

END-INTELLIGENT EXPERIENCE IMPLEMENTATION METHOD AND APPARATUS, AND MEDIUM AND DEVICE

TECHNICAL FIELD

5 **[0001]** This application relates to the technical field of application development, and in particular, to an end-intelligent experience implementation method and apparatus, a medium, and a device.

BACKGROUND

10 **[0002]** Users engage in work, communication, shopping, and other activities on terminals (e.g., smartphones) via various clients (e.g., APPs, web pages, websites, mini-programs, etc.), which is an important part of modern life. The real-time experience of users when using clients may involve a series of functions such as interface design, page loading, function response, usability, smoothness, and user feedback. How to develop and improve applications based on enhancing user experience is a technical issue that needs to be
15 considered by those skilled in the art.

SUMMARY

[0003] In view of this, the present application provides an end-intelligent experience implementation method and apparatus, a medium, and an electronic device, with the main purpose of improving user experience by implementing end-side intelligence.

20 **[0004]** According to an aspect of the present application, an end-intelligent experience implementation method is provided, which is configured for front-end code applicable to multiple types of clients to perform perception, decision-making and handling of user experiences at a front end, where the method is applied to a client among any type of clients in the multiple types of clients and includes:

25 **[0005]** starting the client, sending an end-intelligent experience configuration request to a server, and obtaining an end-intelligent experience configuration and storing the end-

intelligent experience configuration locally; and

[0006] running an end-intelligent suite in the front-end code, including: starting a perception engine in the end-intelligent suite to obtain target data in real time; triggering a decision engine in the end-intelligent suite to make an experience decision on the target data based on the end-intelligent experience configuration; and running a touchpoint engine in the end-intelligent suite to trigger a corresponding handling scheme according to an experience decision result.

[0007] According to another aspect of the present application, an end-intelligent experience implementation apparatus is provided, which is configured for front-end code applicable to multiple types of clients to perform perception, decision-making and handling of user experiences at a front end, where the apparatus is applied to a client among any type of clients in the multiple types of clients and includes:

[0008] a configuration request unit, configured to start the client, send an end-intelligent experience configuration request to a server, and obtain an end-intelligent experience configuration and store the end-intelligent experience configuration locally; and

[0009] an end-intelligent execution unit, configured to run an end-intelligent suite in the front-end code, including: starting a perception engine in the end-intelligent suite to obtain target data in real time; triggering a decision engine in the end-intelligent suite to make an experience decision on the target data based on the end-intelligent experience configuration; and running a touchpoint engine in the end-intelligent suite to trigger a corresponding handling scheme according to an experience decision result.

[0010] According to an aspect of the present application, a storage medium is provided, which stores a computer program, where the computer program is configured to perform, when running, the above-mentioned end-intelligent experience implementation method.

[0011] According to an aspect of the present application, an electronic device is provided, which includes a memory and one or more processors, where the memory stores a computer program, and the one or more processors are configured to run the computer program to perform the above-mentioned end-intelligent experience implementation method.

BRIEF DESCRIPTION OF DRAWINGS

[0012] FIG. 1 shows a flowchart of an end-intelligent experience implementation method according to an embodiment of the present application.

5 [0013] FIG. 2 shows a schematic diagram of end-intelligent experience handling schemes according to an embodiment of the present application.

[0014] FIG. 3 shows a flowchart of an end-intelligent experience monitoring method according to an embodiment of the present application.

10 [0015] FIG. 4 shows a schematic diagram of complex event configuration and matching logic in end-intelligent experience monitoring according to an embodiment of the present application.

[0016] FIG. 5 shows a flowchart of timing matching in end-intelligent experience monitoring according to an embodiment of the present application.

[0017] FIG. 6 shows a flowchart of an end-intelligent experience stratification method according to an embodiment of the present application.

15 [0018] FIG. 7 shows a schematic diagram of an end-intelligent experience stratification implementation logic according to an embodiment of the present application.

[0019] FIG. 8 shows a flowchart of an end-intelligent experience enhancement method according to an embodiment of the present application.

20 [0020] FIG. 9 shows a flowchart of another end-intelligent experience implementation method according to an embodiment of the present application.

[0021] FIG. 10 shows a schematic structural diagram of an end-intelligent experience implementation apparatus according to an embodiment of the present application.

DETAILED DESCRIPTION

25 [0022] To enable those skilled in the art to better understand the solutions of the present application, the technical solutions in the embodiments of the present application will be

clearly and completely described below in conjunction with the accompanying drawings of the embodiments of the present application.

[0023] All types of clients have user experience issues. Therefore, how to improve user experience is a subject that requires continuous research in application development.

5 **[0024]** Traditional HTML 5 application development offers high efficiency but insufficient user performance experience. Consequently, attempts have emerged to develop Web applications with user experiences similar to native applications using Web technologies. The advent of MiniApps has quickly gained popularity among users. A MiniApp is a small, install-free, and fast-loading program that typically runs within a host application or
10 operating system (e.g., a mini-program, a quick app), or can be a JS-native application supporting cross-end deployment. However, limited by the architecture, it is difficult for a MiniApp to achieve the smooth experience of a native application. Particularly for large-scale MiniApps, dynamic experience degradation during user operations is a difficult problem to solve. Since user experience issues of MiniApps are relatively prominent, the
15 embodiments of the present application take MiniApps as an example for illustration. It should be understood, however, that the solutions are also applicable to other types of clients.

[0025] In existing solutions, user experience improvement is usually achieved by collecting data at the front end and performing data calculation at the server end and then feeding back to the front end. This approach, which can be referred to as a server intelligence or cloud
20 intelligence (cloud AI) solution, although featuring strong data processing capabilities, has limitations such as high latency and high maintenance costs.

[0026] Therefore, the inventors of the present application innovatively propose improving user experience through end-side intelligence (also referred to as "mobile intelligence" or "front-end intelligence," hereinafter abbreviated as "end intelligence"). End intelligence can
25 be understood as the ability to think and make decisions on the end side. In short, end intelligence involves real-time perception, calculation, decision-making, and intervention on the end side. By equipping the end side with machine learning capabilities, user experience is enhanced. The end-intelligent experience in the embodiments of the present invention is a comprehensive solution that deeply perceives and understands user behaviors through
30 technologies such as data analysis and machine learning, globally detects and coordinates

various user device environments and complex operating scenarios, and optimizes client product experience through real-time feedback loops and timely adjustment interventions. The end-intelligent experience features real-time performance, intelligence and handleability.

- 5 **[0027]** Through analysis and practice, end intelligence has outstanding technical advantages:
1. real-time performance: the real-time performance on the end side is incomparable to that on the cloud, and for many interactive and streaming scenarios, cloud latency can hardly meet real-time experience requirements; 2. offline capability: although 4G and 5G have become widespread, unstable network connections or even no network access often occur
- 10 in real-life situations (e.g., in cars, on planes, or in suburban areas), and end intelligence can effectively avoid problems caused by poor networks; 3. data privacy: due to data privacy requirements, much user data cannot be uploaded to the cloud and thus can only be used on the end side; 4. cost efficiency: while cloud computing provides massive computing power, the costs of AI computing and network bandwidth are extremely high when serving a large
- 15 number of users, and currently, with the popularization of end-side devices, there are numerous devices (e.g., mobile phones, smart cars, set-top boxes) with computing capabilities available, which can reduce the cost of cloud deployment.

- [0028]** The inventors of the present application further researched and found that there are several challenges in implementing end intelligence. The first is the package size challenge:
- 20 implementing technologies on the end side involves occupying the installation package size and storage resources of the APP, which imposes high requirements on capabilities such as end computing engine slimming, end model size reduction, end model compression and dynamic release. The second is the performance challenge: since computing resources of mobile devices are used, it is necessary to balance user experience and computing efficiency,
- 25 requiring continuous optimization for central processing unit/graphics processing unit (CPU/GPU), memory, power consumption, instruction sets, thread scheduling, etc. of mobile devices. The third is the adaptation challenge: facing fragmentation of devices and system versions, end intelligence technology requires extensive adaptation to ensure stability and availability during operation. Therefore, the embodiments of the present
- 30 application propose an end-intelligent experience implementation solution, aiming to overcome the defects of cloud AI and optimize the implementation effect of the solution by

addressing the above challenges.

[0029] In addition, in existing network applications, the same network product often corresponds to different types of clients (released through different channels). For example, in addition to a dedicated APP, a delivery product also has multiple client forms such as MiniApp, website, and web page. For the front-end code of the same product, if different end-intelligent experience mechanisms are adopted for different types of clients, code development and release efficiency will be significantly affected. Therefore, the embodiments of the present application further improve in this aspect to achieve a set of end-intelligent experience implementation solutions applicable to cross-end scenarios. To develop a solution applicable to multiple types of clients, the embodiments of the present application rely on cross-end compatible programming languages (e.g., JavaScript, JS) for front-end code development. Since JS development is compatible with various client types such as APP, MiniApp and H5, cross-end applicability can be achieved. In the specific implementation process, issues such as processing the source code to eliminate differences between various types of clients also need to be considered.

[0030] Referring to FIG. 1, which is a flowchart of an end-intelligent experience implementation method in an embodiment of the present application. The method is configured for front-end code applicable to multiple types of clients to perform perception, decision-making and handling of user experiences at the front end. The method is applied to a client among any type of clients in the multiple types of clients, and the method includes the following steps S101-S102.

[0031] In the embodiment of the present application, the front-end code applicable to multiple types of clients (also referred to as "cross-end"), such as JavaScript (JS) front-end code, is obtained by statically compiling the source code to eliminate differences between various types of clients. Static compilation means that when a compiler compiles an executable file, the compiler extracts the parts of the corresponding static library that the executable file needs to call and links the parts to the executable file, so that the executable file does not depend on dynamic link libraries during operation. In the embodiment of the present application, static compilation of the source code mainly deals with parts of the JS source code that have strong constraints and cannot be dynamically modified, which may include three aspects: 1. module reference: for example, replacing module references in the

JS source code and modifying suffix names; 2. template attribute mapping or syntax compatibility processing: for example, in JS, template attribute mapping usually refers to mapping object attributes to template strings to generate a complete string, which can be used to create dynamic HTML strings or other types of text; and in JS, the purpose of syntax compatibility processing is to ensure the source code can run in different environments, requiring attention to potential syntax differences across environments, and syntax compatibility processing can adopt multiple methods, such as using strict mode, using JSON objects for serialization and deserialization, using optional chaining operators, using specific syntax, and using template import/export; 3. configuration mapping: for example, through page configuration. Through the above operations, differences between various types of clients are eliminated, providing a foundation for implementing cross-end end-intelligent experience.

