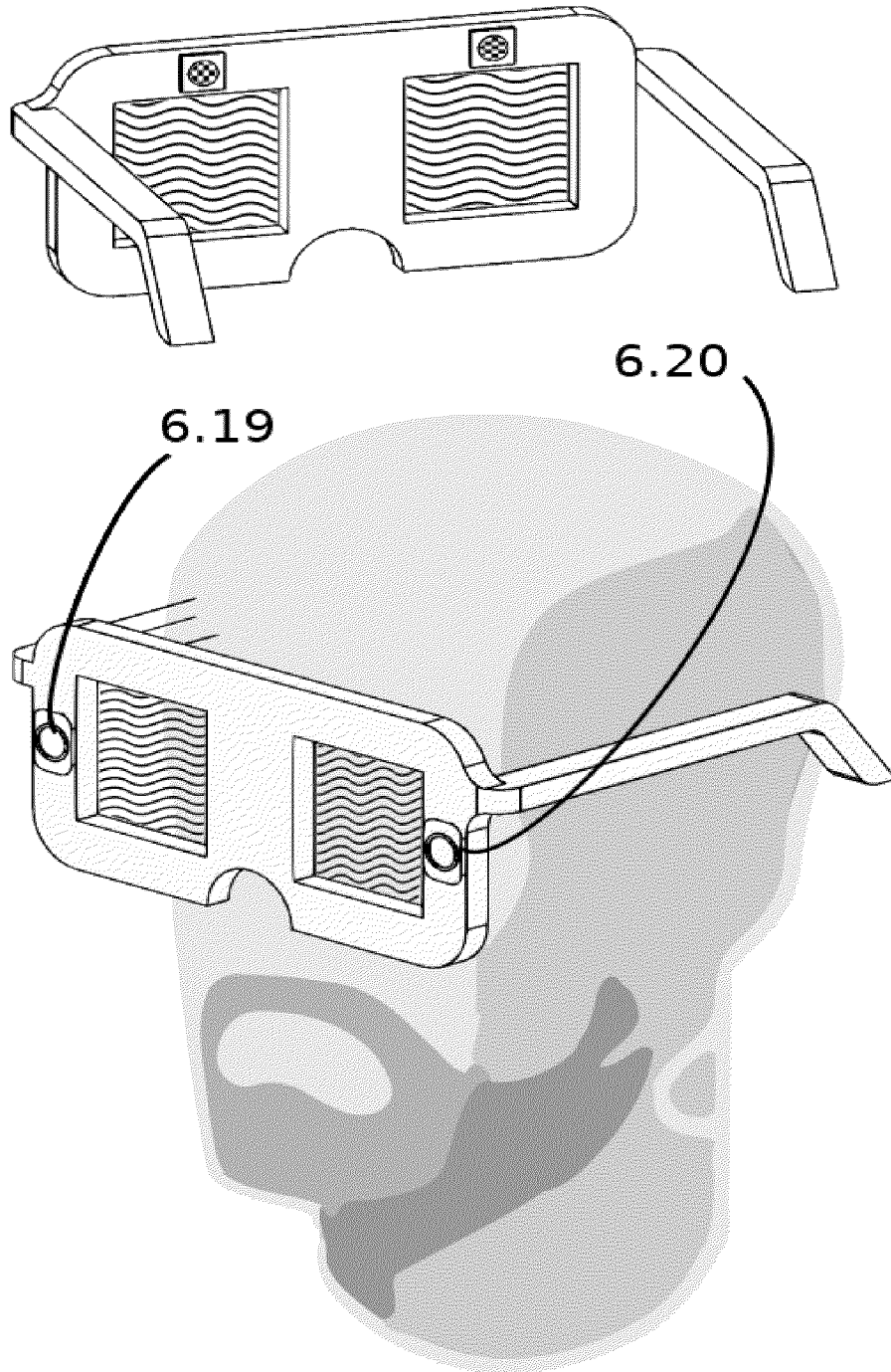


Figure 6

7/15

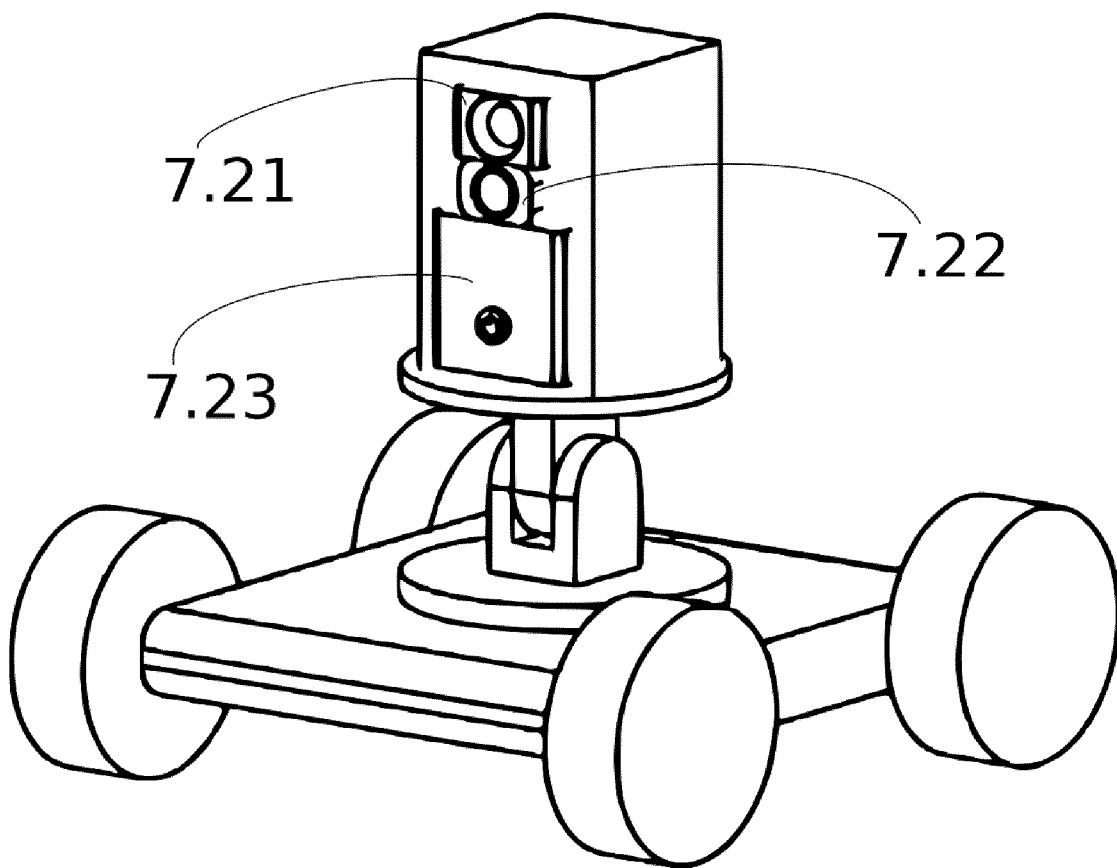


