MODEL TRAINING SYSTEM AND METHOD, AND ELECTRONIC DEVICE

TECHNICAL FIELD

[0001] This application relates to the field of computer technologies, and in particular, to a model training system and method, and an electronic device.

BACKGROUND

[0002] In the field of artificial intelligence, large language models (Large Language Models, LLMs) are widely used. The LLMs have provided a blueprint for the development of artificial general intelligence.

[0003] The LLMs implement alignment through supervised training. However, a supervised training paradigm requires a large amount of labeled data, which is costly and cannot ensure labeling quality and efficiency. Therefore, a training method for the LLMs is required, to improve training efficiency and alignment performance of the LLMs.

SUMMARY

[0004] For a problem of how to improve training efficiency and alignment performance of an LLM, this application provides a model training system and method, and an electronic device. This application further provides a computer-readable storage medium.

[0005] The following technical solutions are used in embodiments of this application.

[0006] According to a first aspect, this application provides a model training system. The model training system includes:

an inference module, configured to generate a training sample used for model training;

an experience buffer, configured to store a training sample; and

a training module, configured to perform model training based on the training sample stored in the experience buffer, where the inference module and the training module run asynchronously and in parallel.

[0007] According to the model training system in the first aspect, an asynchronous architecture in which the training module is separated from the inference module is used, so that a training process is decoupled from an inference process, a conventional optimization technology can be flexibly applied to training and inference separately, and good scalability is provided. This may resolve a problem of low sampling efficiency and improve efficiency of model training.

[0008] In an implementation of the first aspect, the model training system is configured to implement a reinforcement learning from human feedback (Reinforcement Learning from Human Feedback, RLHF) fine-tuning framework.

[0009] In an implementation of the first aspect, the inference module includes a plurality of parallel inference nodes, and/or the training module includes a plurality of parallel training nodes.

[0010] In an implementation of the first aspect, the model system further includes:

a coordination controller, configured to adjust, based on throughputs of the inference module and the training module, compute resource allocation of each component within the inference module, and/or adjust compute resource allocation of the inference module and the training module.

[0011] According to the foregoing implementation of the first aspect, an adaptive resource coordination mechanism is designed for the model system, to provide a function of dynamically adjusting a compute resource ratio within an inference node and a compute resource ratio between a training node and the inference node, thereby avoiding training resource waiting and inference data waste, and effectively improving a training throughput and resource utilization.

[0012] In an implementation of the first aspect, the inference module includes a reference model, a reward model, and a first policy model, and the training module includes a second policy model; and

the coordination controller is configured to: perform optimization solving, based on a training throughput of the second policy model per unit resource, an inference throughput of the first policy model per unit resource, and inference throughputs of the reference model and the reward model per unit resource, with an objective of maximizing a total training throughput and an inference throughput, to obtain a compute resource allocation ratio of each model within the inference module and a compute resource allocation ratio between the inference module and the training module.

[0013] In an implementation of the first aspect,

the inference module is further configured to add a timestamp to the training sample; and

the experience buffer is further configured to: detect the timestamp of the training sample, and delete the training sample when the timestamp of the training sample exceeds a preset threshold.

[0014] According to the foregoing implementation of the first aspect, the timestamp is added to the training sample, and the experience buffer automatically discards an expired training sample based on the timestamp of the training sample, to limit a version difference between a model for generating a sample and a model of the training sample, ensure a model training convergence

capability in large-scale distributed cluster training, and improve alignment performance.

[0015] In an implementation of the first aspect,

the inference module is further configured to add a timestamp to the training sample; and

the training module is further configured to: continuously sample the training sample from the experience buffer, evaluate a difference between the training sample and a current training policy model version, and drop the training sample when the difference exceeds a preset threshold.

[0016] According to the foregoing implementation of the first aspect, the timestamp is added to the training sample. When obtaining the training sample, the training module automatically compares a version timestamp of the training sample with a version timestamp of a current model, to limit a version difference between a model for generating the sample and the model of the training sample, ensure a model training convergence capability in large-scale distributed cluster training, and improve alignment performance.

[0017] In an implementation of the first aspect, the inference module is deployed on a homogeneous or heterogeneous compute resource.

[0018] In an implementation of the first aspect, the inference module includes the reference model, the reward model, and the first policy model.

[0019] The reference model and the reward model are deployed on a first-type compute resource, and the first policy model is deployed on a second-type compute resource.

[0020] In an implementation of the first aspect, the inference module includes the reference model, the reward model, and the first policy model.

[0021] The reference model is deployed on the first-type compute resource, the reward model is deployed on the second-type compute resource, and the first policy model is deployed on a third-type compute resource.

[0022] According to the foregoing implementation of the first aspect, the inference module uses a heterogeneous framework, so that different types of compute resources may be effectively used, thereby avoiding waste of cloud resources.

[0023] In an implementation of the first aspect, the training module is deployed on a homogeneous compute resource.

[0024] According to a second aspect, this application provides a model training method, and the method is applied to a model system. The model system includes an inference module, an experience buffer, and a training module. The method includes:

generating, by using the inference module, a training sample used for model training;

storing the training sample generated by the inference module in the experience buffer; and

performing model training by using the training module based on the training sample stored in the experience buffer, where the inference module and the training module run asynchronously and in parallel.

[0025] In an implementation of the second aspect, the model system is configured to implement an RLHF framework.

[0026] In an implementation of the second aspect, the method further includes:

adjusting, based on throughputs of the inference module and the training module, compute resource allocation of each component within the inference module, and/or adjusting compute resource allocation of the inference module and the training module.

