Chapter 1:

Computation verification

Introduction

Outsourcing the computation of large language models (LLMs) to third-party hardware providers is a promising approach to address the resource constraints of companies seeking powerful AI solutions. However, this introduces critical challenges in verifying that the specified model was executed faithfully, and the output is authentic. Ensuring trust, security, and accountability in such outsourced computations is essential, particularly when dealing with proprietary models or sensitive data.

To tackle these challenges, we propose two complementary solutions:

1. **Zero-Knowledge Proofs for LLMs (zkLLM)**:
    - zkLLM leverages advanced cryptographic protocols to allow third-party executors to generate a cryptographic proof that the computation was performed correctly. The proof ensures that the correct model and input query were used without revealing sensitive details about the model or the data. This solution offers high trust and security but requires specialized infrastructure and computational resources.
2. **Machine Learning-Based Verification**:
    - In this approach, a separate machine learning model is trained to analyze the output and identify whether it was generated by the specified LLM. By learning the unique patterns and characteristics of the target model's outputs, this method provides a scalable and flexible solution without modifying the LLM or the inference process. While simpler to implement, it may face challenges in handling adversarial modifications or unseen cases.

These solutions address the problem from two distinct angles: zkLLM ensures cryptographic proof of execution, while the machine learning model provides a probabilistic method for output verification. Together, they offer a comprehensive framework to establish trust in outsourced LLM computations.

Solution 1: Zero-Knowledge Proofs for LLMs (zkLLM)

zkLLM is a cryptographic framework designed to ensure verifiable execution of large language models (LLMs) using **Zero-Knowledge Proofs (ZKPs)**. The key idea is that a third-party executor can prove they used the specified model to compute the given input and produce the output without revealing the underlying model parameters or the input data. zkLLM achieves this through efficient protocols tailored for LLM operations, such as transformer attention mechanisms, enabling secure and scalable verification.

This innovative approach is detailed in the paper, *zkLLM: Zero Knowledge Proofs for Large Language Models*, which introduces core components like **tlookup** for non-arithmetic operations and **zkAttn** for attention mechanisms. The official implementation is available on: https://github.com/jvhs0706/zkllm-ccs2024

Detailed Explanation of zkLLM

The **zkLLM framework** introduces an innovative method for verifiable computations of large language models (LLMs) using **Zero-Knowledge Proofs (ZKPs)**. This solution allows third-party executors to prove that they have correctly executed a specified LLM on a given input without revealing sensitive model parameters or input data.

How zkLLM Works

1. **Model Commitment**:
    - The model parameters are hashed into a cryptographic commitment, creating a "fingerprint" of the LLM. This ensures that any tampering with the model parameters can be detected.
2. **Input Execution**:
    - The LLM processes the input query to generate an output. While doing so, the zkLLM framework captures intermediate computations for proof construction.
3. **Proof Generation**:
    - The computations performed by the LLM (e.g., matrix multiplications, attention mechanisms) are converted into **arithmetic circuits**, representing the operations as a series of mathematical equations.
    - zkLLM uses two specialized protocols:
        - **tlookup**: Verifies non-arithmetic operations, such as activation functions, efficiently.
        - **zkAttn**: Ensures the correctness of attention mechanisms, balancing computational complexity and proof size.
4. **Proof Submission**:
    - The executor submits the output and the generated proof to the verifier.
5. **Proof Verification**:
    - The verifier uses zkLLM's algorithms to confirm that:
        - The committed model was used for the computation.
        - The provided output matches the input and the model.

Applications and Use Cases

- **Auditing AI Workflows**:
    - Ensures that third-party executors faithfully execute specified LLMs.
- **Privacy-Preserving AI Services**:
    - Verifies computations while keeping the model and inputs confidential.

Limitations

- **Early Stage Development**:
    - Although their gitbhub is public and we can use their repo, but zkLLM has been tested with **LLaMA-2 models** (up to 13 billion parameters) and achieves proof generation in under 15 minutes with proofs under 200 kB. However, the authors

have explicitly stated that the framework is **not yet ready for industrial applications** due to:
    - Lack of security audits.
    - The interactive nature of the current implementation, which is less practical for many real-world scenarios.

Integration in Our Use Case

To use zkLLM in our project:

1. **Deployment**:
    - The specified LLM and input query are securely provided to the executor.
    - The executor runs the zkLLM framework during inference to generate the proof.
2. **Verification**:
    - The proof is validated by our system, ensuring the correct model and inputs were used.
3. **Challenges**:
    - Executors must have robust hardware capable of generating ZKPs efficiently.
    - zkLLM's interactive nature requires modifications or extensions (e.g., implementing the Fiat-Shamir heuristic) to enable non-interactive proofs for scalability.