[0032] S101: starting the client, sending an end-intelligent experience configuration request to a server, obtaining an end-intelligent experience configuration, and storing the end-intelligent experience configuration locally.

[0033] S102: running an end-intelligent suite in the front-end code, including: starting a perception engine in the end-intelligent suite to obtain target data in real time; triggering a decision engine in the end-intelligent suite to make an experience decision on the target data based on the end-intelligent experience configuration; and running a touchpoint engine in the end-intelligent suite to trigger a corresponding handling scheme according to an experience decision result.

[0034] In the embodiment of the present application, to reduce the computational processing pressure of the end, the end-intelligent experience configuration is preconfigured on the server. After the client is started, the end-intelligent experience configuration is loaded and stored locally to prepare for subsequent local operations such as decision-making.

[0035] The end-intelligent experience configuration preconfigured on the server (or cloud) can be understood as including two dimensions of configuration: 1. event and rule configuration: the server needs to predefine various user behavior events and related rules, including event trigger conditions, rule execution logic, etc.; 2. timing and strategy configuration: in terms of scenarios, timing refers to the execution of a specific operation

when a series of conditions are met; in terms of implementation, timing consists of complex event (composite event) rules plus trigger configuration; strategy refers to the conditions that the trigger timing needs to satisfy, or the deployment of a composite event rule model: for strategies requiring machine learning model support, the server is responsible for model
5 deployment and delivery. These models are trained based on a large amount of historical data to predict user behavior or evaluate user experience quality.

[0036] In client code development, an end-intelligent suite is embedded. The end-intelligent suite can be understood as a collection of functional modules for implementing end-intelligent experience. The end-intelligent suite deeply perceives and understands user
10 behaviors, combines technologies such as data analysis and machine learning, comprehensively detects and coordinates the user device environments and operation scenarios, and optimizes client experience through real-time feedback loops and timely adjustment interventions. A perception engine, a decision engine and a touchpoint engine are built in the end-intelligent suite. According to the end-intelligent experience
15 configuration loaded locally, the corresponding engines are automatically triggered at relevant processing nodes to achieve real-time perception, real-time decision-making and real-time handling of data. In an implementation, the end-intelligent suite can be developed based on an end-intelligent experience architecture. The end-intelligent experience architecture may include the following parts.

[0037] 1. End-intelligent experience solution layer: a set of solution strategies and tools provided for specific problems. Relying on the monitoring and decision-making capabilities provided by basic capabilities, the end-intelligent experience solution layer specifically solves various problems encountered by users during use. Classified by experience
20 dimensions, the end-intelligent experience solution layer may include three functional modules: experience monitoring, experience stratification and experience enhancement.
25

[0038] 2. End-intelligent experience plug-in system: a platform that provides a plug-in market and plug-in development capabilities, allowing third-party or internal developers to develop plug-ins for specific functions. This system can be regarded as an extension platform, relying on services and application programming interfaces (APIs) provided by
30 the basic architecture and solutions for plug-in development and distribution.

[0039] 3. End-intelligent experience core: the foundation for operation of the entire system, which may consist of four parts, i.e., perception, modeling, decision-making and handling, and is responsible for collecting and processing information and making intelligent responses. This core directly supports the intelligent implementation of experience monitoring, experience enhancement and experience stratification.

[0040] 4. Cross-end intelligent layer: an advanced functional layer of the system, which integrates capabilities such as unified event stream, model inference, feature engineering, decision engine and touchpoint engine.

[0041] The logical relationships between the above parts and layers reflect the service hierarchy from basic to advanced. Each layer provides support and services for the upper layer, and they work together to ensure the overall intelligence and efficient operation of the system.

[0042] It can be seen that the end-intelligent suite developed through the above end-intelligent experience architecture is constructed based on a unified experience foundation. Different from scattered solutions (e.g., developing specific solutions for individual experience problems), the end-intelligent experience implementation solution models based on a unified experience foundation. This approach of using a unified experience foundation offers the advantages of cost reduction and efficiency improvement. Moreover, at the scenario level, personalized scenario requirements can be met simply by calling the foundation and rendering, featuring scalability and personalization advantages.

[0043] Below is a detailed description of the three main actions (data collection, decision-making, and handling) in S102.

[0044] 1. Data Collection

[0045] The end-intelligent suite includes a perception engine for data collection. Perceiving real-time user data on end-side is the foundation of end-intelligent experience. The target data in S102 refers to a collection of different types of data that need to be perceived in different experience solutions (including, for example, experience monitoring, experience stratification, and experience enhancement, which will be detailed later). Generally, the target data may include user basic data, user behavior data, scenario data, performance data,

etc.

[0046] User basic data can be obtained through certain APIs, such as using `getSystemInfo` to obtain the device model, using `getNetworkType` to obtain the current network type, and using `onNetworkStatusChange` to monitor events of network status changes.

5 **[0047]** User behavior data can be tracked through user tracking (UT). Based on the event identifiers to which the behaviors belong, the data is placed in different event streams for tracking and reporting. User behavior data is relatively complex and requires specific calculation logic. Taking the "page stay duration event" as an example, it involves the following key points: (1) timing starts when the user enters the page, and the page stay event
10 is triggered after the user browses the page for more than xx seconds; (2) if the stay duration of the user in the page does not meet the standard (i.e., the user exits the page), the stay duration timing pauses and resumes accumulating when the user returns to the page; (3) at most one page stay event is triggered during a single page access (from `pageAppear` to `pageDisappear`).

15 **[0048]** Scenario data may involve behaviors such as page access (2001), click (2101), and exposure (2201). Other behaviors are classified into custom behaviors, and different scenario data is organized according to specific scenario rules.

[0049] For performance data, for example, interface latency can be collected and reported through the basic network software development kit (SDK); rendering-related performance
20 data can rely on the client framework and container (e.g., MiniApp framework and container), and combined with the rendering scenario of the scene/page, data collection and reporting at each stage are completed. Taking MiniApps as an example, it is specifically divided into: (1) MiniApp performance plug-in: which collects the time consumption of application/page startup and standard lifecycle, and reports automatically, among which:
25 (1.1) extended capability provides performance data of partial container startup (mainly startup start time and routing latency) through the MiniApp `performanceApi`; (1.2) for some channels lacking container collection APIs (making it impossible to obtain the startup time point), the scenario T2 latency with the application's `onLaunch` as the start point can be customized; (2) MiniApp performance component mini-performance: which collects the end
30 point of scenario T2 latency, among which: (2.1) settlement method: the component is based

on the IntersectionObserver API and is connected to the "last module of the first-screen complete rendering" in the scenario, and the component records the event (and time) when it is visible (and rendered) in the view as the settlement point for simulated T2, with no additional performance overhead. In addition, the MiniApp framework provides the onError method of the App object to monitor global script errors, and onPromiseError to monitor all uncaught Promise exceptions. For page white screen monitoring, a custom method can be adopted: a timer is started after the interface request is completed and data is returned; if no elements are successfully rendered on the page within the subsequent seconds, this situation is regarded as a page white screen and corresponding statistics are made.

10 [0050] 2. Decision-Making

[0051] The end-intelligent suite includes a decision engine for making experience decisions, i.e., making experience decisions on real-time collected target data based on the downloaded end-intelligent experience configuration. The end-intelligent experience configuration may include event configuration information or model configuration information, so as to perform event matching based on the event configuration information or prediction based on the model. The event configuration information may be complex event configuration information (corresponding to the "experience monitoring" solution, detailed later). The model may be an experience stratification model (corresponding to the "experience stratification" solution, detailed later) or a full-link trajectory prediction model (corresponding to the "experience enhancement" solution, detailed later). In the embodiments of this invention, event configuration, model training and deployment are offline, while inference is performed at the front end. Front-end enables faster response speed, reduces network request latency, protects user data privacy, and allows services to be provided even in environments with no network connection.

25 [0052] 3. Handling

[0053] The end-intelligent suite includes a touchpoint engine for triggering experience handling based on an experience decision result. In the end-intelligent experience solution provided in the embodiments of the present application, after problems are perceived through data collection and experience decision-making, the problems can be directly solved or reported on the end side, providing comprehensive and flexible handling capabilities. For

example, in some cases, user interface (UI) forms (such as pop-up windows and bottom bars) are required to interact with users, enabling real-time message pushes on the application interface to inform users of important information or operation feedback (e.g., real-time message prompts, user guidance). The end-intelligent experience framework provides complete UI components, which are highly customizable. These components are built into the host in the form of dynamic plug-ins/sub-packages, enabling rapid delivery of information to users. The end-intelligent experience framework can intercept network request data during sending or receiving, customize request headers or process returned data, thereby enabling flexible network intervention. The end-intelligent experience framework can dynamically proxy and rewrite the basic library api in different scenarios to extend or modify original behaviors. For example, it is possible to proxy changes to `hashToUrl` to return images of different qualities for device models in different operation status.

[0054] Through the above three steps, real-time perception and handling of user experiences can be achieved on the end side. The following uses a specific "experience stratification" example to illustrate the above steps.

[0055] Step 1: preparation and model training. Data collection: collect user behavior data of page access, including information such as page loading time, user device type, network environment, etc. Feature extraction: extract key features from the collected data, such as device performance indicators (CPU, memory usage rate), network speed, etc. Model training: use these features and page loading time as training data to build a machine learning model for predicting the operation status of different device models. Model release: convert the trained model to the open neural network exchange (ONNX) format and release the model file to the content delivery network (CDN) for loading and prediction by the decision engine.

[0056] Step 2: real-time feature collection. Data collection: real-time monitor and collect data such as page loading time, device CPU usage rate and memory usage rate when users use the application. Data normalization: adjust the value range of the data to a specific interval (usually 0–1 or -1–1).

[0057] Step 3: decision-making using the model. Using the collected feature data, the operation status of the device model is determined based on the prediction result of the

model. If the prediction result exceeds the set threshold, the device model is determined to be in a low operation status (poor or average operation status).

