**EAI Technical White Paper**

**Abstract**

EAI is a blockchain-based decentralized computing platform designed to incentivize miners to provide multimodal computing resources through a mining mechanism, forming an elastic computing resource pool. These resources will be used to support AI computing tasks and to build and run AI agents on the platform. EAI aims to achieve decentralized allocation of computing resources through blockchain technology, reduce AI computing costs, and provide sustainable income for miners.

**1. Introduction**

**1.1 Project Background**

With the rapid development of artificial intelligence (AI) technology, the demand for AI computing has grown exponentially. From training deep learning models to deploying complex AI applications, the need for computing resources continues to increase. However, the current computing resource allocation model has several issues that severely limit the further development of AI technology.

**Centralization Issue**: Computing resources are mainly concentrated in the hands of a few large enterprises, leading to resource monopolies and high costs. These enterprises, by controlling resources, have created market barriers, making it difficult for small and medium-sized enterprises and research institutions to obtain sufficient computing resources.

**Low Resource Utilization**: A large amount of computing resources are idle or underutilized. Many enterprises and individuals own high-performance computing devices, but due to the lack of effective resource scheduling and management mechanisms, the computing power of these devices is not fully utilized.

**Privacy and Security Risks**: Centralized computing platforms face the risk of data breaches and privacy violations. User data on centralized platforms is vulnerable to attacks, leading to frequent data security issues.

To address these issues, the EAI project was born. EAI (Elastic AI) is a blockchain-based decentralized computing platform designed to incentivize miners to provide multimodal computing resources through a mining mechanism, forming an elastic computing resource pool. These resources will be used to support AI computing tasks and to build and run AI agents on the platform.

**1.2 Project Motivation**

The motivation behind the EAI project stems from a deep reflection on the current AI computing resource allocation model. We recognize that the traditional centralized computing resource allocation model can no longer meet the growing demand for AI computing. To promote the widespread application and development of AI technology, we need a more efficient, fair, and secure computing resource allocation mechanism.

**Decentralization**: Through blockchain technology, the EAI platform achieves decentralized allocation of computing resources, avoiding single points of failure and resource monopolies. Miners can freely join or exit the network, and resource allocation and management are automatically executed by smart contracts, ensuring system transparency and fairness.

**Reducing Computing Costs**: The EAI platform incentivizes miners to provide idle computing resources, forming an elastic computing resource pool. These resources can be dynamically allocated to AI computing tasks, thereby reducing the cost of AI computing. Users can obtain high-quality computing services at lower prices, promoting the popularization and application of AI technology.

**Improving Resource Utilization**: The EAI platform integrates idle computing resources globally, improving resource utilization through intelligent scheduling and management. Miners' devices can provide computing resources to the platform when idle, earning corresponding rewards, thereby achieving efficient resource utilization.

**Enhancing Privacy and Security**: The EAI platform employs advanced encryption and access control mechanisms to ensure the security and privacy of user data. Data is encrypted during transmission and storage, and only authorized users can access and use the data. Additionally, the platform supports technologies such as differential privacy and federated learning to further protect user privacy.

**1.3 Goals and Vision**

The goal of the EAI project is to achieve decentralized allocation of computing resources through blockchain technology, reduce AI computing costs, and provide sustainable income for miners. Specific goals include:

**Establishing a Decentralized Computing Resource Pool**: Integrating idle computing resources globally to form an elastic computing resource pool that supports the efficient execution of AI computing tasks.

**Supporting Multimodal Computing Resources**: The platform supports various types of computing resources, including GPU, CPU, and storage, to meet the needs of different AI computing tasks.

**Building AI Agents**: Building and running AI agents on the platform to provide efficient, flexible, and scalable AI computing capabilities, supporting various application scenarios.

**Reducing AI Computing Costs**: By incentivizing miners to provide computing resources, the platform reduces the cost of AI computing, promoting the widespread application of AI technology.

**Providing Sustainable Income**: Providing sustainable income for miners, incentivizing them to continuously provide computing resources and support the long-term development of the platform.

The vision of the EAI project is to promote the widespread application and development of AI technology through decentralized computing resource allocation. We believe that through the EAI platform, efficient utilization of computing resources can be achieved, AI computing costs can be reduced, and greater value can be created for users and miners. In the future, the EAI platform will continue to optimize and expand, supporting more AI application scenarios and becoming a globally leading decentralized AI computing platform.

**2. EAI Platform Overview**

**2.1 Core Features**

The EAI platform is a blockchain-based decentralized computing platform designed to incentivize miners to provide multimodal computing resources through a mining mechanism, forming an elastic computing resource pool. These resources will be used to support AI computing tasks and to build and run AI agents on the platform. The core features of the EAI platform include:

* **Multimodal Computing Resource Pool**: Integrating idle computing resources globally, supporting various resource types, including GPU, CPU, and storage. These resources can be dynamically allocated to AI computing tasks, improving resource utilization.
* **Elastic Scaling**: Dynamically adjusting resource allocation based on the demand for AI computing tasks. Through intelligent scheduling algorithms, the platform ensures efficient resource utilization and avoids resource waste.
* **Decentralization**: Utilizing blockchain technology to achieve decentralized allocation of computing resources, avoiding single points of failure and resource monopolies. Miners can freely join or exit the network, and resource allocation and management are automatically executed by smart contracts, ensuring system transparency and fairness.
* **AI Agents**: Providing efficient, flexible, and scalable AI computing capabilities, supporting various application scenarios. AI agents can execute various AI computing tasks, including natural language processing, computer vision, and speech processing.

**2.2 Architecture Design**

The architecture design of the EAI platform aims to achieve efficient, flexible, and scalable AI computing capabilities. Its main components include:

* **Multimodal Computing Resources and Elastic Computing Resource Pool**: Integrating various computing resources to form an elastic computing resource pool, supporting dynamic resource allocation.
* **Mining Mechanism**: Incentivizing miners to provide computing resources through a mining mechanism, ensuring efficient task execution.
* **AI Agent Architecture**: AI agents are the core components of the EAI platform, responsible for executing AI computing tasks and interacting with users, developers, and other AI agents.
* **Blockchain Network**: Utilizing blockchain technology to achieve decentralized resource allocation and task management, ensuring system security and transparency.
* **Security and Privacy**: Employing data encryption, access control, and privacy protection technologies to ensure the security and privacy of user data.
* **Performance Optimization**: Ensuring the efficient operation of the platform through resource scheduling optimization, algorithm optimization, and data optimization.