Solution 2: Training a Machine Learning Model for Output Verification

An effective approach to verifying the authenticity of outputs generated by large language models (LLMs) is to train a machine learning model that distinguishes between outputs produced by a specific LLM and those from other sources. This solution relies on learning the unique patterns, styles, and statistical characteristics inherent in the outputs of the target model.

How This Solution Works

1. **Data Collection**:
    - Build a labeled dataset comprising:
        - Outputs from the target LLM (e.g., LLaMA-3).
        - Outputs from other LLMs (e.g., GPT, T5) or human-written text.
    - Incorporate diverse prompts covering various domains (e.g., question answering, summarization, conversational responses).
    - Reference datasets such as **HC3**, **GROVER**, or **MGTBench**, which provide examples of LLM-generated and human-authored text, can be valuable for training detectors: https://github.com/NLP2CT/LLM-generated-Text-Detection,

https://ar5iv.org/html/2411.06248v1

2. **Feature Extraction**:
    - Utilize features like:
        - **Statistical Metrics**:
            - Perplexity (likelihood of the sequence under a language model).
            - Token distribution and usage frequencies.
        - **Linguistic Patterns**:
            - Stylistic tendencies such as sentence length, structure, and vocabulary diversity.
            - Semantic coherence and repetitiveness.
    - Tools like **GLTR** (Giant Language Model Test Room) and **DetectGPT** analyze token-level probabilities to identify artifacts in LLM-generated text: https://cacm.acm.org/research/the-science-of-detecting-llm-generated-text/, https://ar5iv.org/abs/2303.07205

3. **Model Selection**:
    - Fine-tune a Transformer-based model (e.g., BERT, RoBERTa) on the labeled dataset to classify outputs as originating from the target LLM or another source.
    - Lightweight classifiers (e.g., Logistic Regression) can also be used for simpler applications but may lack robustness.
4. **Training and Evaluation**:
    - Train the detection model using supervised learning techniques.
    - Validate its performance on unseen data to ensure generalization.
    - Conduct adversarial testing by introducing paraphrased or transformed outputs to evaluate robustness.
5. **Deployment**:
    - Integrate the trained detection model into the pipeline. Upon receiving an output from a third-party executor, the model predicts whether the output likely originates from the LLM.

Real-World Applications

Several practical efforts and research works highlight the feasibility of this solution:

- **DetectGPT**:
    - A probabilistic method that leverages the curvature of log probabilities under the LLM to identify whether the text is machine-generated: https://ar5iv.org/abs/2305.16617

- **GROVER**:
    - Trained a model to detect outputs from GROVER itself, demonstrating the effectiveness of self-detection mechanisms:

https://github.com/NLP2CT/LLM-generated-Text-Detection ,

- **AI Text Classifiers**:
  - OpenAI developed a tool to classify whether text was generated by its GPT models, underscoring the utility of such detectors for practical scenarios like content moderation or academic integrity: https://cacm.acm.org/research/the-science-of-detecting-llm-generated-text/ .
- **Academic Use**:
  - Benchmarks like HC3 (Hello ChatGPT Comparative Corpus) have been used to train and evaluate detectors on LLM outputs across domains: https://ar5iv.org/html/2411.06248v1

Advantages

- **No Modification to the LLM**:
  - Works directly with outputs without requiring changes to the LLM or its sampling process.
- **Flexible and Scalable**:
  - Adaptable to multiple LLMs by retraining or fine-tuning the detection model.
- **Established Research**:
  - Builds on existing research and datasets, reducing the need for custom data collection.

Challenges

1. **Dataset Limitations**:
   - High-quality labeled datasets are required to capture the nuances of the target LLM's outputs.
   - Outputs may need to be augmented to handle adversarial modifications.
2. **Generalization**:
   - Detecting outputs from evolving LLMs or unseen prompts remains challenging.
3. **Adversarial Evasion**:
   - Sophisticated transformations (e.g., paraphrasing) can obscure identifying patterns.

Integration in Our Use Case

1. **Dataset Preparation**:
   - Generate a labeled dataset using different LLMs under diverse prompts and collect outputs from other LLMs for comparison.
2. **Model Training**:

    - Fine-tune an existing detector or train a new classifier using features tailored to the LLM's outputs.
3. **Deployment**:
    - Integrate the trained model into the pipeline to verify outputs received from third-party executors.

This solution complements cryptographic approaches like zkLLM by providing a probabilistic verification mechanism that is simpler to implement and scale, particularly for non-critical applications.

Handling Complex Inputs like PDFs and Documents

In scenarios where the user provides complex inputs such as PDF files or other document formats, accompanied by a task-specific prompt (e.g., summarization or question answering), additional steps are required to ensure accurate verification. These types of inputs demand a tailored approach for both the generation and verification of outputs.

