(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
03 April 2025 (03.04.2025)

WIPO | PCT

(10) International Publication Number
WO 2025/071925 A1

(51) International Patent Classification:
G06N 3/045 (2023.01) *G06N 3/0475* (2023.01)
G06N 3/0455 (2023.01) *G06N 3/09* (2023.01)

(21) International Application Number:
PCT/US2024/046118

(22) International Filing Date:
11 September 2024 (11.09.2024)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
18/374,424 28 September 2023 (28.09.2023) US

(71) Applicant: **MICROSOFT TECHNOLOGY LICENSING, LLC** [US/US]; One Microsoft Way, Redmond, Washington 98052-6399 (US).

(72) Inventors: **HOFMANN, Katja**; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, Washington 98052-6399 (US). **KANERVISTO, Anssi Samuli**; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, Washington 98052-6399 (US). **DEVLIN, Sam Michael**; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, Washington 98052-6399 (US). **RASHID, Tabish**; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, Washington 98052-6399 (US). **GUPTA, Tarun**; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, Washington 98052-6399 (US). **PEARCE, Timothy**; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, Washington 98052-6399 (US). **WHITE, Ryen W.**; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, Washington 98052-6399 (US).

(54) **Title:** GENERATIVE NEURAL APPLICATION ENGINE

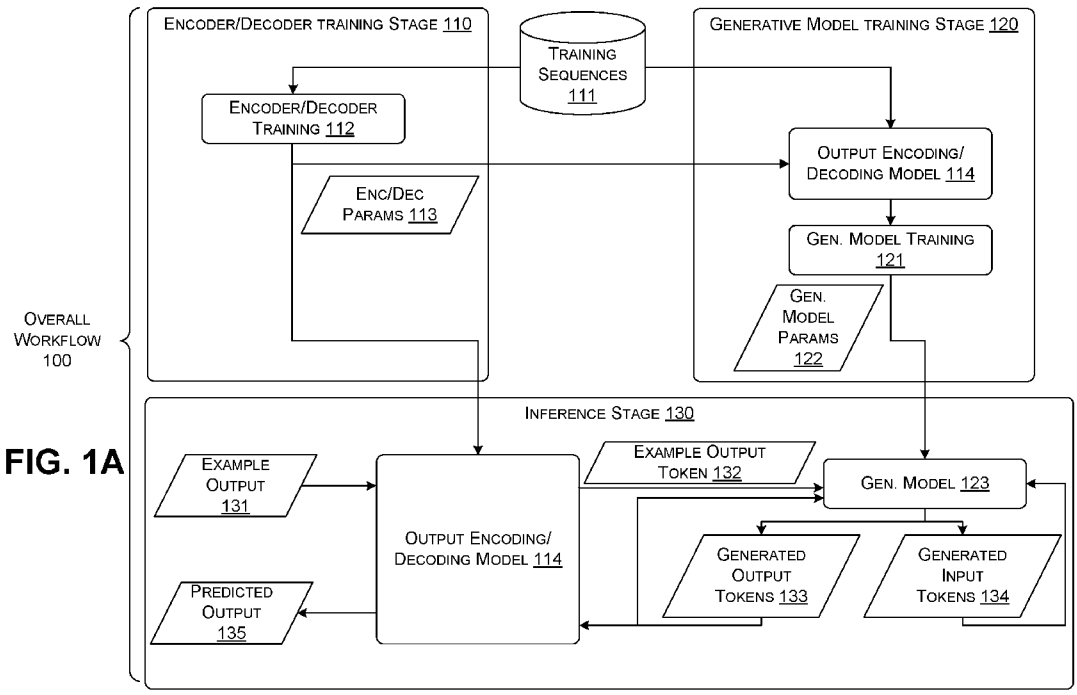


FIG. 1A

(57) **Abstract:** The disclosed concepts relate to implementation of application and application engine functionality using machine learning. One example method involves obtaining a seed image representing a seeded application state and mapping the seed image to at least one seed image token using an image encoder. The example method also involves inputting the at least one seed image token as a prompt to a neural dreaming model that has been trained to predict training sequences obtained from one or more executions of one or more applications, the training sequences including images output by the one more applications during the one or more executions and inputs to the one or more applications during the one or more executions. The example method also involves generating subsequent image tokens with the neural dreaming model, and decoding the subsequent image tokens with an image decoder to obtain subsequent images.

WO 2025/071925 A1

[Continued on next page]

(74) **Agent: CHATTERJEE, Aaron C.** et al.; Microsoft Technology Licensing, LLC, One Microsoft Way, Redmond, Washington 98052-6399 (US).

(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).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

GENERATIVE NEURAL APPLICATION ENGINE

BACKGROUND

[0001] Traditionally, interactive applications such as video games were developed by hand-coding the entire application. Subsequently, supporting technologies such as application engines were developed. An application engine allows a developer to develop their own code for part of their application while offloading certain functions, such as graphics rendering or physics simulations, to the application engine. However, while application engines have greatly simplified application development, coding of complex interactive applications is still a very intensive process.

SUMMARY

[0002] This Summary is provided to introduce a selection of concepts in a simplified form that are further described below in the Detailed Description. This Summary is not intended to identify key features or essential features of the claimed subject matter, nor is it intended to be used to limit the scope of the claimed subject matter.

[0003] The description generally relates to using a generative model to implement application or application engine functionality. One example includes a method or technique. The method or technique can include obtaining a seed image representing a seeded application state and inputting the at least one seed image token as a prompt to a neural dreaming model that has been trained to predict training sequences obtained from one or more executions of one or more applications, inputting the at least one seed image token as a prompt to a neural dreaming model that has been trained to predict training sequences obtained from one or more executions of one or more applications. The method or technique can also include generating subsequent image tokens with the neural dreaming model. The method or technique can also include decoding the subsequent image tokens with an image decoder to obtain subsequent images.

[0004] Another example includes a system that entails a hardware processing unit and a storage resource. The storage resource can store computer-readable instructions which, when executed by the hardware processing unit, cause the hardware processing unit to obtain a seed image representing a seeded application state. The computer-readable instructions can also cause the hardware processing unit to input the at least one seed image token as a prompt to a generative model that has been trained to predict image tokens and input tokens of training sequences obtained from one or more executions of one or more applications. The computer-readable instructions can also cause the hardware processing unit to generate subsequent image tokens with the generative model.

[0005] Another example includes a computer-readable storage medium storing computer-

readable instructions which, when executed by a hardware processing unit cause the hardware processing unit to perform acts. The acts can include accessing training data reflecting one or more executions of one or more applications, the training data including training sequences of images output by the one or more applications and inputs provided to the one or more applications during the one or more executions. The acts can also include mapping the images to training image tokens and the inputs to training input tokens. The acts can also include training a generative model to predict the training image tokens and the training input tokens sequentially according to the training sequences. The acts can also include outputting the trained generative model.

[0006] The above-listed examples are intended to provide a quick reference to aid the reader and are not intended to define the scope of the concepts described herein.

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] The Detailed Description is described with reference to the accompanying figures. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The use of similar reference numbers in different instances in the description and the figures may indicate similar or identical items.

[0008] FIG. 1A illustrates an example overall workflow for training and employing a machine learning model to implement application engine functionality, consistent with some implementations of the present concepts.

[0009] FIG. 1B illustrates an example decoder-based architecture for a generative model, consistent with some implementations of the present concepts.

[0010] FIGS. 2A-2D illustrate an example training sequence obtained by executing a first interactive application, consistent with some implementations of the present concepts.

[0011] FIGS. 3A-3D illustrate an example training sequence obtained by executing a second interactive application, consistent with some implementations of the present concepts.

[0012] FIGS. 4A illustrates an example of a seed image that can be input to a trained machine learning model, consistent with some implementations of the present concepts.

[0013] FIGS. 4B-4D illustrate an example sequence output by a trained machine learning model, consistent with some implementations of the present concepts.

[0014] FIG. 5 illustrates an example of an encoder/decoder and transformer architecture that can be employed as machine learning models, consistent with some implementations of the present concepts.

[0015] FIGS. 6A-6C, 7A-7C, 8A-8C, and 9 illustrate experimental results that were obtained using specific implementations of the present concepts.

[0016] FIG. 10 illustrates an example system, consistent with some implementations of the present concepts.

[0017] FIG. 11 illustrates a method for training a machine learning model to predict application outputs and inputs, consistent with some implementations of the present concepts.

[0018] FIG. 12 illustrates a method for employing a machine learning model to predict application outputs and inputs, consistent with some implementations of the present concepts.

DETAILED DESCRIPTION

OVERVIEW

[0019] As noted above, application engines can provide various useful functions for application developers. For instance, application engines can perform complex physics or rendering calculations that save application developers a great deal of effort, as the application developers can rely on physics or rendering routines provided by the application engines. This relieves the application developers from having to hand-code their own physics or rendering routines.

[0020] Interactive applications, such as video games or training simulations, are one type of application that can benefit greatly from an application engine. These types of applications often involve complex physics or rendering calculations that are computed in real time, and can be implemented very efficiently in an application engine. However, extensive development efforts are still often employed to develop the rest of the code for an interactive application.

[0021] Machine learning has been employed in limited contexts to model application and/or user behavior in interactive applications. For instance, behavioral cloning techniques can train a machine learning model to predict how a human would interact with an application, and world modeling can predict future outputs from an application given user input. However, neither behavioral cloning nor world modeling fully captures the interactions between human users and applications in a unified manner.

[0022] The disclosed implementations can train a generative model to predict application output together with the inputs that a human being would provide in response to the application output. Because the generative model jointly learns both user and application behaviors, the generative model can generate new sequences of application and user interactions that can be employed for various purposes. For instance, the trained generative model can be employed to prototype application scenarios that can be subsequently hand-coded for execution within an application engine. The trained generative model can also be used to replace an application and/or application engine partly or entirely, e.g., relying on the trained generative model to generate application outputs at runtime in response to received user inputs. The trained generative model can also be employed to temporarily take over providing input to an application on behalf of a given user, e.g., in the event of a network disruption that prevents or delays network communications between an application and a user device.

MACHINE LEARNING ALGORITHMS

[0023] There are various types of machine learning frameworks that can be trained to perform a given task. Support vector machines, decision trees, and neural networks are just a few examples of machine learning frameworks that have been used in a wide variety of applications, such as image processing and natural language processing. Some machine learning frameworks, such as neural networks, use layers of nodes that perform specific operations.

[0024] In a neural network, nodes are connected to one another via one or more edges. A neural network can include an input layer, an output layer, and one or more intermediate layers. Individual nodes can process their respective inputs according to a predefined function, and provide an output to a subsequent layer, or, in some cases, a previous layer. The inputs to a given node can be multiplied by a corresponding weight value for an edge between the input and the node. In addition, nodes can have individual bias values that are also used to produce outputs. Various training procedures can be applied to learn the edge weights and/or bias values. The term "parameters" when used without a modifier is used herein to refer to learnable values such as edge weights and bias values that can be learned by training a machine learning model, such as a neural network.

[0025] A neural network structure can have different layers that perform different specific functions. For example, one or more layers of nodes can collectively perform a specific operation, such as pooling, encoding, or convolution operations. For the purposes of this document, the term "layer" refers to a group of nodes that share inputs and outputs, e.g., to or from external sources or other layers in the network. The term "operation" refers to a function that can be performed by one or more layers of nodes. The term "model structure" refers to an overall architecture of a layered model, including the number of layers, the connectivity of the layers, and the type of operations performed by individual layers. The term "neural network structure" refers to the model structure of a neural network. The term "trained model" and/or "tuned model" refers to a model structure together with parameters for the model structure that have been trained or tuned. Note that two trained models can share the same model structure and yet have different values for the parameters, e.g., if the two models are trained on different training data or if there are underlying stochastic processes in the training process.

[0026] There are many machine learning tasks for which there is a relative lack of training data. One broad approach to training a model with limited task-specific training data for a particular task involves "transfer learning." In transfer learning, a model is first pretrained on another task for which significant training data is available, and then the model is tuned to the particular task using the task-specific training data.

[0027] The term "pretraining," as used herein, refers to model training on a set of pretraining

data to adjust model parameters in a manner that allows for subsequent tuning of those model parameters to adapt the model for one or more specific tasks. In some cases, the pretraining can involve a self-supervised learning process on unlabeled pretraining data, where a "self-supervised" learning process involves learning from the structure of pretraining examples, potentially in the absence of explicit (e.g., manually-provided) labels. Subsequent modification of model parameters obtained by pretraining is referred to herein as "tuning." Tuning can be performed for one or more tasks using supervised learning from explicitly-labeled training data, in some cases using a different task for tuning than for pretraining.

TERMINOLOGY

[0028] For the purposes of this document, the term "application" refers to any type of executable software, firmware, or hardware logic. The term "interactive application" refers to an application that performs processing responsive to received user input and iteratively, frequently, or continually adjusts application output in response to the received user input. Video games and flight simulators are two examples of interactive applications. The term "application output" refers to outputs such as video, audio, or haptic output by an application to provide a user experience. The term "actual user input" refers to input actually provided by a user during a course of interaction with an application. Application outputs and inputs can also be generated by a generative machine learning model, as described elsewhere herein.

[0029] The term "machine learning model" refers to any of a broad range of models that can learn to generate automated user input and/or application output by observing properties of past interactions between users and applications. For instance, a machine learning model could be a neural network, a support vector machine, a decision tree, a clustering algorithm, etc. In some cases, a machine learning model can be trained using labeled training data, a reward function, or other mechanisms, and in other cases, a machine learning model can learn by analyzing data without explicit labels or rewards. The term "user-specific model" refers to a model that has at least one component that has been trained or constructed at least partially for a specific user. Thus, this term encompasses models that have been trained entirely for a specific user, models that are initialized using multi-user data and tuned to the specific user, and models that have both generic components trained for multiple users and one or more components trained or tuned for the specific user. Likewise, the term "application-specific model" refers to a model that has at least one component that has been trained or constructed at least partially for a specific application.

[0030] The term "generative model," as used herein, refers to a machine learning model employed to generate new content. As discussed below, generative models can be trained to predict items in sequences of training data. When employed in inference mode, the output of a generative model can include new sequences of items that the model generates. The term "multi-

modal model," as used herein, refers to a machine learning model that operates on multiple categories or "modalities" of data. For instance, the following describes a generative model that is trained to produce application outputs, such as images, as well as application inputs, such as inputs representing controls from a video game controller.

[0031] A "neural dreaming model" is a generative model, based on a neural network architecture, that produces future output sequences of an application, e.g., future images from a video game. A neural dreaming model can be multi-modal, e.g., the neural dreaming model can also generate future sequences of inputs, such as video game controller inputs. A neural dreaming model can employ a transformer architecture. The term "transformer decoder" is used to refer to decoding layers of a transformer-based neural dreaming model to distinguish from decoders employed for purposes such as decoding of images.

EXAMPLE WORKFLOW

[0032] FIG. 1A shows an overall workflow 100, which involves an encoder/decoder training stage 110, a generative model training stage 120, and an inference stage 130. Generally speaking, the encoder/decoder training stage can train an encoder/decoder to encode and decode application outputs (e.g., images). The generative model training stage can train a generative model, such as a neural dreaming model, to predict sequences of application outputs and user inputs. The inference stage can employ the trained generative model to generate future sequences of application outputs and user inputs starting from a given point, e.g., a seed image.

[0033] The encoder/decoder training stage 110 and the generative model training stage 120 can employ training sequences 111, which can include training inputs and training outputs. For example, the training sequences can be obtained by logging executions of one or more applications. The training inputs can include inputs provided by human users during the executions, e.g., video game controller inputs, keyboard inputs, mouse inputs, spoken inputs, gestures, etc. The training outputs can include any type of output by the applications, e.g., video, audio, and/or haptic output produced by the applications in response to the user inputs.

[0034] The encoder/decoder training stage can involve accessing training outputs in the training sequences 111. As described more below, encoder/decoder training 112 can involve training an encoder/decoder model to represent application output, such as images, as tokens in a vector space. For instance, the encoder/decoder can map training images to tokens and then decode those tokens into corresponding images. A training objective can be defined that encourages the encoder/decoder to learn encoder/decoder parameters 113 that reduce or minimize the differences between the training images and the decoded or "reconstructed" images. Once the encoder/decoder training stage is complete, the encoder/decoder parameters can be output for use by output encoding/decoding model 114, which can be employed in both the generative model

training stage 120 and the inference stage 130 as described more below.

[0035] Generative model training stage 120 involves performing generative model training 121 to obtain generative model parameters 122. During generative model training, the generative model attempts to predict the input tokens and output tokens that are present in a given training sequence. As described more below, in some cases, the generative model is a transformer, and techniques for self-supervised learning of transformer parameters are employed, e.g., next token prediction. In other cases, bidirectional training techniques can be employed, e.g., by predicting preceding as well as subsequent tokens. To obtain input tokens representing inputs in the training sequences, each training input can be represented as one or more values, e.g., a string of bits. A deterministic function can be employed to map different user input mechanisms (e.g., button presses, joystick direction, etc.) to different values, such as bits, in a given input token. To obtain output tokens representing outputs in the training sequences, the output encoding/decoding model can be employed to map the outputs in the training sequences into corresponding output tokens. When generative model training stage 120 is complete, the generative model parameters 122 can be output for use in the inference stage 130 by generative model 123.

[0036] Inference stage 130 involves receiving an example output 131. For instance, the example output can be a seed image that conveys a seeded application state that a developer wishes to use as a starting point for subsequent predictions. The example output is processed by the output encoding/decoding model 114 using the learned encoder/decoder parameters 113 to obtain an example output token 132, which represents the example output in a vector space. The example output token is input to generative model 123. The generative model uses the generative model parameters 122 to produce generated output tokens 133 and generated input tokens 134, which are input back to the generative model to continue generating sequences of generated output and input tokens. The generated output tokens are also decoded by the output encoding/decoding model to produce generated output 135, e.g., by decoding the generated output tokens to obtain images.

[0037] Note that FIG. 1A shows an example where the generative model 123 employs its own generated inputs to predict future inputs and outputs during inference stage 130. Said another way, the generative model can "dream" by generating new sequences of user/application interactions, in some cases without receiving any input from a user. As described more below, however, in other cases, the generated inputs can be discarded and actual user inputs can be provided to the generative model. In this case, the user is effectively interacting with the model acting directly as an interactive application. Also, note that FIG. 1A shows separate training stages for encoding/decoding and generative model training. However, as discussed more below, in some cases, joint training of encoder/decoder and generative models can be employed.

EXAMPLE DECODER-BASED LANGUAGE MODEL

[0038] FIG. 1B illustrates an exemplary generative model 150 that can be employed using the disclosed implementations. Generative model 150 can receive input tokens 152, e.g., input tokens representing user input or output tokens representing application output. Position encoding 154 represents the location of each token in order relative to the other tokens in a sequence of input tokens.

[0039] The tokens and position encodings are processed in one or more transformer decoder blocks 156. Each transformer decoder block implements masked multi-head self-attention 158, which is a mechanism relating different positions of tokens within a given sequence of tokens to compute the similarities between those tokens. Each token is represented as a weighted sum of other tokens in the input. Attention is only applied for already-decoded values, and future values are masked. Layer normalization 160 normalizes features to mean values of 0 and variance to 1, resulting in smooth gradients. Feed forward layer 162 transforms these features into a representation suitable for the next iteration of decoding, after which another layer normalization 164 is applied. Multiple instances of transformer decoder blocks can operate sequentially on input tokens, with each subsequent transformer decoder block operating on the output of a preceding transformer decoder block. After the final transformer decoding block, token prediction layer 166 can predict the next token in the sequence, which is output as output token 168 and also fed back into the generative model.

[0040] Generative model 150 is an example of a transformer-based generative model. For example, generative model 150 could be implemented using one or more versions of models such as ChatGPT, BLOOM, PaLM, and/or LLaMA. Note that other types of generative models, e.g., recurrent models, can also be employed.

FIRST EXAMPLE TRAINING SEQUENCE

[0041] FIGS 2A-2D collectively show an example training sequence from a first application, e.g., a driving game. For the following discussion, assume that a user played a driving video game and that the output of the driving video game was logged together with the user's inputs while they played the game. FIG. 2A shows example output 200(1) of the driving video and associated example input 210(1) provided by the user at a first time. FIG. 2B shows example output 200(2) of the driving video game and associated example input 210(2) provided by the user at a second time. FIG. 2C shows example output 200(3) of the driving video game and associated example input 210(3) provided by the user at a third time. FIG. 2D shows example output 200(4) of the driving video game and associated example input 210(4) provided by the user at a fourth time.

[0042] Referring to FIG. 2A, output 200(1) shows a car 201 moving on a road 202 with a map inset 203 showing the position of the car on the road, with a tree 204 in the background. Example

input 210(1) shows controller inputs that were entered by a user while example output 200(1) was being output by the driving game. The controller inputs include a directional input 211, which steers the car, and an acceleration input 212, which accelerates the car. In FIG. 2A, the car is heading almost straight ahead toward a left turn, and the user is not accelerating the car.

[0043] FIG. 2B shows a subsequent point in the training sequence, where the user has corrected their steering by turning more sharply to the left, as shown by directional input 211 in example input 210(2). The car correspondingly turns more sharply into the left turn, as shown by example output 200(2). FIG. 2C continues the training sequence, with the user having corrected their steering and deciding to accelerate the car, as shown by acceleration input 212 in by example input 210(3). FIG. 2D shows the car moving further down the road while the user continues to accelerate.