[0058] Step 4: scenario handling. Interface loading optimization: for device models determined to be in poor operation status, automatically optimize the network loading strategy (e.g., delaying the loading of non-critical interfaces). Content adaptation: dynamically adjust page content according to the operation status of the device (e.g., reducing image quality or simplifying page elements). Data feedback: continuously collect handling results and user experience feedback as new data input to continuously optimize the model and strategy.

[0059] As described earlier, from the perspective of user experience, there are three dimensions: experience monitoring, experience stratification and experience enhancement. Referring to FIG. 2, which is a schematic diagram of an end-intelligent experience solution according to an embodiment of the present application. The experience monitoring is to proactively discover user experience problems (e.g., whether a specific complex event occurs) in the process of using the client through real-time collection, analysis and early warning of user experience data, and performs real-time feedback and on-site handling to improve the user experience. The experience stratification is to evaluate the user device operation status from the perspective of actual user usage, and different from the previous rigid device classification based on factory hardware information, intelligent experience stratification uses multi-dimensional latency data of the actual operation of the application as the basis, and uses an experience stratification model (obtained through the end-intelligent experience configuration) to perform comprehensive real-time weighted scoring on user device data and conducts corresponding handling based on the score. The experience enhancement is to use a full-link trajectory prediction model (obtained through the end-intelligent experience configuration) to realize, based on real-time user interaction behavior information, real-time prediction of the future trajectory of the user and performs interventions such as preloading sub-packages and pages that the user is likely to access to enhance the sense of surprise in user experience.

[0060] Therefore, in an implementation, the above handling schemes include: an experience monitoring handling scheme, an experience stratification handling scheme, and/or an experience enhancement handling scheme.

[0061] The experience monitoring handling scheme includes: in response to determining according to the experience decision that a complex event occurs, performing real-time message push in an interface through UI interaction and/or performing a page jump operation.

5 [0062] The experience stratification handling scheme includes: in response to determining according to the experience decision that a device operation status is a low operation status, performing interface loading optimization and/or a dynamic page content adjustment scheme. The prediction results of the experience decision can be divided into three levels: good operation status, average operation status, and poor operation status. The "good
10 operation status" belongs to a high operation status, while the "average operation status" and the "poor operation status" belong to a low operation status. The "average operation status" can be understood as a medium-level operation status with room for experience improvement, while the "poor operation status" urgently requires experience improvement. Those skilled in the art can set and distinguish the operation status of each level according
15 to specific performance parameters. In response to determining a low operation status, corresponding handling schemes can be executed, for example, improving via interface loading optimization (including optimizing the network loading strategy) or dynamically adjusting page content (including reducing image quality, simplifying page elements, and/or simplifying page interactions).

20 [0063] The experience enhancement handling scheme includes: obtaining a future behavior trajectory of a user according to the experience decision, determining sub-package resources or page resources that the user is likely to access, and performing preloading operations on the sub-package resources or the page resources.

[0064] In an implementation, the priorities of different handling schemes can be set to
25 determine the execution logic relationship between them. Therefore, the above method further includes: setting different priorities for different handling schemes, and determining an execution logical relationship between different handling schemes according to the priorities. For example, the experience stratification handling scheme is set to correspond to a first priority, and whether to execute the experience monitoring handling scheme and/or
30 the experience enhancement handling scheme is determined according to a processing result of the experience stratification handling scheme. The advantage of this processing approach

is that the experience stratification handling scheme is executed first (e.g., reducing image quality for low-performance devices). After the above experience stratification handling, if the operation status of the device is still "poor" (which can be obtained by prediction through the experience stratification model again), it is considered inappropriate to perform experience monitoring and experience enhancement at this time to save the resource overhead and reduce the data processing pressure.

[0065] In addition, in an implementation, after executing the handling scheme, the method may further include: determining the executed handling scheme, collecting user feedback data, and sending the handling scheme and the user feedback data to the server as new input to enable the server to optimize the end-intelligent experience configuration. Thus, handling results and user experience feedback are continuously collected as new data input to continuously optimize the model and strategy.

[0066] In addition, to optimize the implementation effect, a "white-box + black-box" mode is used for decision-making.

[0067] For a white-box model, the weights are fixed during online operation, and there is no need to load an inference model for inference. The white-box model is mainly used in the cold start phase when the user opens the client (e.g., MiniApp) to pre-screen low-performance devices, avoiding additional burden on the device caused by model operation. This strategy ensures that more advanced models can be used for further operation status diagnosis and experience optimization in subsequent processes. A black-box model is a deep network architecture constructed through methods such as multi-layer nonlinear transformation and abstract global feature learning, which simulates the layered information processing method of the neural system of the human brain and is used to solve tasks such as classification and regression. For example, in the embodiments of the present application, the full-link trajectory prediction model may mainly rely on a black-box model (e.g., a deep learning network such as long short-term memory (LSTM)) to predict the behavior path of a user. By analyzing the historical behavior data of the user, such model attempts to predict the next possible operation or preference of the user. Since deep learning models usually involve complex network structures and a large number of parameters, the internal decision logic is opaque to users, hence the name "black-box model". Such models play an important role in predicting the future behavior trajectory of a user and realizing intelligent preloading,

aiming to improve the fluency and personalization of user experience. For the experience stratification model, on the other hand, white-box models (e.g., the gradient boosting decision tree (GBDT) algorithm) are predominantly used to manage user experience stratification according to the operation status of the device. By analyzing the performance parameters of the device (e.g., memory usage, processor performance), the model can real-time evaluate the performance level of the device and adjust the performance of the application accordingly (e.g., reducing image quality or simplifying interaction effects) to adapt to devices of different performance levels. The decision process of the white-box model is transparent, allowing developers to clearly understand the decision logic and basis of the model, facilitating tuning and maintenance.

[0068] Therefore, in an implementation, an experience decision for experience monitoring, experience stratification or experience enhancement is made through a white-box model or a black-box model, a preliminary decision is made according to the white-box model, and whether to use the black-box model for further refined decision-making is determined according to a preliminary decision result of the white-box model.

[0069] In an implementation, an experience stratification model is set to a white-box model mode, and in a cold start phase of the client, the experience stratification model in the white-box model mode is run, and whether to continue executing the end-intelligent suite or close the end-intelligent suite is determined according to a prediction result of the experience stratification model. Accordingly, a full-link trajectory prediction model is set to a black-box model mode, and in a hot start phase of the client, the full-link trajectory prediction model in the black-box model mode is run.

[0070] It can be seen that the end-intelligent experience implementation method provided in the embodiments of the present application emphasizes a cross-end, real-time, and intelligent experience optimization method. The method not only focuses on improving a single aspect of performance but also starts from the overall perspective of user experience, providing better services by real-time perceiving the status of the user and dynamically adjusting the experience. The core advantages of the method include the following aspects.

[0071] 1. Cross-end intelligent experience. The method supports cross-end use (e.g., MiniApps, web ends with different technology stacks). By converting front-end code into

multi-end supported code through static compilation, consistent optimization effects on different platforms and devices can be provided.

[0072] 2. Real-time response capability. The method makes it possible to real-time perceive the behavior of the user and the device status (e.g., network conditions, memory usage) and make real-time adjustments based on this information. This real-time response capability continuously optimizes the experience during using by the user, rather than performing one-time optimization only when the page is loaded.

[0073] 3. Intelligent decision-making and automatic optimization. The end-intelligent experience framework has built-in black-box and white-box models, which can perform intelligent analysis based on collected data to make targeted optimization decisions. This data-driven optimization method can more accurately identify and solve performance bottlenecks.

[0074] 4. Global strategy support. The method provides a series of global strategies (e.g., image quality degradation, package preloading, weak network switching), which can be flexibly used in different scenarios to achieve end-to-end performance optimization.

[0075] In summary, compared with traditional performance optimization (which often focuses on technical optimization for specific problems, such as code splitting, lazy loading, and resource compression), the embodiments of the present invention based on the end-intelligent experience framework focus more on providing a comprehensive, dynamic, and adaptive optimization solution from the perspective of user experience through real-time data analysis and intelligent decision-making. This method can not only solve some limitations encountered in traditional performance optimization but also provide more personalized and high-quality services according to actual usage scenarios and user needs.

[0076] Below is an exemplary description of the end-intelligent experience implementation method provided in the embodiments of the present application from three perspectives: experience monitoring, experience stratification, and experience enhancement.

[0077] First, with reference to FIG. 3 to FIG. 5, the experience monitoring will be described.

[0078] The experience monitoring is to proactively discover user experience problems (e.g.,

whether a specific complex event occurs) in the process of using the client through real-time collection, analysis, and early warning of user experience data, and performs real-time feedback and on-site handling to improve the user experience.

5 [0079] Referring to FIG. 3, which shows a flowchart of an end-intelligent experience monitoring method according to an embodiment of the present application.

[0080] S301: starting a client, sending an end-intelligent experience configuration request to a server, obtaining complex event configuration information, and initializing the complex event configuration information, where the complex event configuration information includes event configuration information and timing configuration information of at least
10 one target complex event.

[0081] S302: starting a perception engine, capturing tracking data based on a unified event stream processing tool in a front-end code, and determining whether a target complex event corresponding to the event configuration information is triggered according to the tracking data.

15 [0082] S303: in response to triggering of the target complex event, running a decision engine to perform timing matching on the tracking data based on the timing configuration information, and determining whether a timing of the target complex event is satisfied.

[0083] S304: after the timing matching is successful, running a touchpoint engine to trigger an experience monitoring handling scheme corresponding to the target complex event.

20 [0084] A complex event can be understood as a composite event. For example, in practical applications in the field of big data analysis, it may be necessary to detect the occurrence of a "consecutive login failure event", which is essentially a composite of "login failure" events. Another example is detecting the order placement and payment behavior of a user, which is also a composite event: a "payment event" occurs within a certain period after an "order
25 placement event," which further includes a time constraint. Such composite event involving multiple individual events is referred to as a complex event.