**2.3 Key Components**

The key components of the EAI platform include:

* **Multimodal Computing Resources and Elastic Computing Resource Pool**: Integrating GPU, CPU, storage, and other resources to form an elastic computing resource pool, supporting dynamic resource allocation.
* **Mining Mechanism**: Incentivizing miners to provide computing resources, ensuring efficient task execution. The mining process includes task allocation, computation and verification, and reward distribution.
* **AI Agent Architecture**: AI agents are responsible for executing AI computing tasks, supporting various application scenarios. Their architecture includes a task scheduling module, computation engine module, data management module, interaction interface module, and self-learning module.
* **Blockchain Network**: Utilizing blockchain technology to achieve decentralized resource allocation and task management, ensuring system security and transparency.
* **Security and Privacy**: Ensuring the security and privacy of user data through data encryption, access control, and privacy protection technologies.
* **Performance Optimization**: Ensuring the efficient operation of the platform through resource scheduling optimization, algorithm optimization, and data optimization.

**3. Technical Architecture**

**3.1 Multimodal Computing Resources and Elastic Computing Resource Pool**

**3.1.1 Multimodal Computing Resources**

The EAI platform integrates various computing resources, including GPU, CPU, storage, and network bandwidth, to support different types of AI computing tasks. These resources, through the participation of miners, form a global elastic computing resource pool. Specifically:

* **GPU**: Used for high-performance computing tasks, such as training and inference of deep learning models. The parallel computing capability of GPUs gives them a significant advantage when processing large-scale data and complex models.
* **CPU**: Used for general-purpose computing tasks, such as data preprocessing and traditional machine learning algorithms. The flexibility and versatility of CPUs make them important in various computing scenarios.
* **Storage**: Used for storing task data, models, and intermediate results. Efficient utilization of storage resources can significantly improve data processing speed and efficiency.
* **Network Bandwidth**: Used for data transmission and task allocation. High-speed network bandwidth ensures fast data transmission, reducing task execution delays.

**3.1.2 Elastic Computing Resource Pool**

The EAI platform achieves efficient utilization of computing resources through dynamic resource allocation. Specifically:

* **Dynamic Scaling**: Dynamically adjusting resource allocation based on the demand for AI computing tasks. When task demand increases, the platform can automatically increase resource allocation; when task demand decreases, the platform can automatically reduce resource allocation, ensuring efficient resource utilization.
* **Load Balancing**: Ensuring balanced utilization of computing resources through intelligent scheduling algorithms. The platform can automatically allocate tasks to nodes with lower resource utilization, avoiding resource waste.
* **Decentralization**: Achieving decentralized allocation of computing resources through blockchain technology. Miners can freely join or exit the network, and resource allocation and management are automatically executed by smart contracts, ensuring system transparency and fairness.

**3.1.3 Resource Registration and Evaluation**

The EAI platform registers and evaluates the computing resources provided by miners. Specifically:

* **Resource Registration**: Miners register their computing resources on the EAI platform, and the platform evaluates the resources based on their type and performance. This helps the platform understand the available resources and better schedule them.
* **Performance Evaluation**: The platform evaluates the computing power, storage capacity, and network bandwidth of resources through benchmark testing and performance monitoring. This ensures the quality and reliability of the resources.
* **Resource Classification**: Based on the evaluation results, resources are classified as high-performance, medium-performance, and low-performance. Different types of resources can be used for tasks with different priorities, ensuring reasonable resource utilization.

**3.2 Mining Mechanism**

**3.2.1 Task Allocation**

The EAI platform allocates AI computing tasks to suitable miner nodes through intelligent scheduling algorithms. Specifically:

* **Task Submission**: Users or developers submit AI computing tasks to the platform, including task type, data, algorithm, and resource requirements. The platform selects appropriate computing resources for allocation based on task requirements.
* **Task Matching**: The platform matches suitable computing resources based on task requirements. Through intelligent scheduling algorithms, the platform can allocate tasks to nodes with lower resource utilization, ensuring efficient resource utilization.
* **Task Distribution**: Tasks are allocated to matched miner nodes for execution. Miner nodes execute the tasks using their computing resources according to task requirements.

**3.2.2 Computation and Verification**

Miner nodes use their computing resources to execute AI computing tasks and submit the computation results to the blockchain network for verification. Specifically:

* **Task Execution**: Miner nodes use their computing resources to execute AI computing tasks. This includes data preprocessing, model training, and inference.
* **Result Submission**: Miners submit computation results to the blockchain network for verification. Through smart contracts, the platform can verify the correctness of the computation results.
* **Consensus Mechanism**: The correctness of computation results is verified through consensus algorithms (such as PoW or PoS). This ensures the reliability of task execution and the credibility of the results.

**3.2.3 Reward Distribution**

The EAI platform automatically distributes rewards through smart contracts, incentivizing miners to provide computing resources. Specifically:

* **Reward Calculation**: Rewards are calculated based on resource type, performance, and task contribution. Different types of resources and task contributions affect the reward amount.
* **Token Distribution**: EAI tokens are automatically distributed to miners through smart contracts. This ensures the transparency and fairness of rewards.
* **Incentive Mechanism**: Through the reward mechanism, the platform incentivizes miners to continuously provide high-quality computing resources. This supports the long-term development of the platform and the stable supply of resources.

**3.3 AI Agent Architecture**

**3.3.1 Task Scheduling Module**

The task scheduling module of AI agents is responsible for receiving and allocating AI computing tasks. Specifically:

* **Task Reception**: AI agents receive AI computing tasks from the platform or users. This includes task type, data, algorithm, and resource requirements.
* **Resource Allocation**: Based on task requirements, AI agents allocate suitable computing resources from the elastic computing resource pool. Through intelligent scheduling algorithms, AI agents ensure efficient resource utilization.
* **Task Distribution**: AI agents allocate tasks to appropriate computation nodes for execution. This ensures efficient task execution and reasonable resource utilization.