[0027] In an implementation of the second aspect, the inference module includes a reference model, a reward model, and a first policy model, and the training module includes a second policy model; and

the adjusting, based on the throughputs of the inference module and the training module, the compute resource allocation of each component within the inference module, and/or adjusting the compute resource allocation of the inference module and the training module includes:

performing optimization solving, based on a training throughput of the second policy model per unit resource, an inference throughput of the first policy model per unit resource, and inference throughputs of the reference model and the reward model per unit resource, with an objective of maximizing a total training throughput and an inference throughput, to obtain a compute resource allocation ratio of each model within the inference module, and a compute resource allocation ratio between the inference module and the training module.

[0028] In an implementation of the second aspect, the method further includes:

deploying the inference module, where the inference module includes a plurality of parallel inference nodes; and/or

deploying the training module, where the training module includes a plurality of parallel training nodes.

[0029] In an implementation of the second aspect, the method further includes:

deploying the inference module, where the inference module is deployed on a homogeneous or heterogeneous compute resource.

[0030] In an implementation of the second aspect, the inference module includes the reference model, the reward model, and the first policy model; and

the deploying the inference module includes:

deploying the reference model and the reward model on a first-type compute resource, and deploying the first policy model on a second-type compute resource.

[0031] In an implementation of the second aspect, the inference module includes the reference

model, the reward model, and the first policy model; and

the deploying the inference module includes:

deploying the reference model on the first-type compute resource, deploying the reward model on the second-type compute resource, and deploying the first policy model on a third-type compute resource.

[0032] In an implementation of the second aspect, the method further includes:

deploying the training module, where the training module is deployed on a homogeneous compute resource.

[0033] In an implementation of the second aspect, the method further includes:

before the training sample is stored in the experience buffer, adding a timestamp to the training sample; and

detecting the timestamp of the training sample in the experience buffer, and deleting the training sample from the experience buffer when the timestamp of the training sample exceeds a preset threshold.

[0034] In an implementation of the second aspect, the method further includes:

before the training sample is stored in the experience buffer, adding a timestamp to the training sample; and

continuously sampling the training sample from the experience buffer, and performing model training fine-tuning, where a difference between the training sample and a current training policy model version is evaluated, and the training sample is dropped when the difference exceeds a preset threshold.

[0035] According to a third aspect, this application provides an electronic device. The electronic device includes a memory configured to store computer program instructions and a processor configured to execute the computer program instructions, and when the computer program instructions are executed by the processor, the electronic device is triggered to perform the steps of the method according to the second aspect.

[0036] According to a fourth aspect, this application provides a computer-readable storage medium. The computer-readable storage medium stores a computer program, and when the computer program is run on a computer, the computer is enabled to perform the method according to the second aspect.

BRIEF DESCRIPTION OF DRAWINGS

[0037] FIG. 1 is a diagram of a structure of a model system according to an embodiment of this application;

[0038] FIG. 2 is a schematic flowchart of creating a model system according to an embodiment of this application;

[0039] FIG. 3 is a diagram of a structure of a model system according to an embodiment of this application;

[0040] FIG. 4 is a schematic flowchart of an RLHF method according to an embodiment of this application; and

[0041] FIG. 5 is a diagram of a structure of an electronic device according to an embodiment of this application.

DESCRIPTION OF EMBODIMENTS

[0042] To make objectives, technical solutions, and advantages of this application clearer, the following clearly and completely describes the technical solutions in this application with reference to specific embodiments and the accompanying drawings in this application. It is clear that the described embodiments are merely some but not all of embodiments of this application. All other embodiments obtained by a person of ordinary skill in the art based on embodiments of this application without creative efforts shall fall within the protection scope of this application.

[0043] Terms used in implementations of this application are only used to explain specific embodiments of this application, but are not intended to limit this application.

[0044] A primary objective of large language models (Large Language Models, LLMs) is to train a human-centered (honest, helpful, and harmless) intelligent assistant based on a large amount of corpus data. Therefore, it is crucial that the LLMs align with human preferences.

[0045] Based on the above principles, a feasible solution for training of the LLMs is to use a reinforcement learning from human feedback (Reinforcement Learning from Human Feedback, RLHF) fine-tuning method.

[0046] Reinforcement learning is one of paradigms and methodologies in machine learning, which is used to describe and resolve a problem of maximizing rewards or achieving specific goals through learning strategies during interaction between intelligent agents and environment. The RLHF, reinforcement learning from human feedback, is a technology that aligns large language models with human preferences. Based on human preference data, a reward model is trained to model human preferences for model responses, and a reinforcement learning algorithm is used to optimize a language model to maximize cumulative rewards, thereby aligning language model outputs with human preferences.

[0047] The RLHF uses the reward model to learn value orientations from human preference data and guides a fine-tuning process of a language model, serving as a key technical paradigm for

achieving human preference alignment. The RLHF can make full use of human feedback information, extract human subjective preferences, and automatically optimize the model to generate higher-quality text that is more consistent with human preferences and mainstream values. This significantly improves training efficiency and alignment performance of the large language models.

[0048] The RLHF has significant advantages in the alignment of large language models, but also introduces many challenges. First, sampling efficiency is low. An RLHF sampling process includes training and inference of a plurality of large models, including a policy model, a value model, a reference model, and a reward model. Serial logic exists among these models, leading to device idling. Second, training efficiency is low. Due to frequent switching between training and inference in reinforcement learning, a quantity of training samples is restricted by the sampling efficiency, and a plurality of large models occupy video memory resources, limiting a training throughput.

[0049] To resolve the above problems of the RLHF, a feasible solution is to optimize an RLHF framework in a plurality of directions, such as video memory optimization and efficient switching between training and inference.