SECOND EXAMPLE TRAINING SEQUENCE

[0044] FIGS 3A-3D collectively show an example training sequence from a second application, e.g., a fantasy game. For the following discussion, assume that a user played the fantasy video game and that the output of the fantasy video game was logged together with the user's inputs while they played the game, similarly to the driving video game example discussed previously. FIG. 3A shows example output 300(1) of the fantasy video game and associated example input 310(1) provided by the user at a first time. FIG. 3B shows example output 300(2) of the fantasy video game and associated example input 310(2) provided by the user at a second time. FIG. 3C shows example output 300(3) of the fantasy video game and associated example input 310(3) provided by the user at a third time. FIG. 3D shows example output 300(4) of the fantasy video game and associated example input 310(4) provided by the user at a fourth time.

[0045] Referring to FIG. 3A, example output 300(1) shows a character 301 on a hoverboard 302. Example input 310(1) shows directional input 211 pointed directly forward. At this time, the user is employing the directional input to move the character forward through the scene. FIG. 3B shows a subsequent point in the training sequence, where the user has moved the directional input slightly to the right. In response, the character turns slightly to the right in the example output 300(2). In FIGS. 3C and 3D, the inputs remain consistent while the application output shows the character continuing to move through the scene.

[0046] Note that FIGS. 3A-3D do not show the use of the acceleration input 212. In the fantasy video game, the directional input alone controls the movement of the character and a separate acceleration input is not utilized to move the character. Thus, the two different applications use different control paradigms.

EXAMPLE GENERATED SEQUENCE

[0047] Consider a generative model that has been trained on thousands of training sequences

similar to the two training sequences described above. The generative model could learn what types of images the applications tend to output in response to user inputs, as well as what user inputs the users tend to provide in response to the images they see. For instance, the generative model could learn that users tend to move the directional input to the left when heading into a left turn, that applications can move objects such as cars or characters forward in response to acceleration inputs or directional inputs, etc. Given an example output from which to start, the generative model can then produce its own generated sequences of outputs and inputs.

[0048] Consider a developer that wishes to see a gaming experience where a character on a hoverboard drives the hoverboard on a road, in a manner similar to a car. The developer can provide an example output 400(1) to the generative model as shown in FIG. 4A. The example output shows a character 401 on a hoverboard 402 driving on a road 403. FIG. 4B-4D collectively show an example generated sequence that could be generated by a generative model trained on the training sequences described above. Note that the developer can also provide an example input but does not necessarily need to do so, and no example input is shown in FIG. 4A.

[0049] FIG. 4B shows generated output 400(2), which shows character 401 moving along the road 403. The generative model can generate map inset 404 and tree 405, e.g., as learned from the driving game. The generative model can also adjust the orientation of the character and the hoverboard to match the trajectory on the road, e.g., as learned from the fantasy game. The generative model can also produce generated input 410(2), which shows the directional input steering the hoverboard character to the left. FIGS. 4C and 4D show that the generative model predicts the character will continue to move down the road via generated outputs 400(3) and 400(4), respectively while the input gently steers the character through the turn as shown via generated inputs 410(3) and 410(4).

[0050] Note that in FIGS. 4B, 4C, and 4D, the generative model predicts that the input will gently steer the character along the road using the directional input 211, without using the acceleration input. This could be, for example, because the generative model has learned the control paradigm from the fantasy application, e.g., that the acceleration input is not necessary to move the character forward and that the directional input alone can do so. The generative model also correctly updates the position of the character on map inset 404 as the character travels down the road. This could be, for example, because the generative model has learned to update the map inset from the driving application. More generally, the trained generative model has learned to utilize concepts from both applications that were used to train the model to provide a coherent user experience. By starting with the example shown in FIG. 4A, the trained generative model is able to generate a sequence of later application outputs and inputs independently. As described more below, this allows the generative model to be employed for a wide range of applications,

from prototyping of new applications that are subsequently hand-coded to running new applications entirely within the trained model.

SPECIFIC IMPLEMENTATION

[0051] The following describes a specific implementation of the disclosed concepts, named WHAM (World and Human Action Model). WHAM is a model that jointly learns to do both behavioral cloning and world modeling. WHAM encodes image observations as discrete tokens using an image encoder, and the image tokens are interleaved with action tokens (representing user inputs) to form training sequences. A transformer model is then trained to do next-token prediction on a large-scale human gameplay dataset.

[0052] FIG. 5 presents an overview of WHAM, which structures trajectories as an interleaved sequence of observation and action tokens. An encoder/decoder 510 includes an encoder 511 and a decoder 512. The encoder performs tokenization of images, from observation space, $\mathbf{o}_t \in \mathrm{R}^{H \times W \times 3}$, to a compact discrete latent space $\mathbf{z}_t \in \{1,2,\ldots,V_o\}^{dz}$, for vocabulary size V_o and bottleneck size d_z. Each image can be encoded as one or more image tokens. The decoder can transform one or more image tokens into a corresponding image in the observation space. A transformer 520 (e.g., a causal transformer decoder employed as a generative model) is then trained to predict these latent observation tokens $\hat{\mathbf{z}}_t$ and discretized action tokens, $\hat{\boldsymbol{a}}_t$, for successive timesteps, $t \in \mathrm{N}$.

[0053] Notation. $\mathbf{z}_t$ refers to all tokens encoding an observation $\mathbf{o}_t$ at timestep t, and z_t^i when referring to the i^{th} token of that latent observation. A similar convention applies to $\mathbf{a}_t$ and a^i_t. Hatted variables denote model predictions.

[0054] Observation tokenization. WHAM's image encoder provides a deterministic mapping, $\mathrm{Enc}_\theta(\mathbf{o}_t) \rightarrow \mathbf{z}_t$, while the decoder deterministically maps, $\mathrm{Dec}_\theta(\mathbf{z}_t) \rightarrow \hat{\boldsymbol{o}}_t$. To allow the transformer model to operate on discrete tokens, a model such as the VQVAE (Van Den Oord, et al., "*Neural discrete representation learning,*" Advances in neural information processing systems, 30, 2017) encoder/decoder architecture can be employed. VQVAE is a convolutional autoencoder, with a quantization layer at the bottleneck. Observations are first mapped to a continuous latent vector, $\mathbf{w}_t \in \mathrm{R}^{dz \times d}$. These continuous vectors are then quantized to the nearest VQVAE codebook vector, $e_j \in \mathrm{R}^d$ for $j \in \{1,2,\ldots,V_o\}$, via, $z_t^i = arg\,min_j \|e_j - \omega_t^i\|2$. VQVAE uses two reconstruction losses; pixelwise MSE, and a perceptual loss (Zhang, et al., "*The unreasonable effectiveness of deep features as a perceptual metric,*" In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 586–595, 2018).

[0055] Another model that can be employed for the encoder/decoder is VQGAN (Esser, et al., "*Taming transformers for high-resolution image synthesis*," In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 12873–12883, 2021). VQGAN introduced several innovations. For instance, VQGAN adds a further reconstruction loss: a GAN

discriminator is trained to distinguish patches from the reconstruction from patches from the original image. In preliminary experiments discussed below, VQGAN produced higher-quality images compared to VQVAE. Comparing the two provides insights on whether perceptual quality correlates with overall model performance.

[0056] Action tokenization. The action space is an Xbox controller, which has 12 binary buttons and two joysticks. Each joystick is decomposed into an x and y component, which are discretized into 11 buckets. This creates 16 = 12 + 4 total action dimensions. Vocabulary is assigned so that each action dimension has its own unique tokens, two for buttons, and 11 for each joystick dimension, giving a total action vocabulary size of, $V_a := 68 = (12 \times 2) + (4 \times 11)$.

Specific Transformer Example

[0057] Following the tokenization of observations and actions as described above, training sequences, **c**, take the form:

$$\mathbf{c}[z_t^0, z_t^1, \ldots, z_t^{d_z-1}, a_t^0, a_t^1, \ldots, a_t^{15}, z_{t+1}^0, z_{t+1}^1, \ldots z_{t+1}^{d_z-1}, a_{t+1}^0, a_{t+1}^1, \ldots, a_{t+1}^{15}, \ldots], \quad (1)$$

where each item of the sequence, $c_i \in \{0, 1, \ldots, V_o + V_a\}$, is simply an integer within the vocabulary. These sequences allow for training of transformer models on next-token prediction, e.g. by maximizing the likelihood, $p(\hat{c}_i = c_i | c_{<i})$.

[0058] WHAM employs a causal transformer architecture with 205M parameters. No explicit encoding is employed to distinguish whether an observation or action token should be generated next. Rather, the model learns to predict either observation or action tokens from its learned position embeddings. At test time, illegal token selections are masked out – e.g. when predicting an observation token, action token logits are set to negative infinity. Sequences are constructed so that a complete observation, beginning from z_t^0, comes first. WHAM was trained on sequences of ten observation tokens and ten action tokens. WHAM samples dimensions autoregressively allowing samples from the joint distribution, $p(zti, zti+1, zti+2) = p(zti)p(zti+1|zti)p(zti+2|zti, zti+1)$.

Training Regimes

[0059] Two regimes for training WHAM are described and compared below. The first regime involves training the encoder/decoder first, with only a reconstruction loss, then freezing the encoder/decoder weights and training the transformer – as in Micheli, et al., "*Transformers are sample efficient world models*," In International Conference on Learning Representations, 2023). The second regime involves beginning with the pretrained encoder/decoder checkpoint, continuing to train the encoder/decoder jointly with the transformer – similar to Hafner, et al., "*Mastering atari with discrete world models*," arXiv preprint arXiv:2010.02193, 20. Here, the encoder receives gradients from both the decoder reconstruction loss *and* the next-token prediction

loss. Joint training might encourage the extracted observation representations, $\mathbf{z}_t$, to prioritize information that helps action prediction (such as salient game details). However, this comes at the cost of higher VRAM requirements/smaller batch sizes. One useful property of the first training regime is that image observations can be tokenized ahead of time, streamlining dataloading and training.

Game Environments and Dataset

[0060] The following details the environment and dataset used throughout the experiments described below. The video game Bleeding Edge was used as a testbed environment. Bleeding Edge is a team-based 4v4 online multi-player video game. Players select from thirteen possible heroes, each with different abilities. The camera is set in a third-person view, which makes the environment partially observable. Experiments were conducted using the *Power Collection* game mode. Players compete to collect power cells that spawn at random locations on the map at fixed time intervals. Points are scored by delivering these power cells to hand-in platforms, which activate for a limited time period. Throughout this, the two teams engage in melee and range-style combat, which can also earn points. The game dynamics are complex, both moment-by-moment through the precise control needed for fights, and also through longer-term strategies required for the power cell objective, which benefits from team coordination. The data and experiments focus on a single map called *Skygarden*, which is spread over several islands each with multiple elevation levels.

[0061] Bleeding Edge visuals are highly complex. The 3D view provides important game-relevant information about the map geometry, power cells and other players. Heads-up-display (HUD) elements such as the mini-map and health information are small details on the screen but carry important information. Overall, the Bleeding Edge environment combines several types of complexity that are absent in the benchmark environments used in prior world modeling research.

Human Data Collection

[0062] Human gameplay data was recorded as part of the regular gameplay process, in order to enable in-game functionality as well as to support future research. Games were recorded on the servers that hosted the games in the form of so-called *replay files*. Recordings include a representation of the internal game state and controller actions of all players. To minimize risk to human subjects, any personally identifiable information (Xbox user ID) was removed when extracting the data used for this study from the original replays.

Dataset

[0063] For the experiments outlined below, a dataset was extracted from the replay files. Human gameplay actions were extracted as hybrid (discrete and continuous) controls. Joystick actions were represented as continuous values that control the player direction (left joystick) and

camera direction (right joystick). Controller buttons were represented by discrete values. Image data was stored as MP4s. The resulting data was cleaned to remove errors and data from bad actors as detailed below, after which data remained for 8,788 matches, yielding 71,940 individual player trajectories, totaling 3.87TiB on disk. These matches were recorded between 09-02-2020 and 10-19-2022, and amount to 56.3 days of match time, or 9875 hours (1.12 years) of individual game play. Sampled at a rate of 10Hz, this equates to 355.5M frames. This individual game play data was divided into training / validation / test sets by dividing the 8788 matches with an 80:10:10 split.

Experimental Results

[0064] The following results evaluate the ability of the trained models to play a version of the game. Game relevant statistics such as (human normalized) damage dealt to opponents, objectives (power cells) collected, and healing done per episode are reported. These statistics convey the ability of the trained model to generate inputs to the game that cause corresponding results, such as damaging opponents, collecting power cells, and healing. The results are averaged over 750 episodes. Starting states are sampled randomly from the dataset. Episodes last for 30 seconds of game-time.

[0065] Linear probing was performed to obtain insights into what game-relevant concepts the models' internal representations capture. Using privileged game state information, multiple binary classification tasks were constructed, such as 'is the current hero Daemon?' or 'will the character die in the next two seconds?'. Linear logistic regression probes were then trained using the model's hidden activations as input. Results were normalized relative to a randomly initialized CNN (convolutional neural network), and an oracle achieving 100% accuracy.

[0066] Dreaming evaluates the ability of WHAM to generate plausible future observations. Dreams were conditions on a ground-truth context of observations and actions from a reference human trajectory *up to* timestep t. O*ne-step dreaming* – the model's capability to autoregressively predict the immediate next observation $\hat{\mathbf{z}}_{t+1}$, was evaluated, and also *multi-step dreaming* – predicting $\hat{\mathbf{z}}_{t+1}, \hat{\mathbf{z}}_{t+2}, \ldots$, but conditioned on *ground-truth* human actions, $\mathbf{a}_{t+1}, \mathbf{a}_{t+2}, \ldots$. To quantify the quality of the imagined sequences, cosine similarity between the true and predicted encoder embeddings was computed at each step.

[0067] These metrics allow quantification of the performance of different models relative to one another and enable understanding of the impact of key modeling choices. The following discussion evaluates a range of modeling choices that have been trained for a fixed compute budget. Unless detailed otherwise, performance is reported after $20k$ update steps.

Training & Evaluation Compute

[0068] The encoder/decoder models were trained on 4×A6000 GPUs for up to four days, while

the transformer models each used 8×V100 GPUs for four days. For all evaluation runs we used virtual machines with either M60, A6000 or A100 NVIDIA GPU cards.

[0069] For reference, the models and variants are referred to using the form {W}HAM-(Joint)-64{VQVAE|VQGAN}, where WHAM denotes a World and Human Action Model, and HAM denotes an action-only model. Joint indicates that the encoder was trained jointly with the sequence model. 64 denotes the encoder's bottleneck size, d_z, and VQVAE|VQGAN denotes the encoder/decoder architecture.

Results of Predicting Actions and Observations

[0070] As described below, WHAM can learn to predict both user inputs (actions) and application outputs (observations) in a single model, without necessarily compromising performance on either component. This type of joint model can provide additional capabilities and sidesteps the need to train two separate models. Ideally, this should not come at the expense of a tradeoff of model capacity. The following confirms that additionally predicting the more complex observation tokens does not negatively impact the model's ability to predict actions.

[0071] To investigate this, the following compares a model that predicts only actions (HAM-64VQVAE), with a model that predicts both actions and observations (WHAM-64VQVAE). Other hyperparameters are shared – using a VQVAE trained independently from the transformer, and set $d_z = 64$. FIGS. 6A shows a training loss graph 610, FIG. 6B shows a rollout performance graph 620, and FIG. 6C shows a linear probe results graph 630. Both models achieve similar cross-entropy loss over action tokens, and similar online performance. Linear probe results graph 630 shows that the representations of the WHAM model are encoding more game-relevant information. In particular, the WHAM model is encoding which character is currently being used – information that the WHAM model can capture as a side effect of predicting the observation tokens. Note that WHAM does not appear to compromise on the ability to predict actions relative to the HAM model that predicts only actions.

Training of Encoder/Decoder and Transformer

[0072] As noted above, several training regimes were implemented: 1) training the encoder/decoder on a reconstruction loss, and then training the transformer separately, and 2) continuing to train the encoder/decoder *jointly* with the transformer. FIG. 7A shows training loss graph 710 and FIG. 7B shows rollout performance graph 720 for the two training regimes. These results compare the performance of WHAM-64VQVAE with the pre-trained VQVAE described previously to that of the jointly trained WHAM-Joint-64VQVAE model. Note that WHAM-Joint-64VQVAE exhibits significantly more unstable training than using the pre-trained encoder. One potential explanation is that having the encoder provide both the inputs and the targets for training creates a harmful feedback loop. This results in significantly worse performance for WHAM-

Joint-64VQVAE when it is rolled out online. Turning to the FIG. 7C, which shows linear probe results graph 730, note that the representations for the joint training regime suffer, with WHAM-Joint-64VQVAE performing significantly worse at character identification.

[0073] Joint training again leads to a significantly less stable cross-entropy loss over the observation tokens. The joint loss appears lower, but this kind of direct comparison is misleading – unlike for the action tokens, the observation token targets are now generated by different encoders, and for the joint model, these change through training. For a more grounded comparison of model quality, the reconstructions of both models can be inspected. The result is substantially worse for the jointly trained model, with highly blurred reconstructed images that miss game relevant details. It does not appear that a joint training regime allows models to capture more game relevant detail at the current scale of compute. This is a promising result, because pre-training and freezing the observation encoder leads to very large gains in training efficiency and thus more scalable models.

Choice of Encoder/Decoder Model

[0074] WHAM has the ability to model the world at future timesteps based on its own generated inputs, also referred to as 'dreaming.' The decoder maps from generated latent tokens $\hat{\boldsymbol{z}}_t$ to images, $\hat{\boldsymbol{o}}_t$, allowing unique insight into what WHAM can represent and predict. For instance, inspection of the dreamed trajectories can be employed to qualitatively assess how well WHAM models the game physics, geometry, and other game-play details. If some aspect of the game can be visually identified in the dreamed trajectory, then that aspect of the game has been represented and generated by WHAM. Generally speaking, VQGAN provides higher output quality than VQAE when rated by human users.

[0075] FIG. 8A shows a training loss graph 810, FIG. 8B shows a rollout performance graph 820, and FIG. 8C shows linear probe results graph 830, comparing the relative performance of using VQVAE vs. VQGAN as the encoder/decoder. These results show that the qualitative improvement in visual reconstruction by VQGAN indeed translates to improved model performance when rolled out online. This is true even though the action and image losses are slightly higher for VQGAN than for VQVAE. For example, this difference is particularly pronounced in terms of number of power cells collected.

[0076] FIG. 9 shows dream results graphs 910, 920, 930, and 940, which compare the dreaming ability of the two models. The cosine similarity quantitative metric suggests that the dream quality of VQVAE and VQGAN is similar. However, from a qualitative perspective, there is a stark difference in image quality between the two. In particular, identifying the character is much easier for a human based on the VQGAN reconstructions compared to the VQVAE reconstructions.

[0077] However, FIG. 8C suggests that the model's internal representation is not linearly separable in terms of game-specific features, including the character identity. Therefore, it is likely that the additional discriminator in the decoder of VQGAN causes a qualitative visual benefit and quantitative improvement in ability to reproduce human behavior (FIG. 8B). Thus, VQGAN resulted in the highest performing model overall, even though its internal representation could not be interpreted via linear probing.

Further Model Details

[0078] Encoder model details. Both VQGAN and VQVAE encoders take as input an image of size128 × 128 × 3, and output codebook indices with latent vocab size of $V_o = 4096$. The VQVAE encoder and decoder that were employed are adaptations of the code by Van Den Oord, et al., "*Neural discrete representation learning*," Advances in neural information processing systems, 30, 2017. An additional convolutional layer in the encoder and decoder was added to reduce the bottleneck size to 64, and also add perceptual image loss to improve reconstruction quality (Zhang, et al., "*The unreasonable effectiveness of deep features as a perceptual metric*," In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 586–595, 2018). For training VQGAN models, the taming-transformers repository was employed (Esser, et al., "*Taming transformers for high-resolution image synthesis*," In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 12873–12883, 2021). Encoder/decoder training parameters are detailed in the following Table 1:

Parameter	VQVAE	VQGAN
Num optimizer updates	100,000	100,000
Learning rate	0.0001 → 0.00001	0.00054 (4.5e-06*Batch Size)
LR schedule	Cosine annealing with warmup	Constant with warmup
Warmup steps	500	1000
Optimizer	AdamW	AdamW
Vocab size	4096	4096
Embedding dimension	512	512
Batch size	150	120
Gradient clip threshold	1.0	1.0
Discriminator	N/A	0.25

Weight GAN Loss Starting Step	N/A	20,000

[0079] Hyperparameters for the transformer training are described in Table 2 below:

Parameter	Value
Num. optimizer updates	20,000
Num. timesteps trained	160M
Learning rate	0.0001
LR schedule	Constant with warmup
Warmup steps	1000
Optimizer	AdamW
Weight decay	0.0
Gradient clip threshold	1.0
Num. transformer blocks	16
Num. attention heads	16
Transformer embedding size	1024
Parameter count	208M
Control rate	10Hz
Context length in steps	10
Batch size	800

For each run, a mini-batch size that fits into the V100's VRAM was employed, and gradient accumulation was adjusted to reach an effective batch size of 800.

Tokenization details

[0080] Separate tokens for images and actions were employed. For image encodings, the codebook indices from the VQ encoder were employed. For actions, each of the 16 dimensions was represented separately (12 for buttons, 4 for the discretized gamepad sticks). All tokens were then embedded using a single embedding layer.

[0081] The main training hyperparameters for the WHAM model are detailed above. The transformers employed the NanoGPT implementation. The selected model size (208M parameters) gave significant improvement in terms of training/validation loss over smaller models (10 blocks, 10 heads, 640 embedding size, 74M parameters), but not significantly worse than a larger one (18 blocks, 11 heads, 1408 embedding size, 500M parameters). FlashAttention was employed to speed up training as described in Dao et al., "*Flashattention: Fast and memory efficient exact attention with io-awareness*," Advances in Neural Information Processing Systems, 35:16344–16359, 2022.