Figure 7

8/15

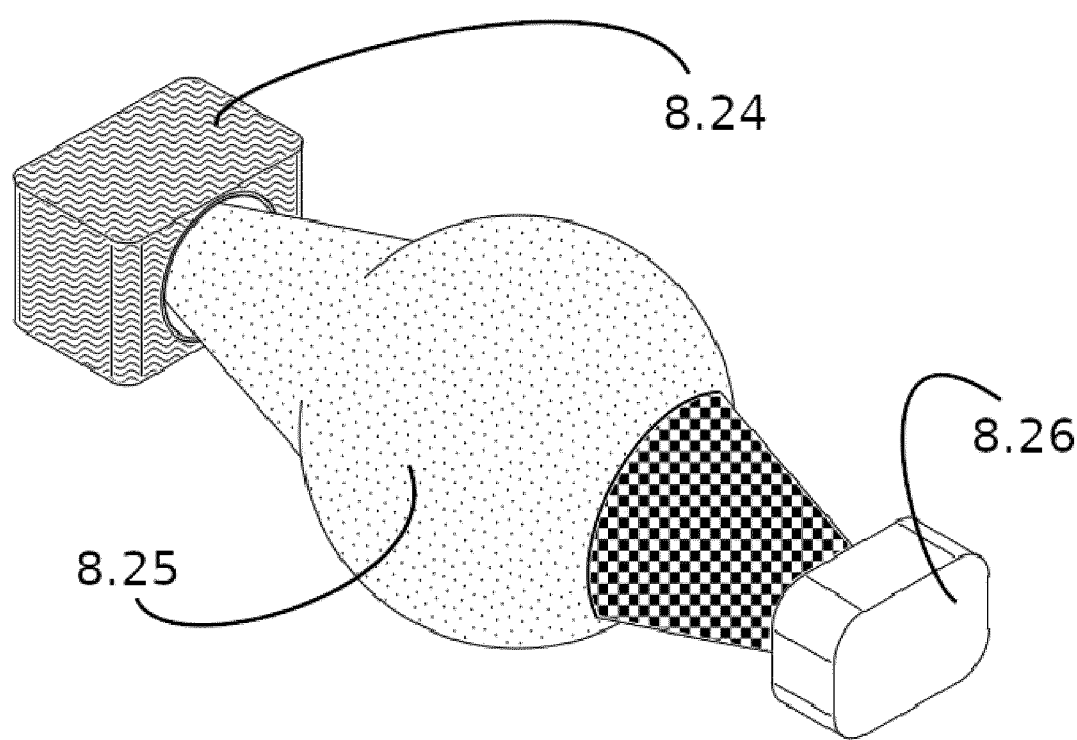