Preprocessing the Input

1. **Text Extraction**:
    - Extract the text from the input file using tools like **PyPDF2**, **PDFMiner**, or OCR technologies for scanned documents. This ensures the extracted content matches the user's input.
    - Address potential errors in OCR by applying preprocessing techniques such as spell-checking and format normalization.
2. **Input Structuring**:
    - For lengthy documents, preprocess the content into smaller, manageable sections or key points before passing it to the LLM. This ensures that the outputs are coherent and aligned with the prompt.

Generating and Verifying Outputs

1. **Primary Task Verification**:
    - The LLM processes the extracted text based on the user's prompt (e.g., summarization). The output is compared to the expected patterns learned by the detection model.
    - Features specific to the primary task, such as **content overlap**, **compression ratio**, and **linguistic coherence**, are analyzed.
2. **Secondary Prompt Integration**:
    - A secondary, unrelated prompt is sent to the LLM alongside the primary task to independently verify that the executor used the intended model.
    - The detection model evaluates the output of the secondary prompt to determine if it aligns with the characteristics of the target LLM.

Training the Detection Model

1. **Dataset Expansion**:
    - Include document-based inputs (e.g., academic papers, reports) and their corresponding LLM outputs in the training dataset.
    - Incorporate both primary task outputs (e.g., summaries or answers) and secondary task responses (e.g., answers to unrelated questions).
2. **Task-Specific Features**:
    - Develop features tailored to document-based tasks:
        - **Compression Ratio**: The reduction in text length during summarization.
        - **Content Overlap**: Degree of alignment between key points in the document and the summary.
        - **Question Relevance**: Accuracy and completeness of responses to document-specific queries.
3. **Adversarial Testing**:
    - Introduce adversarially modified examples (e.g., paraphrased summaries, reordered content) to ensure robustness.

Challenges and Mitigations

1. **Diversity of Input Formats**:
    - Documents may vary in length, style, and format. Preprocessing pipelines must handle this diversity effectively.
2. **Secondary Prompt Design**:
    - Select secondary prompts that are challenging enough to differentiate between models yet do not require additional context.

Scalability

This approach can be scaled by regularly updating the training dataset with new document types and prompts, ensuring the detection model remains accurate as user demands evolve.

While solutions like watermarking and behavior-based detection exist, they come with limitations that make them less suitable for our specific use case. **Watermarking**, for example, involves embedding hidden patterns in the generated outputs during the model's inference process. While effective in controlled environments, watermarks can be removed or obscured through adversarial transformations like paraphrasing or summarization. Moreover, implementing watermarking requires modifications to the LLM's generation process, which may not always be feasible.

Behavior-based detection methods, such as zero-shot approaches like **DetectGPT**, rely on analyzing the statistical or semantic properties of outputs. These methods can provide useful insights but often lack robustness when faced with outputs from advanced LLMs or adversarial modifications: https://ar5iv.org/html/2411.06248v1, https://ar5iv.org/abs/2305.16617

Given these challenges, we focus on zkLLM and machine learning-based detection as more practical and effective solutions that align better with our requirements for scalability, security, and reliability. Let me know if you'd like me to refine or expand this section further.

Chapter 2:

blockchain integration

1- Token Implementation

To facilitate the reward mechanism for verified computations, we propose the creation of a custom cryptocurrency token using the widely adopted **ERC-20 standard**. This token will serve as the primary medium of exchange within our platform, rewarding users for successful verification of tasks and enabling transparent, secure, and efficient transactions.

Token Design

1. **Token Name and Symbol**:
    - The token will have a unique name and ticker symbol to distinguish it from other cryptocurrencies.
2. **Token Properties**:
    - **Fungible**: All tokens will be identical in value and functionality, making them interchangeable.
    - **Divisible**: Tokens will support fractional values to allow flexible rewards.
    - **Mintable**: The token smart contract will include minting functionality for controlled creation of new tokens as needed.
3. **ERC-20 Standard**:
    - The ERC-20 standard ensures compatibility with existing wallets, exchanges, and blockchain tools. It defines the essential functions for token creation, transfer, and balance management, such as:
        - `totalSupply`: Tracks the total number of tokens in circulation.
        - `balanceOf`: Returns the token balance of a specific address.
        - `transfer`: Moves tokens between addresses.
        - TransferFrom
        - Approve: approve a user to transfer from my account
        - allowance

Smart Contract Implementation

The token smart contract will be developed and deployed on a blockchain that supports Ethereum Virtual Machine (EVM), such as Ethereum, Binance Smart Chain, or Polygon. Using libraries like **OpenZeppelin**, we will implement secure and audited contract templates, ensuring reliable token behavior.