[0050] For example, a trlX framework is used. The trlX framework is a serial iteration solution for a training and inference process. The trlX framework is a distributed RLHF framework built upon a Hugging Face (Hugging Face) ecosystem. A training process of the trlX framework is based on the Accelerate (Accelerate) backend of the Hugging Face, supporting distributed training of a model up to 20B parameters.

[0051] For another example, a ColossalChat framework is used. The ColossalChat framework is a serial iteration solution for a training and inference process. The ColossalChat framework uses low-rank adaptation (Low-Rank Adaptation, LoRA) and zero redundancy optimizer (Zero Redundancy Optimizer, ZeRO)+Gemini (Gemini) technology to improve memory usage efficiency and reduce communication frequency. In addition, the ColossalChat framework can offload optimizer states from a graphics processing unit (graphics processing unit, GPU) memory to a central processing unit (Central Processing Unit, CPU) memory or hard disk space, enabling larger models to be trained with fewer compute resources.

[0052] For another example, a DeepSpeedChat framework is used. The DeepSpeedChat framework is a serial iteration solution for a training and inference process. The DeepSpeedChat framework proposes a hybrid training-inference engine to implement seamless switching between an inference mode and a training mode, so that various optimization methods of DeepSpeed training (such as ZeRO and LoRA) and inference (such as tensor parallelism and high-performance operators) can be used.

[0053] However, the foregoing optimization solutions have the following problems.

[0054] (1) Poor scalability: The foregoing framework fails to fully consider mechanisms such as the distributed training, thereby making it difficult to effectively scale sizes and compute resources of a plurality of models.

[0055] (2) Serial iteration of training and inference: The training and inference processes of the foregoing framework are highly coupled, thereby limiting utilization of an inference optimization technology.

[0056] (3) Low resource utilization: RLHF training and inference throughputs differ significantly. The foregoing framework fails to consider optimal allocation and heterogeneous integration of training resources and inference resources.

[0057] For the foregoing problems, an embodiment of this application provides a model system. The model system is configured to implement a reinforcement learning from human feedback (Reinforcement Learning from Human Feedback, RLHF) fine-tuning method (RLHF framework).

[0058] Specifically, in an embodiment of this application, the model system uses asynchronous distributed RLHF.

[0059] "Asynchronous distributed" refers to achieving efficient parallel processing among a plurality of nodes through asynchronous communication and coordination. "Asynchronous distributed" may improve a throughput and scalability of a system while reducing performance degradation due to communication bottlenecks or synchronization delays.

[0060] In an embodiment of this application, the model system uses an asynchronous distributed RLHF framework. To be specific, in the model system, a training node and an inference node of the RLHF framework run independently and in parallel, without performing synchronous communication or waiting.

[0061] Specifically, in an embodiment, the RLHF framework in the model system includes a plurality of parallel training nodes and a plurality of parallel inference nodes. The training nodes and the inference nodes run independently and in parallel, without performing synchronous communication or waiting.

[0062] FIG. 1 is a diagram of a structure of a model system according to an embodiment of this application.

[0063] As shown in FIG. 1, in an embodiment, an RLHF framework of the model system includes an inference module 110, a training module 120 (including a plurality of parallel training nodes), and an experience buffer (Experience Buffer) 140.

[0064] The inference module 110 is configured to implement a self-sampling inference process of RLHF. The inference module 110 includes parallel inference nodes, which are responsible for

self-sampling to generate sample data used for model training.

[0065] The inference module 110 includes a reference model (Reference Model, Ref Model) 111, a reward model (Reward Model, RM) 112, and a policy model (Policy Model) 113.

[0066] The reference model 111, in the RLHF, refers to an original language model used to initialize a policy model. The reference model 111 is configured to provide a reference answer for a given user query for comparison with a model response generated by the policy model 113, to limit a change magnitude of the policy model 113.

[0067] The reward model 112, in the RLHF, represents a reward-guided policy optimization process. The reward model 112 takes {a user query, a model output} as an input, and outputs a scalar score, to evaluate a response generated by the policy model 113.

[0068] The policy model 113, in the RLHF, is a large language model to be optimized. The policy model 113 is configured to take a user query as an input and output a model response. An ultimate goal of the RLHF is to optimize the policy model, such that the model outputs high-quality text and aligns with human preferences.

[0069] In an inference phase of the inference module 110, a user question is sampled from a dataset and input to the policy model 113, and forward inference is performed to obtain a question-answer pair. The question-answer pair is transmitted to the reference model 111 and the reward model 112 to infer and compute a KL penalty value and a reward value respectively, so as to obtain a training sample, where the training sample is stored in the experience buffer 140.

[0070] The training module 120 is used in an RLHF fine-tuning training process. The training module 120 includes a plurality of parallel training nodes, and is configured to perform model training fine-tuning based on a reinforcement learning method.

[0071] In a training phase, the training module 120 loads the policy model, continuously performs sampling from the experience buffer 140, and performs model training fine-tuning based on a reinforcement learning algorithm (for example, proximal policy optimization (Proximal Policy Optimization, PPO) algorithm).

[0072] The PPO is a reinforcement learning algorithm based on actor-critic (Actor-Critic), and is used to train policy models to complete complex tasks.

[0073] The experience buffer 140 is stored in a compute resource memory, and is configured to receive sample data constructed by the inference module 110, and provide, for the training module 120, training data required for RLHF fine-tuning.

[0074] An asynchronous distributed RLHF framework provided in an embodiment of this application uses an asynchronous heterogeneous multi-node cluster architecture in which the training module 120 is separated from the inference module 110 (training and inference are separated). This framework has good scalability, effectively resolving a problem of low sampling

efficiency of an RLHF method for large models and improving model training efficiency.