Figure 8

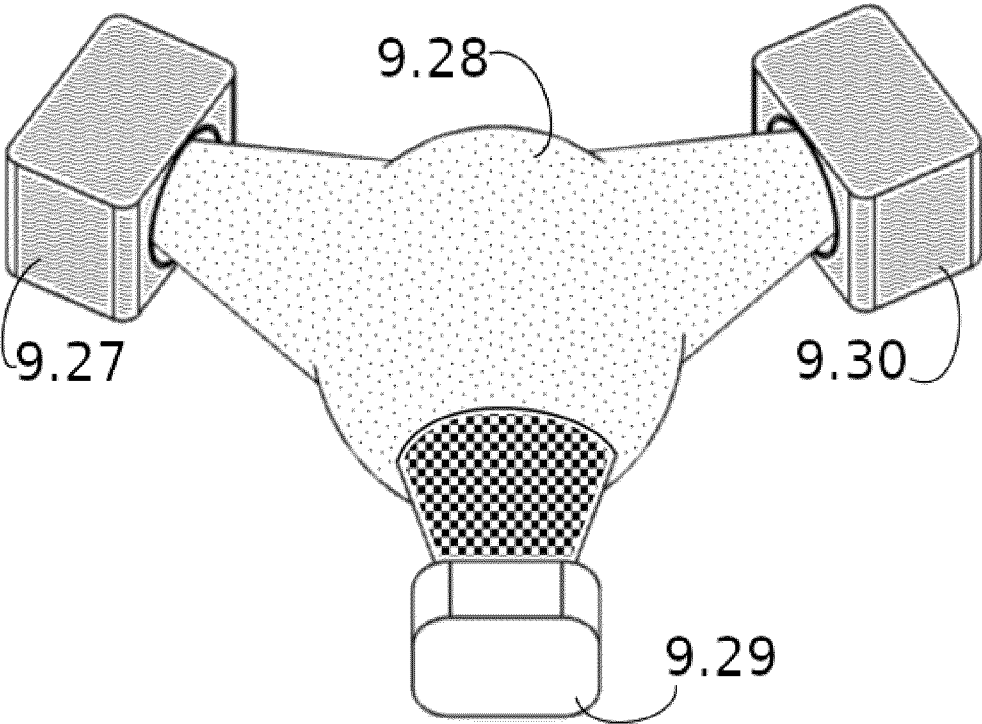


Figure 9

10/15