[0082] With a training context of 10 timesteps, 64 tokens for images and 16 tokens for actions, the final context length of the models is 800 tokens. After tokenizing the context, the cross-entropy loss of predicting the next token given all the previous ones was calculated, and the loss was averaged over the full token sequence. During inference, invalid tokens were masked for the current prediction step. For instance, when predicting an image token, action-related tokens were masked out.

[0083] For HAM models, the same setup was employed as for WHAM, but the prediction loss for image tokens was set to zero. To enable joint training of the encoder and transformer model, the previously-described setup was modified. Instead of using the codebook indices, a single linear layer that mapped the codebook *vectors* into transformer vectors was employed, effectively serving as an embedding layer but allowing backpropagation into the encoder. During joint training the encoder and decoder were also updated with the VQVAE's own reconstruction loss using the same batches. For actions, a separate trainable embedding layer was employed.

Training Details

[0084] All models were trained using an Azure virtual machine cluster with 56 32GB Nvidia V100 GPUs in total, distributed across 7 ND40rs-series nodes with 40 CPUs each and 671 GB RAM. The models were trained on either 8 GPUs at a time, over a period of 6 months, including all prototyping and hyperparameter search. Each WHAM/HAM model training took 5 to 7 days to complete, depending on the model settings.

[0085] Each WHAM/HAM model was trained for 20k optimizer updates. With batch size of 800 and sequence length of 10 (one step is a single image and action), this corresponds to training for 160M timesteps. This is bit over half of an epoch over the training dataset (284M timesteps total).

EXAMPLE SYSTEM

[0086] The present concepts can be implemented in various technical environments and on various devices. FIG. 10 shows an example system 1000 in which the present concepts can be employed, as discussed more below. As shown in FIG. 10, system 1000 includes a console 1010, a client device 1020, a model training server 1030, and a model execution server 1040. Console 1010, client device 1020, model training server 1030, and model execution server 1040 are connected over one or more networks 1050.

[0087] Certain components of the devices shown in FIG. 10 may be referred to herein by parenthetical reference numbers. For the purposes of the following description, the parenthetical (1) indicates an occurrence of a given component on console 1010, (2) indicates an occurrence of a given component on client device 1020, (3) indicates an occurrence on model training server 1030, and (4) indicates an occurrence on model execution server 1040. Unless identifying a

specific instance of a given component, this document will refer generally to the components without the parenthetical.

[0088] Generally, the devices shown in FIG. 10 may have respective processing resources 1001 and storage resources 1002, which are discussed in more detail below. The devices may also have various modules that function using the processing and storage resources to perform the techniques discussed herein, as discussed more below.

[0089] The model training server 1030 can include a model training module 1031, which is configured to train one or more machine learning models as described elsewhere herein. For instance, the model training module can train an encoder/decoder model to encode/decode application outputs such as images, and a generative model such as a transformer to predict outputs and inputs. The model training server can distribute the trained model(s) to other devices in system 1000, e.g., console 1010 and/or model execution server 1040. Generally speaking, larger models that implement complex application behavior may tend to be implemented remotely via cloud resources, e.g., by running on remote model execution module 1041 on model execution server 1040. Conversely, smaller models that implement limited application behavior, such as relatively simple applications or limited parts of an application, may tend to be implemented locally on devices such as console 1010.

[0090] Console 1010 can include a local model execution module 1011 and a control interface module 1012. The local model execution module can obtain one or more trained machine learning models from model training server 1030 and execute the machine learning model(s) locally on the console. The control interface module 1012 can obtain control inputs from controller 1013, which can include a controller circuit 1014 and a communication component 1015. The controller circuit can digitize inputs received by various controller mechanisms such as buttons or analog input mechanisms. The communication component can communicate the digitized inputs to the console over the local wireless link 1016. The control interface module on the console can obtain the digitized inputs and send them to the local model execution module or to the model execution server 1040.

[0091] Client device 1020 can have a model configuration module 1021. The model configuration module can be employed to configure any aspect of model training and/or execution. For instance, the model configuration module can provide training data or hyperparameters to the model training server 1030, seed images for initiating dreaming or gameplay sequences to the console 1010 or model execution server 1040, etc.

TRAINING DATA SELECTION

[0092] Generally speaking, machine learning models can be taught various concepts by being exposed to sufficient training examples prior to being employed for inference. Given a large model

with a sufficient number of training examples that show a wide range of concepts, it is possible to build a general-purpose generative model that learns generalized representations of inputs and outputs that can be employed effectively across a wider range of application scenarios. Thus, it can be useful to obtain training data for various scenarios, including outputs from many different types of applications and inputs provided by users with different abilities, demographic characteristics, etc.

[0093] For instance, consider video games. There are strategy games, shooting games, sports games, games where users control vehicles such as race cars or fighter planes, etc. In addition, there are games where users have a limited field of view, e.g., first person shooter games or a view out of the cockpit of a plane. There are other games where users see a top-down view of an entire playing surface, e.g., some soccer or football games. There are different ways that players can score points, get injured, heal damage, or obtain various in-game achievements. By exposing a generative model to a very wide range of games with different experiences, visuals, achievements, etc., a very general model can be developed. Then, the generative model will be able to generate plausible output and input sequences for a wide range of seed outputs.

[0094] In still further cases, a large generative model that has been trained using a wide range of training data from a variety of applications can be subsequently tuned to obtain a generative model that is adapted for specific types of games. For instance, a generative model could be trained on hundreds of games of various types, strategy, racing, fantasy, shooting, sports, etc. Then, that generative model could be further tuned using a specific subset of additional sports games to obtain a tuned generative sports model, tuned again on a specific subset of fantasy games to obtain a tuned generative fantasy model, etc. Then, seed outputs of example sports game scenarios could be input to the tuned generative sports model to implement various sports games, and seed outputs of example fantasy game scenarios could be input to the tuned generative fantasy model to implement various fantasy games.

[0095] In addition, user skill level, preferences, and other tendencies can vary widely. For instance, some users are very dedicated and skilled game players, whereas other users are novices. In addition, even two equally skilled game players may have very different tendencies, e.g., one might drive very aggressively and another might drive very carefully. By ensuring that the generative model sees sufficient training examples of varying user behaviors during training, the generative model can learn to generate input sequences that approximate those of a wider range of users.

PROTOTYPING USE CASE

[0096] The techniques described herein can be employed for a number of different use cases. For example, consider a rapid prototyping scenario where a developer wishes to evaluate how a

new game idea might look and feel. The developer can obtain a few seed images for various points in the game and input the seed images into a trained generative model. The generative model can generate new sequences of outputs and inputs to show how game play might proceed when humans play the game.

[0097] One way to generate outputs and inputs involves a random sampling approach from output distributions provided by the generative model. Thus, for example, the generative model could output a probability distribution of (token A == 0.7, token B == 0.2 token C == 0.1) for three future output or input tokens. A random number between 0 and 1 can be generated, and token A can be selected with a probability of 70%, token B with a probability of 20%, and token C with a probability of 10%. By executing the model several times, different generated gameplay sequences can be generated by the same model.

[0098] The developer can choose whether to keep or discard different sequences. For instance, referring back to FIG. 4A, assume the developer wishes to visualize how a racing game with a character riding a hoverboard might look. The trained generative model could output the sequences of outputs shown in FIGS. 4B-4D. If the developer likes the output, the developer could use those images for developing actual game code. If not, the developer could run the model again and generate a different sequence of images, e.g., perhaps with a sharper turn, fewer trees, a larger map inset, etc.

[0099] The developer could also choose to modify the seed image to obtain different generated sequences. For example, the developer might decide that a skateboard might be more realistic than a hoverboard, and replace the hoverboard shown in FIG. 4A with a skateboard. The generative model could then generate one or more new generated sequences where the character is riding a skateboard instead of a hoverboard. Or the developer could decide to use a narrower road, a different type of background (e.g., city buildings, etc.), insert an obstacle such as a boulder or another character on the road, etc. By modifying the seed image in this way, the developer can see how different sequences might play out for actual users if the game is implemented as generated by the model.

APPLICATION EXECUTION USE CASE

[00100] In further implementations, one or more seed outputs can be employed to cause the trained generative model to replace part or all of a hand-coded application. For instance, as noted above, inputs generated by the model can be discarded, and instead inputs received from an actual user can be used by the model to predict future output sequences. Thus, the generative model itself can serve directly as the application.

[00101] In some cases, the generative model can serve as an entire application, e.g., all application output is generated by the model. In other cases, the generative model can implement

some of the application functionality and other parts of an application can be hand-coded. For instance, consider a scenario where a client-side device is relatively resource constrained and can execute a relatively small generative model. The generative model might perform well for about 100 milliseconds of predicting application output, but thereafter might tend to diverge substantially from realistic outputs. Such a generative model could still be useful to address issues such as network disruptions.

[00102] Referring back to FIG. 10, a small generative model could be provided to the console 1010. During normal network conditions, the remote model execution module 1041 could receive user inputs from the console over network 1050, and provide application output to the console over the network. When a network disruption is detected that delays one or more packets longer than a threshold (e.g., 100 milliseconds), the local model execution module 1011 on the console can temporarily take over and provide output to the user using the small generative model.

[00103] As another example, certain portions of a given application could be executed within a conventional game engine or hand-coded logic, while other portions are executed using a trained generative model. For instance, a trained generative model could be employed for actually racing on a racetrack during a racing game, but the initial countdown to starting the race and maintaining of a leaderboard with race results could be implemented within a game engine or using hand-coded logic. As another example, a flight simulator might use a trained generative model to control the flight pattern of a virtual aircraft, but weapons functionality could be implemented in an application engine to give the developer full control over weapons functionality.

[00104] There are various techniques that could be employed to obtain a smaller generative model suitable for client-side execution. For instance, a full-scale generative model that generalizes to a wide range of games could be trained and deployed on model execution server 1040. That model could be pruned to remove nodes or weights that do not significantly contribute to model performance for specific client-side scenarios. For instance, pruning could be performed on an application-specific basis, e.g., removing nodes or weights that do not significantly affect performance of the model for a specific application. Alternatively or in addition, distillation techniques could be employed to teach a smaller student generative model to learn from the output distribution of a larger teacher model over a limited range of scenarios. Generative models obtained via pruning or distillation may offer adequate performance for limited (e.g., short-duration) client-side execution scenarios. As yet another example, for applications with relatively simple characteristics, e.g., simply logic rules, relatively simple graphics, etc., a smaller stand-alone generative model can be trained for scratch specifically for that application.

FURTHER IMPLEMENTATIONS

[00105] In addition to video output, other types of application output can also be generated. For

instance, audio and/or haptic output can be encoded using an encoder/decoder model, and corresponding audio or haptic output tokens can be used to train a generative model to predict audio or haptic output by an application.

[00106] In other cases, a generative model can be trained using tokens representing individual users. Thus, the generative model can learn how different types of users have different abilities, tendencies, preferences, etc. When the trained model is executed for a new user, a token representing that user can be learned. Then, the outputs (and inputs) generated by the generative model can be conditioned not only on the previous outputs and inputs, but also the token representing the user that is playing the game. Thus, that user can receive an experience that is tailored to their own characteristics or preferences.

[00107] In addition, note that the encoder/decoder and generative model architectures and training techniques described above are examples, and other types of models and/or training techniques can also be employed. For instance, token prediction training can be performed bidirectionally, e.g., by masking out individual output or input tokens from a training sequence and training the generative model to predict both preceding and subsequent tokens. As another example, a long short-term memory model can be employed instead of a transformer for generating inputs and outputs. Still further, an encoder/decoder model approach could be employed for representing user inputs instead of directly mapping input mechanisms to specific bits or other values, as discussed previously.

[00108] In addition, note that some implementations may employ a text-to-image synthesis model to generate seed outputs. For instance, instead of feeding a model a picture of a troll fighting a dragon, a developer could ask a text-to-image synthesis model to automatically generate a picture of a troll fighting a dragon. The image produced by the text-to-image synthesis model could be used as a seed input for prototyping or direct gameplay.

[00109] In some cases, the developer could provide individual game checkpoints via example outputs. For instance, the developer could provide an image of a character at a first location on a path through the woods, a second image of the character on a boat, a third image of the character on an airplane, and so on. Then, the developer could have the generative model generate a sequence for each example image, so that the gameplay experience for the user would involve traveling through the woods, getting on a boat, and then getting on an airplane.

[00110] In addition, note that the disclosed techniques can also be employed for augmented or virtual reality experiences. Consider a generative model that is trained on a videoconferencing application of participants co-located in conference rooms and also of remote participants working from home. A trained generative model could be employed to patch together images from multiple home offices to create an experience that appears as if the users are all co-located in the same

conference room.

TECHNICAL EFFECT

[00111] As noted above, the disclosed techniques can be employed to train a generative model that jointly learns to predict application outputs and user inputs. As a consequence, the trained generative model can generate plausible sequences of outputs and/or inputs for various use cases, such as rapid prototyping or executing the generative model directly as an application.

[00112] Prior techniques such as behavioral cloning or world modeling have limitations that are overcome by the present techniques. For instance, while behavioral cloning techniques can reasonably approximate human inputs to an application, the application itself needs to be developed to train and utilize these techniques. Conversely, while world modeling can predict how an application might behave in response to user inputs, the user inputs need to be obtained separately.

[00113] Because the disclosed techniques can generate both application output and user inputs, the disclosed techniques can generate future trajectories without a separate application or source of user inputs. Moreover, unlike some prior predictive techniques that require internal state information such as the location of different characters, the disclosed techniques can be implemented without accessing any internal application state.

[00114] In addition, as noted previously, some implementations can train the encoder/decoder and generative models separately. This is more efficient than joint training and can save a great deal of processor and/or memory resources that would otherwise be expended during joint training.

EXAMPLE TRAINING METHOD

[00115] FIG. 11 illustrates an example method 1100 that can be used to train a model to predict application outputs and inputs, consistent with the present concepts. As discussed elsewhere herein, method 1100 can be implemented on many different types of devices, e.g., by one or more cloud servers, by a client device such as a laptop, tablet, or smartphone, or by combinations of one or more servers, client devices, etc.

[00116] Method 1100 begins at block 1102, where training data is accessed. For instance, the training data can include training sequences of images or other application outputs of the one or more applications and inputs provided to the one or more applications during the one or more executions. In some cases, the applications are interactive applications, such as video games, flight simulators, etc.

[00117] Method 1100 continues at block 1104, where the images are mapped to training image tokens. For instance, a trained image encoder/decoder can be employed to map the images to tokens, e.g., embeddings in a vector space. Other types of application output (e.g., audio or haptic)

in the training sequences can also be mapped to corresponding tokens. In addition, inputs in the training sequences can be mapped to training input tokens, e.g., by representing different input mechanisms as different bits or other values in the training input tokens.

[00118] Method 1100 continues at block 1106, where a generative machine learning model (such as a transformer-based neural dreaming model) is trained to predict the training image tokens and the training input tokens that are obtained from the training sequences. For instance, in some cases, the generative machine learning model is a transformer that is trained to predict sequences of tokens representing the inputs and outputs. Block 1104 can also involve separate or joint training of an encoder/decoder model to generate tokens representing the outputs.

[00119] Method 1100 continues at block 1108, where the trained generative machine learning model is output. For instance, the generative model can be output to storage, shared in memory with another process or thread, or sent over a network to a separate device for later execution.

EXAMPLE INFERENCE METHOD

[00120] FIG. 12 illustrates an example method 1200 that can be used to generate application outputs and inputs with a trained generative machine learning model, consistent with the present concepts. As discussed elsewhere herein, method 1200 can be implemented on many different types of devices, e.g., by one or more cloud servers, by a client device such as a laptop, tablet, or smartphone, or by combinations of one or more servers, client devices, etc.

[00121] Method 1200 begins at block 1202, where a seed image is obtained. For instance, the seed image can be an image representing a starting point for an application scenario, e.g., a seeded application state. Other types of seed output (e.g., audio or haptic) can also be obtained at block 1202. In some cases, a seed input representing a user input responsive to the seed image can also be obtained at block 1202.

[00122] Method 1200 continues at block 1204, where the seed image is mapped to at least one seed image token using an encoder. For instance, as noted above, the encoder can be part of an image encoder/decoder that has been trained to represent images in vector space using training images. The encoder can have been trained using reconstruction loss. Other types of seed output obtained at block 1202 can also be mapped into corresponding tokens.

[00123] Method 1200 continues at block 1206, where the at least one seed image token is input as a prompt to a generative machine learning model, such as a neural dreaming model. For instance, the generative machine learning model can have previously been trained to predict application outputs (e.g., image, audio, and/or haptic) and inputs that are present in training sequences obtained from one or more executions of one or more applications. For instance, the generative machine learning model can include one or more transformer decoder blocks.

[00124] Method 1200 continues at block 1208, where subsequent image tokens are generated.

For instance, the subsequent image tokens can be sampled randomly from an output distribution of the generative machine learning model. Audio and/or haptic tokens can also be generated at block 1208. In some cases, input tokens are also selected from the output distribution. In other cases, inputs are received from a user and the output distribution of the model for inputs is discarded.

[00125] Method 1200 continues at block 1210, where the subsequent image tokens are decoded using an image decoder, e.g., from the image encoder/decoder employed at block 1204. The decoded image tokens can be images in an image space. Audio and/or haptic tokens can also be decoded at block 1208.

[00126] Method 1200 continues at block 1212, where the subsequent images are displayed. Decoded audio and/or haptic tokens can also be output at block 1210, e.g. by playing audio over a speaker and/or causing a video game controller or other device to generate haptic feedback.

DEVICE IMPLEMENTATIONS

[00127] As noted above with respect to FIG. 10, system 1000 includes several devices, including a console 1010, a controller 1013, a client device 1020, a model training server 1030, and a model execution server 1040. As also noted, not all device implementations can be illustrated, and other device implementations should be apparent to the skilled artisan from the description above and below.

[00128] The term “device,” "computer," "computing device," “client device,” and or “server device” as used herein can mean any type of device that has some amount of hardware processing capability and/or hardware storage/memory capability. Processing capability can be provided by one or more hardware processors (e.g., hardware processing units/cores) that can execute data in the form of computer-readable instructions to provide functionality. Computer-readable instructions and/or data can be stored on storage, such as storage/memory and or the datastore. The term “system” as used herein can refer to a single device, multiple devices, etc.

[00129] Storage resources can be internal or external to the respective devices with which they are associated. The storage resources can include any one or more of volatile or non-volatile memory, hard drives, flash storage devices, and/or optical storage devices (e.g., CDs, DVDs, etc.), among others. As used herein, the term "computer-readable media" can include signals. In contrast, the term "computer-readable storage media" excludes signals. Computer-readable storage media includes "computer-readable storage devices." Examples of computer-readable storage devices include volatile storage media, such as RAM, and non-volatile storage media, such as hard drives, optical discs, and flash memory, among others.

[00130] In some cases, the devices are configured with a general purpose hardware processor and storage resources. In other cases, a device can include a system on a chip (SOC) type design.

In SOC design implementations, functionality provided by the device can be integrated on a single SOC or multiple coupled SOCs. One or more associated processors can be configured to coordinate with shared resources, such as memory, storage, etc., and/or one or more dedicated resources, such as hardware blocks configured to perform certain specific functionality. Thus, the term "processor," "hardware processor" or "hardware processing unit" as used herein can also refer to central processing units (CPUs), graphical processing units (GPUs), controllers, microcontrollers, processor cores, or other types of processing devices suitable for implementation both in conventional computing architectures as well as SOC designs.

[00131] Alternatively, or in addition, the functionality described herein can be performed, at least in part, by one or more hardware logic components. For example, and without limitation, illustrative types of hardware logic components that can be used include Field-programmable Gate Arrays (FPGAs), Application-specific Integrated Circuits (ASICs), Application-specific Standard Products (ASSPs), System-on-a-chip systems (SOCs), Complex Programmable Logic Devices (CPLDs), etc.

[00132] In some configurations, any of the modules/code discussed herein can be implemented in software, hardware, and/or firmware. In any case, the modules/code can be provided during manufacture of the device or by an intermediary that prepares the device for sale to the end user. In other instances, the end user may install these modules/code later, such as by downloading executable code and installing the executable code on the corresponding device.

[00133] Also note that devices generally can have input and/or output functionality. For example, computing devices can have various input mechanisms such as keyboards, mice, touchpads, voice recognition, gesture recognition (e.g., using depth cameras such as stereoscopic or time-of-flight camera systems, infrared camera systems, RGB camera systems or using accelerometers/gyroscopes, facial recognition, etc.). Devices can also have various output mechanisms such as printers, monitors, etc.

[00134] Also note that the devices described herein can function in a stand-alone or cooperative manner to implement the described techniques. For example, the methods and functionality described herein can be performed on a single computing device and/or distributed across multiple computing devices that communicate over network(s) 1050. Without limitation, network(s) 1050 can include one or more local area networks (LANs), wide area networks (WANs), the Internet, and the like.

ADDITIONAL EXAMPLES

[00135] Various examples are described above. Additional examples are described below. One example includes a method comprising obtaining a seed image representing a seeded application state, mapping the seed image to at least one seed image token using an image encoder, inputting

the at least one seed image token as a prompt to a neural dreaming model that has been trained to predict training sequences obtained from one or more executions of one or more applications, the training sequences including images output by the one or more applications during the one or more executions and inputs to the one or more applications during the one or more executions, generating subsequent image tokens with the neural dreaming model, and decoding the subsequent image tokens with an image decoder to obtain subsequent images.