[0075] According to an embodiment of this application, the RLHF framework uses an asynchronous distributed framework, so that an RLHF training process is decoupled from an RLHF inference process. The training node and the inference node run in an asynchronous and parallel manner, which has good scalability. In addition, a conventional optimization technology can be flexibly applied to training and inference separately.

[0076] Optionally, in an embodiment, the RLHF framework of the model system uses a heterogeneous distributed framework. The heterogeneous distributed framework refers to a framework formed by nodes of different types or different platforms. The heterogeneous distributed framework performs efficient parallel computing by simultaneously using respective advantages of nodes of different types or different platforms.

[0077] Specifically, in an embodiment, the inference module 110 is compatible with a homogeneous or heterogeneous compute resource configuration. For example, inference node compute resources of the RLHF framework may perform parallel computing by using both a GPU and a neural-network processing unit (Neural-network Processing Unit, NPU).

[0078] In an embodiment, in the inference module 110, the policy model 113 is independently deployed on a compute resource, while the reference model 111 and the reward model 112 are jointly deployed on a same compute resource.

[0079] According to an embodiment of this application, the RLHF framework uses the heterogeneous distributed framework, so that different types of compute resources (such as a GPU, an NPU, and a CPU) may be effectively used, thereby avoiding waste of cloud resources.

[0080] Further, in an embodiment, the model system further includes a coordination controller (Controller) 130. The coordination controller (Controller) 130 is configured to adjust resource occupancy of each component of the RLHF framework in the model system.

[0081] Specifically, based on throughputs of the training module 120 and the inference module 110, the coordination controller 130 adaptively and dynamically adjusts compute resource allocation of each component within the inference module 110 (for example, adjusts compute resource allocation ratios of the reference model 111, the reward model 112, and the policy model 113), and adjusts compute resource allocation between the training module 120 and the inference module 110, to implement dynamic balance of data flow, ensure that the training module 120 and the inference module 110 run at full capacity, and maximize utilization of the compute resources.

[0082] In the asynchronous distributed RLHF framework provided in an embodiment of this application, an adaptive resource coordination mechanism is designed, to provide a function of dynamically adjusting a compute resource ratio within an inference node and a compute resource ratio between a training node and an inference node, thereby effectively improving an RLHF

training throughput and resource utilization.

[0083] FIG. 2 is a schematic flowchart of creating a model system according to an embodiment of this application.

[0084] In an embodiment, an electronic device performs the following process shown in FIG. 2 to construct and train the RLHF framework shown in FIG. 1.

[0085] S210: Construct an inference module 110 based on available compute resources, and set a maximum available resource threshold N^{infer} for the inference module 110.

[0086] S220: Construct a training module 120 based on the available compute resources, and set a maximum available resource threshold N^{train} for the training module 120.

[0087] S230: Construct a coordination controller 130, and use the coordination controller 130 to compute optimization of a resource allocation ratio between the inference module 110 and the training module 120.

[0088] Specifically, based on a policy model training throughput t_p^{train} per unit resource, a policy model inference throughput t_p^{infer} per unit resource, and a reference/reward model inference throughput t_r^{infer} per unit resource, the coordination controller 130 performs optimization solving with an objective of maximizing a total training throughput and inference throughput, to obtain an optimal compute resource allocation ratio between the policy model and the reference/reward model within the inference module 110 and an optimal compute resource allocation ratio between the training module 120 and the inference module 110.

[0089] In the descriptions of this embodiment of this application, the training throughput refers to a maximum quantity of samples or tokens that may be processed per unit time. The inference throughput refers to a maximum quantity of samples or tokens that may be inferred per unit time.

[0090] Preferably, in an embodiment, the coordination controller 130 performs optimization solving on the compute resource allocation ratio, with an optimization objective of maximizing the training and inference throughputs of the RLHF framework under given training and inference resources.

[0091] Specifically, an optimization function is formulated as follows:

$$\max_{N_p^{infer}, N_r^{infer}, N_p^{train}} \min\left(T^{train}, T^{infer}\right) \cdot \quad \text{(Formula 1)}$$

[0092] In Formula 1, N_p^{train} represents a quantity of training resources, N_p^{infer} represents a quantity of policy model inference resources, N_r^{infer} represents a quantity of reference/reward model inference resources, and T^{train} and T^{infer} respectively represent total throughputs of the training module and the inference module.

[0093] Between the training module and the inference module, to avoid resource idling caused

by the training module waiting for the inference module, a total training throughput is required to be no greater than a total inference throughput. The specific constraints are as follows:

$$\begin{cases} T^{train} \le T^{infer} \\ T^{train} = t_p^{train} \cdot N_p^{train} \cdot K_t \\ T^{infer} = \min\left(T_p^{infer}, T_r^{infer}\right) \\ N_p^{train} \le N^{train} \end{cases} \cdot \quad \text{(Formula 2)}$$

[0094] In Formula 2, t_p^{infer} and N_p^{infer} respectively represent a single-card inference throughput and a quantity of cards of the policy model, t_r^{infer} and N_r^{infer} respectively represent a single-card inference throughput and a quantity of cards of the reference/reward model, and K_i represents a minimum quantity of cards required for model inference.

[0095] S240: Initiate a training job based on the training and inference resource allocation ratio obtained by the coordination controller 130 through optimization solving, and monitor throughputs of the inference module 110 and the training module 120 in real time.

[0096] Specifically, when a network status experiences significant fluctuations, the coordination controller 130 may dynamically optimize the resource allocation ratio and quantities of the inference module 110 and the training module 120.