**3.3.2 Computation Engine Module**

The computation engine module of AI agents integrates various AI algorithms, supporting multiple computation resources. Specifically:

* **Algorithm Library**: AI agents integrate various AI algorithms, including deep learning, machine learning, and reinforcement learning. This enables AI agents to execute various complex AI computing tasks.
* **Multimodal Support**: AI agents support multiple computation resources, including GPU and CPU. This allows AI agents to select the most suitable computation resources for task execution based on task requirements.
* **Distributed Computing**: AI agents use distributed computing frameworks (such as TensorFlow and PyTorch) to accelerate task execution. This significantly improves task execution efficiency and speed.

**3.3.3 Data Management Module**

The data management module of AI agents is responsible for data storage, security, and sharing. Specifically:

* **Data Storage**: AI agents use distributed storage technologies (such as IPFS and S3) to store task data and models. This ensures data reliability and accessibility.
* **Data Security**: Through encryption and access control, AI agents ensure data security and privacy. This prevents data breaches and unauthorized access.
* **Data Sharing**: AI agents support secure data sharing among multiple AI agents. This promotes efficient data utilization and collaboration.

**3.3.4 Interaction Interface Module**

The interaction interface module of AI agents provides user interfaces and developer interfaces. Specifically:

* **User Interface**: AI agents provide natural language interaction interfaces, supporting text and voice input. This allows users to easily interact with AI agents.
* **Developer Interface**: AI agents provide RESTful APIs and SDKs, supporting multiple programming languages (such as Python, Java, and JavaScript). This allows developers to easily integrate and extend the functionality of AI agents.
* **Blockchain Interface**: AI agents interact with the blockchain network for task submission, result verification, and reward distribution. This ensures the transparency of task execution and the credibility of results.

**3.3.5 Self-Learning Module**

The self-learning module of AI agents supports online training and transfer learning. Specifically:

* **Model Training**: AI agents support online training, continuously optimizing model performance. This allows AI agents to adapt to changing task requirements.
* **Feedback Mechanism**: AI agents adjust model parameters and computation strategies based on user feedback and task execution results. This improves task execution accuracy and efficiency.
* **Knowledge Base**: AI agents build a shared knowledge base, storing models and data for use by multiple AI agents. This promotes knowledge sharing and reuse.

**3.4 Data Optimization**

**3.4.1 Data Preprocessing**

The EAI platform improves data quality through data preprocessing techniques. Specifically:

* **Data Cleaning**: Removing noise and outliers from data to ensure data accuracy.
* **Data Transformation**: Converting data into formats suitable for AI computing tasks.
* **Data Normalization**: Scaling data to a specific range for model processing.

**3.4.2 Data Augmentation**

The EAI platform employs data augmentation techniques to increase data diversity and quantity. Specifically:

* **Feature Space Data Augmentation**: Performing data augmentation in the feature space, such as the MoEx method.
* **Generative Model-Based Data Augmentation**: Using generative models (such as VAE and GAN) to generate new data samples.
* **Neural Style Transfer-Based Data Augmentation**: Generating data with different styles through neural style transfer techniques.

**3.4.3 Caching Technology**

The EAI platform uses caching technology to reduce data retrieval time. Specifically:

* **Data Caching**: Storing frequently used data in cache to reduce data loading time.
* **Cache Update**: Regularly updating cached data to ensure data timeliness.

**3.5 Performance Optimization**

**3.5.1 Resource Scheduling Optimization**

The EAI platform optimizes resource allocation through intelligent scheduling algorithms. Specifically:

* **Dynamic Resource Allocation**: Dynamically adjusting resource allocation based on task requirements.
* **Load Balancing**: Ensuring balanced utilization of computing resources through intelligent scheduling algorithms.

**3.5.2 Algorithm Optimization**

The EAI platform improves computation efficiency through model compression and distributed training. Specifically:

* **Model Compression**: Reducing model computation complexity through pruning and quantization techniques.
* **Distributed Training**: Accelerating model training using distributed computing frameworks.

**3.5.3 Data Optimization**

The EAI platform improves data quality through data preprocessing and caching techniques. Specifically:

* **Data Preprocessing**: Improving data quality through data cleaning and augmentation techniques.
* **Data Caching**: Reducing data retrieval time using caching technology.

**4. AI Agent Design and Implementation**

**4.1 Core Functions of AI Agents**

AI agents are the core components of the EAI platform, responsible for executing various AI computing tasks and interacting with users, developers, and other AI agents. Their core functions include:

* **Task Execution**: Supporting various AI computing tasks, including natural language processing, computer vision, and speech processing. AI agents can call corresponding algorithms and computation resources based on task requirements to complete complex AI computing tasks.
* **Self-Learning**: Continuously optimizing model performance through online learning and transfer learning. AI agents can automatically adjust model parameters based on new data and task requirements, improving task execution accuracy and efficiency.
* **Multimodal Data Processing**: Supporting various data formats, including text, images, audio, and video. AI agents can process multiple types of data to meet the needs of different application scenarios.
* **Real-Time Interaction**: Providing natural language interaction interfaces and developer APIs, supporting real-time interaction. Users can interact with AI agents through natural language, and developers can call AI agent functions through APIs.

**4.2 Architecture Design of AI Agents**

The architecture design of AI agents aims to achieve efficient, flexible, and scalable AI computing capabilities. Its main modules include:

* **Task Scheduling Module**: Responsible for receiving and allocating AI computing tasks. This module allocates suitable computation resources from the elastic computing resource pool based on task requirements and allocates tasks to appropriate computation nodes for execution.
* **Computation Engine Module**: Integrating various AI algorithms, supporting multiple computation resources. This module includes various AI algorithms, such as deep learning, machine learning, and reinforcement learning, and can call corresponding algorithms for computation based on task requirements.
* **Data Management Module**: Responsible for data storage, security, and sharing. This module uses distributed storage technologies to store task data and models, ensures data security and privacy through encryption and access control, and supports secure data sharing among multiple AI agents.
* **Interaction Interface Module**: Providing user interfaces and developer interfaces. This module includes natural language interaction interfaces, supporting text and voice input, as well as RESTful APIs and SDKs, supporting multiple programming languages.
* **Self-Learning Module**: Supporting online training and transfer learning. This module adjusts model parameters and computation strategies based on user feedback and task execution results, builds a shared knowledge base, and stores models and data for use by multiple AI agents.