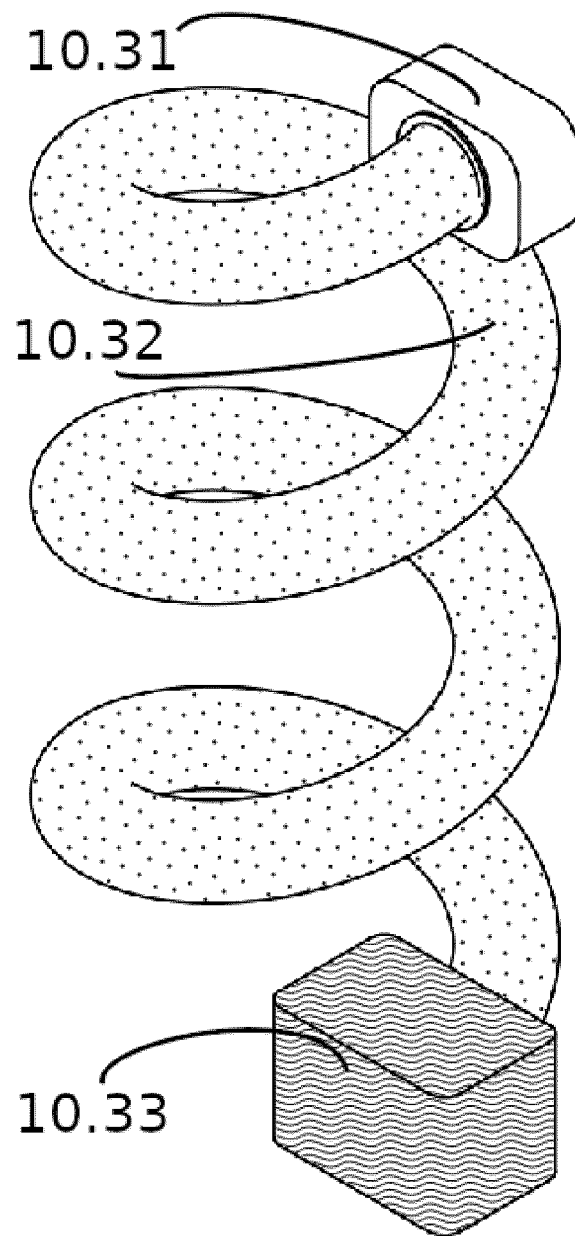


Figure 10

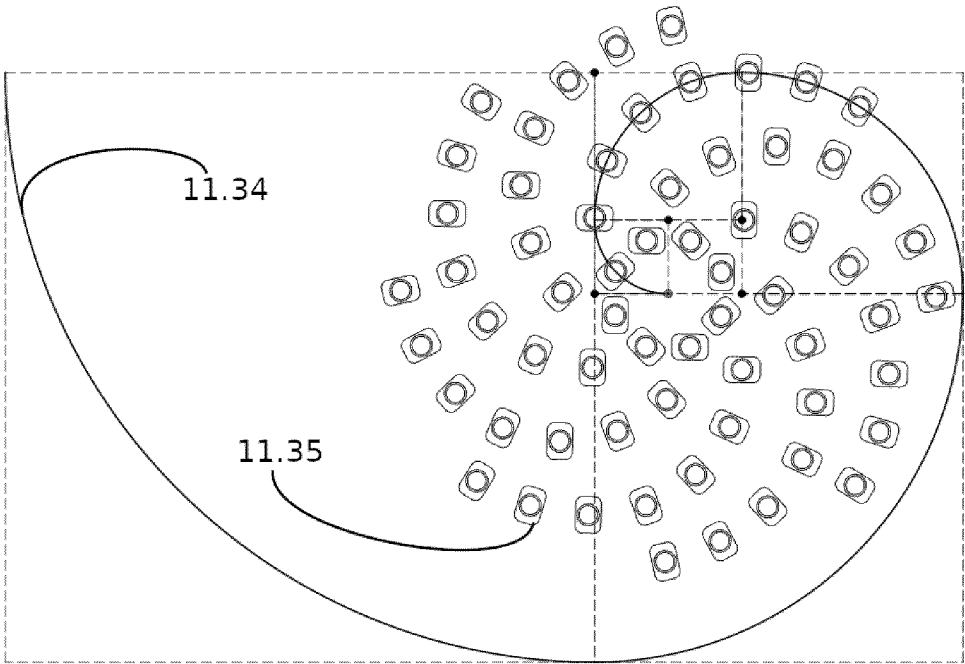