[0097] According to an embodiment of this application, the RLHF framework uses adaptive resource coordination to dynamically adjust a compute resource ratio within an inference node, optimize a throughput of the policy model and the reference/reward model, and dynamically adjust a compute resource allocation ratio between a training node and an inference node, to avoid training resource waiting and inference data waste.

[0098] Optionally, in an embodiment, a timestamp is added to a training sample generated in an inference phase of the inference module 110.

[0099] The timestamped training sample is stored in an experience buffer 140. Sample data format:

$$x_s = \{\text{prompt}, \text{response}, \text{logprob}, \text{value}, \text{reward}, \text{timestamp}\}. \quad \text{(Formula 3)}$$

[00100] In an embodiment, when generation time of the sample data in the experience buffer 140 exceeds a Staleness threshold (for example, 10 training cycles), the experience buffer 140 automatically deletes the data.

[00101] According to an embodiment of this application, the RLHF framework uses a timestamp. The experience buffer automatically discards an expired training sample based on the timestamp, thereby limiting a version difference between a model for generating a sample and a training sample model, ensuring an RLHF training convergence capability in large-scale distributed cluster training, and improving alignment performance.

[00102] In another embodiment, the training module 120 continuously samples the training

sample from the experience buffer 140, evaluates a difference between a timestamp (timestamp) of the training sample and a current version of the policy model being trained, and directly drops the data if the difference exceeds a given Staleness threshold (for example, 10 version iteration cycles).

[00103] According to an embodiment of this application, the RLHF framework uses the timestamp. When obtaining a training sample, the training module automatically compares a version timestamp of the training sample with a version timestamp of a current model, thereby limiting the version difference between the model for generating a sample and the training sample model, ensuring the RLHF training convergence capability in the large-scale distributed cluster training, and improving the alignment performance.

[00104] In another embodiment, when the generation time of the sample data in the experience buffer 140 exceeds the Staleness threshold (for example, 10 training cycles), the experience buffer 140 automatically deletes the data. In addition, the training module 120 continuously samples the training sample from the experience buffer 140, evaluates the difference between the timestamp (timestamp) of the training sample and the current version of the policy model being trained, and directly drops the data if the difference exceeds the given Staleness threshold (for example, 10 version iteration cycles).

[00105] The RLHF framework provided in this embodiment of this application may be applied to RLHF training scenarios in a cloud environment with a single compute resource (only a GPU or an NPU) or with hybrid compute resources (GPU+NPU) or (GPU+NPU+CPU). The following describes the present invention in detail with reference to an embodiment of an LLM having a parameter size on the order of tens of billions. The following embodiments will help a person skilled in the art further understand the present invention, but do not limit the present invention in any form. It should be noted that a person of ordinary skill in the art may further make several variations and improvements without departing from the concept of the present invention. These all fall within the protection scope of the present invention.

[00106] FIG. 3 is a diagram of a structure of a model system according to an embodiment of this application.

[00107] As shown in FIG. 3, in an embodiment, the model system includes an inference module 310, a training module 320, an experience buffer 340, and a coordination controller 330.

[00108] The experience buffer 340 is stored in a memory of a compute resource, and the experience buffer 340 is configured to store a reinforcement learning experience sample.

[00109] In an inference process, the inference module 310 generates training data through self-sampling, and stores the training data in the experience buffer 340.

[00110] The experience buffer 340 receives sample data constructed by the inference module

310, and provides, for the training module 320, training data required for RLHF fine-tuning.

[00111] As shown in FIG. 3, the inference module 310 includes a policy model 313, a reference model 311, and a reward model 312.

[00112] The inference process of the inference module 310 is as follows: The policy model 313 autoregressively generates responses (responses) based on a prompt (prompt) of a raw sample buffer 314. The prompt (prompt) and the response (responses) are transmitted to the reference model 311 and the reward model 312, to respectively compute a KL penalty value and a reward value, and to store a generated sample {prompt, response, logprob, value, reward, timestamp} in the experience buffer 340.

[00113] The training module 320 includes a policy model 321 and a set model (SET Model) 322.

[00114] The training module 320 performs model initialization on the policy model 321 by using a supervised fine-tuning model (supervised fine-tuning Model, SFT Model) 322.

[00115] The training module 320 continuously samples a training sample from the experience buffer 340, evaluates a difference between a sample timestamp and a current version of the policy model being trained, and directly drops the data if the difference exceeds a given threshold (Staleness=2).

[00116] Further, in a distributed parallel RLHF framework, policy models (the policy model 313 and the policy model 321) of the training module 320 and the inference module 310 are different to some extent. A timestamp of a sample stored in the experience buffer 340 is a timestamp of the policy model 313 in the current inference module 310, and is used to record a source of a sample.

[00117] In a single compute resource scenario, both the training module 320 and the inference module 310 use a same hardware resource, for example, a GPU or an NPU.

[00118] In a hybrid compute resource scenario, the training module 320 uses a single type of compute resource, for example, a GPU or an NPU. The inference module 310 may use a plurality of types of compute resources, for example, a GPU+an NPU+a CPU.

[00119] Specifically, when the RLHF framework is constructed, one or more compute resources are selected for the inference module 310. Based on a communication problem of parallel training, a compute resource is selected for the training module 320.

[00120] For example, considering the video memory footprint and computational workload requirements for tens-of-billions-parameter LLM inference, available cloud compute resources on the HUAWEI CLOUD platform include NVIDIA A800 GPU resources and Ascend D910 NPU compute resources, both of which have a minimum inference resource requirement of a single card in a tens-of-billions-parameter LLM inference scenario. Ascend D910 NPU is a Huawei-

developed AI computing chip, and NVIDIA A800 GPU is a graphics processing unit developed by NVIDIA Corporation. One or more compute resources may be selected for the inference module. Due to the communication problem of parallel training, only one type of compute resources may be selected for the training module.