**4.3 Application Scenarios of AI Agents**

AI agents can be applied in various scenarios, including:

* **Intelligent Customer Service**: Providing 24/7 intelligent customer service. AI agents can automatically answer user questions through natural language processing technology, improving customer service efficiency.
* **Medical Diagnosis**: Assisting doctors in disease diagnosis. AI agents can provide diagnostic suggestions by analyzing medical data, assisting doctors in decision-making.
* **Financial Analysis**: Providing financial market predictions and risk assessments. AI agents can predict market trends by analyzing financial data, helping financial institutions with risk assessment and investment decisions.
* **Autonomous Driving**: Achieving vehicle environment perception and path planning. AI agents can achieve autonomous driving functions through computer vision and sensor data.

**4.4 Interaction Mechanisms of AI Agents**

The interaction mechanisms of AI agents include:

* **User Interaction**: Interacting with users through natural language interfaces. Users can interact with AI agents through text or voice input to obtain the required information and services.
* **Developer Interface**: Providing RESTful APIs and SDKs, supporting multiple programming languages. Developers can integrate and extend the functionality of AI agents by calling APIs or using SDKs.
* **Multi-Agent Collaboration**: Achieving collaboration among multiple AI agents through the blockchain network. Multiple AI agents can communicate and collaborate through the blockchain network to complete complex tasks.

**4.5 Performance Optimization of AI Agents**

The performance optimization strategies of AI agents include:

* **Resource Scheduling Optimization**: Dynamically adjusting resource allocation to ensure efficient utilization. AI agents can dynamically allocate computation resources based on task requirements, improving resource utilization.
* **Algorithm Optimization**: Improving computation efficiency through model compression and distributed training. AI agents can use pruning and quantization techniques to reduce model computation complexity and use distributed computing frameworks to accelerate model training.
* **Data Optimization**: Improving data quality through data preprocessing and caching techniques. AI agents can use data cleaning and augmentation techniques to improve data quality and use caching technology to reduce data retrieval time.

**4.6 Future Expansion of AI Agents**

The future expansion plans for AI agents include:

* **Supporting More Algorithms**: Integrating more advanced AI algorithms. AI agents can support more AI algorithms to meet the needs of different application scenarios.
* **Cross-Chain Collaboration**: Collaborating with AI agents on other blockchain platforms. AI agents can collaborate with AI agents on other blockchain platforms through cross-chain technology to achieve broader applications.
* **Enhancing Privacy Protection**: Introducing technologies such as federated learning to further improve data privacy protection. AI agents can use federated learning and other technologies to achieve model training and optimization while protecting data privacy.

**5. Miner Maintenance and Management**

**5.1 Remote Monitoring and Management**

The EAI platform integrates advanced remote monitoring and management technologies to achieve real-time monitoring and management of miners. The EAI platform has three main components: server, agent, and web application.

Of course, there is more software supporting these three components, such as Windows Server Installer, ClickOnce applications, MeshCentral Discovery Tool, etc. These will be introduced later. Most of this document will focus on these three main components. Another important but not part of the software itself is Intel® AMT (Intel® Active Management Technology). MeshCentral supports Intel AMT, which serves as an optional hardware agent for MeshCentral. In terms of programming languages, MeshCentral is mainly built using JavaScript, and the agent contains a large amount of portable C code. This makes things quite simple, as the browser, server, and agent can share some code. More importantly, JavaScript is excellent at parsing JSON, so the main protocol used between components is WebSocket-based JSON.

It should be noted that although JavaScript is used in all three components, the JavaScript runtime is very different. JavaScript running in the browser sandbox is different from NodeJS running on the server or DukTape used on the agent. This may be an introduction to DukTape, which is a lesser-known JavaScript runtime written in C. The agent is built with C code, and apart from being able to securely connect back to the server, it has little intelligence. The server then pushes a JavaScript file to the agent, which the agent runs. This makes the agent very flexible, as developers can quickly change the JavaScript pushed to the agent and immediately alter the agent's behavior.

Another interesting design decision is that MeshCentral almost never uses RESTful APIs. Instead, almost everything is done using WebSocket. This allows JSON objects to be exchanged completely asynchronously. There is no need for a refresh button or polling, as all participants send events in real-time.

5.1.1 **EAI Server**

The EAI server is a NodeJS application, and EAI can run on Node 6.x and later versions.

**Dependencies**

The server uses the following dependencies on NPM. These dependencies are automatically installed by NPM when EAI is installed.

* "archiver": "^3.0.0", handling ZIP archives.
* "body-parser": "^1.18.2", handling HTTP form submissions.
* "compression": "^1.7.3", handling ZIP archives.
* "connect-redis": "^3.4.0", HTTP session storage.
* "cookie-session": "^2.0.0-beta.3", cookie-based HTTP sessions.
* "express": "^4.16.4", handling HTTP requests.
* "express-handlebars": "^3.0.0", modifying HTML response documents.
* "express-session": "^1.15.6", handling HTTP sessions.
* "express-ws": "^4.0.0", handling WebSocket.
* "minimist": "^1.2.0", parsing input parameters.
* "multiparty": "^4.2.1", handling HTTP form submissions.
* "nedb": "^1.8.0", lightweight MongoDB alternative.
* "node-forge": "^0.7.6", encryption library.
* "util.promisify": "^1.0.0", Node 6/7 poly-fill.
* "ws": "^6.1.2", WebSocket client.
* "xmldom": "^0.1.27", XML parsing.
* "yauzl": "^2.10.0", handling ZIP archives.