Figure 11

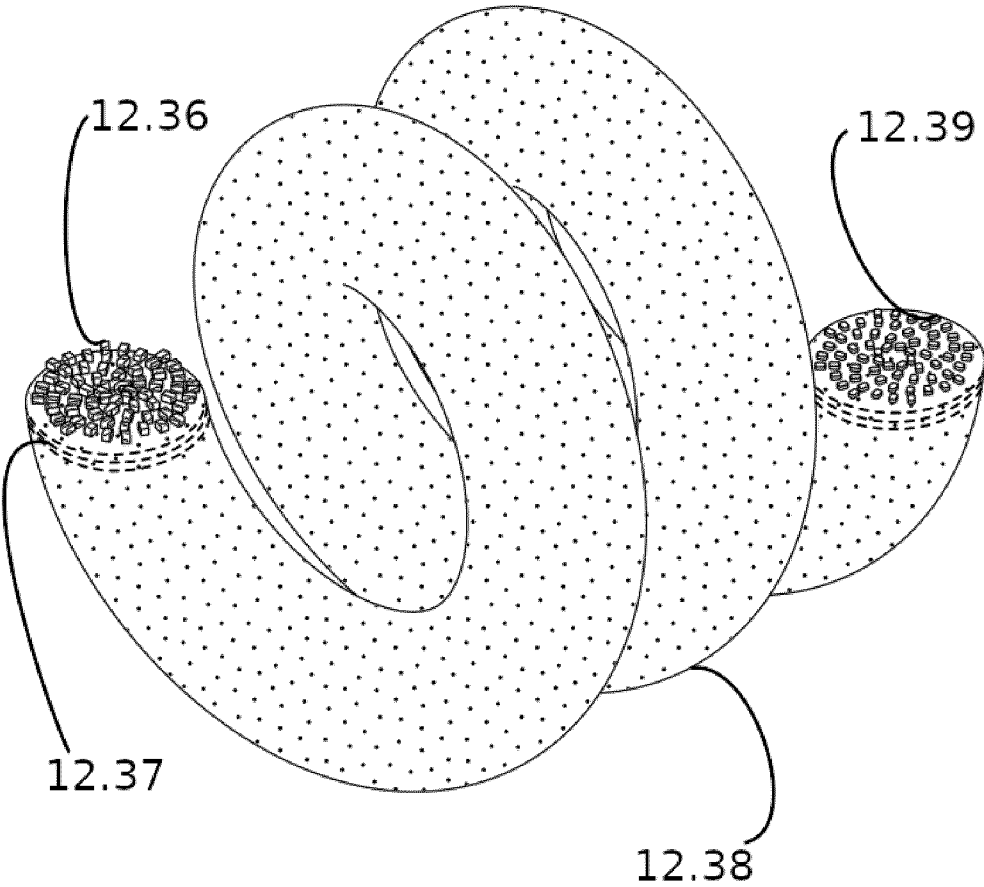


Figure 12

13/15