[0085] In the embodiments of the present application, complex event configuration information can be pre-configured or real-time configured through an end-intelligent

configuration platform. This end-intelligent configuration platform, which can be a visual platform provided on the server, allows users (e.g., developers) to configure complex event information. For example, the end-intelligent configuration platform provides timing decision configuration, including event configuration and timing configuration. The event configuration refers to matching rules for end-side user events. For example, using "spm == a.b.c && click" to correspond to a click event at a fixed position. The timing configuration refers to a sequence of events used for complex matching of the occurrence sequence of end-side user event streams, such as "click to enter store", "add to cart", and "return to homepage". Notably, for personalized configuration of different types of clients, the end-intelligent configuration platform can be used to configure corresponding complex event sets for different types of clients. A complex event set includes multiple complex events, and each complex event includes event configuration information and timing configuration information.

[0086] Referring to FIG. 4, which is a schematic diagram of complex event configuration and matching logic in end-intelligent experience monitoring. In this example, complex event configuration (including event configuration and timing configuration) is first performed on the configuration management platform. The event configuration (which can be understood as feature configuration) refers to accumulation and statistics of user behaviors, such as the number of times a user enters a store within 3 days. The timing configuration refers to specific conditions that can trigger decision-making and responses, such as entering the store twice and adding items to the cart. After completing the configuration on the configuration management platform, the configuration information can be written into a database (DB), and then a configuration gateway service reads data from the DB. The client automatically loads the configuration information from this configuration gateway service after startup. After obtaining the configuration information, the end-intelligent suite of the client makes relevant decisions according to the configuration information. Specifically, the decision engine performs event mapping/filtering, pattern matching, behavior accumulation, etc., to complete the timing decision. After determining that the timing matching is successful, the touchpoint engine is run to trigger the corresponding handling scheme, for example, performing real-time message push in an interface through UI interaction, or performing a page jump operation automatically for a current operation failure event (such as switching to a pre-checkout counter after multiple payment failures).

[0087] When the client starts, it automatically loads the complex event configuration information. As in step S301, the client sends a request to the server to obtain and initialize the complex event configuration information, where the complex event configuration information includes event configuration information and timing configuration information for at least one target complex event. The complex event configuration request may include longitude and latitude information, client information and a user ID, where the client information refers to client type information. After receiving the complex event configuration request, the server can determine the client type based on the client information, thereby matching and delivering the complex event configuration information corresponding to that type of client.

[0088] In the embodiments of the present application, the unified event stream processing tool refers to an event stream standardization processing tool based on super position tracking. It is possible to uniformly summarize user behaviors into standard events according to the tracking information of various types of clients, including information fields such as tracking information, event ID (eventId) and parameter information (properties). Functions in other pages or plugins can obtain, by monitoring unified events, the operations performed by the current user on the client.

[0089] The super position tracking includes the following aspects of settings.

[0090] Event marking

[0091] For example, a.b.c.d is used to mark an event in a certain page, where the 'a' position represents the user client level, the 'b' position represents a certain page within the client, the 'c' position represents a certain block in a certain page, and the 'd' position represents a specific click position in a certain block.

[0092] Event type

[0093] The above a.b.c.d positions can only mark a specific location. To distinguish different event types and record different general parameter information based on specific event types, which can cover all statistical needs and help reduce the difficulty and resource consumption of data processing, three types such as page type, click type and exposure type can be adopted. The page type is to count user behaviors at the page granularity. When

entering and exiting a page, parameters such as user ID, device ID, timestamp, and entry or exit status need to be recorded, so that data such as unique visitor (UV), page view (PV) and user stay duration for each page can be counted. For the click type, long press, swipe, etc., are all considered click types. The exposure type refers to the exposure at a specific module level, such as product exposure and video exposure. Combined with click-type events for analysis, the click-through rate of a product, a post, or a video can be calculated.

[0094] Jump relationship

[0095] For the jump relationship, it is only necessary to record the relationship between the previous jump and the jump before the previous one in the uniform resource locator (URL).

For example, for "clicking the first product on the shopping cart page", a click event can be reported. The previous jump may have come from a certain product detail page, so the previous jump can be recorded. The jump before the previous one may have come from a certain product in the "You May Like" information flow on the homepage, so the jump before the previous one can be recorded. Based on this, the movement path of the user is connected in series.

[0096] Traffic channel

[0097] For example, if a channel on the homepage includes multiple pages, a traffic channel is introduced. For example, multiple pages included in a flash sale channel are all marked as a traffic channel. Then, by deduplicating the number of visitors across these pages, the total UV of this channel can be obtained. Other metrics such as clicks and total page visits can be processed using the same statistical logic.

[0098] After the unified event stream processing tool captures the tracking data in step S302, the method may further include: performing client type analysis and format analysis on the tracking data, and performing standardization processing on the tracking data so that data of different types of clients have a unified format and structure. Performing the standardization processing on the tracking data includes: parsing tracking information, event ID and parameter information of the tracking data; and performing unified format encapsulation on the tracking information, the event ID and the parameter information to convert the tracking information, the event ID and the parameter information into event information in a standard

format.

[0099] It can be seen that the unified event stream processing tool in the embodiments of the present application can be understood as a client standardized event stream processing tool, which uses super position tracking in cross-end (APP, MiniApp, or H5) source codes to capture tracking data, encapsulate the tracking data to convert the tracking data into standardized events. By introducing the unified event stream processing tool to monitor the target complex events triggered by the processing tool, when a target complex event occurs, the filtering and matching of the complex event timing decision engine is triggered. This unified event stream processing method is an important part of realizing cross-end processing of complex events. By capturing tracking data based on super position tracking, event information of various types and positions can be captured on the client, and by standardizing the tracking data, it becomes possible to subsequently use a unified complex event timing decision engine to perform complex event detection and handling based on the same set of complex event matching rule base and the same matching logic.

[0100] In the embodiments of the present application, the decision engine is used for timing decision-making for complex events (which can be called a "timing decision engine"). The decision engine can be a complex event processing component based on a self-developed complex event matching rule base and declared as a static function, where the self-developed complex event matching rule base can be developed using JavaScript to be compatible with multiple types of clients. The advantage of the self-developed complex event matching rule base is that personalized configuration or modification is possible according to scenario requirements, and specific personalized rule matching can be performed based on this complex event matching rule base. To implement a cross-end compatible timing decision engine, the complex event processing component can be declared as a static function through the aforementioned JS static compilation method to eliminate differences between multiple ends, thereby obtaining a cross-end timing decision engine.

[0101] In an implementation, the specific processing procedure of the complex event timing decision engine is to perform event processing and timing matching on the target complex events according to pre-configured complex event matching rules, where the complex event matching rules include event rules, pattern rules and timing rules. For example, for the triggered target complex events, event filtering is performed according to the event rules,

and the filtered events is converted to obtain event IDs in a standard format; pattern filtering is performed on the converted events according to the pattern rules, events corresponding to the timing rules are retained, and pattern matching is performed on the retained events, where rule matching is performed according to multiple rules and items corresponding to the pattern rules to determine whether a round of pattern matching is successful, and whether the pattern matching is successful is determined according to the progress of multiple rounds of pattern matching; after successful pattern matching, fatigue detection is performed according to the timing rules, where the latest fatigue detection is performed according to fatigue cache information, and if the fatigue detection passes, the timing matching is determined to be successful.

[0102] The pattern matching in complex event processing can be understood in two aspects: the characteristics of each simple event and the combination relationship between simple events. In addition, the function of the pattern can be expanded, such as the time limit for matching detection, whether each simple event can appear repeatedly, and for patterns where events can appear repeatedly, whether to skip subsequent matches after a match is found. Pattern matching detects consecutive events or discontinuous but successively occurring events according to the neighbor relationship and conditions in the events. The pattern may also have a time limit. If the matching conditions are not met within the set time range, the pattern matching will be overtime.

[0103] Referring to FIG. 5, which shows an example of a processing procedure of the timing decision engine.

[0104] S501: event triggering. The target complex events triggered by the unified event stream processing tool are monitored.

[0105] S502: event filtering. Filtering is performed according to event rules. For example, if only click events are configured, all exposure events and scroll events will be filtered out to reduce the pressure of subsequent matching.

[0106] S503: event conversion. The filtered events are converted into unified events that can be matched, and the front-end events are mapped to the event IDs configured in the backend.

[0107] S504: pattern filtering. Multiple complex rules exist for one timing, and each rule configures different events. To reduce matching pressure, further filtering is performed here to retain only the events used by this rule.

[0108] S505: pattern matching. For example, by using a depth-first traversal approach, events are matched in sequence, including multiple rules. Pattern matching further includes rule matching and item matching, where a rule refers to one of the complex rules, containing multiple items. For the item matching, the items correspond one-to-one with events configured in the backend. Bottom-level event matching is performed here, and matching results are returned hierarchically upon completion.

[0109] S506: matching progress check. The matching progress increments by one for each pattern matching completed, and when all pattern matchings are completed, the procedure proceeds to the next step.

[0110] S507: matching result. Whether the matching is successful or failed is determined.

[0111] S508: fatigue detection. Fatigue detection is performed if the matching is successful. For example, the rule may be set as: trigger placing an order three times within 3 days; if the number of orders within 3 days is less than or equal to 3, the fatigue is true, allowing subsequent callback actions; if the number of orders within 3 days is greater than 3, the fatigue is false, and subsequent callback actions are no longer allowed. Historical fatigue can be cached locally and will be loaded the next time the client is launched.

[0112] S509: decision completed.

[0113] In an implementation, after successful timing matching, a pre-registered callback function of the complex event timing decision engine can be automatically invoked. The touchpoint engine is started through the callback function to execute the handling scheme corresponding to the target complex event. As described earlier, the end-intelligent configuration platform can implement complex event configuration. Correspondingly, through the end-intelligent configuration platform, the timing matching success information of a target complex event can be received, and the corresponding handling scheme can be determined according to the scenario attributes of the target complex event and/or the end attributes of the client of the type corresponding to the target complex event. The scenario

attributes may include application scenario type information, and the end attributes of the client are used to distinguish client types. Therefore, through the above method, a personalized handling scheme for the target complex event can be implemented. For example, the handling scheme can be in the form of a prompt window or a floating layer.

- 5 **[0114]** Taking “automatically switching to the pre-checkout counter after three consecutive user payment failures” as an example, the experience monitoring is illustratively explained as follows.