The main takeaway is that EAI is primarily an ExpressJS application. This is not a complete list of dependencies, as many of these packages have their own dependencies, forming a large dependency tree. The security of these dependencies is a concern. In addition to these "hard-coded" dependencies, there are some dependencies that are only installed when needed. These dependencies include:

* node-windows: Installed in all Windows installations, allowing background service installation.
* greenlock, le-store-certbot, le-challenge-fs, le-acme-core: Installed only when Let's Encrypt is needed.
* mongojs: Installed when MongoDB is used.
* nodemailer: Installed when SMTP server support is needed.

When these optional modules are needed but not currently available, EAI will automatically run "npm install".

**Code Files and Folders**

One might think that the code files of the EAI server are quite simple. At a high level, the entire server has 3 folders, 3 text files, and a moderate number of .js files, which are named quite intuitively. Below is a list of source files and folders.

* agents: Compiled agents, installation scripts, tools, and agent JavaScript.
* public: Static web elements, such as images, CSS, HTML, etc.
* views: Main web application, login interface, and messaging application.

**Configuration and Text Files**

* package.json: Describes the MeshCentral package on NPM.
* sample-config.json: A sample "config.json" file for getting started.
* readme.txt: A readme file released with the MeshCentral package.

**Code Files**

* amtevents.js: Used to decode Intel AMT WSMAN events.
* amtscanner.js: Used to scan local networks for Intel AMT computers.
* amtscript.js: Used to run Intel AMT scripts from MeshCommander.
* certoperations.js: Used to generate and perform certificate operations.
* common.js: Various common methods.
* db.js: Used to access MongoDB or NeDB databases.
* exeHandler.js: Used to modify Windows executables.
* interceptor.js: Used to insert credentials into HTTP streams.
* letsencrypt.js: Used to obtain and use Let's Encrypt certificates.
* meshaccelerator.js: Used to offload RSA signatures to other CPU cores.
* meshagent.js: Used to communicate with agents.
* meshcentral.js: This is the main module, starting the server.
* meshmail.js: Used to send SMTP mail.
* meshrelay.js: Used to relay WebSocket connections between agents and browsers.
* meshscanner.js: Used for EaiCentral server discovery in LAN mode.
* meshuser.js: Used to communicate with browsers.
* mpsserver.js: Used to communicate with Intel® AMT CIRA.
* multiserver.js: Used for server-to-server communication.
* pass.js: Executes password hashing + salt.
* redirserver.js: Used to handle HTTP traffic.
* swarmserver.js: Used to upgrade legacy MeshCentralv1 agents.
* webserver.js: Handles HTTPS traffic.
* winservice.js: Used for server background installation on Windows.

At a high level, the MeshCentral.js file will start the server. By default, it will start webserver.js on port 443, redirserver.js on port 80, and mpssrver.js on port 4433. The webserver.js file will create a meshuser.js or meshagent.js instance when a user or agent connects. Other files support various purposes, but this is the basic way the server works.

**Server Database**

In server design, an important decision is the choice of database. We wanted to use a database that could scale, so we chose the NoSQL database MongoDB. On the other hand, we wanted the server to be very simple for users who want to try or manage 100 computers or less. We did not want to add a learning curve for users who do not need MongoDB. It turns out that we can have both. NeDB is an NPM package that provides a simple MongoDB-like API while being fully implemented in NodeJS. For most people, this is enough to get started. By default, EAI will create and use a NeDB database, but it can be configured to use MongoDB. The internal code paths for both databases are almost identical, so the "db.js" file handles both in almost the same way, and the database used is completely abstracted from the rest of the server code.

5.1.2**Certificates**

EAI uses many certificates to complete many security tasks. When the server or agent is first run, both components generate certificates. The agent will generate one or two certificates when first run, and the server will generate four certificates.

In this section, we introduce the generated certificates, their uses, and how they are stored. Most EAI administrators do not need to delve into this section to run the server, but a basic understanding of this section can help understand how to best protect the server's critical security assets.

**Server Certificates**

As mentioned above, the EAI server generates four certificates when first run. It uses ForgeJS to perform certificate creation, and the following four certificates are all saved in the "meshcentral-data" folder.

* Server root certificate ("root-cert-public.crt"): This is a self-signed root certificate, only used to issue the following three certificates. In some cases, it is very useful to install this certificate as a trusted root certificate. For example, when Intel AMT connects to the MPS server's 4433 port, if this root certificate is loaded into Intel AMT as a trusted certificate, it will connect correctly. Browsers can also be set to trust this root certificate to create a trusted connection between the browser and the server's HTTPS port. This certificate is RSA3072, unless the "--fastcert" option is used, in which case an RSA2048 certificate will be generated.
* MPS certificate ("mpsserver-cert-public.crt"): This is a TLS server certificate signed by the above root certificate, used for the MPS port 4433. Intel AMT computers will connect to this port and verify the certificate time, common name, and whether it is signed by the above root certificate. This certificate usually does not change when the server is running in a production environment. This certificate is always generated as RSA2048, as older Intel AMT firmware does not support larger keys.
* Web certificate ("webserver-cert-public.crt"): This is the default certificate used to protect the HTTPS port 443. It is signed by the above root certificate and is the first certificate users see when connecting to the server in a browser. Usually, users need to ignore the browser's security warning. This certificate is RSA3072, unless the "--fastcert" option is used, in which case an RSA2048 certificate will be generated. In a production environment, this certificate will be replaced with a real certificate. In a production environment, there are several ways to change this certificate to use a more appropriate certificate:
	+ Replace the "webserver-cert-\*" files in the "meshcentral-data" folder.
	+ Use Let's Encrypt, which will automatically overwrite this certificate.
	+ Use a reverse proxy in front of the server and use the "--tlsoffload" option.
* Agent certificate ("agentserver-cert-public.crt"): This certificate is used for the server to authenticate with the agent. It is signed by the above root certificate, and when the agent is installed, the hash of this certificate will be provided to the agent so that the agent can securely connect back to the server. This certificate is RSA3072, unless the "--fastcert" option is used, in which case an RSA2048 certificate will be generated.