[00121] Optionally, in an embodiment, the policy model 313 exclusively occupies compute resources, and the reward model 312 shares compute resources with the reference model 311. In another embodiment, the reward model 312 and the reference model 311 exclusively occupy the compute resources.

[00122] The coordination controller 330 is deployed on a CPU. Based on throughputs of the training module 320 and the inference module 310, the coordination controller 330 adaptively and dynamically adjusts a compute resource ratio within the inference module 310 and a compute resource ratio between the training module 320 and the inference module 310 (adjusting a ratio of an inference node to a training node), to implement dynamic balance of data flow, ensure that the training module 320 and the inference module 310 run at full capacity, and maximize utilization of the compute resources.

[00123] Specifically, in an embodiment, within the inference module 310, a total throughput of the policy model 313 is required to be less than or equal to a total throughput of the reference model 311/the reward model 312. As shown in Formula 2, that is,

$$\begin{cases} T_p^{infer} \le T_r^{infer} \\ T_p^{infer} = t_p^{infer} \cdot N_p^{infer} \cdot 1 \\ T_r^{infer} = t_r^{infer} \cdot N_r^{infer} \cdot 1 \\ N_p^{infer} + N_r^{infer} \le N^{infer} \end{cases}. \quad \text{(Formula 4)}$$

[00124] In Formula 4, t_p^{infer} and N_p^{infer} respectively represent a single-card inference throughput and a quantity of cards of the policy model, t_r^{infer} and N_r^{infer} respectively represent a single-card inference throughput and a quantity of cards of the reference/reward model, and T_p^{infer} and T_r^{infer} respectively represent a total inference throughput of the policy model and the reference/reward model.

[00125] Between the training module 320 and the inference module 310, a total training throughput is required to be not greater than a total inference throughput, that is,

$$\begin{cases} T^{train} \le T^{infer} \\ T^{train} = t_p^{train} \cdot N_p^{train} \cdot 8 \\ T^{infer} = \min\left(T_p^{infer}, T_r^{infer}\right) \\ N_p^{train} \le N^{train} \end{cases}. \quad \text{(Formula 5)}$$

[00126] In Formula 5, T^{train} and T^{infer} respectively represent total throughputs of the training module and the inference module, and t_p^{train} and N_p^{train} respectively represent a

single-card throughput and a quantity of cards of model training.

[00127] An optimization objective of the coordination controller 330 is to maximize training and inference throughputs of the RLHF framework under a given training and inference resource, to calculate an optimal quantity of policy model inference resources N_p^{infer}, a quantity of reference/reward model inference resources N_r^{infer}, and a quantity of training resources N_p^{train}, as shown in Formula 1.

[00128] FIG. 4 is a schematic flowchart of an RLHF method according to an embodiment of this application.

[00129] S410: Select a required inference compute resource type, and set an upper limit for a quantity of resources (with reference to S210).

[00130] S420: Select a required training compute resource type, and set an upper limit for a quantity of resources (with reference to S220).

[00131] S430: Perform a trial run on RLHF by using a minimum training and inference unit, where a coordination controller computes an initial resource allocation ratio based on training and inference throughputs (with reference to S230).

[00132] S440: Adjust training nodes based on a computed resource allocation scheme, and continuously sample from an experience buffer to fine-tune a model.

[00133] Specifically, in the S440, an initial resource allocation ratio computed in S430 is initially used.

[00134] S450: The coordination controller computes an optimal resource allocation ratio between a current training node and a current inference node based on throughputs of a training node and an inference node (with reference to S230).

[00135] After the S450, the process returns to S440, and in S440, uses the optimal resource allocation ratio computed in S450.

[00136] In addition to a large model RLHF scenario, the RLHF framework provided in this embodiment of this application is also applicable to a general reinforcement learning training scenario in a large model scenario, thereby enabling flexible expansion of training and inference node resources in reinforcement learning, and improving training efficiency in reinforcement learning.

[00137] A person skilled in the art knows that, in addition to implementing the system and the apparatuses provided in the present invention purely in a form of computer-readable program code, the system and the apparatuses provided in the present invention may also be implemented in a form of a logic gate, a switch, an application-specific integrated circuit, a programmable logic controller, an embedded microcontroller, and the like, through logic programming method steps to achieve a same function. Therefore, the system and the apparatuses thereof provided in the present

invention may be considered as a hardware component, and an apparatus included in the system and configured to implement various functions may also be considered as a structure within a hardware component; or an apparatus configured to implement various functions may be considered either a software module for implementing a method or a structure within a hardware component.

[00138] The embodiments disclosed herein are merely example embodiments of the present invention. These embodiments are selected and described in detail in this specification to better explain the principle and actual application of the present invention, but are not intended to limit the present invention. Any modification or variation made by a person skilled in the art within the scope of the specification shall fall within the protection scope of the present invention.

[00139] In the descriptions of embodiments of this application, for ease of description, the apparatus is described by dividing functions into various modules. Division into the modules is merely logical function division. During implementation of embodiments of this application, the functions of the modules may be implemented in a same piece of or a plurality of pieces of software and/or hardware.

[00140] Specifically, in actual implementation, all or some of the apparatuses provided in embodiments of this application may be integrated into one physical entity (for example, a GPU or a processor of another type), or may be physically separated. In addition, the modules may be all implemented in a form of software invoking processing elements; or the modules may be all implemented in a form of hardware; or some modules may be implemented in a form of software invoking processing elements, and some modules may be implemented in a form of hardware. For example, a detection module may be an independently disposed processing element, or may be integrated in one chip of an electronic device for implementation. Implementations of other modules are similar. In addition, all or some of the modules may be integrated together, or may be implemented independently. In an implementation process, steps in the foregoing methods or the foregoing modules may be implemented by using a hardware integrated logical circuit in a processor element, or by using instructions in a form of software.