10 **[0115]** Step 1: on the end-intelligent configuration platform, configure the user payment failure event (if already configured, it can be reused). Here, it is assumed that this event is Event F.

[0116] Step 2: on the end-intelligent configuration platform, set the occurrence count of the user payment failure event to three, and set the matching count of Event D to 3.

[0117] Step 3: after the client is started, the client requests configuration information and performs initialization.

- 15 **[0118]** Step 4: after the user fails to pay once (determined by the unified event stream processing tool through tracking), the timing decision engine is triggered for timing matching, and the count is recorded as 1.

[0119] Step 5: after the user fails to pay twice, the timing decision engine is triggered for timing matching, and the count is recorded as 2.

- 20 **[0120]** Step 6: after the user fails to pay three times, the timing decision engine is triggered for timing matching, and the count is recorded as 3.

[0121] Step 7: after the matching degree detection is passed, call the registered callback function.

- 25 **[0122]** Step 8: run the touchpoint engine to automatically switch to the front-end checkout counter. The front-end checkout counter allows users to select a specific payment platform on the merchant page to complete the payment, avoiding jumping to intermediate pages and helping to optimize the online banking payment experience. The front-end checkout counter

supports integration on Web, wireless application protocol (WAP), and App ends.

[0123] In the example of "three consecutive user login failure", the timing decision engine is an engine that performs matching according to matching rules and only performs data parsing and rule matching, and can be understood as a core component declared as a static function but caching part of the execution process. The matching rules and associations between complex events can be pre-configured on the server, while the intermediate steps in the execution process (triggering complex events and timing matching) are not associated and can be understood as independent black-box implementations. Taking "monitoring repeated clicks on the same position of a page without response" as another example, first, the monitoring strategy is configured. For example, the monitoring strategy is configured according to scenario needs, and matching rules and triggering timing are specified. The event matching logic is to distinguish pages through the 'ab' position, distinguish scenarios through the 'c' position, and distinguish slots through the 'd' position. Through a single timing configuration, all clickable areas on the page can be covered. The trigger strategy is three or more consecutive clicks within 2 seconds. Then, user abnormalities are reported in real-time through tracking, and the abnormalities that can be handled are intervened in real-time. For example, targeted processing is made for different positions, such as page degradation or redirecting to H5 page details. Thus, compared with traditional monitoring that hardly detects occasional abnormalities, the experience monitoring in the embodiments of the present application can identify problems in the first place based on user sessions and intervene before complaints arise, improving the user experience.

[0124] In addition, "intelligent correction of bill mobile phone numbers" is taken as an example. In a user order placement scenario, decisions are made based on the distribution of similarity scores. When it is determined that the user is likely to have entered the mobile phone number incorrectly (for example, the score is between 0.7 and 1), through the mobile phone number correction, the user is reminded to modify the mobile phone number with one click (which can be implemented on the front end). For example, the correction is performed through a bubble prompt, and the function of one-click entry is provided through predicting the mobile phone number the user will enter.

[0125] In addition, there are many other scenario examples for experience monitoring, such as automatically switching to mobile data in weak WiFi scenarios, which will not be

described one by one in the embodiments of this application.

[0126] In an implementation, the above method further includes the following steps: determining whether the target complex event is a real-time behavior event or a cumulative behavior event; in response to determining that the target complex event is a real-time behavior event, obtaining user behavior data in real-time, and performing complex event trigger judgment and timing matching in real-time based on tracking data corresponding to the user behavior data; in response to determining that the target complex event is a cumulative behavior event, obtaining user behavior data multiple times, and accumulating the user behavior data; and performing complex event trigger judgment and timing matching based on tracking data corresponding to the currently accumulated user behavior data. It can be seen that by using user behavior data as the basis for matching in the above manner, more intelligent decision-making can be realized. Different from processing real-time user behavior events, processing cumulative behavior events (such as placing an order 5 times within a week) can realize domain linkage, that is, accumulated user data can be used in the timing decision process, and a complete access scheme can be provided.

[0127] It can be seen that through the above experience monitoring technical solution, in the embodiments of the present application, complex events on the front end are detected and handled for JavaScript front-end code applicable to multiple types of clients. After startup, the client sends a complex event configuration request to the server to obtain and initialize complex event configuration information. Then, based on the unified event stream processing tool in the front-end code, the client captures tracking data and determines whether the target complex event corresponding to the event configuration information is triggered according to the tracking data. Next, in response to the triggering of the target complex event, the decision engine is run to perform timing matching on the tracking data according to the timing configuration information to determine whether the timing of the target complex event is satisfied. Finally, after the timing matching is successful, a callback function is triggered to execute the handling scheme corresponding to the target complex event. In the embodiment of the present application, compared with the method that mainly relies on the server for complex event detection and handling, processing complex events on the client enables real-time rule matching based on real-time acquired data, thereby reducing processing time and having the advantages of low latency and low maintenance

costs.

[0128] Moreover, different from the existing complex event processing solutions that can only be applied to one type of client, the embodiments of the present application improve in this aspect to realize a set of cross-end compatible complex event processing solutions. To achieve the complex event processing solutions applicable to multiple types of clients, the embodiments of the present application rely on JavaScript (JS) for front-end code development. Since JS development is compatible with various types of clients such as APPs, MiniApps and H5, the cross-end purpose can be achieved. A single event processing mechanism can be used universally for various types of clients, greatly reducing the development pressure.

[0129] In addition, to conveniently configure personalized complex events for different types of clients, visual configuration can be performed through an end-intelligent configuration platform, thereby conveniently configuring matching rules for complex events and making strategies more flexible and easier to use. With the end-intelligent configuration platform, a simple implementation method of "input configuration" – "output timing matching result" is achieved.

[0130] Next, the experience stratification is explained with reference to FIG. 6 to FIG. 7.

[0131] Referring to FIG. 6, which is a flowchart of an end-intelligent experience stratification method according to an embodiment of the present application.

[0132] S601: starting the client, sending an end-intelligent experience configuration request to a server, and obtaining an experience stratification model.

[0133] S602: starting a perception engine to obtain device hardware static data, device real-time status data and scenario real-time operation data.

[0134] S603: triggering a decision engine, inputting the obtained device hardware static data, device real-time status data and scenario real-time operation data into the experience stratification model to predict a device operation status.

[0135] S604: running a touchpoint engine to trigger a corresponding experience stratification handling scheme according to a prediction result of the device operation status.

[0136] In practical applications, an attempt is made to identify the operation status of user devices and provide strategic optimization for devices in low operation status, so as to achieve normal operation status and user experience. Taking MiniApps as an example, limited by the MiniApp container, framework and complex scenario logic, it is difficult to achieve consistent performance and experience across all device models. Based on the long-tail theory, the solution focuses on low-performance devices, using minimal resources to address the problem of experience degradation. Existing methods for judging low-performance devices rely on the following key factors: hardware specification, price, performance, function, etc. These judgment criteria are relatively static and cannot make more fine-grained distinctions according to the real-time operation status of user devices. For example, when a high-performance mobile phone opens several large-scale game apps and then opens the target MiniApp, the page becomes very laggy, but the traditional judgment method still considers it a high-performance device at this moment. However, in the embodiments of the present application, real-time judgments are made based on the performance of page rendering. In the experience stratification solution of the embodiments of the present application, the operation status of a user device is evaluated from the perspective of actual user usage. Different from the previous rigid device classification based on factory hardware information, intelligent experience stratification relies on multi-dimensional time-consuming data in the actual operation of the application, uses an end-intelligent experience stratification model to perform comprehensive real-time weighted scoring on user device data, and conducts corresponding handling according to the score.

[0137] For the experience stratification solution, it is first necessary to collect and process data on the server and train the experience stratification model. Then, the experience stratification model is delivered to the front end. During the operation of the client, based on the obtained target data, real-time stratification prediction is performed through the experience stratification model at the front end, and scenario degradation and other handling are carried out based on the operation status of the user device. The solution is introduced below from two perspectives: data collection and model structure determination.

[0138] I. Data Collection and Processing

[0139] Data collection and processing are key steps in model training. The following process divisions can be made for data processing: feature dimension division, data

collection and reporting, data reading, data cleaning, data division and data preprocessing. Through these steps, standardized input is provided to the model.

[0140] (1) Feature Dimension Division

5 **[0141]** Different from existing low-performance device judgment solutions, the data in this solution includes not only static data such as mobile phone model and pixel ratio, but also real-time dynamic operation data of the mobile phone, such as current available memory, number of alarms, loading time of each page, memory changes at each stage, etc. The operation of the mobile phone has more real-time characteristics, and this solution can promptly capture moments when the operation status is not good, and with the help of
10 degradation means, prevent the operation status from deteriorating or even improve the operation status.

[0142] Taking the experience stratification model as an example, the collected data can be divided into static data and dynamic data. Static data includes the hardware conditions of the user's mobile phone, such as mobile phone model, configuration, pixel ratio, etc.
15 Dynamic data is divided into device-related data and scenario-related data.

[0143] In the embodiments of the present application, according to types, the collected data can be classified into: device hardware static data, device real-time status data and scenario real-time operation data. The device hardware static data includes parameters such as mobile phone model and memory capacity. The device real-time status data refers to dynamically
20 changing device data, including parameters such as battery level or network status. The scenario real-time operation data refers to data related to scenario operation, including but not limited to home page scenarios, store scenarios, etc., including parameters such as home page loading time. The embodiments of the present application do not limit the specific parameters of each type of data, for example, including parameter a, parameter b, parameter
25 c, etc.

[0144] (2) Data Collection and Reporting

[0145] Data collection methods vary according to feature dimensions. Static data is read only once and cached to reduce time consumption, while dynamic data is extracted at the collection node to ensure accuracy.

[0146] Taking MiniApps as an example, data reporting supports different dimensions.

[0147] Ordinary tracking data provides daily-dimensional reports, which is suitable for T+1 offline analysis scenarios.

5 [0148] Highway can return data at the minute level, which is suitable for online real-time analysis scenarios.

[0149] Answer has high flexibility and can customize data formats, which is suitable for richer scenarios.

[0150] Mtop entrainment has high flexibility and real-time performance, but is limited by interface qps and trigger timing, and is suitable for end-cloud collaboration scenarios.