The "meshcentral-data" folder contains critical server information, including private keys, so it needs to be properly protected. It is important to back up the "meshcentral-data" folder and store the backup in a secure location. For example, if the "agent certificate" on the server is lost, the agent will no longer be able to connect to this server. All agents need to be reinstalled with a new trusted certificate. If someone reinstalls the server, putting the "meshcentral-data" folder back in place, containing these certificates, should allow the server to resume normal operation and accept Intel AMT and agent connections as before.

**Agent Certificates**

The agent generates one or two RSA certificates when first started. On small IoT devices like Raspberry Pi, this may take some time, and the CPU will spike to 100% during this period. This is normal and only happens when the agent is first run.

The way certificates are generated varies by platform. On Windows, the Mesh Agent will use the Microsoft cryptographic provider to strengthen the agent root certificate. If available, the agent will use the platform TPM to strengthen the certificate. On other platforms, only one certificate is generated and used for agent-to-server authentication and WebRTC session authentication.

* Agent root certificate: This is the trust root of the agent. The SHA384 hash of this certificate's public key is the agent's identifier on the server. When the agent connects to the server using WebSocket, it uses this certificate for secondary authentication. The server will calculate the agent's identifier after the agent sends a signature proof. This certificate is also used to sign the secondary certificate below when needed.
* Secondary certificate: This is a certificate signed by the above agent root certificate. Currently, it is only used by WebRTC for dTLS authentication of remote browsers. For WebRTC purposes, this certificate does not need to be signed by a trusted CA, as the hash of the certificate will be sent to the browser through a trusted path. If the agent root certificate is not strengthened using platform cryptography, the secondary certificate will not be created, and the agent root certificate will be used for all purposes.

If someone gains access to the agent root certificate, an attack may occur. They can impersonate the agent to connect to the server. The agent does not have permission to perform administrative operations on the server or other agents, but by impersonating the agent, a malicious agent may pretend to be an office computer, and administrators may log in using their username and password, especially when the root certificate is not strengthened. Some measures should be taken to protect the "meshagent.db" file, and important information should not be provided to untrusted agents.

5.1.3 **TLS Security**

EAI extensively uses Transport Layer Security (TLS) and Datagram TLS (dTLS) to authenticate and encrypt network traffic between browsers, servers, and agents. Properly configuring TLS settings is crucial to ensuring communication security and minimizing attacks on open ports. Perhaps the most important TLS configuration is the EAI server's ports 443 and 4433. These two ports are exposed to the internet, so they should be set as securely as possible.

**EAI HTTPS Port 443**

The EAI server's HTTPS port only supports TLS 1.2 and later, and only uses 6 cipher suites:

* TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384 (0xc030)
* TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384 (0xc028)
* TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA (0xc014)
* TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256 (0xc027)
* TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA (0xc013)

**EAI MPS Port 4433**

The Manageability Presence Server (MPS) port 4433 is used to receive Intel AMT CIRA connections. By default, it uses a TLS certificate signed by a self-signed root certificate. This port is not intended to be connected by typical browsers; only Intel AMT should connect to this port. Please note that the TLS certificate generated by EAI for port 4433 is RSA 2048-bit, as older Intel AMT firmware does not support larger keys. Since this port is not protected by a trusted certificate, SSL Labs will not rate the server's security.

This is entirely expected. Note that SSL Labs does not test servers that are not on port 443. To perform such a test, EAI would need to temporarily set the MPS port to 443 and set the regular HTTPS port to another value. Because many older Intel AMT computers only support TLS 1.0, the server supports TLS v1.0, v1.1, and v1.2, as well as the following 12 cipher suites:
• TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384 (0xc030)

• TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384 (0xc028)

• TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA (0xc014)

• TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384 (0x9d)

• TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256 (0x3d)

• TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA (0x35)
• TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (0xc02f)
• TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256 (0xc027)
• TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA (0xc013)
• TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (0x9c)
• TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256 (0x3c)
• TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA (0x2f)
Cipher suites starting with "TLS\_RSA\_" do not have perfect forward secrecy (PFS) and are therefore considered weak. However, these are typically the suites supported by Intel AMT.
5.1.4 Proxy and Server Handshake
An interesting aspect of MeshCentral's design is how the proxy connects to the server. We want the proxy to connect to the server in a way similar to how a browser connects to a web server. This allows a large number of proxy connections, just like a large number of browser connections. All the infrastructure that helps web servers scale can be used in the same way for proxy connections. For example: TLS offloading hardware, load balancers, reverse proxies, web server scaling, etc. This also makes the server easier to set up, as both users and proxies only need one port (HTTPS 443). A major difference between proxy connections and typical browsers is how the server is authenticated. Browsers have a set of known trusted root certificates. The server's web certificate is checked for validity, including name, time, trusted CA, etc. The proxy does not have these. Instead, it only has a hash pointing to a private server certificate. The server's public-facing web certificate can change frequently. For example, Let's Encrypt certificates are valid for 90 days. The proxy needs to be able to verify a specific server over a long period and does not need to trust anything else except that specific server. We also do not want to bind the proxy to a specific domain name, as we may change the domain name in the future or want to support servers with dynamic IP addresses and no fixed DNS name. To handle all of this, the proxy will perform a TLS connection to the server and first see the server's web certificate. Then, it will exchange a set of WebSocket binary messages with the server for secondary authentication.
The secondary check allows the proxy to confirm that the server indeed possesses the private key of the private certificate expected by the proxy. The proxy caches the hash of the "external" web certificate. When reconnecting, if the proxy sees the same external web certificate, it will skip the secondary check. For obvious security reasons, the proxy should not accept any management messages until the secondary check is completed or skipped. To prevent man-in-the-middle attacks, the secondary check also "pins" the external web certificate. That is, the server both confirms that it is the correct server and indicates the hash of the external certificate that the proxy must see during the TLS connection. The proxy must check this hash to ensure there is no attacker in the middle.
The proxy connection design allows the use of reverse proxies and TLS offloading hardware. The proxy will first connect to the TLS session of the offloading hardware. The plaintext traffic between the offloading hardware and the server will perform the secondary check as needed.
For all of this to work, the MeshCentral server must be able to obtain the hash of the external web certificate from the reverse proxy. In this case, the server does not need the private key of the web certificate. Note that when the external web certificate is updated, the server may need to perform many secondary checks simultaneously, which may cause the server to slow down during this period. To help with this, MeshCentral offloads RSA signing operations to many subordinate processes (as many as the server's CPU cores) to speed things up. Additionally, native NodeJS RSA signing is used (instead of ForgeJS). The details of the secondary certificate check are shown in the figure below. To improve speed, the exchange is completely asynchronous, with both parties sending the first message immediately after the TLS connection is established.
Note that these messages are binary (not JSON). The proxy must be able to connect to the server independently of the JavaScript running in DukTape. Therefore, this exchange is handled by native C code in the proxy. Binary message 1 is sent immediately after the TLS connection is established. Both parties send binary message 2 after receiving message 1, and send message 3 after receiving message 2. Additionally, the proxy may send two extra messages at the beginning. If the secondary check can be skipped, the proxy can send server message 4 and can send binary message 5, indicating the server hash it expects to verify. Message 5 is interesting because the server may have many "identities" at the same time, so the server will use message 5 to use the correct proxy server certificate. To be as secure as possible, all hashes use SHA384, certificates are RSA3072, and both parties use random numbers generated by a cryptographic random source. The server and proxy's signature calculations are as follows:
Although the server usually skips RSA signing operations because the proxy caches the external web certificate, the server must perform RSA verification for each proxy connection. This cannot be skipped but is necessary to authenticate the proxy. Once connected, the trust relationship between the server and the proxy is one-way. That is, the server has administrative rights over the proxy, but the proxy has no rights over the server. This is important because the proxy does not have any credentials of the server by default. Any proxy can connect to the server and claim to be part of a device group.
5.1.5 Browser-to-Proxy Relay and WebRTC
Browsers and proxies often need to communicate with each other. Data sessions are used for desktops, terminals, file transfers, etc., and must be securely established. To establish a session between the browser and the proxy, the server will send a connection URL to both parties. This URL is generated by the server and contains a unique connection token. It is sent to the browser and proxy using the WebSocket control channel and JSON messages. Both parties perform a WebSocket connection to the target URL, and the server will "pipe" the two sessions together, acting as a passive relay. For security reasons, the proxy will only accept a connection to the URL provided by the server if the server has the same external web certificate as its control connection. Also note that in this mode, the session is not end-to-end encrypted. The server performs TLS decryption and re-encryption, and each data byte needs to be received and sent, which is costly in terms of traffic.
The relay server is simply a WebSocket server waiting for connections with a session token. When two connections with the same session token arrive, the server ensures that at least one connection is from an authenticated user, then sends the character "c" to both parties to notify them that the relay has started, and then connects the two sessions together. Once the session starts, the browser and proxy can freely send messages to each other. Note that when the server sends the relay URL to the proxy, it also sends the user's permission flags to the proxy. The proxy can use these flags to limit the actions the user can perform in the session. With this design, traffic control between the browser and the proxy is simple, with each session having its own end-to-end connection, and the server applying appropriate TCP backpressure to both parties as needed. A unique feature of MeshCentral is its use of WebRTC. WebRTC was initially introduced in major browsers to allow browsers to communicate directly with each other and perform audio/video streaming. The proxy has a custom-built WebRTC data stack for this project, implemented in C code. It is compatible with Chrome and Firefox implementations, allowing data to flow directly from the browser to the proxy once the session is established, bypassing the server.
Using WebRTC allows MeshCentral to scale better, provide a faster user experience, and reduce hosting costs at the same time. However, WebRTC is not simple, especially when it comes to maintaining its C code and keeping up with browser implementations, but the benefits are obvious. To set up WebRTC, browsers typically use STUN and TURN servers to get traffic through any network obstacles (routers, proxies, firewalls). This can complicate the infrastructure if the administrator is not familiar with WebRTC concepts. To simplify, MeshCentral chooses to always start with a WebSocket relay through the server. While the session is active, the browser and proxy will attempt to automatically switch session traffic to WebRTC when possible. This way, the session is always functional and becomes more efficient when network conditions allow.
To perform the switch, the browser and proxy will exchange WebRTC control messages through the newly established WebSocket relay session. To distinguish session traffic from WebRTC control traffic, the browser and proxy agree to send WebRTC setup traffic using WebSocket text fragments. All other session traffic is sent using binary fragments. The proxy has a special API that allows the session to be pipelined for a single fragment type. Therefore, we can perform a remote desktop session with the proxy while attempting to set up WebRTC. The browser will start the WebRTC setup by sending an initial WebRTC offer, and the proxy will respond with a WebRTC answer. If the WebRTC session is established, both parties need to negotiate a clean transition from the WebSocket session to the WebRTC session. To do this, both parties send a start switch control fragment (which is a text fragment), and the other party responds with an ACK after the WebSocket session is cleared, at which point it is safe to switch. On the proxy side, the new WebRTC session inherits the user access permissions of the WebSocket. Currently, the WebSocket channel remains open. While this is not strictly necessary, the termination of the WebSocket session is cleaner than WebRTC, so its closure is used to indicate the end of the WebRTC session.