[00136] Another example can include any of the above and/or below examples where the neural dreaming model comprises a transformer decoder.

[00137] Another example can include any of the above and/or below examples where the neural dreaming model is a multi-modal model and the generating also involves generating subsequent input tokens.

[00138] Another example can include any of the above and/or below examples where the method further comprises sequentially generating further subsequent image tokens and further subsequent input tokens with the neural dreaming model conditioned on previously-generated image tokens and previously-generated input tokens.

[00139] Another example can include any of the above and/or below examples where the method further comprises receiving actual user input tokens representing actual user inputs, inputting the actual user input tokens to the neural dreaming model, and generating the subsequent image tokens based at least on the actual user inputs.

[00140] Another example can include any of the above and/or below examples where the method further comprises sequentially generating further subsequent image tokens with the neural dreaming model conditioned on previously-generated image tokens and previously-received actual user input tokens.

[00141] Another example can include any of the above and/or below examples where the neural dreaming model has been trained using token prediction loss when predicting image tokens and input tokens from the training sequences.

[00142] Another example can include any of the above and/or below examples where the image encoder and the image decoder have been trained using reconstruction loss from the images in the training sequences.

[00143] Another example can include any of the above and/or below examples where the method further comprises displaying the subsequent images.

[00144] Another example can include a system comprising a hardware processing unit and a storage resource storing computer-readable instructions which, when executed by the hardware processing unit, cause the hardware processing unit to obtain a seed image representing a seeded application state, map the seed image to at least one seed image token, input the at least one seed

image token as a prompt to a generative model that has been trained to predict image tokens and input tokens of training sequences obtained from one or more executions of one or more applications, and generate subsequent image tokens with the generative model.

[00145] Another example can include any of the above and/or below examples where the seed image represents output by a video game that is at least partially implemented by the generative model.

[00146] Another example can include any of the above and/or below examples where the generative model has been trained to predict images output by video games and video game controller inputs that are present in the training sequences.

[00147] Another example can include any of the above and/or below examples where the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to map the seed image to the at least one seed image token using an image encoder.

[00148] Another example can include any of the above and/or below examples where the image encoder having been trained using reconstruction loss from the image tokens in the training sequences and the generative model having been trained using token prediction loss when predicting the image tokens and the input tokens in the training sequences.

[00149] Another example can include any of the above and/or below examples where the input tokens obtained from the training sequences have values representing different input mechanisms of a video game controller.

[00150] Another example can include any of the above and/or below examples where the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to generate future image tokens and future input tokens given past image tokens produced by the generative model and past input tokens produced by the generative model.

[00151] Another example can include any of the above and/or below examples where the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to generate future image tokens given past image tokens produced by the generative model and actual user inputs received from a video game controller.

[00152] Another example can include any of the above and/or below examples where the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to receive a natural language description of an application scenario and generate the seed image from the natural language description using a text-to-image synthesis model.

[00153] Another example can include a computer-readable storage medium storing computer-readable instructions which, when executed by a hardware processing unit, cause the hardware

processing unit to perform acts comprising accessing training data reflecting one or more executions of one or more applications, the training data including training sequences of images output by the one or more applications and inputs provided to the one or more applications during the one or more executions, mapping the images to training image tokens and the inputs to training input tokens, training a generative model to predict the training image tokens and the training input tokens sequentially according to the training sequences, and outputting the trained generative model.

[00154] Another example can include any of the above and/or below examples where the acts further comprise training an image encoder/decoder to map the images in the training sequences to the training image tokens using reconstruction loss and training the generative model using next token prediction loss for the training image tokens and the training input tokens.

[00155] Another example can include a method comprising accessing training data reflecting one or more executions of one or more applications, the training data including training sequences of application outputs of the one or more applications and inputs provided to the one or more applications during the one or more executions, training a predictive model to predict the application outputs and the inputs that are present in the training sequences, and outputting the trained predictive model.

[00156] Another example can include any of the above and/or below examples where the training comprises sequentially predicting future application outputs and future inputs given past application outputs and past inputs that are present in the training sequences.

[00157] Another example can include any of the above and/or below examples where the predictive model comprises a neural network.

[00158] Another example can include any of the above and/or below examples where the predictive model comprises a transformer.

[00159] Another example can include any of the above and/or below examples where the method further comprises determining past output tokens representing past application outputs in the training sequences and past input tokens representing past inputs in the training sequences and training the predictive model to predict future application output tokens and future input tokens given the past output tokens and the past input tokens.

[00160] Another example can include any of the above and/or below examples where the application outputs include images output by the one or more applications during the one or more executions.

[00161] Another example can include any of the above and/or below examples where the method further comprises mapping the images into the past output tokens using another machine learning model.

[00162] Another example can include any of the above and/or below examples where the another machine learning model comprises an image encoder/decoder.

[00163] Another example can include any of the above and/or below examples where the method further comprises training the image encoder/decoder using reconstruction loss for the images.

[00164] Another example can include any of the above and/or below examples where the application is a video game.

[00165] Another example can include any of the above and/or below examples where the inputs in the training sequences are provided by a video game controller.

[00166] Another example can include any of the above and/or below examples where the method further comprises representing respective input mechanisms of the video game controller as corresponding bits in the past input tokens and the future input tokens.

[00167] Another example can include a system comprising a hardware processing unit and a storage resource storing computer-readable instructions which, when executed by the hardware processing unit, cause the hardware processing unit to obtain a seed application output, input the seed application output to a predictive model that has been trained to predict application outputs and inputs that are present in training sequences obtained from one or more executions of one or more applications, and generate subsequent application outputs with the predictive model.

[00168] Another example can include any of the above and/or below examples where the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to, starting from the seed application output, sequentially generate the subsequent application outputs and subsequent application inputs with the predictive model.

[00169] Another example can include any of the above and/or below examples where the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to sequentially generate further subsequent application outputs and further subsequent application inputs with the predictive model, conditioned on previously-generated application outputs and previously-generated application inputs.

[00170] Another example can include any of the above and/or below examples where the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to receive actual user inputs, input the actual user inputs to the predictive model, and generate the subsequent application outputs based at least on the actual user inputs.

[00171] Another example can include any of the above and/or below examples where the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to sequentially generate further subsequent application outputs with the predictive model conditioned on previously-generated application outputs and previously-

received actual user inputs.

[00172] Another example can include any of the above and/or below examples where the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to receive a natural language description of an application scenario and generate the seed application output from the natural language description using a text-to-image synthesis model.

[00173] Another example can include any of the above and/or below examples where the subsequent application outputs comprise video, audio, or haptic output.

[00174] Another example can include a computer-readable storage medium storing computer-readable instructions which, when executed by a hardware processing unit, cause the hardware processing unit to perform acts comprising obtaining a seed image, mapping the seed image to a seed image token using an encoder, inputting the seed image token to a transformer that has been trained to predict training sequences obtained from one or more executions of one or more applications, the training sequences including images output by the one more applications during the one or more executions and inputs to the one or more applications during the one or more executions, generating subsequent image tokens and subsequent input tokens with the transformer, and decoding the subsequent application image tokens with a decoder to obtain subsequent output images.

CONCLUSION

[00175] Although the subject matter has been described in language specific to structural features and/or methodological acts, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or acts described above. Rather, the specific features and acts described above are disclosed as example forms of implementing the claims and other features and acts that would be recognized by one skilled in the art are intended to be within the scope of the claims.

CLAIMS

1. A method (1200) comprising:

obtaining (1202) a seed image representing a seeded application state;

mapping (1204) the seed image to at least one seed image token using an image encoder;

inputting (1206) the at least one seed image token as a prompt to a neural dreaming model that has been trained to predict training sequences obtained from one or more executions of one or more applications, the training sequences including images output by the one or more applications during the one or more executions and inputs to the one or more applications during the one or more executions;

generating (1208) subsequent image tokens with the neural dreaming model; and

decoding (1210) the subsequent image tokens with an image decoder to obtain subsequent images.

2. The method of claim 1, wherein the neural dreaming model comprises a transformer decoder.

3. The method of claim 1, wherein the neural dreaming model is a multi-modal model and the generating also involves generating subsequent input tokens.

4. The method of one of claims 1-3, further comprising:

sequentially generating further subsequent image tokens and further subsequent input tokens with the neural dreaming model conditioned on previously-generated image tokens and previously-generated input tokens.

5. The method of claim 1, further comprising:

receiving actual user input tokens representing actual user inputs;

inputting the actual user input tokens to the neural dreaming model; and

generating the subsequent image tokens based at least on the actual user inputs.

6. The method of claim 5, further comprising:

sequentially generating further subsequent image tokens with the neural dreaming model conditioned on previously-generated image tokens and previously-received actual user input tokens.

7. The method of claim 1, wherein the neural dreaming model has been trained using token prediction loss when predicting image tokens and input tokens from the training sequences.

8. The method of claim 1, wherein the image encoder and the image decoder have been trained using reconstruction loss from the images in the training sequences.

9. The method of claim 1, further comprising displaying the subsequent images.

10. A system(1000, 1010, 1040) comprising:

a hardware processing unit (1001); and

a storage resource (1002) storing computer-readable instructions which, when executed by the hardware processing unit, cause the hardware processing unit to:

obtain a seed image representing a seeded application state;

map the seed image to at least one seed image token;

input the at least one seed image token as a prompt to a generative model that has been trained to predict image tokens and input tokens of training sequences obtained from one or more executions of one or more applications; and

generate subsequent image tokens with the generative model.

11. The system of claim 10, wherein the seed image represents output by a video game that is at least partially implemented by the generative model.

12. The system of claim 11, wherein the generative model has been trained to predict images output by video games and video game controller inputs that are present in the training sequences.

13. The system of claim 12, wherein the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to:

map the seed image to the at least one seed image token using an image encoder.

14. The system of claim 13, the image encoder having been trained using reconstruction loss from the image tokens in the training sequences and the generative model having been trained using token prediction loss when predicting the image tokens and the input tokens in the training sequences.

15. The system of claim 14, the input tokens obtained from the training sequences having values representing different input mechanisms of a video game controller.

16. The system of one of claims 12-15, wherein the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to:

generate future image tokens and future input tokens given past image tokens produced by the generative model and past input tokens produced by the generative model.

17. The system of claim 12, wherein the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to:

generate future image tokens given past image tokens produced by the generative model and actual user inputs received from a video game controller.

18. The system of claim 10, wherein the computer-readable instructions, when executed by the hardware processing unit, cause the hardware processing unit to:

receive a natural language description of an application scenario; and

generate the seed image from the natural language description using a text-to-image synthesis model.

19. A computer-readable storage medium (1002) storing computer-readable instructions

which, when executed by a hardware processing unit (1001), cause the hardware processing unit to perform acts comprising:

accessing training data reflecting one or more executions of one or more applications, the training data including training sequences of images output by the one or more applications and inputs provided to the one or more applications during the one or more executions;

mapping the images to training image tokens and the inputs to training input tokens;

training a generative model to predict the training image tokens and the training input tokens sequentially according to the training sequences; and

outputting the trained generative model.

20. The computer-readable storage medium claim 19, the acts further comprising:

training an image encoder/decoder to map the images in the training sequences to the training image tokens using reconstruction loss; and

training the generative model using next token prediction loss for the training image tokens and the training input tokens.

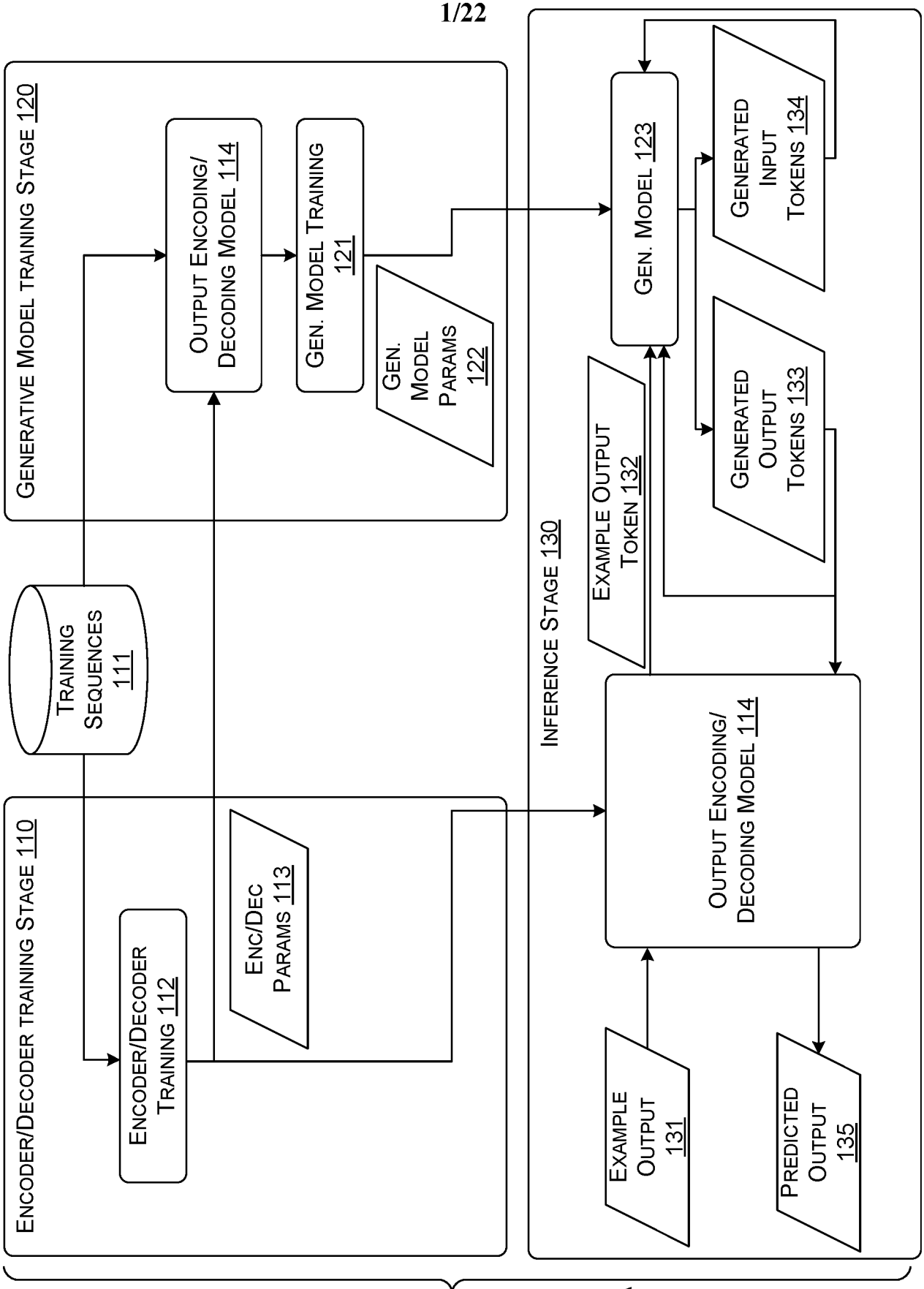

Overall Workflow 100
Encoder/Decoder training Stage 110
Encoder/Decoder Training 112
Enc/Dec Params 113
Training Sequences 111
Generative Model training Stage 120
Output Encoding/ Decoding Model 114
Gen. Model Training 121
Gen. Model Params 122
Inference Stage 130
Gen. Model 123
Example Output Token 132
Output Encoding/ Decoding Model 114
Generated Output Tokens 133
Generated Input Tokens 134
Example Output 131
Predicted Output 135

FIG. 1A

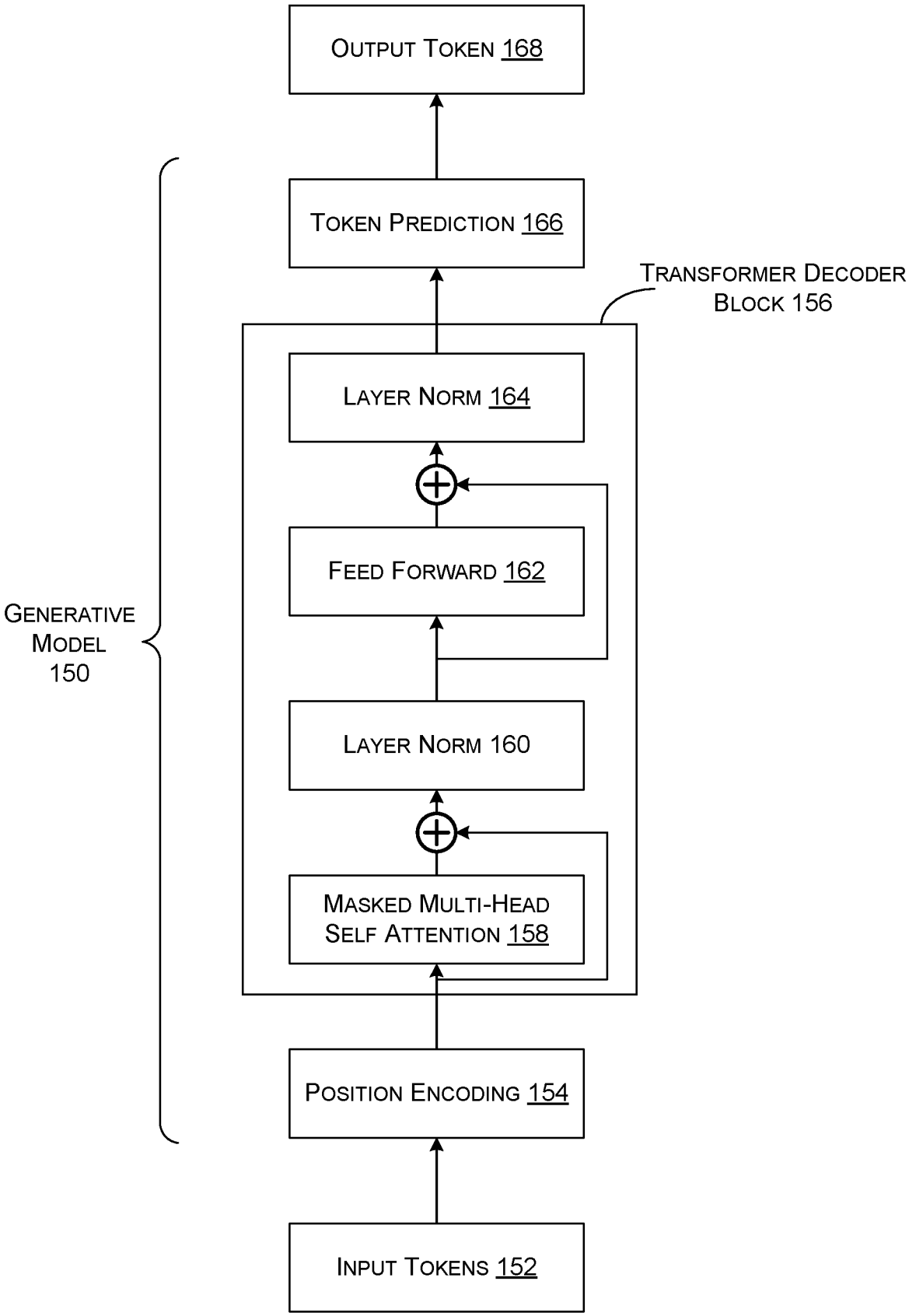

OUTPUT TOKEN 168
TOKEN PREDICTION 166
TRANSFORMER DECODER BLOCK 156
LAYER NORM 164
FEED FORWARD 162
LAYER NORM 160
MASKED MULTI-HEAD SELF ATTENTION 158
POSITION ENCODING 154
INPUT TOKENS 152
GENERATIVE MODEL 150