10 [0151] (3) Data Preprocessing

[0152] The quality of data directly determines the prediction effect and generalization ability of the model, and determines the ceiling of the model. Therefore, data processing involves several dimensions: accuracy, completeness, consistency, timeliness, credibility, and interpretability. Although the data for the experience stratification model has undergone
15 basic data cleaning, the data still contains a lot of noise.

[0153] Therefore, preprocessing is required before the model is used. Through means such as filling missing values, smoothing noisy data, smoothing or deleting outliers and resolving inconsistent data, standard, clean and smooth data is obtained. In addition, continuous data can be discretized by means of binning, and continuous data can be divided into
20 corresponding bins according to certain rules. Binning rules include unsupervised binning (equal-frequency binning, equal-width binning), supervised binning (chi-square binning, decision tree binning), etc. Equal-frequency binning and equal-width binning are commonly used. Combined with the characteristics of experience stratification data, equal-frequency binning can be adopted.

25 [0154] II. Model Structure and Training

[0155] In the embodiments of the present application, a convolutional neural network (CNN) can be used to construct the experience stratification model. The training process of CNN

includes forward propagation (the network calculates the output based on input data and current weights), loss calculation (based on the output and real labels), back propagation (calculates the gradient of each layer according to the loss), and weight update (adjusts the network weights using gradient descent algorithm or other optimization algorithms).

5 **[0156]** Through further research and exploration, it is found that the experience stratification model first needs to solve the classification problem, and classic algorithms such as GBDT, CNN and DNN are candidate models. The convolutional neural network (CNN) and the deep neural network (DNN) have been tried successively and have their own advantages and disadvantages. CNN is mainly designed to process data with spatial continuity (such as
10 images). By introducing convolution operations and pooling operations, CNN can effectively process spatial data such as images, but may not be as efficient as DNN in processing other types of data. DNN has a simpler and more general structure, but may face problems such as a huge parameter space and overfitting, which can be avoided through optimization. Based on the collected static data and dynamic data such as memory of mobile
15 phones, the comprehensive performance of DNN is better than that of CNN. Combining the advantages of the two algorithms, in the optimization solution, CNN can be used for feature extraction, and then the extracted features are input into DNN for final model training.

[0157] In an implementation, the experience stratification model is obtained through training in a "white-box + black-box model" manner. As described earlier, the white-box
20 model has specific weights during online operation and does not need to load an inference model for inference. The white-box model is mainly used in the cold start phase when the user opens the client (such as a MiniApp) to pre-screen low-performance devices, so as to avoid additional burden on the device caused by model operation. This strategy ensures that more advanced models can be used for further operation status diagnosis and experience
25 optimization in subsequent processes. Black-box model is a deep network architecture constructed through methods such as multi-layer nonlinear transformation and abstract global feature learning, which simulates the layered information processing method of the neural system of the human brain and is used to solve tasks such as classification and regression. Specifically, the experience stratification model can be obtained by training a
30 GBDT structure with a white-box model and a DNN structure with a black-box model. The advantage of this method is that first, a white-box model (such as the GBDT algorithm) is

used to manage user experience stratification according to the operation status of the device. By analyzing the performance parameters of the device (such as memory usage, processor performance, etc.), the model can real-time evaluate the performance level of the device and adjust the performance of the application accordingly, such as reducing the image quality or simplifying interaction effects, to adapt to devices of different performance levels. The decision process of the white-box model is transparent, allowing developers to clearly understand the decision logic and basis of the model, which is convenient for tuning and maintenance. Then, the use of a black-box model (such as DNN) usually involves a complex network structure and a large number of parameters, and the internal decision logic of the black-box model is opaque to users. Such models have strong prediction capabilities and aim to improve the fluency and personalization of user experience. Therefore, by training different network structures to obtain the experience stratification model through the "white-box + black-box" method, the characteristics of each model can be exerted. The white-box model is used for pre-screening low-performance devices, and the black-box model is used for accurate prediction to implement real-time stratification processing solutions.

[0158] Therefore, in an implementation, a first experience stratification model is obtained by training a first network structure (such as GBDT or CNN), and a second experience stratification model is obtained by training a second network structure (such as DNN). During the decision-making process, the first experience stratification model is used for white-box pre-screening and the second experience stratification model is used for black-box fine-screening, so as to predict the device operation status. For example, through white-box pre-screening, 15% of the devices are initially determined to be in poor operation status and eligible for degradation handling. Then, through black-box fine-screening, the remaining 85% of the devices are judged in detail, and 30% of them are still eligible for degradation handling. This "white-box + black-box" combined experience stratification method can comprehensively carry out device stratification handling. Experiments show that it can significantly save resources for 50% of the devices, reduce the average number of memory alarms by 10%, greatly improve the operation fluency of the client, and enhance user experience.

[0159] In the end-intelligent experience stratification solution, another technical key is to release and deploy the experience stratification model trained on the server, so that the client

can obtain the experience stratification model during operation and use the model for real-time prediction. Compared with the method where the server or cloud obtains data from the front end and then performs model prediction, prediction at the front end can reduce the steps of data reporting, thereby accelerating prediction speed and reducing latency.

5 Moreover, this method of model prediction at the front end is not limited by the network status. Even when the network is stuck or faulty, model prediction can be completed relying on the data processing capability of the front end. Therefore, running the experience stratification model on the client is a technical point.

[0160] Referring to FIG. 7, which shows a schematic diagram of an end-intelligent experience stratification implementation logic according to an embodiment of the present application. The entire process includes four main stages: model preparation, environment
10 preparation, parameter preparation, and inference scoring.

[0161] The model preparation stage is to obtain the experience stratification model through offline training, such as data collection and processing on the server and determination of
15 the model structure as described earlier, so as to train the model structure using the processed data and obtain the experience stratification model.

[0162] The environment preparation stage refers to preparing the environment for running the model on the client, which mainly includes starting the engine. Taking MiniApps as an example, first, after the MiniApp is started, the model is downloaded according to the end-
20 intelligent configuration information (for example, through the configuration capability of the end-intelligent configuration platform, the model and related control parameters are configured on the platform, then the model is stored in a DB, and a configuration gateway service reads the model from the DB; after the client is started, the client automatically loads the model from the configuration gateway service, which can be referred to FIG. 4 and the
25 description thereof), then Tensor preparation is carried out; then, by accessing functions such as graphics libraries (such as WebGL), programming solutions (such as WASM, a technical solution for writing code in non-Java programming languages and running the code in a browser), and image processing (WebGPU), dynamic switching between JS engine and NA engine is realized.

30 [0163] In the parameter preparation stage, the main task is to prepare model inference

parameters, which may include steps such as computation graph parsing, operator loading, weight loading, and readiness confirmation.

[0164] In the inference running stage, the main task is to input inference parameters into the model to obtain the prediction result, which may include steps such as Tensor input, input preprocessing, operator computation, and result generation.

[0165] Thus, through the above four stages, the experience stratification model can be used for real-time inference and scoring, thereby determining the real-time operation status of the user device (such as a mobile phone).

[0166] It should be noted that, taking MiniApps as an example, since the model file format that can be run in the MiniApp scenario is limited to “.xnn”, it is necessary to perform model structure optimization and heterogeneous model compilation on models in other formats, which essentially involves the conversion of operator computations. Limited by the running environment and framework of MiniApps, the complexity of the model should be kept simple. A complex model structure will not only result in a large model file and long loading time, but also consume a lot of device resources due to complex computations. Therefore, the model should be designed based on the principle of simple computation, avoiding the use of complex operators and complex processes.

[0167] Therefore, after obtaining the experience stratification model, the above method may further include: an environment preparation stage, in which Tensor preparation is performed based on the downloaded experience stratification model, the logic of the model data preprocessing stage is mapped to JS, and an XNN parser is loaded, to realize dynamic JS engine and NA engine; and a parameter preparation stage, in which model parameters are prepared based on the model format supported by the client, including model structure optimization and heterogeneous model compilation.

[0168] Model structure optimization refers to the process of improving efficiency and reducing resource consumption by improving the model architecture, which usually involves the following aspects: pruning, quantization, simplification, knowledge distillation and autoML for model compression. Pruning is to remove unimportant weights or neurons to reduce the model size. Quantization is to reduce the precision of model parameters to reduce

the model size and accelerate inference, such as converting from 32-bit floating-point numbers to 8-bit integers. Simplification is to simplify the model and remove overly complex parts to improve operation efficiency. Knowledge distillation is to transfer the knowledge of a large model to a smaller model. AutoML for model compression is to use automated algorithms to find optimal model compression methods.

[0169] Heterogeneous model compilation is closely related to model structure optimization. Heterogeneous model compilation refers to the process of compiling deep learning models for different hardware platforms (such as GPU, CPU, tensor processing unit (TPU), or field-programmable gate array (FPGA)). With the diversification of hardware devices, a model that runs efficiently on a specific hardware may perform poorly on another type of hardware. Heterogeneous compilation converts the model to adapt to the characteristics of different hardware, thereby realizing model portability and performance optimization.

[0170] In short, model structure optimization mainly focuses on reducing complexity and improving efficiency within the model, while heterogeneous model compilation focuses on model portability and performance optimization across different hardware platforms. These two aspects are crucial for deploying deep learning models to actual production environments.

[0171] In an implementation, the experience stratification model is trained in advance on the server and released to a content delivery network. The decision engine accesses the content delivery network according to the configuration information of the experience stratification model and loads the experience stratification model to the local. For example, through the configuration capability of the end-intelligent configuration platform, the model and related control parameters are configured on the platform, then the model is stored in a DB, and a configuration gateway service reads the model from the DB; after the client is started, the client automatically loads the model from the configuration gateway service, which can be referred to FIG. 4 and the description thereof.