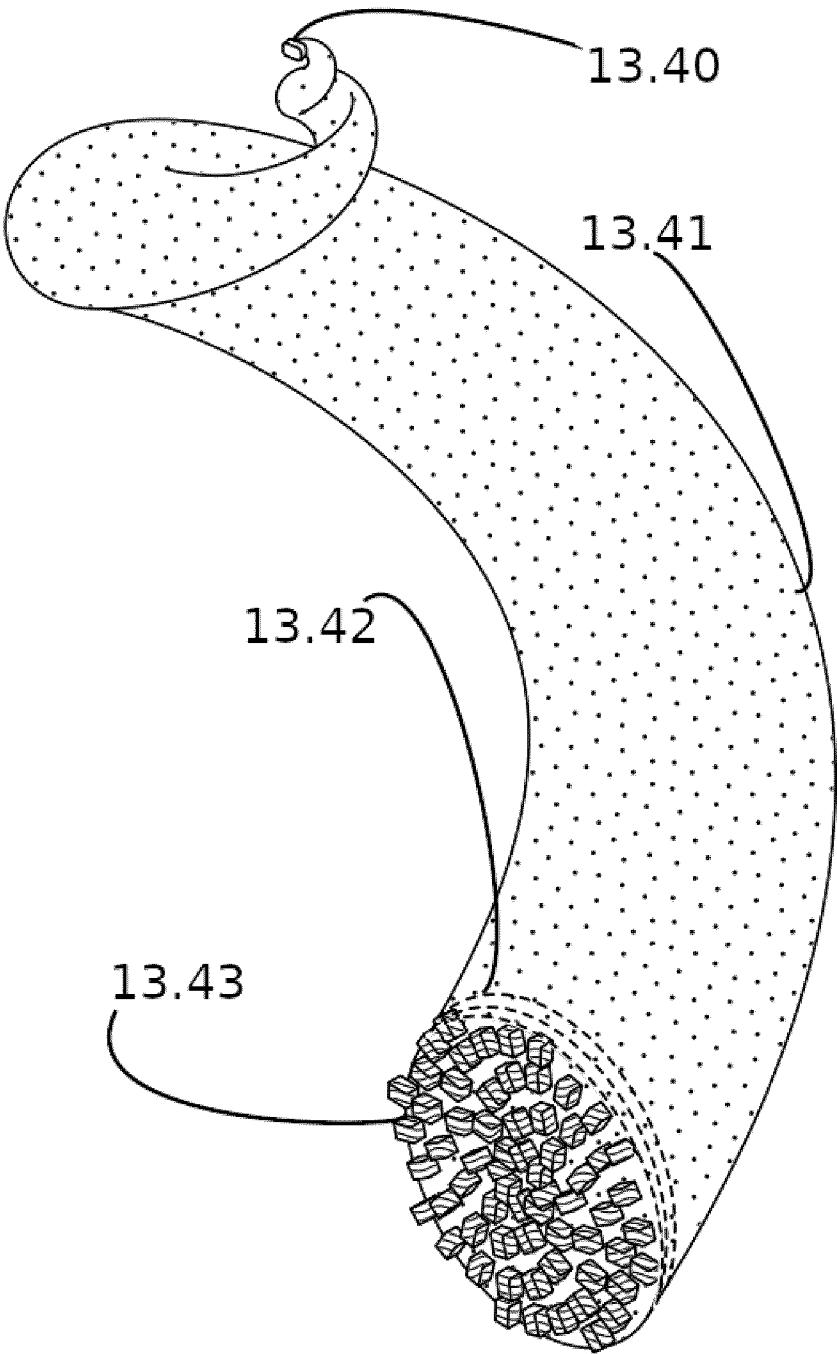


Figure 13

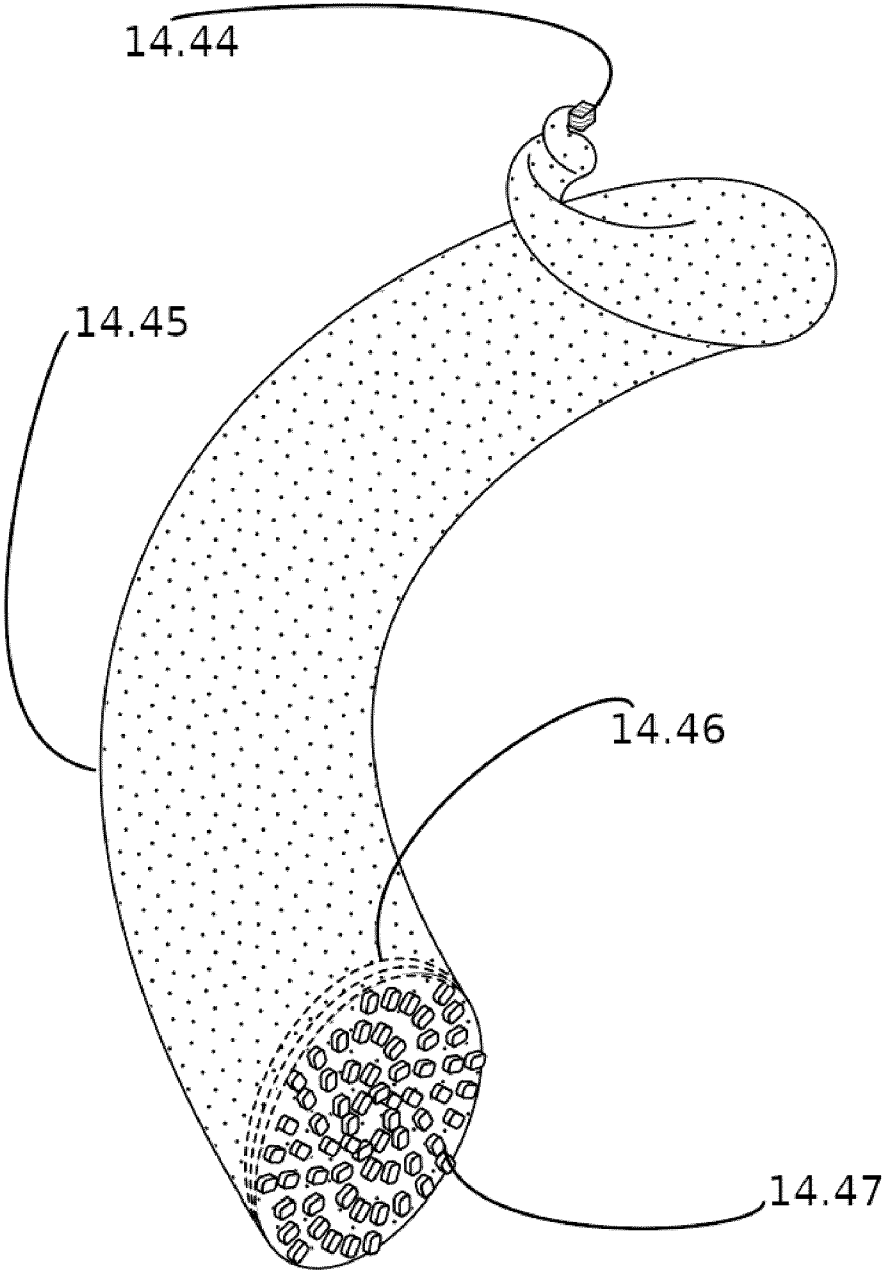