FIG. 1B

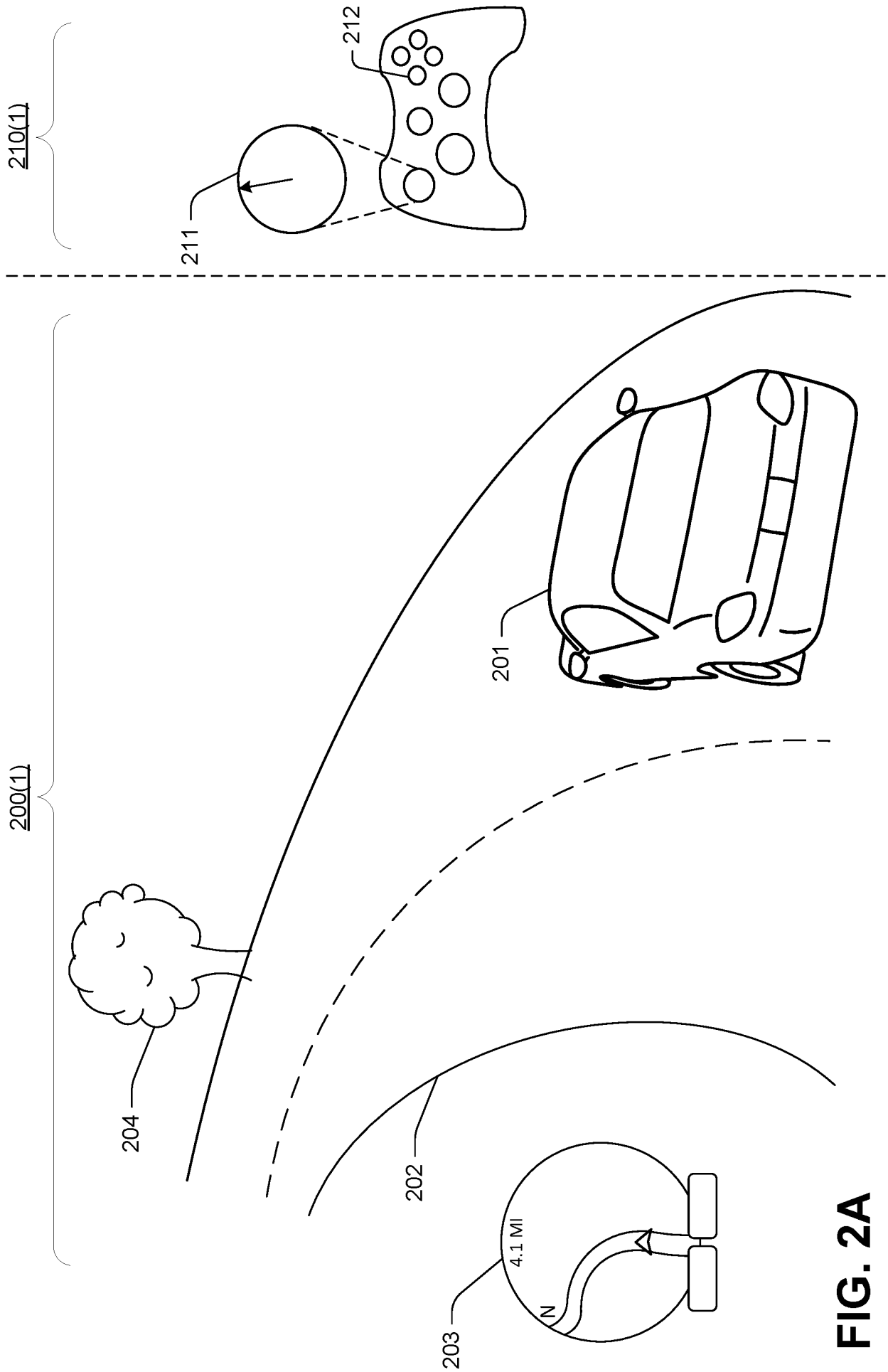


FIG. 2A

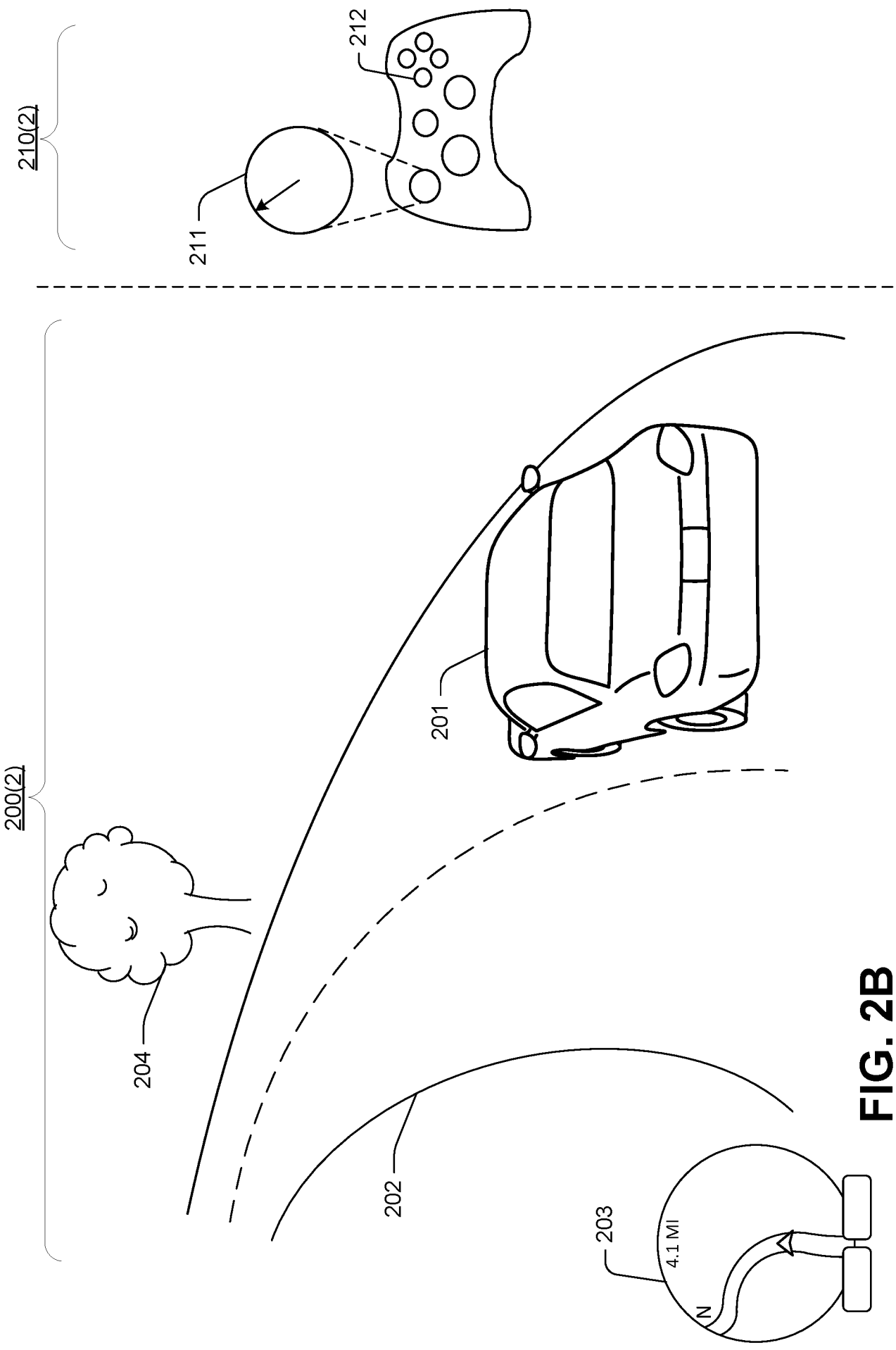


FIG. 2B

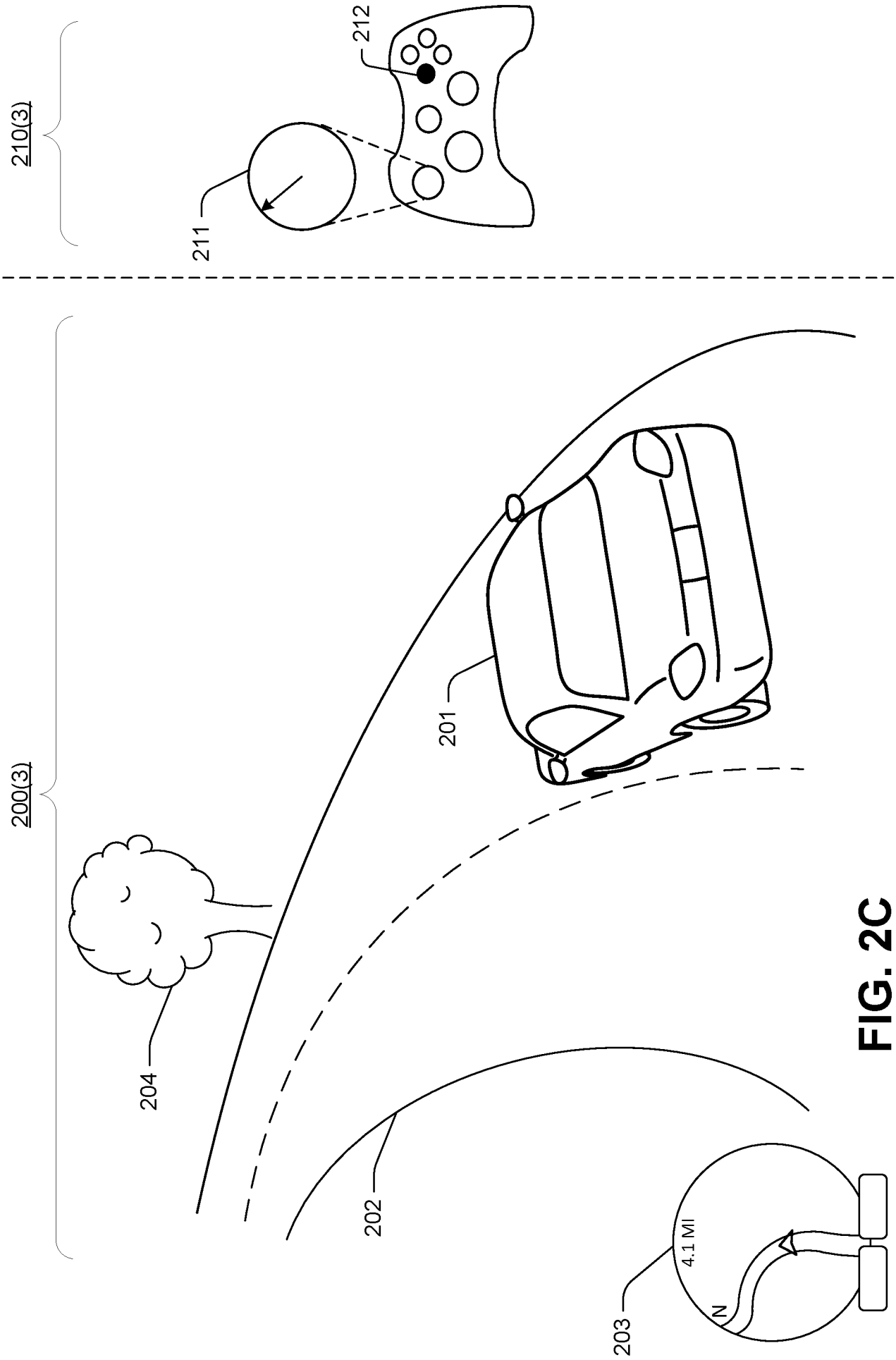


FIG. 2C

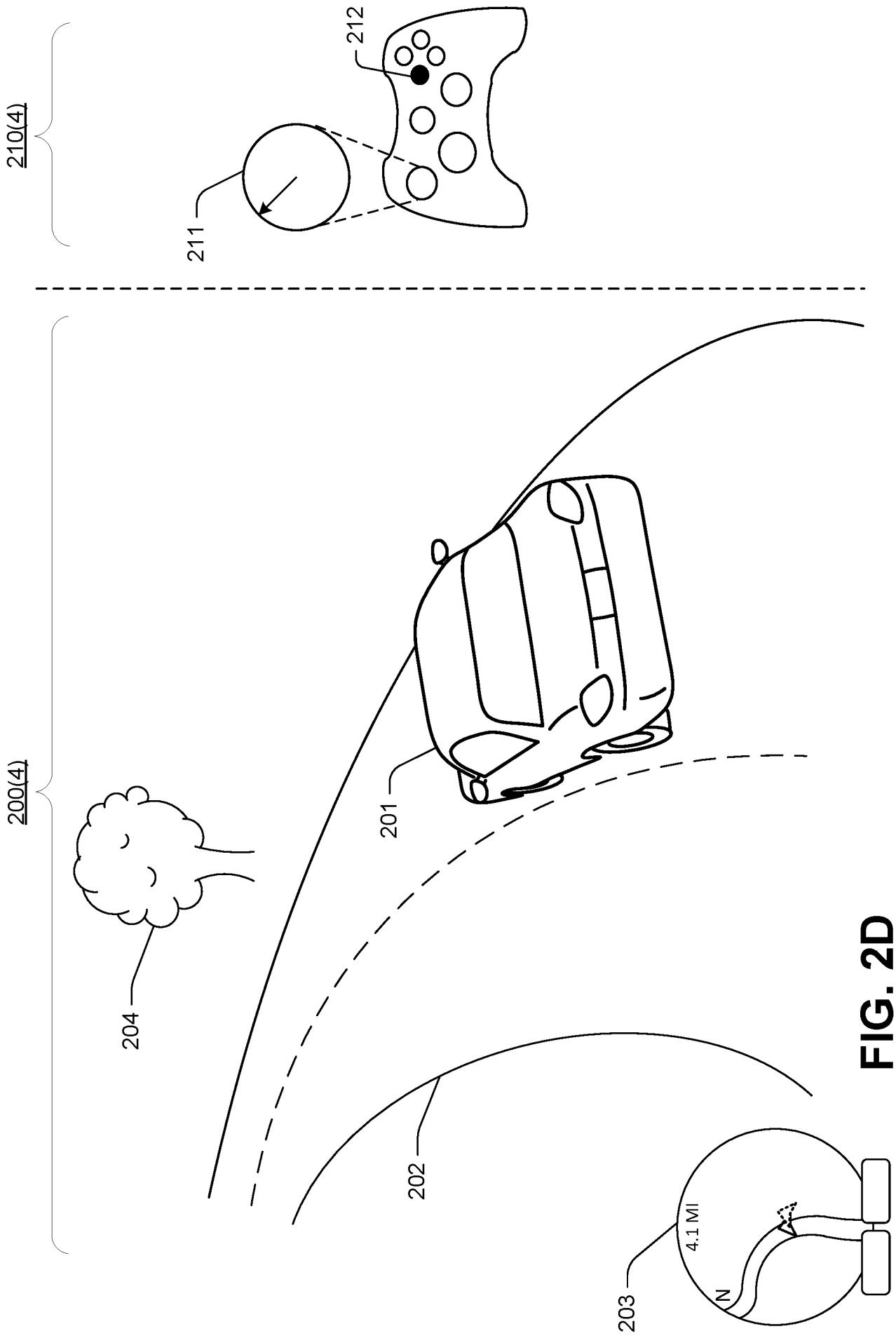


FIG. 2D

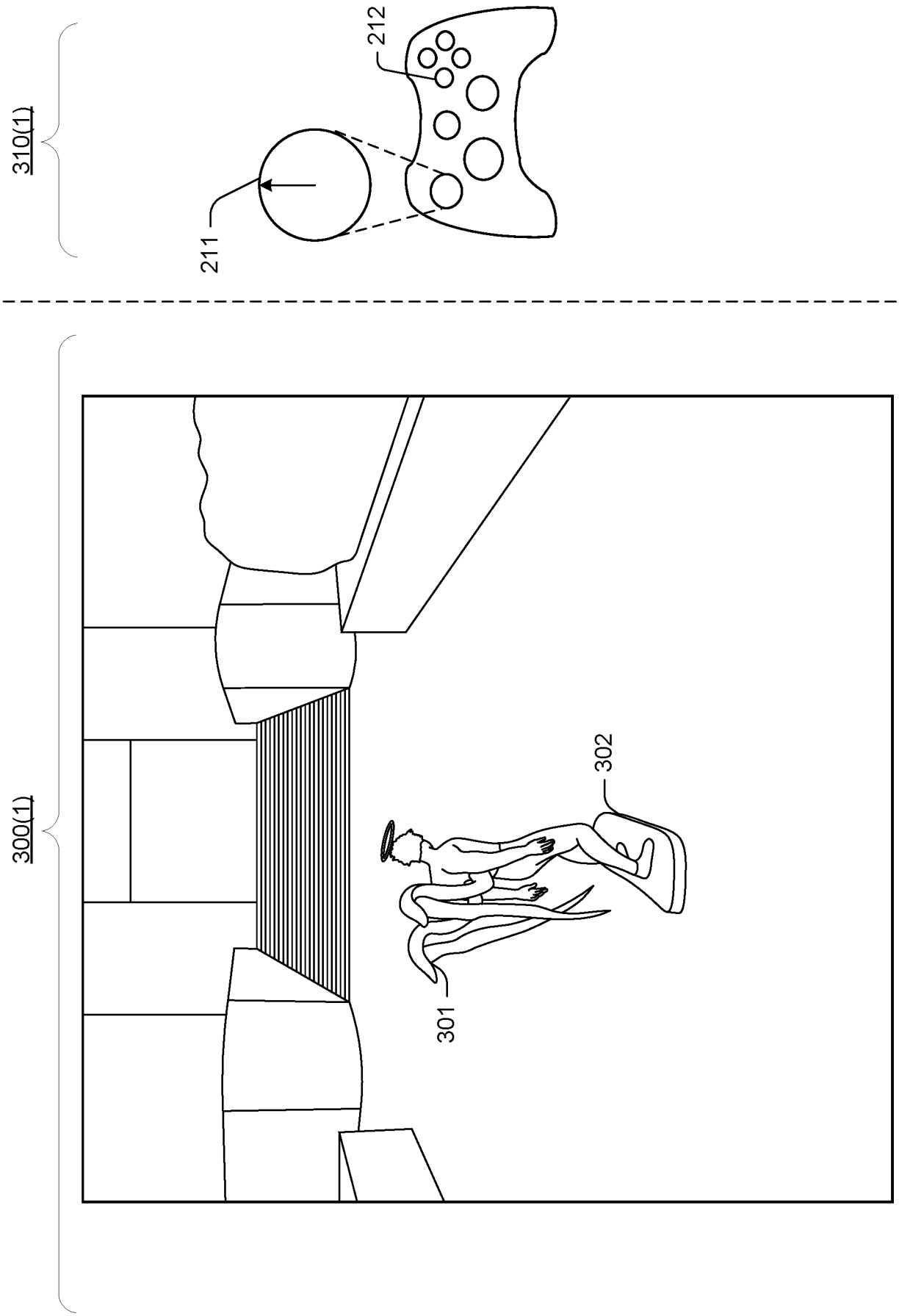
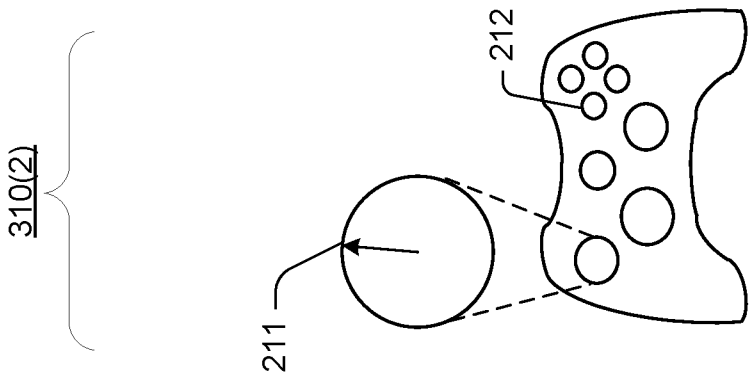


FIG. 3A



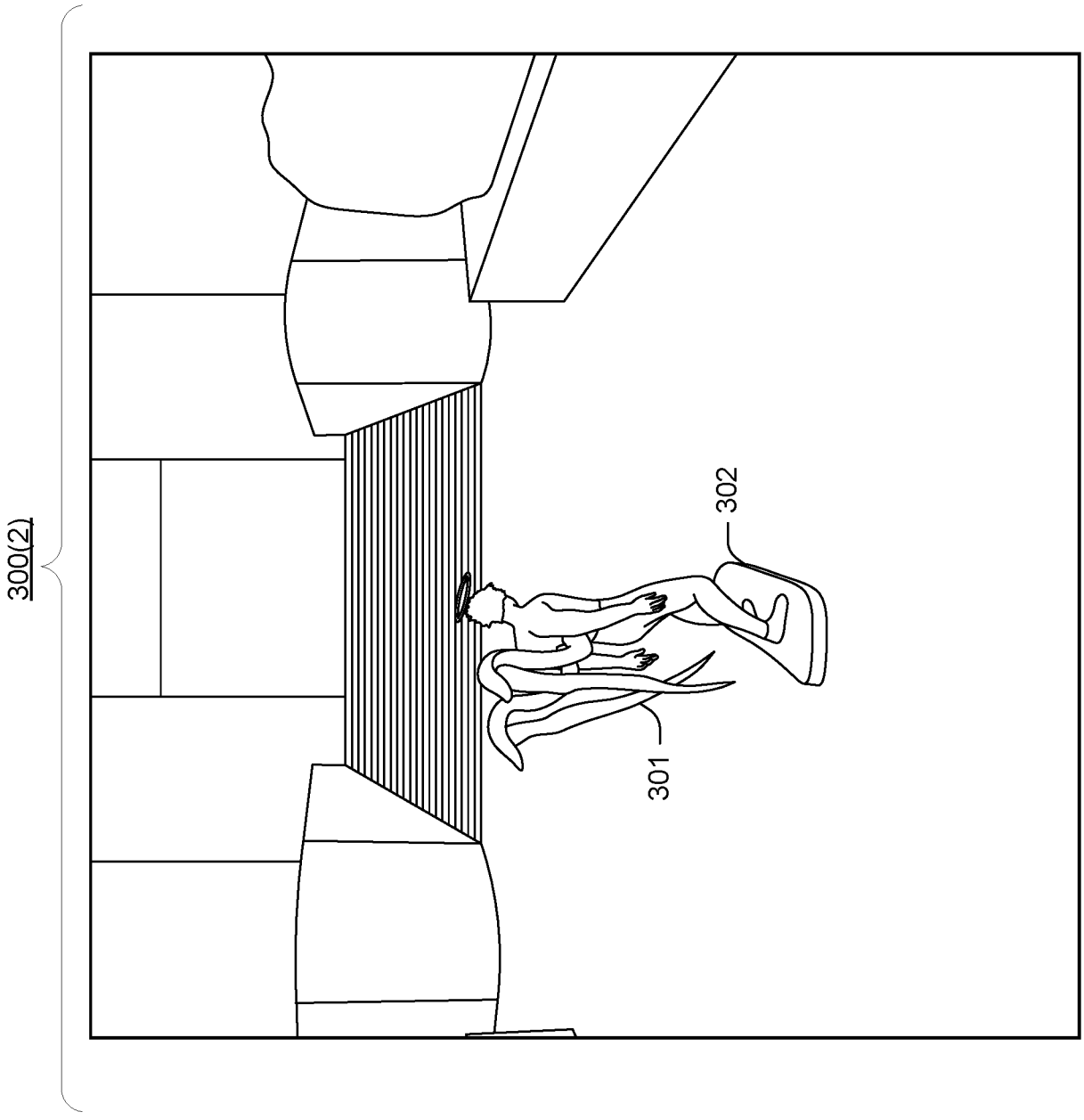
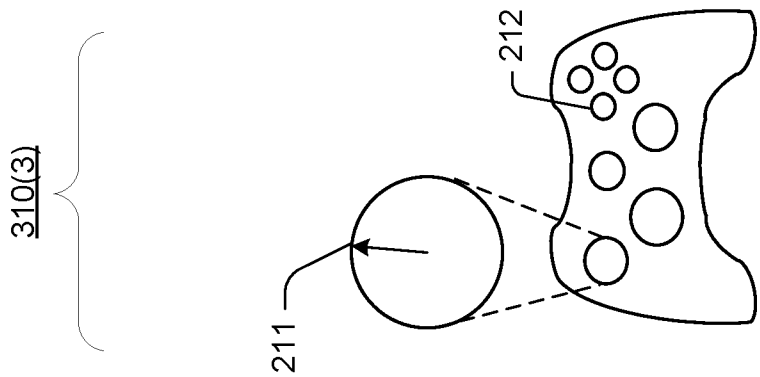