[0172] For example, the prediction results of the experience stratification model are divided into three levels: good operation status, average operation status, and poor operation status. For the latter two levels, handling schemes such as path simplification, rich interaction degradation, and function degradation can be adopted. Path simplification can be disabling

non-critical functions or lowering the priority of non-critical functions. For example, if non-critical functions have no impact on users' order placement or order fulfillment, these functions can be directly disabled; or if non-critical functions have little impact on users' order placement or order fulfillment, the priority of these functions can be lowered. Rich interaction degradation may include aspects such as reducing animation frame rate, reducing image resolution, and simplifying interaction effects. In terms of reducing the animation frame rate, the higher the animation frame rate, the higher the requirement for device performance, therefore, the animation frame rate can be adjusted in real time according to the performance of the user device to achieve a smooth operation effect. in terms of reducing the image resolution, the image resolution can be adjusted in real time according to the performance of the user device to reduce the memory consumption of image rendering. in terms of simplifying the interaction effects, some interaction effects have high requirements for device performance, thus these interaction effects (such as lottie animation effects) can be simplified or disabled in real time according to the performance of the user device. Function degradation refers to degrading functions in the client that do not block users' order placement or order fulfillment.

[0173] The experience stratification solution provided in the embodiments of the present application is a cross-end, real-time, dynamic, intelligent, and full-link low-performance device identification strategy. The experience stratification solution not only focuses on the static data of the device, but also starts from data of various dimensions during the device operation process, to provide more accurate data basis for the identification of low-performance devices through real-time perception and calculation. Compared with traditional methods for evaluating low-performance devices, which are more rigid and less flexible, the embodiments of the present application focus on providing a more intelligent, comprehensive, and accurate evaluation solution from the perspective of real user experience through real-time data analysis and algorithmic decision-making. This solution breaks through the limitations of traditional solutions and is more personalized, intelligent and real-time.

[0174] Finally, the experience enhancement solution is introduced with reference to FIG. 8.

[0175] Referring to FIG. 8, which shows a flowchart of an end-intelligent experience enhancement method according to an embodiment of the present application.

[0176] S801: starting a client, sending an end-intelligent experience configuration request to a server, and obtaining a full-link trajectory prediction model.

[0177] S802: starting a perception engine to obtain user behavior data, user basic data, scenario data and performance data.

- 5 [0178] S803: triggering a decision engine, inputting the obtained user behavior data, user basic data, scenario data and performance data into the full-link trajectory prediction model to predict a future behavior trajectory of a user.

[0179] S804: running a touchpoint engine to trigger a corresponding experience enhancement handling scheme according to a prediction result of the future behavior trajectory of the user.

10

[0180] Experience enhancement refers to realizing real-time prediction of the future trajectory of a user based on real-time user interaction behavior information and using a full-link trajectory prediction model, and performing interventions such as preloading sub-packages and pages that the user is likely to access to enhance the sense of surprise in user experience.

15

[0181] For the experience enhancement solution, it is first necessary to collect and process data on the server and train the full-link trajectory prediction model. Then, the full-link trajectory prediction model is delivered to the front end. During the operation of the client, based on the obtained target data, real-time prediction of future behaviors is performed through the full-link trajectory prediction model at the front end, and handling such as sub-package preloading is carried out based on the prediction results of future behaviors.

20

[0182] Regarding the two perspectives of data collection and model structure determination, the principles and processing processes are similar to those of the experience stratification solution, and reference can be made to the relevant description of the experience stratification solution. The difference lies in the division of data feature dimensions and the selection of model structures. Since experience enhancement mainly considers predicting future behaviors of users, the data feature dimensions mainly include user behavior data, user basic data, scenario data and performance data. For the acquisition methods of various types of data, reference can be made to the previous description. For the network structure

25

of the full-link trajectory prediction model, through research and practice, a network structure with strong computing capabilities, such as a deep learning network (e.g., LSTM), can be selected. Such model attempts to predict the next possible operations or preferences of users by analyzing historical behavior data of the users, thereby predicting the behavior paths of the users. Since deep learning models usually involve complex network structures and a large number of parameters, the internal decision logic of the deep learning models is opaque to users, so a black-box model can be adopted. Therefore, in an implementation, the full-link trajectory prediction model corresponding to the experience enhancement decision is set to a black-box model mode, and in the hot start phase of the client, the full-link trajectory prediction model in the black-box model mode is run.

[0183] Regarding the release and deployment of the full-link trajectory prediction model trained on the server, and the client downloading and performing real-time prediction locally, reference can be made to the relevant description of the experience stratification solution (FIG. 7), and the principles and implementation processes are similar, which will not be repeated here.

[0184] In addition, in an implementation, the full-link trajectory prediction model is trained in advance on the server and released to a content delivery network. The decision engine accesses the content delivery network according to the configuration information of the experience stratification model and loads the experience stratification model to the local. For example, through the configuration capability of the end-intelligent configuration platform, the model and related control parameters are configured on the platform, then the model is stored in a DB, and a configuration gateway service reads the model from the DB; after the client is started, the client automatically loads the model from the configuration gateway service, which can be referred to FIG. 4 and the description thereof.

[0185] For example, after predicting the user's future behavior trajectory through the full-link trajectory prediction model, corresponding experience enhancement handling can be performed on the prediction result. For example, the experience enhancement handling scheme includes: obtaining a future behavior trajectory of a user according to the experience decision, determining sub-package resources or page resources that the user is likely to access, and performing preloading operations on the sub-package resources or page resources.

[0186] In a specific scenario, intelligent preloading can be performed for channel sub-packages. In the client interface, the next page after the home page is usually a sub-package. Forced full-scale preloading not only wastes traffic, but also occupies memory and may even cause freezes. By using the end-intelligent experience enhancement solution to predict the next jump address and perform sub-package preloading according to the proximity principle, the contradiction between memory and experience can be well balanced.

[0187] Corresponding to the end-intelligent experience implementation method applied on the client, the embodiments of the present application further provide an end-intelligent experience implementation method applied on the server. Referring to FIG. 9, which is a flowchart of an end-intelligent experience implementation method applied on the server.

[0188] This end-intelligent experience implementation method is used to configure and manage touchpoints of front-end code for end-intelligent experience, where the front-end code is applicable to multiple types of clients and performs perception, decision-making and handling of user experiences at a front end. The method is applied to a server and includes the following steps.

[0189] S901: receiving an end-intelligent configuration request sent by a client, matching an end-intelligent configuration and delivering the end-intelligent configuration to the client, so that the client executes end-intelligent experience decision-making and handling.

[0190] The client executing the experience decision-making and handling includes running an end-intelligent suite in the front-end code, including: starting a perception engine in the end-intelligent suite to obtain target data in real time; triggering a decision engine in the end-intelligent suite to make an experience decision on the target data based on the end-intelligent experience configuration; and running a touchpoint engine in the end-intelligent suite to trigger a corresponding handling scheme according to an experience decision result.

[0191] S902: in response to triggering of a callback function, receiving an execution result of the handling scheme fed back by the client.

[0192] Regarding the end-intelligent experience implementation method performed on the server, reference can be made to the previous principles and descriptions, which will not be repeated here.

[0193] Referring to FIG. 10, which shows a schematic structural diagram of an end-intelligent experience implementation apparatus according to an embodiment of the present application.

5 [0194] This apparatus is configured for front-end code applicable to multiple types of clients to perform perception, decision-making and handling of user experiences at a front end. The apparatus is applied to a client among any type of clients in the multiple types of clients and includes:

[0195] a configuration request unit 1001, configured to start the client, send an end-intelligent experience configuration request to a server, obtain an end-intelligent experience configuration and store the end-intelligent experience configuration locally; and
10

[0196] an end-intelligent execution unit 1002, configured to run an end-intelligent suite in the front-end code, including: starting a perception engine in the end-intelligent suite to obtain target data in real time; triggering a decision engine in the end-intelligent suite to make an experience decision on the target data based on the end-intelligent experience configuration; and running a touchpoint engine in the end-intelligent suite to trigger a
15 corresponding handling scheme according to an experience decision result.

[0197] For specific examples of each unit in this embodiment, reference can be made to the examples described in the above embodiments and optional implementations, which will not be repeated in this embodiment.

20 [0198] The embodiments of the present application further provide a storage medium, in which a computer program is stored, and the computer program is configured to execute the steps in any of the above method embodiments when running.

[0199] In this embodiment, the above storage medium may be configured to store a computer program for executing the above method embodiments.

25 [0200] In this embodiment, the above storage medium may include but is not limited to: a USB flash drive, a Read-Only Memory (ROM), a Random Access Memory (RAM), a mobile hard disk, a magnetic disk, or an optical disc and other various media that can store a computer program.

[0201] The embodiments of the present application further provide an electronic device, including a memory and one or more processors, where the memory stores a computer program, and the one or more processors are configured to run the computer program to execute the steps in any of the above method embodiments.

- 5 [0202] In an implementation, the above electronic device may further include a transmission device and an input-output device, where the transmission device is connected to the one or more processors, and the input-output device is connected to the one or more processors.

[0203] In this embodiment, the above one or more processors may be configured to execute the steps in any of the above method embodiments through a computer program.

- 10 [0204] For specific examples in this embodiment, reference can be made to the examples described in the above embodiments and optional implementations, which will not be repeated in this embodiment.

[0205] The serial numbers of the embodiments of the present application are only for description, and do not indicate the advantages or disadvantages of the embodiments.

- 15 [0206] In the above embodiments of the present application, the description of each embodiment has its own focus. For parts that are not described in detail in a certain embodiment, reference can be made to the relevant descriptions of other embodiments.

- [0207] In the several embodiments provided in the present application, it should be understood that the disclosed technical content can be implemented in other ways. The apparatus embodiments described above are only illustrative. For example, the division of the units is only a logical function division, and in actual implementation, there may be other division manners. For example, multiple units or components may be combined or integrated into another system, or some features may be ignored or not implemented. In addition, the displayed or discussed mutual coupling or direct coupling or communication connection
20
25 may be through some interfaces, and the indirect coupling or communication connection between units or modules may be in the form of electrical or other forms.

[0208] The units described as separate components may or may not be physically separate, and the components displayed as units may or may not be physical units, that is, they may

be located in one place, or may be distributed to multiple network units. Some or all of the units may be selected according to actual needs to achieve the purpose of the solution of this embodiment.

5 [0209] In addition, the functional units in each embodiment of the present application may be integrated into one processing unit, or each unit may exist alone physically, or two or more units may be integrated into one unit. The above integrated units may be implemented in the form of hardware or in the form of software functional units.