Figure 14

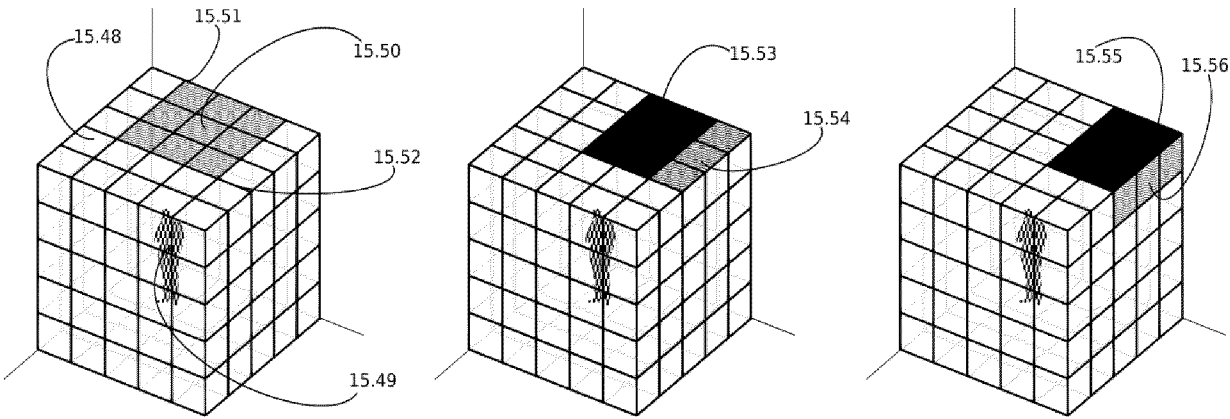


Figure 15