FIG. 3B



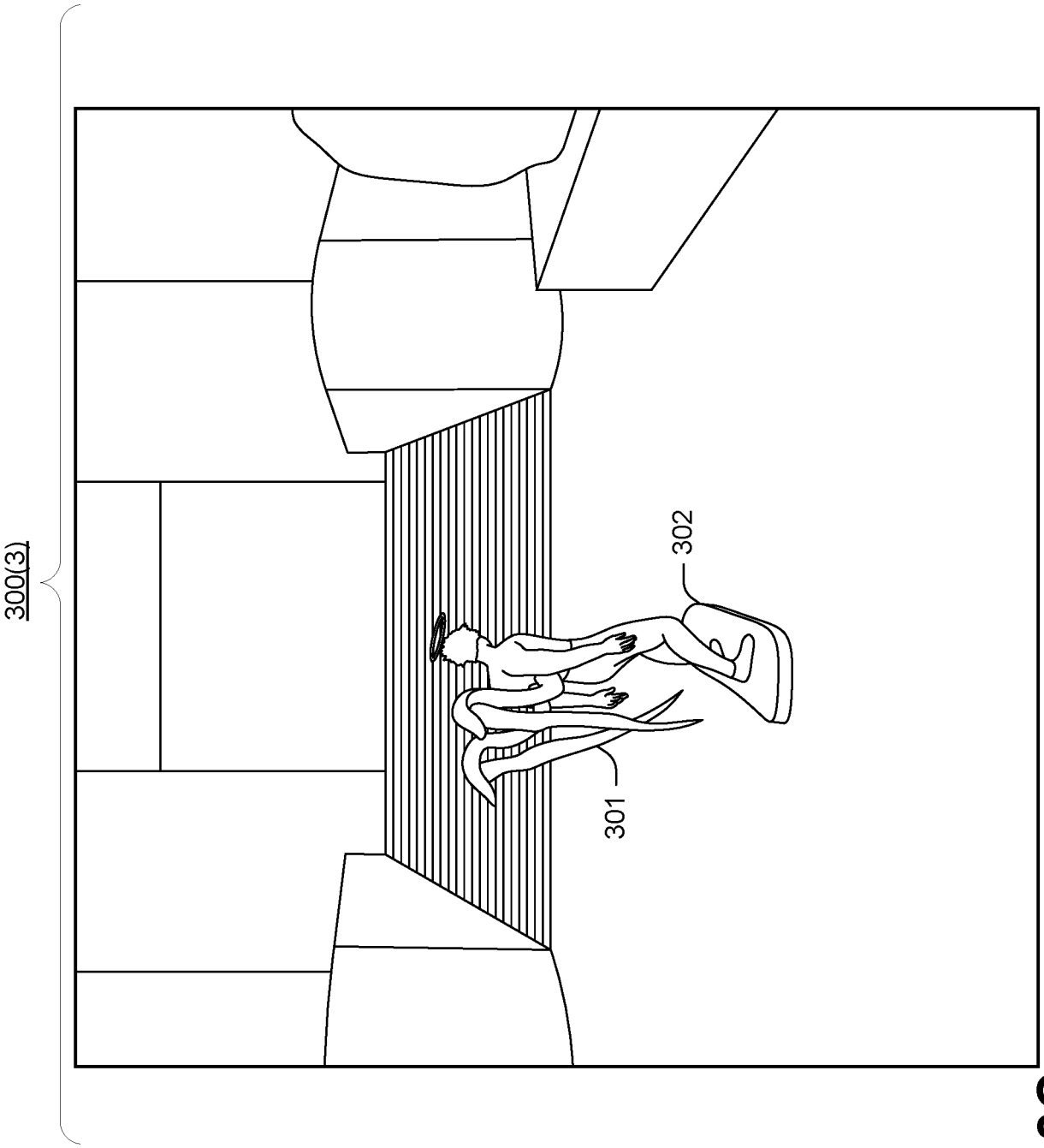
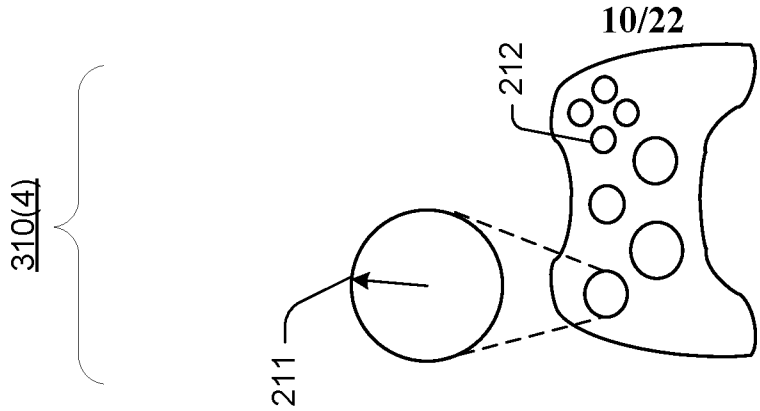