[00141] For example, the foregoing modules may be one or more integrated circuits configured to implement the foregoing method, for example, one or more application-specific integrated circuits (Application-Specific Integrated Circuit, ASIC), one or more digital signal processors (Digital Signal Processor, DSP), or one or more field programmable gate arrays (Field Programmable Gate Array, FPGA). For another example, these modules may be integrated together and implemented in a form of a system-on-a-chip (System-On-a-Chip, SOC).

[00142] An embodiment of this application further provides an electronic device. The electronic device is configured to perform the method procedures or some of the method procedures

according to embodiments of this application.

[00143] FIG. 5 is a diagram of a structure of an electronic device according to an embodiment of this application.

[00144] As shown in FIG. 5, the electronic device 2500 includes a memory 2502 configured to store computer program instructions and a processor 2501 configured to execute the program instructions. When the computer program instructions are executed by the processor 2501, the electronic device 2500 is triggered to perform the steps of the method described in embodiments of this application.

[00145] Specifically, in an embodiment of this application, one or more computer programs are stored in the memory 2502, the one or more computer programs include instructions, and when the instructions are executed by the electronic device 2500, the electronic device 2500 is enabled to perform the steps of the method described in embodiments of this application.

[00146] It may be understood that descriptions of the structure of the electronic device 2500 in this embodiment of this application do not constitute a specific limitation on the electronic device 2500. In some other embodiments of this application, the electronic device 2500 may include another component in addition to the processor 2501 and the memory 2502.

[00147] The processor 2501 may be a system-on-a-chip SOC apparatus. The processor 2501 may include a central processing unit (Central Processing Unit, CPU), and may further include a processor of another type.

[00148] The processor 2501 may include, for example, a CPU, a DSP, a microcontroller, or a digital signal processor, and may further include a GPU, an embedded neural-network processing unit (Neural-network Processing Unit, NPU), and an image signal processor (Image Signal Processor, ISP). The processor may further include a necessary hardware accelerator or a logic processing hardware circuit, such as an ASIC, or one or more integrated circuits configured to control program execution of the technical solutions in this application. In addition, the processor may have a function of executing one or more software programs. The software programs may be stored in a storage medium.

[00149] The processor 2501 may include one or more processing units. For example, the processor may include an application processor (application processor, AP), a modem processor, a graphics processing unit (graphics processing unit, GPU), an image signal processor (image signal processor, ISP), a controller, a video codec, a digital signal processor (digital signal processor, DSP), a baseband processor, a neural-network processing unit (neural-network processing unit, NPU), and/or the like. Different processing units may be independent components, or may be integrated into one or more processors. In some embodiments, the electronic device 2500 may also include one or more processors 2501. The controller may generate an operation

control signal based on instruction operation code and a time sequence signal, to complete control of instruction fetching and instruction execution.

[00150] In some embodiments, the processor 2501 may include one or more interfaces. The interface may include an inter-integrated circuit (inter-integrated circuit, I2C) interface, an integrated circuit sound (integrated circuit sound, I2S) interface, a pulse code modulation (pulse code modulation, PCM) interface, a universal asynchronous receiver/transmitter (universal asynchronous receiver/transmitter, UART) interface, a mobile industry processor interface (mobile industry processor interface, MIPI), a general-purpose input/output (general-purpose input/output, GPIO) interface, a SIM card interface, a USB interface, and/or the like. The USB interface is an interface that conforms to a USB standard specification, and may be specifically a mini USB interface, a micro USB interface, a USB Type-C interface, or the like. The USB interface may be configured to connect to a charger to charge the electronic device, or may be configured to transmit data between the electronic device and a peripheral device.

[00151] The electronic device 2500 may further include an interface for external memory. The interface of external memory may be configured to connect to an external memory card, for example, a micro SD card, to expand storage capability of the electronic device. The external memory card communicates with the processor 2501 through the interface for external memory, to implement a data storage function. For example, files such as music and videos are stored in the external memory card.

[00152] The memory 2502 may include a code storage area and a data storage area. The code storage area may store an operating system. The data storage area may store data created in a process of using the electronic device 2500, and the like. In addition, the memory 2502 may include a high-speed random access memory, and may further include a non-volatile memory, such as one or more magnetic disk storage components, a flash memory component, or a universal flash storage (universal flash storage, UFS).

[00153] The memory 2502 may be a read-only memory (read-only memory, ROM) or another type of static storage device that may store static information and instructions, a random access memory (random access memory, RAM) or another type of dynamic storage device that may store information and instructions, an electrically erasable programmable read-only memory (electrically erasable programmable read-only memory, EEPROM), a compact disc read-only memory (compact disc read-only memory, CD-ROM) or another optical disk storage, an optical disc storage (including a compact disc, a laser disc, an optical disc, a digital versatile disc, a Blu-ray disc, or the like), a disk storage medium or another magnetic storage device, or any other medium that can be used to carry or store desired program code in a form of instructions or a data structure and that can be accessed by a computer.

[00154] The processor 2501 and the memory 2502 may be integrated into one processing apparatus, or more commonly, may be components independent of each other.

[00155] An embodiment of this application further provides an electronic chip. The electronic chip is configured to perform the method procedures or some of the method procedures according to embodiments of this application. For example, the electronic chip may be a GPU, a CPU, or an NPU.