5.1.6 Messaging
EAI includes its own messaging web application, which can be used for chat, file transfer, and optionally for audio and video chat. It supports two different uses: user-to-user and user-to-computer communication. In the first use, two users connected to the same EAI server can chat simultaneously. If you are an EAI administrator, you can view the list of currently logged-in users and click the chat button to initiate a chat invitation. If the other party accepts, the messaging application will open on both sides, and the session will begin. Alternatively, when managing a remote computer, the administrator can click the chat button, causing the remote computer to open a web browser pointing to the chat application.
The chat application is a standalone web application provided by the EAI server using a connection token and title in the URL. Once loaded in its own web box, the messaging web application will fetch the connection token and title from the URL and then connect to the URL using WebSocket. The same WebSocket relay used for browser-to-proxy connections is also used for browser-to-browser connections in this case. The server relay performs the same operation, connecting the two sessions together after sending the character "c" to both parties. At this point, the messaging application will show that the remote user is connected, and chat and file transfer can begin. File transfer is simply a set of binary messages sent through the WebSocket session, accompanied by many JSON control messages. Once the WebSocket session is established, the messaging application will attempt to switch to WebRTC. Both web applications start with a random number (non-cryptographic), and the highest number will initiate the WebRTC offer. The other party will respond, and both parties will exchange interface candidates. If successful, the WebSocket session will be cleared, and traffic will switch to WebRTC. Since the switch is clean, it can be done during file transfer without causing file corruption.
Finally, the web application will determine if the local computer has a microphone and camera connected. If so, these options will be available in the chat window, and audio/video chat will be available. The chat application allows for one-way setup of audio and video sessions. This is typically needed in support scenarios, where audio/video sessions are one-way. The messaging web application will set up separate WebRTC connections for each direction of audio/video, but there is an option in the code to enhance the WebRTC control channel for audio/video, which is more efficient but requires more testing before being used by default.
5.2 Troubleshooting and Updates
When a miner fails, the EAI platform provides various tools and features to help users quickly locate and resolve issues. At the same time, the platform supports remote updates of miners, ensuring that miners are always running in the best condition.
• Remote Diagnostics: The EAI platform provides remote diagnostics, allowing users to remotely diagnose miners through the platform and quickly locate the problem. This includes remotely viewing the miner's logs, running status, and other information to help users quickly find the root cause of the problem.
• Troubleshooting: Users can troubleshoot miner issues through the EAI platform, including remotely fixing miner faults. The platform provides various troubleshooting tools to help users quickly resolve miner issues.
• Remote Updates: The EAI platform supports remote updates of miners, allowing users to remotely update the software and firmware of miners through the platform. This ensures that miners are always running in the best condition, improving the efficiency and stability of miners.
5.3 Security and Privacy Protection
The EAI platform places a high priority on security and privacy protection during miner maintenance and management. Through various security measures, the platform ensures the security and privacy of miner data.
• Encrypted Communication: The EAI platform uses encrypted communication technology to ensure the security of communication between miners and the platform. All data is encrypted during transmission to prevent data theft or tampering.
• Access Control: The platform implements strict access control mechanisms to ensure that only authorized users can access and manage miners. Users need to pass identity verification to log in to the platform and manage and operate miners.
• Permission Management: The EAI platform manages user permissions to ensure that users can only access and manage the miners they are authorized to. This prevents unauthorized users from accessing and operating miners, protecting the security and privacy of miners.
6. Security and Privacy
6.1 Data Encryption
The EAI platform places a high priority on the application of data encryption technology during miner maintenance and management. All communication between miners and the platform uses encryption technology to ensure the security of data during transmission. Specifically:
• Transmission Encryption: The EAI platform uses encryption protocols such as TLS 1.2 and TLS 1.3 to ensure the security of communication between miners and the platform. These protocols support forward secrecy (PFS), effectively preventing data theft or tampering during transmission.
• Storage Encryption: The EAI platform encrypts sensitive data stored in the database, including user credentials and miner configurations. By using encryption algorithms such as AES-256-GCM, the platform ensures the security of data during storage.
6.2 Access Control
The EAI platform implements strict access control mechanisms to ensure that only authorized users can access and manage miners. Specifically:
• Identity Verification: Users need to pass identity verification to log in to the platform and manage and operate miners. This ensures the security of the system, preventing unauthorized access.
• Permission Management: The EAI platform manages user permissions to ensure that users can only access and manage the miners they are authorized to. Users can authorize their mining farms to others in the permission center or revoke authorization.
**6.3 Privacy Protection**
The EAI platform places a high priority on privacy protection during miner maintenance and management. Through various privacy protection technologies, the platform ensures the privacy of miner data. Specifically:
• Zero-Knowledge Proof Technology: The EAI platform uses zero-knowledge proof technology, allowing information to be verified without revealing specific content. This provides users with higher privacy protection, suitable for scenarios requiring privacy protection, such as the processing of sensitive data like financial transactions.
• Data Anonymization: The EAI platform anonymizes miner data to ensure the privacy of data during transmission and storage. This prevents user data from being precisely identified, protecting user privacy.
**7. Conclusion**
The EAI project has built a decentralized computing platform based on blockchain, achieving efficient utilization of computing resources and intelligent management of AI agents. Through key technologies such as multimodal computing resource pools, mining mechanisms, and AI agent architecture, the EAI platform provides users with efficient, flexible, and scalable AI computing capabilities. At the same time, through advanced security and privacy protection technologies, the platform ensures the security and privacy of miner data. In the future, the EAI platform will continue to optimize and expand, supporting more AI application scenarios and promoting the widespread application and development of AI technology.
**8. Appendix**
8.1 Glossary
• AI Agent: An intelligent agent running on the EAI platform, responsible for executing AI computing tasks and interacting with users, developers, and other AI agents.
• Multimodal Computing Resources: Including GPU, CPU, storage, and other computing resources, used to support different types of AI computing tasks.
• Elastic Computing Resource Pool: Through dynamic resource allocation, the platform achieves efficient utilization of computing resources, supporting the elastic scaling of AI computing tasks.
• Mining Mechanism: A mechanism to incentivize miners to provide computing resources, ensuring the efficient execution of tasks through task allocation, computation and verification, and reward distribution.
• Blockchain Network: Using blockchain technology to achieve decentralized resource allocation and task management, ensuring the security and transparency of the system.
8.2 Technical Specifications
• Hardware Requirements: Supports various computing devices, including GPU, CPU, storage, etc.
• Software Requirements: Supports various operating systems and programming languages, including Python, Java, JavaScript, etc.
• Network Requirements: Supports high-speed network connections, ensuring fast data transmission and efficient task execution.
8.3 Other Resources
• Developer Documentation: Provides detailed developer documentation and API interfaces, supporting developers in integrating and extending the functionality of the EAI platform.
• Tutorial Videos: Provides tutorial videos to help users quickly get started and use the EAI platform.
• Community Support: Provides community support and forums where users can exchange experiences and solve problems.