FIG. 3C



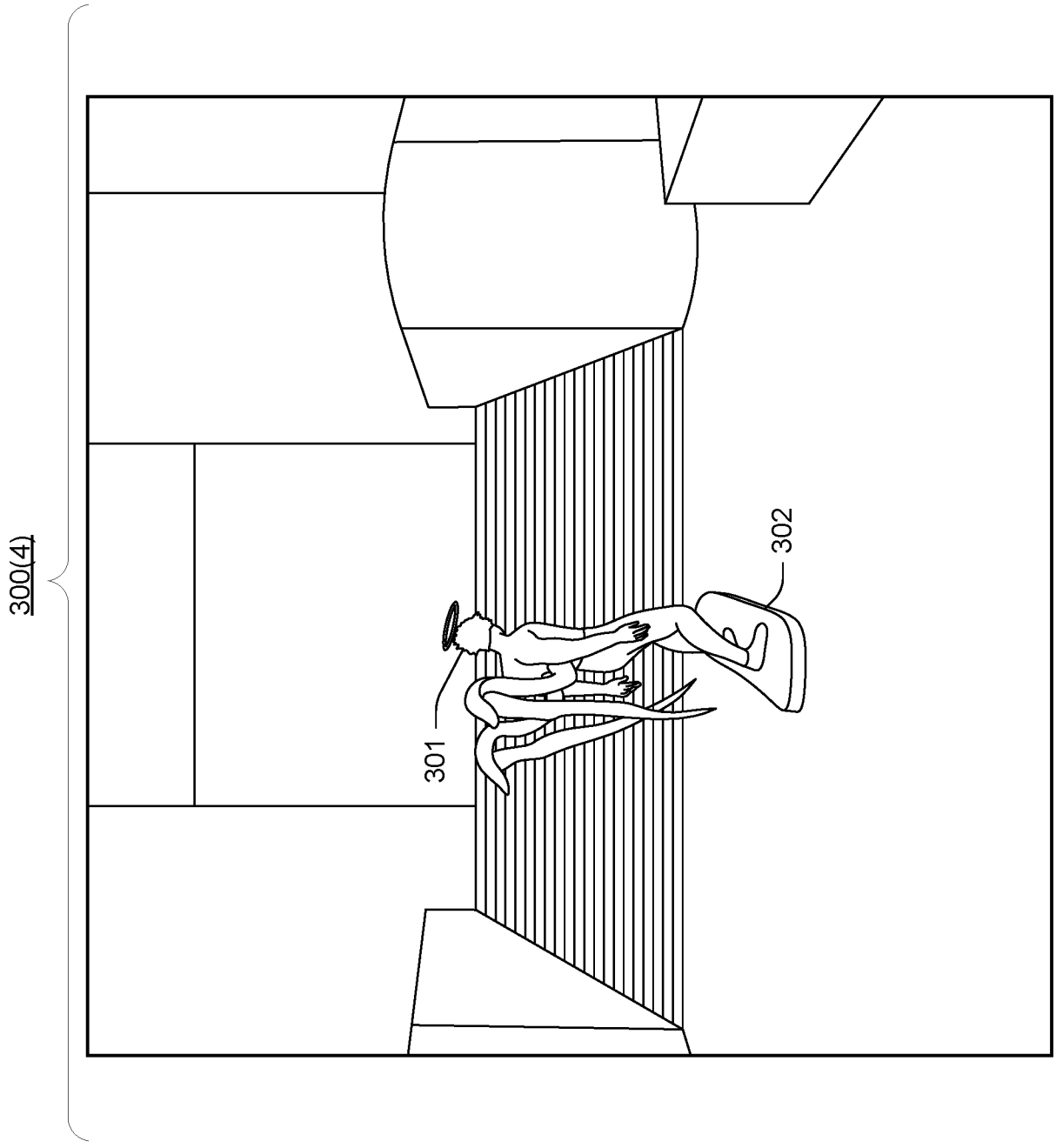


FIG. 3D

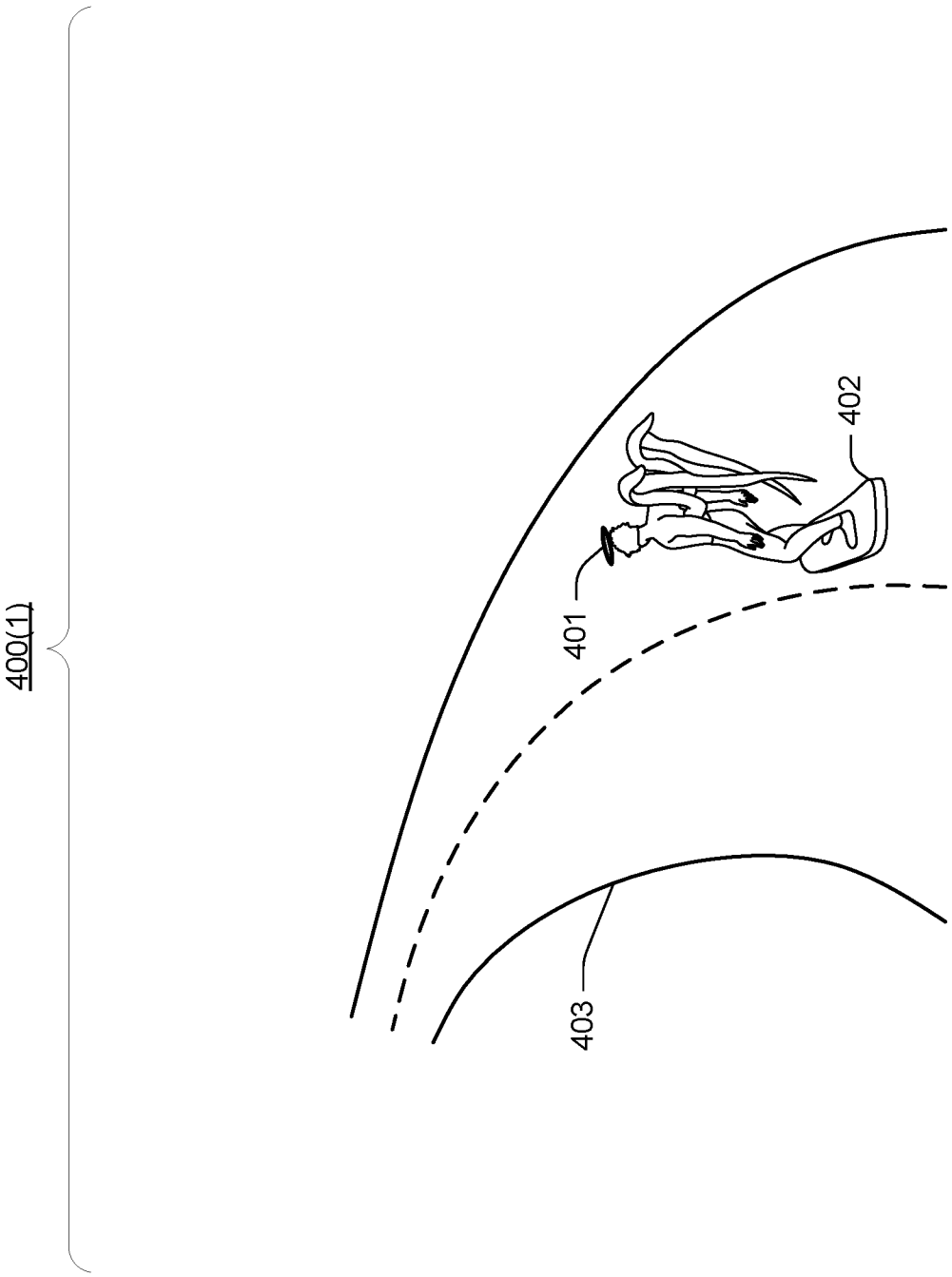


FIG. 4A

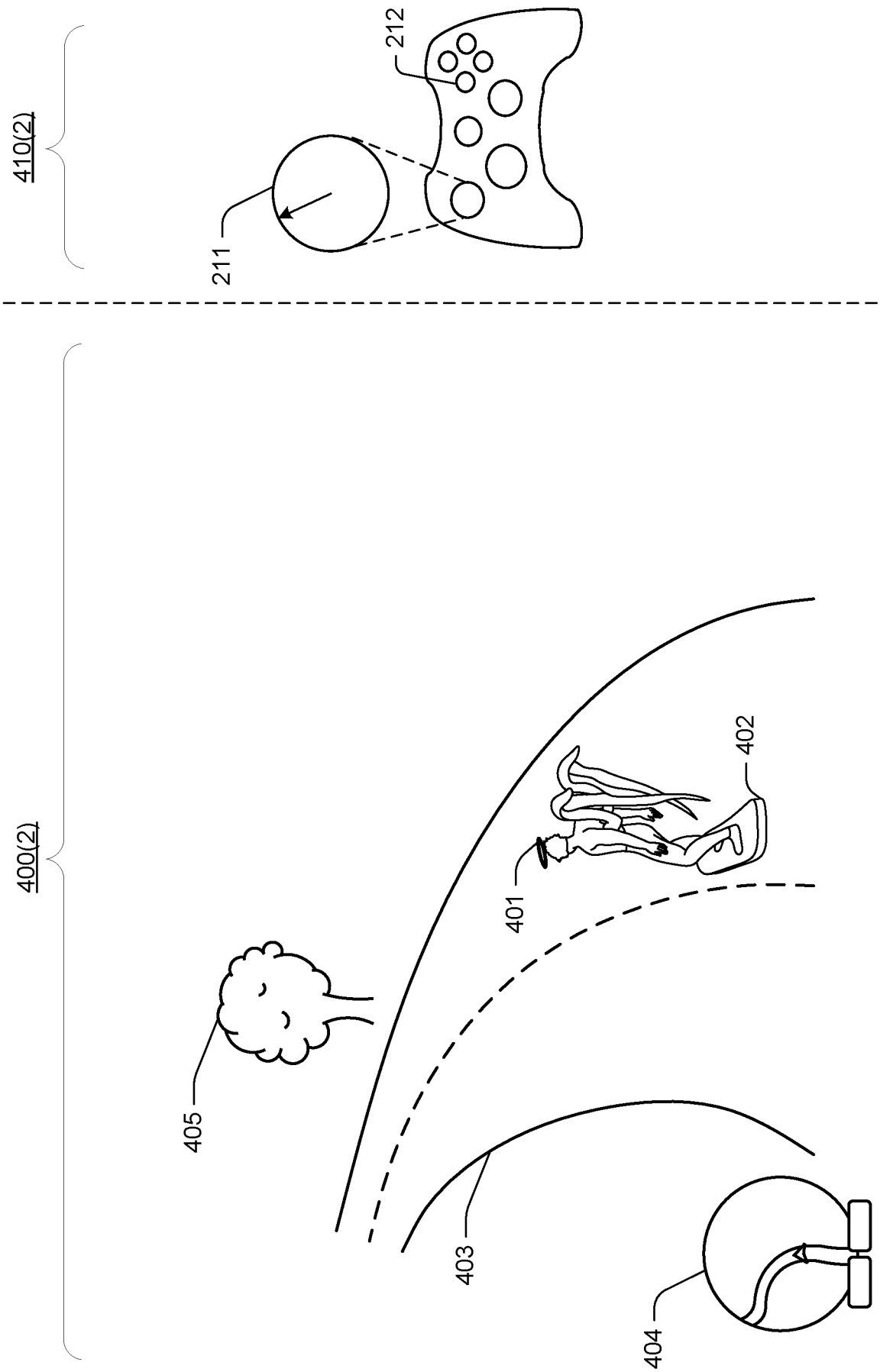


FIG. 4B

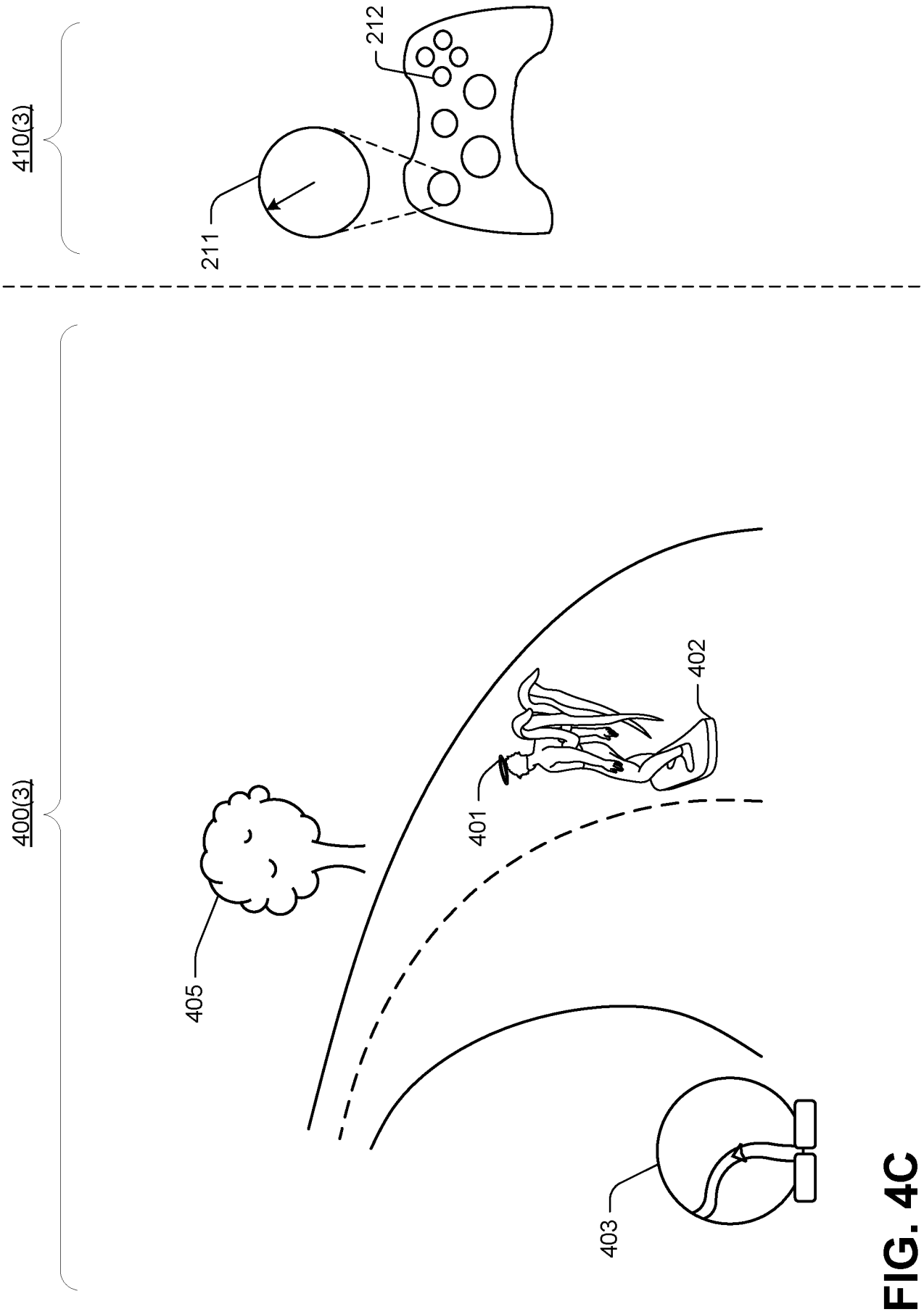


FIG. 4C

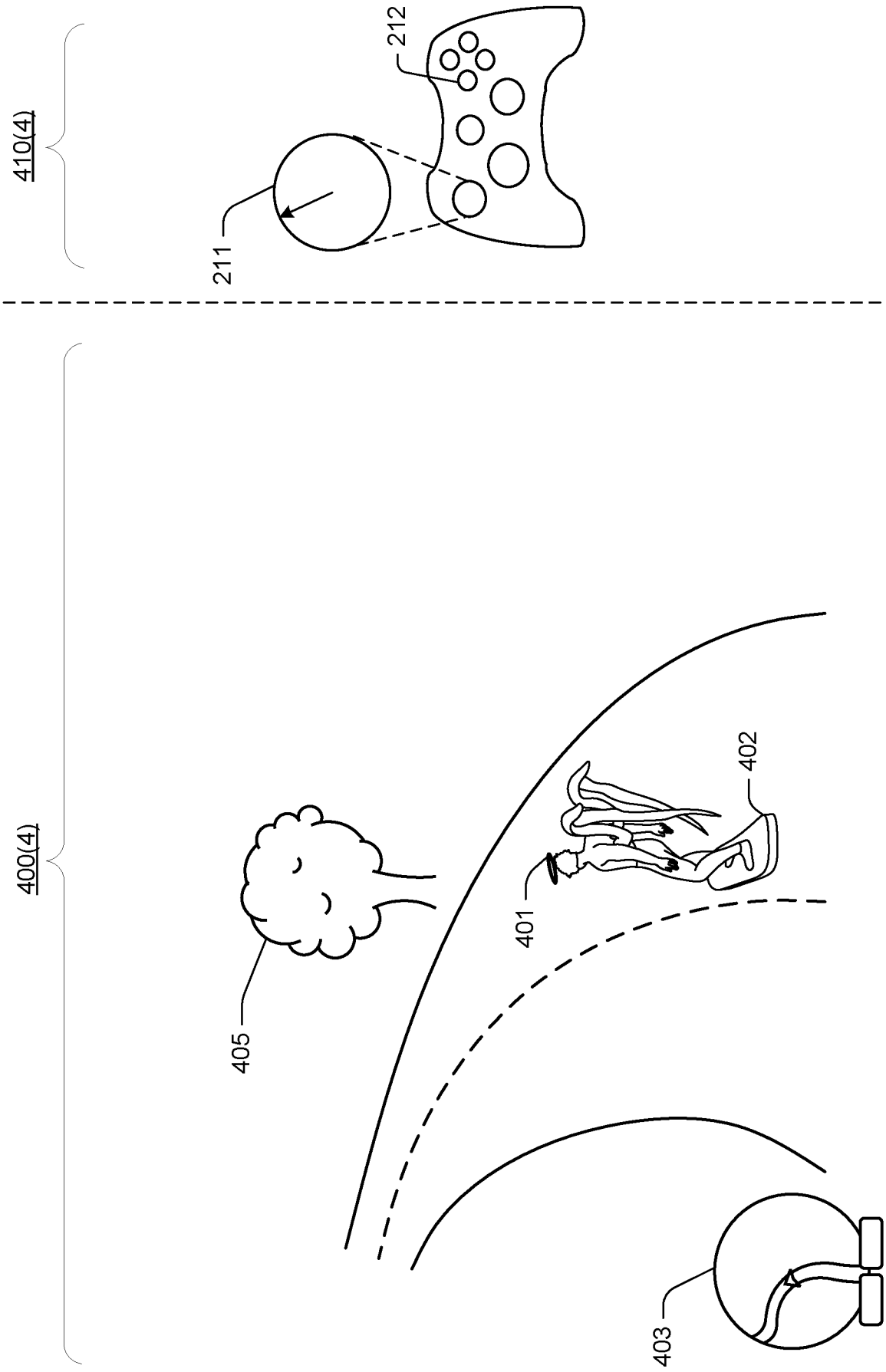


FIG. 4D

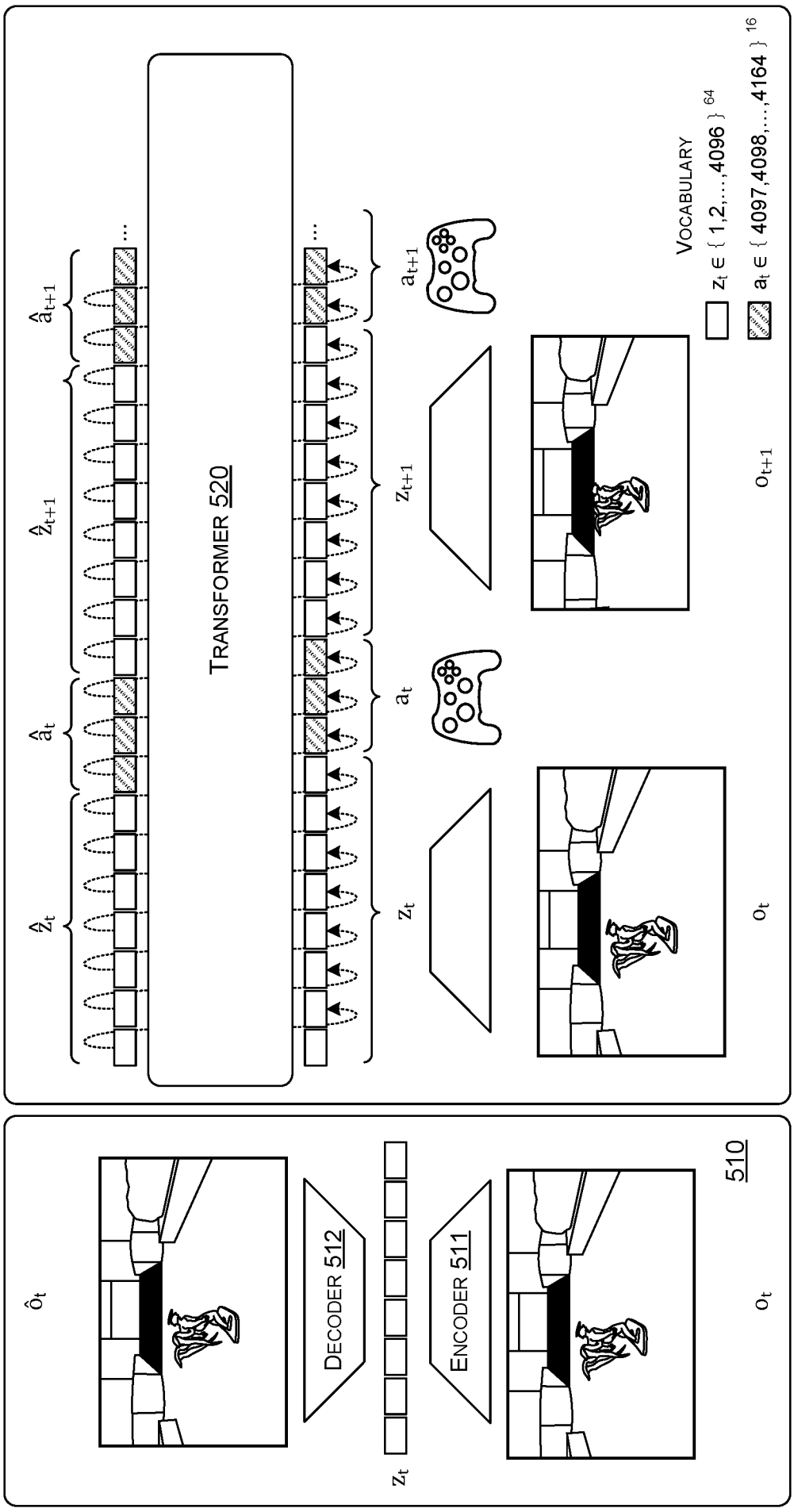

TRANSFORMER 520
VOCABULARY
$z_t \in \{1,2,...,4096\}^{64}$
$a_t \in \{4097,4098,...,4164\}^{16}$
DECODER 512
ENCODER 511
510