[00156] Specifically, the electronic chip includes a processor configured to execute program instructions. When the computer program instructions are executed by the processor, the electronic chip is triggered to perform the steps of the method described in embodiments of this application. For the processor of the electronic chip, refer to the processor of the electronic device described above.

[00157] Optionally, the device, apparatus, and module described in embodiments of this application may be specifically implemented by a computer chip or an entity, or may be implemented by a product having a specific function.

[00158] A person skilled in the art should understand that embodiments of this application may be provided as a method, an apparatus, or a computer program product. Therefore, the present invention may be in a form of a hardware-only embodiment, a software-only embodiment, or an embodiment combining software and hardware. In addition, the present invention may use a form of a computer program product that is implemented on one or more computer-usable storage media that include computer-usable program code.

[00159] In the several embodiments provided in this application, when any function is implemented in a form of a software functional unit and sold or used as an independent product, the function may be stored in a computer-readable storage medium. Based on such an understanding, the technical solutions of this application essentially, or the part contributing to the conventional technology, or a part of the technical solutions may be implemented in a form of a software product. The computer software product is stored in a storage medium, and includes several instructions for instructing a computer device (which may be a personal computer, a server, a network device, or the like) to perform all or some of the steps of the method described in embodiments of this application.

[00160] Specifically, an embodiment of this application further provides a computer-readable storage medium. The computer-readable storage medium stores a computer program, and when the computer program is run on a computer, the computer is enabled to perform the method provided in embodiments of this application.

[00161] An embodiment of this application further provides a computer program product. The computer program product includes a computer program, and when the computer program is run

on a computer, the computer is enabled to perform the method provided in embodiments of this application.

[00162] Embodiments of this application are described with reference to the flowcharts and/or block diagrams of the method, the device (apparatus), and the computer program product according to embodiments of this application. It should be understood that computer program instructions may be used to implement each process and/or each block in the flowcharts and/or the block diagrams and a combination of a process and/or a block in the flowcharts and/or the block diagrams. These computer program instructions may be provided for a general-purpose computer, a special-purpose computer, an embedded processor, or a processor of any other programmable data processing device to generate a machine, so that the instructions executed by a computer or a processor of any other programmable data processing device generate an apparatus for implementing a specific function in one or more procedures in the flowcharts and/or in one or more blocks in the block diagrams.

[00163] These computer program instructions may be stored in a computer-readable memory that can instruct the computer or any other programmable data processing device to work in a specific manner, so that the instructions stored in the computer-readable memory generate an artifact that includes an instruction apparatus. The instruction apparatus implements a function specified in one or more procedures in the flowcharts and/or in one or more blocks in the block diagrams.

[00164] The computer program instructions may alternatively be loaded onto a computer or another programmable data processing device, so that a series of operations and steps are performed on the computer or the another programmable device, so that computer-implemented processing is generated. Therefore, the instructions executed on the computer or the another programmable device provide steps for implementing a specific function in one or more procedures in the flowcharts and/or in one or more blocks in the block diagrams.

[00165] It should be noted that in embodiments of this application, "at least one" means one or more, and "a plurality of" means two or more. "And/or" describes an association relationship between associated objects and indicates that three relationships may exist. For example, A and/or B may indicate the following cases: Only A exists, both A and B exist, and only B exists, where A and B may be singular or plural. The character "/" generally indicates an "or" relationship between the associated objects. "At least one of the following items" or a similar expression thereof means any combination of these terms, including any combination of single or plural terms. For example, at least one of a, b, or c may indicate a, b, and c, a and b, a and c, b and c, or a, b, and c, where a, b, and c may be singular or plural.

[00166] In embodiments of this application, the term "include", "contain", or any other variant

thereof is intended to cover a non-exclusive inclusion, so that a process, a method, a commodity, or a device that includes a series of elements not only includes those elements but also includes other elements not expressly listed, or further includes elements inherent to such a process, method, commodity, or device. An element preceded by "includes a ..." does not, without more constraints, preclude an existence of additional identical elements in the process, method, commodity, or device that includes the element.

[00167] This application may be described in the general context of computer-executable instructions executed by a computer, for example, a program module. Generally, the program module includes a routine, a program, an object, a component, a data structure, and the like for executing a particular task or implementing a particular abstract data type. This application may also be practiced in distributed computing environments in which tasks are performed by remote processing devices that are connected through a communication network. In a distributed computing environment, program modules may be located in both local and remote computer storage media, including storage devices.

[00168] Embodiments in this application are described in a progressive manner, same and similar parts of embodiments may be mutually referenced, and each embodiment focuses on differences from other embodiments. In particular, an apparatus embodiment is basically similar to a method embodiment, and therefore is described briefly; for related parts, refer to partial descriptions in the method embodiment.

[00169] A person of ordinary skill in the art may be aware that units and algorithm steps described in embodiments of this application can be implemented by electronic hardware, or a combination of computer software and electronic hardware. Whether the functions are performed by hardware or software depends on particular applications and design constraint conditions of the technical solutions. A person skilled in the art may use different methods to implement the described functions for each particular application, but it should not be considered that the implementation goes beyond the scope of this application.

[00170] It may be clearly understood by a person skilled in the art that, for the purpose of convenient and brief description, for a detailed working process of the foregoing apparatuses, devices, and units, reference may be made to a corresponding process in the foregoing method embodiments. Details are not described herein again.

[00171] The foregoing descriptions are merely specific implementations of this application. Any variation or replacement readily figured out by a person skilled in the art within the technical scope disclosed in this application shall fall within the protection scope of this application. Therefore, the protection scope of this application shall be subject to the protection scope of the claims.