10 [0210] If the integrated units are implemented in the form of software functional units and sold or used as an independent product, the integrated units may be stored in a computer-readable storage medium. Based on this understanding, the technical solution of the present application essentially or the part that contributes to the prior art or all or part of the technical solution may be embodied in the form of a software product. The computer software product is stored in a storage medium and includes several instructions to enable a computer device (which may be a personal computer, a server, or a network device, etc.) to execute all or part
15 of the steps of the method described in each embodiment of the present application. The aforementioned storage medium includes: a USB flash drive, a ROM, a RAM, a mobile hard disk, a magnetic disk, an optical disc, and other media that can store program codes.

20 [0211] The above is only preferred implementations of the present application. It should be pointed out that for those of ordinary skill in the art, without departing from the principle of the present application, several improvements and modifications can be made, and these improvements and modifications should also be considered within the protection scope of the present application.

CLAIMS

1. An end-intelligent experience implementation method, configured for front-end code applicable to multiple types of clients to perform perception, decision-making and handling
5 of user experiences at a front end, wherein the method is applied to a client among any type of clients in the multiple types of clients and comprises:

starting the client, sending an end-intelligent experience configuration request to a server, and obtaining an end-intelligent experience configuration and storing the end-intelligent experience configuration locally; and

10 running an end-intelligent suite in the front-end code, comprising: starting a perception engine in the end-intelligent suite to obtain target data in real time; triggering a decision engine in the end-intelligent suite to make an experience decision on the target data based on the end-intelligent experience configuration; and running a touchpoint engine in the end-intelligent suite to trigger a corresponding handling scheme according to an
15 experience decision result;

wherein the handling scheme comprises: an experience monitoring handling scheme, an experience stratification handling scheme and/or an experience enhancement handling scheme;

the experience monitoring handling scheme comprises: in response to determining
20 according to the experience decision that a complex event occurs, performing real-time message push in an interface through UI interaction and/or performing a page jump operation;

the experience stratification handling scheme comprises: in response to determining according to the experience decision that a device operation status is a low operation status,
25 performing interface loading optimization and/or a dynamic page content adjustment scheme;

the experience enhancement handling scheme comprises: obtaining a future behavior trajectory of a user according to the experience decision, determining sub-package resources or page resources that the user is likely to access, and performing preloading operations on
30 the sub-package resources or the page resources.

2. The method according to claim 1, further comprising:

setting different priorities for different handling schemes, and determining an execution logical relationship between different handling schemes according to the priorities, wherein the experience stratification handling scheme is set to correspond to a first priority, and whether to execute the experience monitoring handling scheme and/or the experience enhancement handling scheme is determined according to a processing result of the experience stratification handling scheme.

3. The method according to claim 1, further comprising:

making an experience decision for experience monitoring, experience stratification or experience enhancement through a white-box model or a black-box model; and

making a preliminary decision according to the white-box model, and determining whether to use the black-box model for further refined decision-making according to a preliminary decision result of the white-box model.

4. The method according to claim 3, further comprising:

setting an experience stratification model corresponding to an experience stratification decision to a white-box model mode; and in a cold start phase of the client, running the experience stratification model in the white-box model mode, and determining whether to continue executing the end-intelligent suite or close the end-intelligent suite according to a prediction result of the experience stratification model; and

setting a full-link trajectory prediction model corresponding to an experience enhancement decision to a black-box model mode; and in a hot start phase of the client, running the full-link trajectory prediction model in the black-box model mode.

5. The method according to any one of claims 1 to 4, wherein the end-intelligent experience configuration comprises complex event configuration information, and the complex event configuration information comprises event configuration information and timing configuration information of at least one target complex event;

starting the perception engine in the end-intelligent suite to obtain the target data in real time comprises: starting the perception engine, capturing tracking data based on a unified event stream processing tool in the front-end code, and determining whether a target complex event corresponding to the event configuration information is triggered according

to the tracking data;

triggering the decision engine in the end-intelligent suite to make the experience decision on the target data based on the end-intelligent experience configuration comprises: in response to triggering of the target complex event, running the decision engine to perform
5 timing matching on the tracking data based on the timing configuration information, and determining whether a timing of the target complex event is satisfied; and

running the touchpoint engine in the end-intelligent suite to trigger the corresponding handling scheme according to the experience decision result comprises: after the timing matching is successful, running the touchpoint engine to trigger an experience monitoring
10 handling scheme corresponding to the target complex event.

6. The method according to claim 5, after the unified event stream processing tool captures the tracking data, the method further comprises:

performing client type analysis and format analysis on the tracking data, and
15 performing standardization processing on the tracking data so that data of different types of clients have a unified format and structure;

wherein performing the standardization processing on the tracking data comprises: parsing tracking information, event ID and parameter information of the tracking data; and performing unified format encapsulation on the tracking information, the event ID and the
20 parameter information to convert the tracking information, the event ID and the parameter information into event information in a standard format.

7. The method according to claim 5, wherein the decision engine is a complex event processing component based on a self-developed complex event matching rule base and
25 declared as a static function, and the self-developed complex event matching rule base is developed using JavaScript to be compatible with multiple types of clients.

8. The method according to any one of claims 1 to 4, wherein the end-intelligent experience configuration comprises an experience stratification model;

30 starting the perception engine in the end-intelligent suite to obtain the target data in real time comprises: starting the perception engine to obtain device hardware static data, device real-time status data and scenario real-time operation data;

triggering the decision engine in the end-intelligent suite to make the experience decision on the target data based on the end-intelligent experience configuration comprises: triggering the decision engine, and inputting the obtained device hardware static data, device real-time status data and scenario real-time operation data into the experience stratification model to predict a device operation status; and

running the touchpoint engine in the end-intelligent suite to trigger the corresponding handling scheme according to the experience decision result comprises: running the touchpoint engine to trigger a corresponding experience stratification handling scheme according to a prediction result of the device operation status.

9. The method according to claim 8, wherein the experience stratification model is obtained by extracting features of data samples based on a convolutional neural network and inputting the extracted features into a deep neural network model for training.

10. The method according to claim 8, wherein the experience stratification model predicts the device operation status by training a first network structure to obtain a first experience stratification model and training a second network structure to obtain a second experience stratification model, and in a decision-making process of the decision engine, using the first experience stratification model as a white-box model for white-box pre-screening, and using the second experience stratification model as a black-box model for black-box fine-screening.

11. The method according to any one of claims 1 to 4, wherein the end-intelligent experience configuration comprises a full-link trajectory prediction model;

starting the perception engine in the end-intelligent suite to obtain the target data in real time comprises: starting the perception engine to obtain user behavior data, user basic data, scenario data and performance data;

triggering the decision engine in the end-intelligent suite to make the experience decision on the target data based on the end-intelligent experience configuration comprises: triggering the decision engine, and inputting the obtained user behavior data, user basic data, scenario data and performance data into the full-link trajectory prediction model to predict a future behavior trajectory of a user; and

running the touchpoint engine in the end-intelligent suite to trigger the corresponding handling scheme according to the experience decision result comprises: running the touchpoint engine to trigger a corresponding experience enhancement handling scheme according to a prediction result of the future behavior trajectory of the user.

5

12. An end-intelligent experience implementation method, used to configure and manage touchpoints of front-end code for end-intelligent experience, wherein the front-end code is applicable to multiple types of clients and performs perception, decision-making and handling of user experiences at a front end, and the method is applied to a server and comprises:

10

receiving an end-intelligent configuration request sent by a client, matching an end-intelligent configuration and delivering the end-intelligent configuration to the client, so that the client runs an end-intelligent suite in the front-end code, which comprises: starting a perception engine in the end-intelligent suite to obtain target data in real time; triggering a decision engine in the end-intelligent suite to make an experience decision on the target data based on the end-intelligent experience configuration; and running a touchpoint engine in the end-intelligent suite to trigger a corresponding handling scheme according to an experience decision result; and

15

in response to triggering of a callback function, receiving an execution result of the handling scheme fed back by the client;

20

wherein the handling scheme comprises: an experience monitoring handling scheme, an experience stratification handling scheme and/or an experience enhancement handling scheme;

the experience monitoring handling scheme comprises: in response to determining according to the experience decision that a complex event occurs, performing real-time message push in an interface through UI interaction and/or performing a page jump operation;

25

the experience stratification handling scheme comprises: in response to determining according to the experience decision that a device operation status is a low operation status, performing interface loading optimization and/or a dynamic page content adjustment scheme;

30

the experience enhancement handling scheme comprises: obtaining a future behavior

trajectory of a user according to the experience decision, determining sub-package resources or page resources that the user is likely to access, and performing preloading operations on the sub-package resources or the page resources.

- 5 13. An end-intelligent experience implementation apparatus, configured for front-end code applicable to multiple types of clients to perform perception, decision-making and handling of user experiences at a front end, wherein the apparatus is applied to a client among any type of clients in the multiple types of clients and comprises:
- 10 a configuration request unit, configured to start the client, send an end-intelligent experience configuration request to a server, and obtain an end-intelligent experience configuration and store the end-intelligent experience configuration locally; and
- 15 an end-intelligent execution unit, configured to run an end-intelligent suite in the front-end code, comprising: starting a perception engine in the end-intelligent suite to obtain target data in real time; triggering a decision engine in the end-intelligent suite to make an experience decision on the target data based on the end-intelligent experience configuration; and running a touchpoint engine in the end-intelligent suite to trigger a corresponding handling scheme according to an experience decision result;
- 20 wherein the handling scheme comprises: an experience monitoring handling scheme, an experience stratification handling scheme and/or an experience enhancement handling scheme;
- 25 the experience monitoring handling scheme comprises: in response to determining according to the experience decision that a complex event occurs, performing real-time message push in an interface through UI interaction and/or performing a page jump operation;
- the experience stratification handling scheme comprises: in response to determining according to the experience decision that a device operation status is a low operation status, performing interface loading optimization and/or a dynamic page content adjustment scheme;
- 30 the experience enhancement handling scheme comprises: obtaining a future behavior trajectory of a user according to the experience decision, determining sub-package resources or page resources that the user is likely to access, and performing preloading operations on the sub-package resources or the page resources.

14. A storage medium, having a computer program stored thereon, wherein the computer program is configured to perform, when running, the method according to any one of claims 1 to 12.

5

15. An electronic device, comprising a memory and one or more processors, wherein the memory stores a computer program, and the one or more processors are configured to run the computer program to perform the method according to any one of claims 1 to 12.