FIG. 5

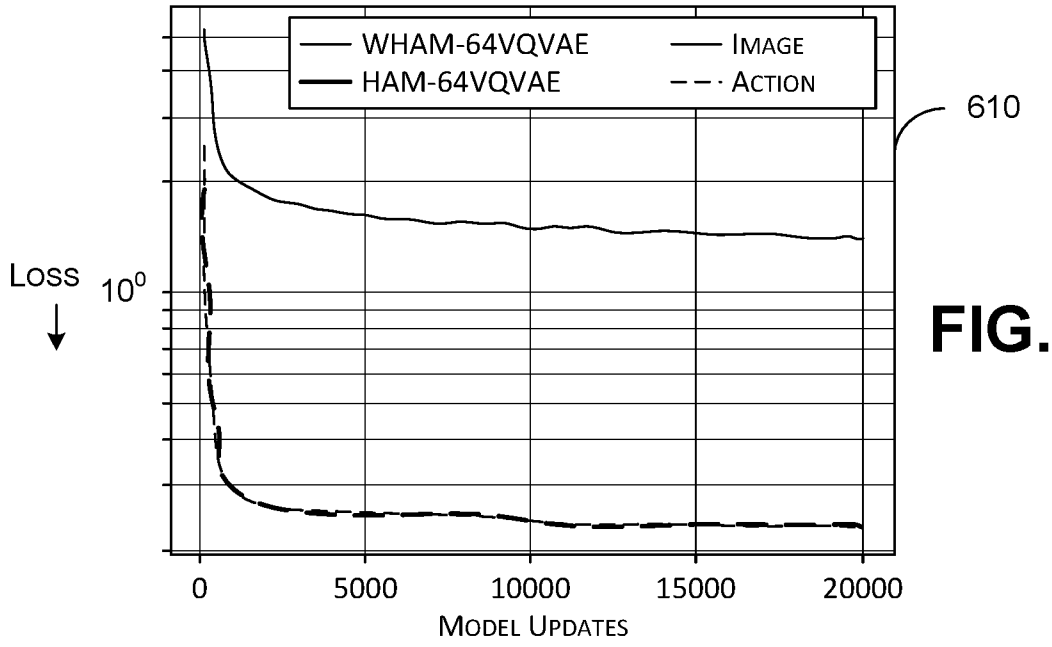


FIG. 6A

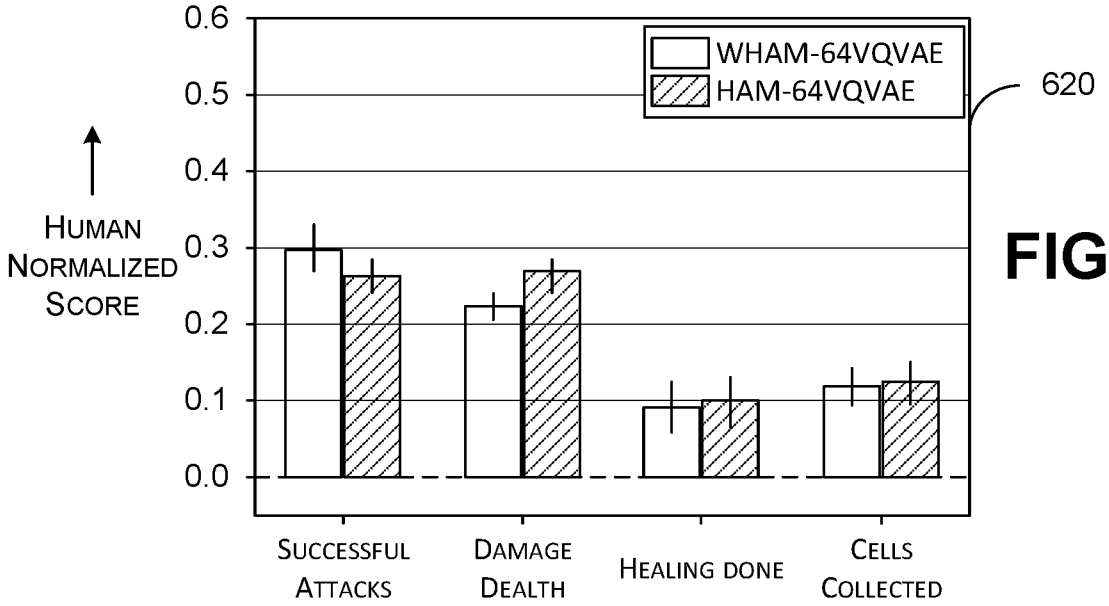


FIG. 6B

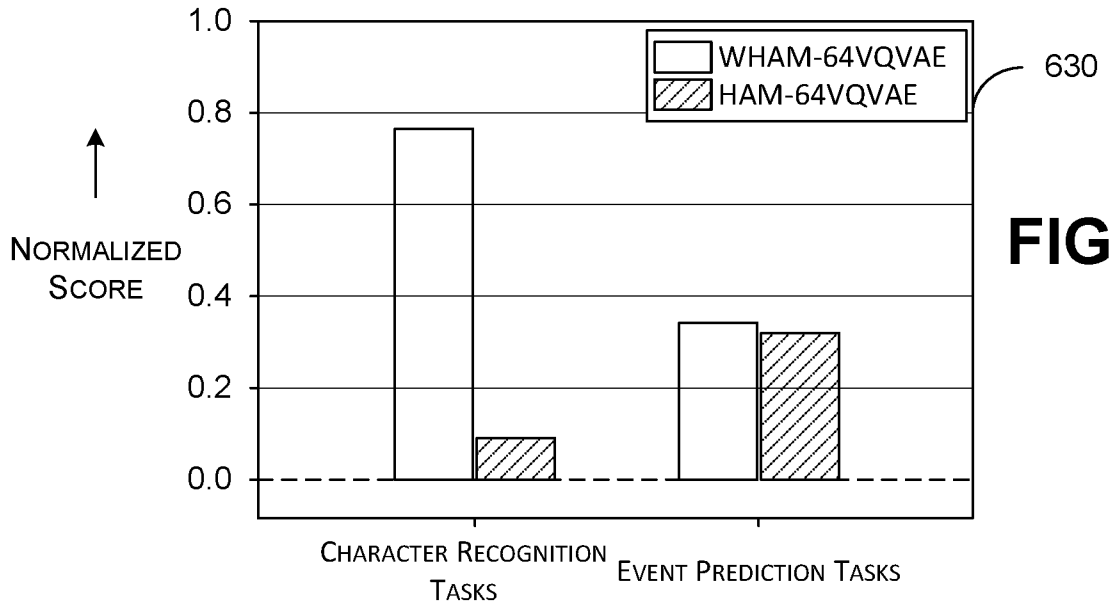


FIG. 6C

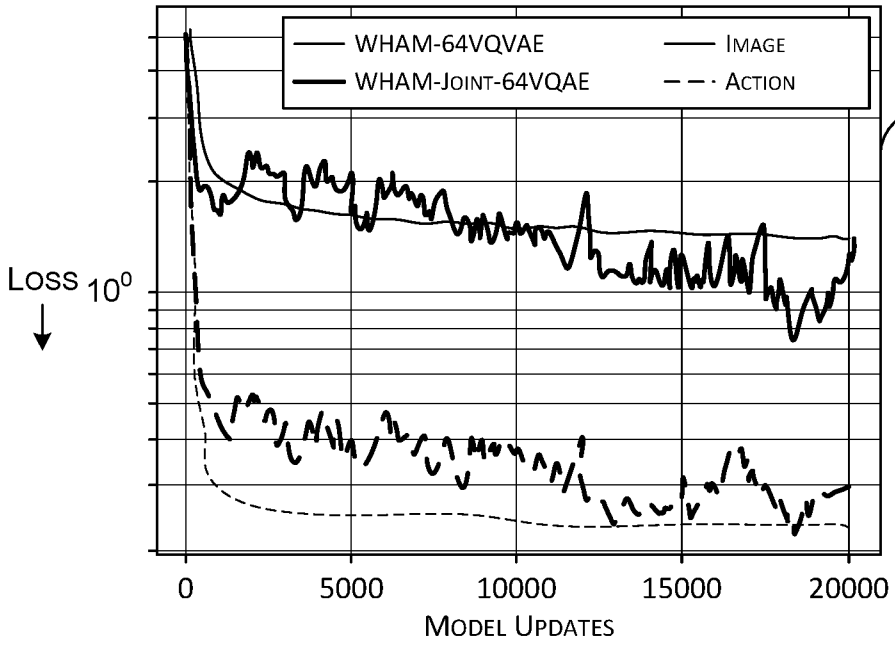


FIG. 7A

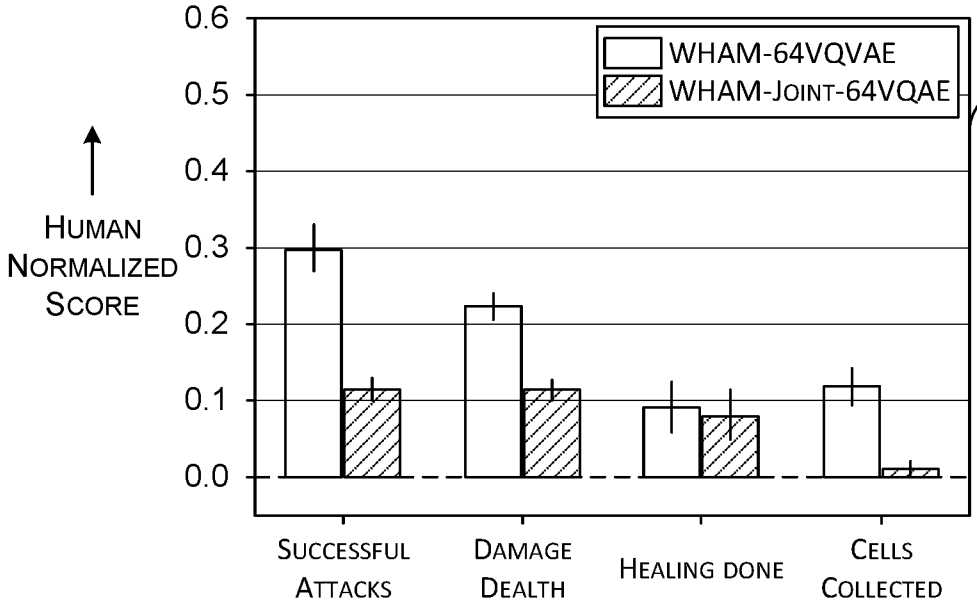


FIG. 7B

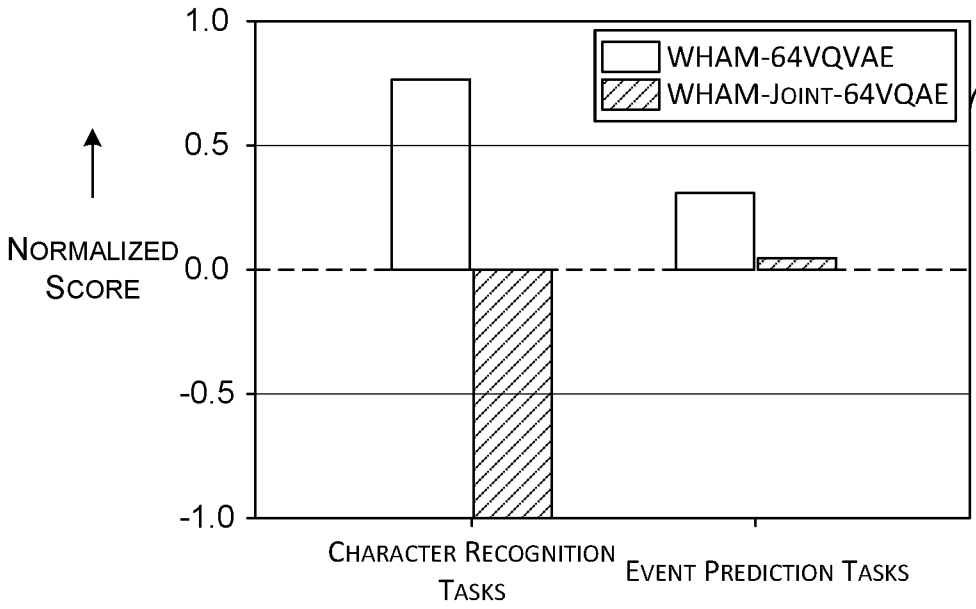


FIG. 7C

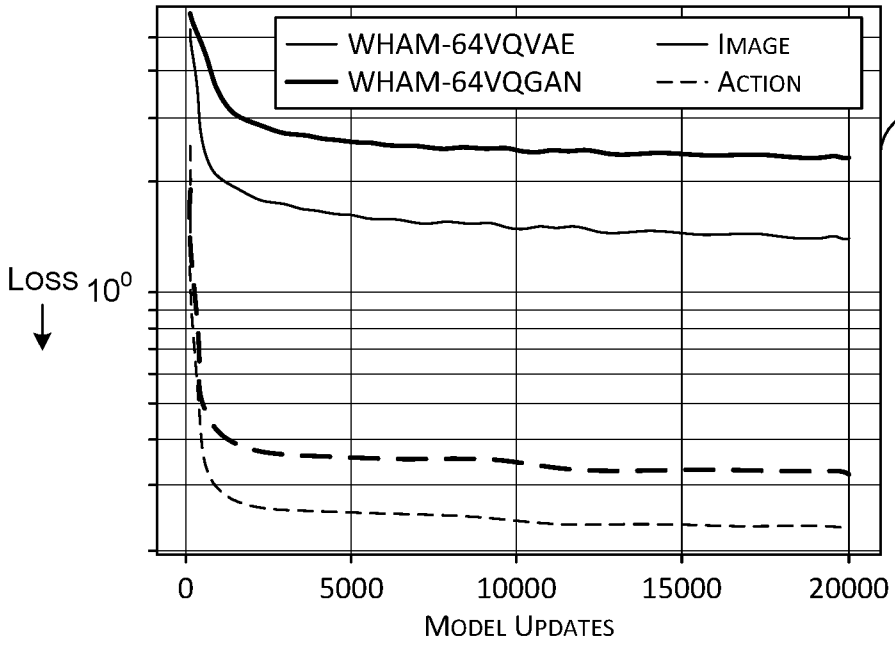


FIG. 8A

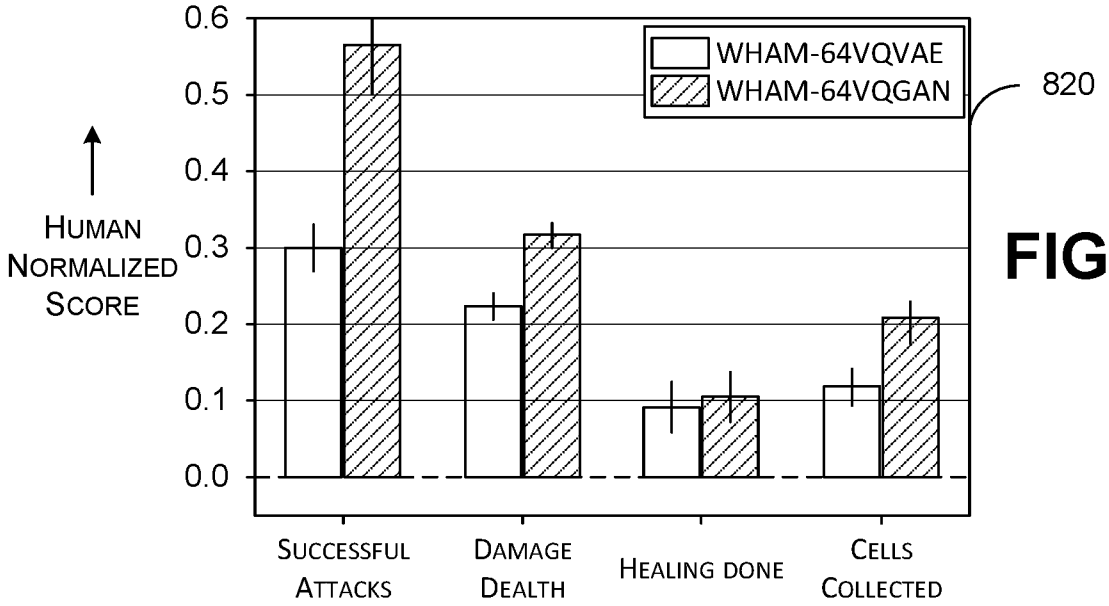


FIG. 8B

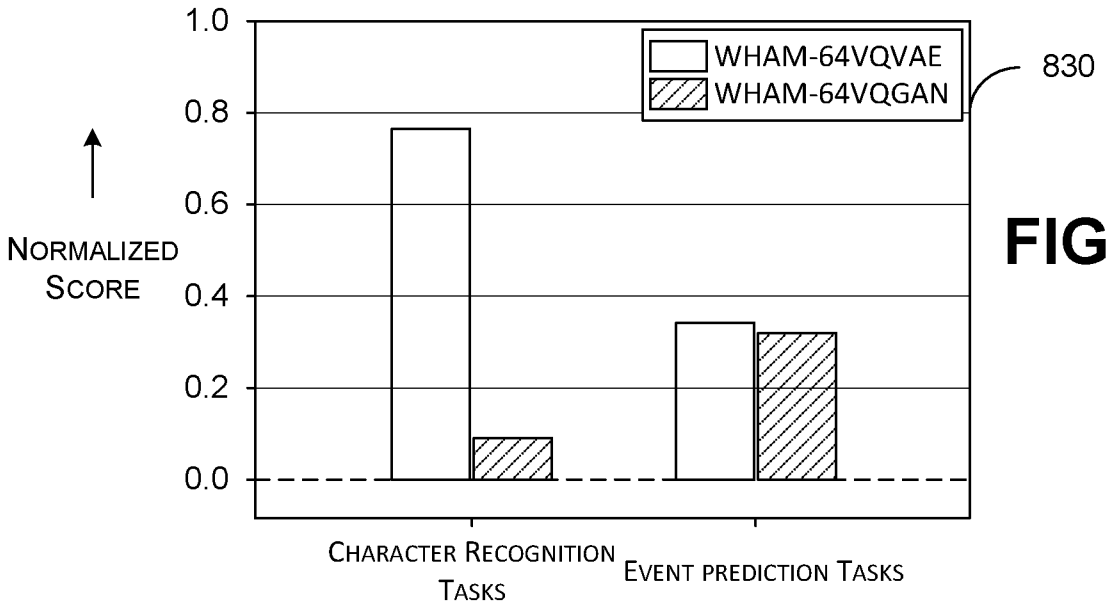


FIG. 8C

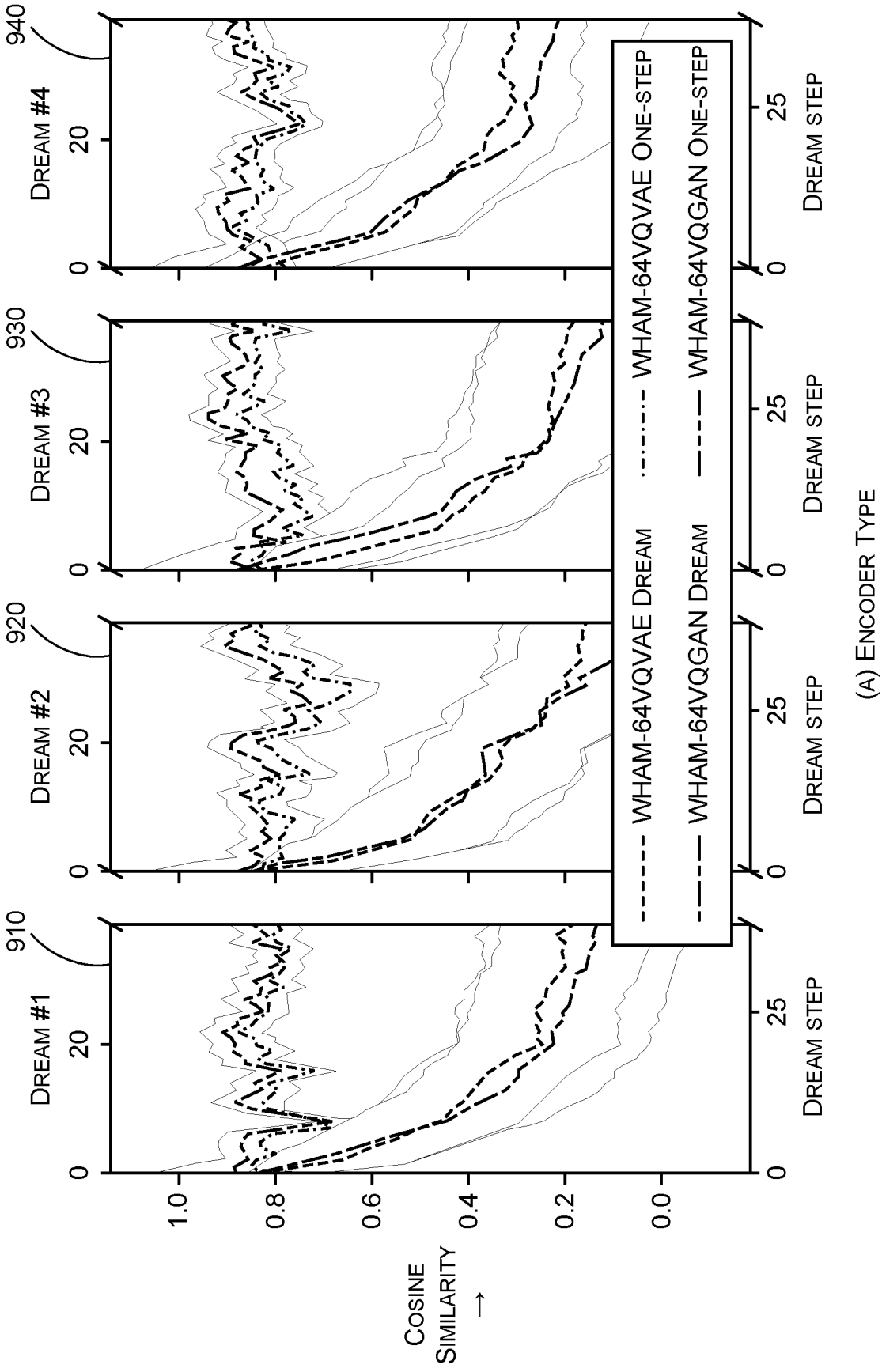

910
920
930
940
DREAM #1
DREAM #2
DREAM #3
DREAM #4
0
20
0
25
DREAM STEP
1.0
0.8
0.6
0.4
0.2
0.0
COSINE SIMILARITY →
WHAM-64VQVAE DREAM
WHAM-64VQGAN DREAM
WHAM-64VQVAE ONE-STEP
WHAM-64VQGAN ONE-STEP
(A) ENCODER TYPE

FIG. 9

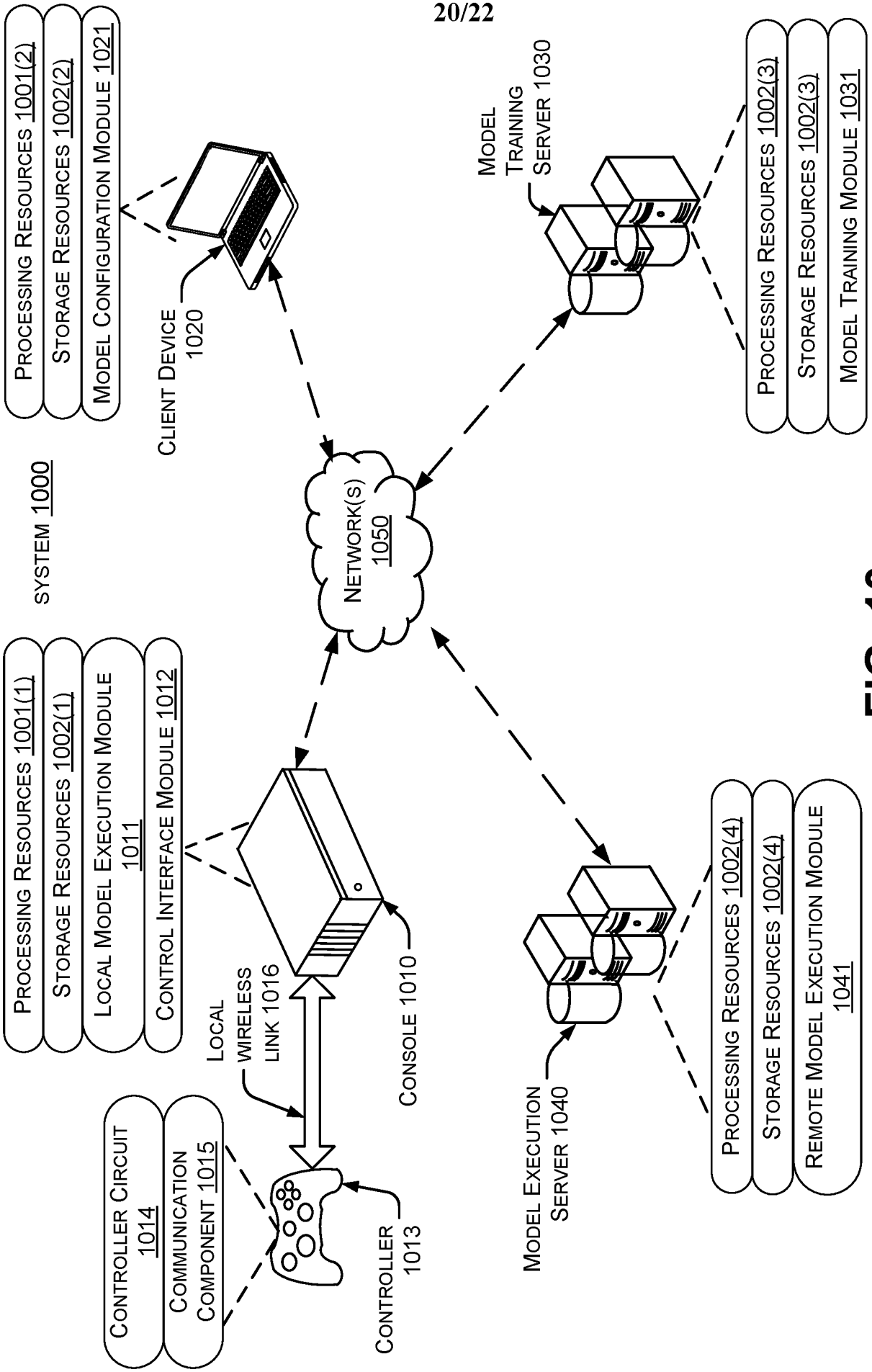

SYSTEM 1000
PROCESSING RESOURCES 1001(1)
STORAGE RESOURCES 1002(1)
LOCAL MODEL EXECUTION MODULE 1011
CONTROL INTERFACE MODULE 1012
CONTROLLER CIRCUIT 1014
COMMUNICATION COMPONENT 1015
LOCAL WIRELESS LINK 1016
CONSOLE 1010
CONTROLLER 1013
PROCESSING RESOURCES 1001(2)
STORAGE RESOURCES 1002(2)
MODEL CONFIGURATION MODULE 1021
CLIENT DEVICE 1020
NETWORK(S) 1050
MODEL TRAINING SERVER 1030
PROCESSING RESOURCES 1002(3)
STORAGE RESOURCES 1002(3)
MODEL TRAINING MODULE 1031
MODEL EXECUTION SERVER 1040
PROCESSING RESOURCES 1002(4)
STORAGE RESOURCES 1002(4)
REMOTE MODEL EXECUTION MODULE 1041

FIG. 10

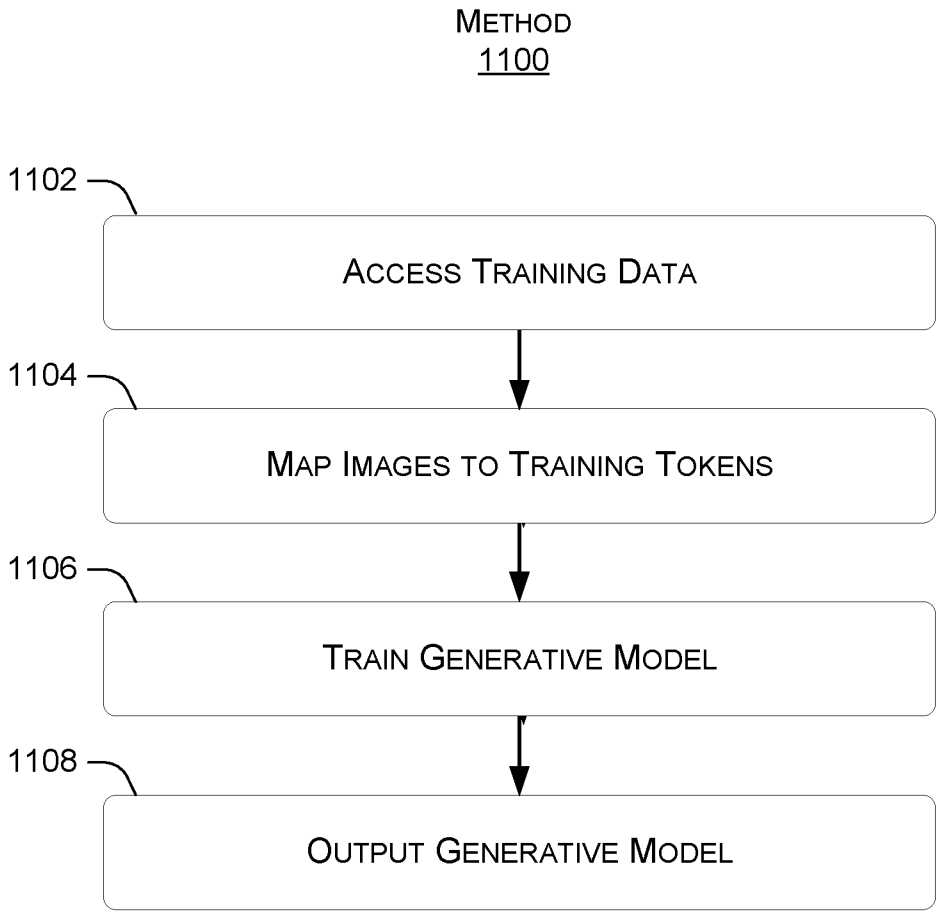

METHOD
1100
1102
ACCESS TRAINING DATA
1104
MAP IMAGES TO TRAINING TOKENS
1106
TRAIN GENERATIVE MODEL
1108
OUTPUT GENERATIVE MODEL

FIG. 11

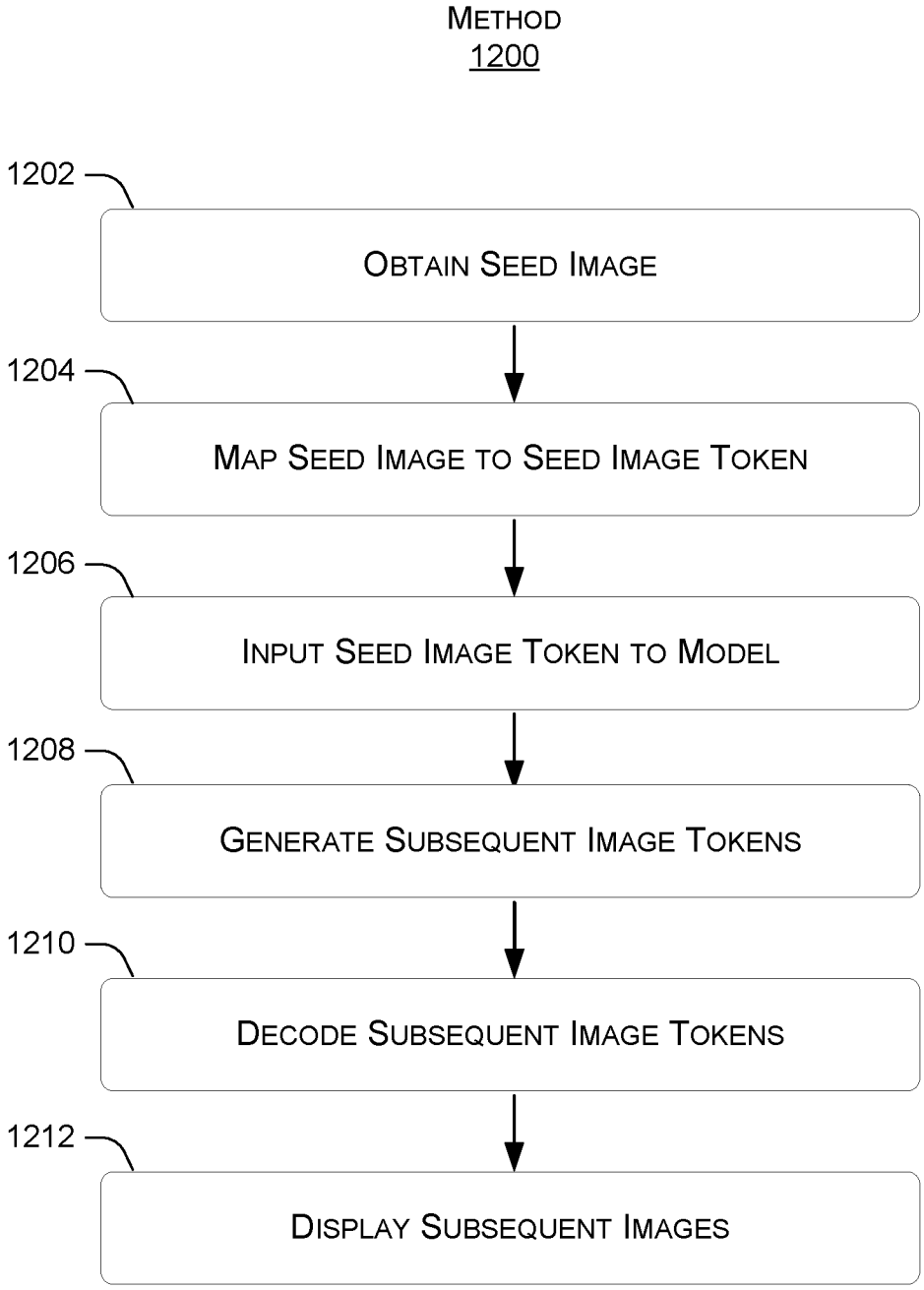

METHOD
1200
1202
OBTAIN SEED IMAGE
1204
MAP SEED IMAGE TO SEED IMAGE TOKEN
1206
INPUT SEED IMAGE TOKEN TO MODEL
1208
GENERATE SUBSEQUENT IMAGE TOKENS
1210
DECODE SUBSEQUENT IMAGE TOKENS
1212
DISPLAY SUBSEQUENT IMAGES

FIG. 12

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2024/046118

A. CLASSIFICATION OF SUBJECT MATTER

INV. G06N3/045 G06N3/0455 G06N3/0475 G06N3/09
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06N

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X A	US 2023/108874 A1 (MORLOT JEAN-BAPTISTE [FR] ET AL) 6 April 2023 (2023-04-06) page 7680 - page 7684; figures 1-4	1-14, 16-20 15

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance;; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance;; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
22 January 2025	29/01/2025

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Falco, Gabriele

Form PCT/ISA/210 (second sheet) (April 2005)

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2024/046118

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2023108874 A1	06-04-2023	EP 4104104 A1	21-12-2022
		US 2023108874 A1	06-04-2023
		WO 2021160686 A1	19-08-2021