Key Features

1. **Ownership Control**:
    - A designated owner account (platform administrator) will control token minting and allocation.
2. **Transparency**:

    - The token smart contract will be deployed on a public blockchain, allowing anyone to verify token transactions and supply.
3. **Extensibility**:
    - The contract will be designed to integrate seamlessly with future components of the platform, such as staking or governance mechanisms.

Deployment Plan

1. **Smart Contract Development**:
    - Use OpenZeppelin's ERC-20 implementation as a base.
    - Customize features to meet platform requirements.
2. **Testing**:
    - Deploy the contract to a testnet (e.g., Ropsten or Mumbai) and simulate transactions to ensure correct behavior.
3. **Mainnet Deployment**:
    - Deploy the final smart contract to the chosen mainnet, initializing the token supply.

This token will be the cornerstone of the reward mechanism, ensuring a seamless and transparent experience for all stakeholders.

Some of the most suitable networks for deploying the ERC-20 token and their respective gas fee considerations:

- **Ethereum**
    - **Gas Fees**: Ethereum's gas fees are the highest among EVM-compatible networks, often ranging between $5 to $50+ per transaction depending on network congestion.
    - **Advantages**: High security and wide adoption, making it ideal for a token with long-term plans to integrate with major DeFi and NFT platforms.
    - **Use Case**: Best for tokens requiring maximum trust and integration into high-value ecosystems.
- **Polygon (MATIC)**
    - **Gas Fees**: Extremely low, typically less than $0.01 per transaction.
    - **Advantages**: Low fees and high throughput, making it cost-effective for frequent token transfers. Polygon also supports zk-rollups for added security and scalability.
    - **Use Case**: Excellent for platforms targeting a large number of transactions or micro-rewards.
- **Binance Smart Chain (BSC)**
    - **Gas Fees**: Around $0.10 to $0.30 per transaction.
    - **Advantages**: Cost-effective and faster than Ethereum while maintaining a robust ecosystem for tokens and dApps.
    - **Use Case**: Ideal for platforms with moderate transaction volumes and compatibility with Binance's extensive ecosystem.

- **Optimism**
    - **Gas Fees**: Ranges from $0.05 to $0.30 per transaction, significantly lower than Ethereum.
    - **Advantages**: Optimistic rollups provide scalability while leveraging Ethereum's security. It's a good balance between cost and reliability.
    - **Use Case**: Suitable for platforms needing scalability without compromising on Ethereum's base-layer trust.
- **Arbitrum**
    - **Gas Fees**: Similar to Optimism, ranging between $0.05 to $0.30.
    - **Advantages**: Uses Optimistic rollups with multi-round fraud-proof systems for added transaction integrity.
    - **Use Case**: Ideal for platforms needing both scalability and a growing ecosystem.

Security Measures

To ensure the security and integrity of our blockchain token implementation, we will adopt a multi-faceted approach that encompasses both cryptographic techniques and best practices in software development. The following security measures will be implemented:

- **Smart Contract Security**
    - Code Audits: Prior to deployment, our smart contract will undergo rigorous code audits conducted by reputable third-party security firms. This process will identify vulnerabilities and ensure compliance with industry standards.
    - Use of Established Libraries: We will utilize well-established libraries such as OpenZeppelin, which provide secure implementations of common functionalities (e.g., ERC-20 token standards). This reduces the risk of introducing vulnerabilities through custom code.

User Interaction with Token

The successful implementation of our blockchain token hinges on creating an intuitive and engaging user experience. Users will interact with the token primarily through the following mechanisms:

- **Earning Tokens:** Users will earn tokens by participating in the verification process of computations performed by third-party executors. The earning process includes:
    - computationTasks: Users can submit computation tasks where they run a LLM and submit the result. If the result is verified, they receive the token as reward.

- **Spending Tokens:**Tokens earned can be utilized within the platform for various purposes:
    - Access to Services: Users can spend their tokens to access premium features or services within the platform, such as advanced analytics tools or priority support.

    - Transaction Fees: Tokens may also be used to pay for transaction fees associated with submitting verification tasks or accessing specific functionalities.
- **Token Management**: Users will have control over their tokens through a user-friendly interface that allows them to:
    - View Balances: Users can easily check their token balances and transaction history through their account dashboard.
    - Transfer Tokens: The platform will enable users to transfer tokens to other users, facilitating peer-to-peer transactions and collaborations.
    - Withdrawals: Users will have the option to withdraw their tokens to external wallets or exchanges, providing flexibility and liquidity.

- **Involvement in Governance:** As token holders, users will have a voice in the governance of the platform:
    - Voting Rights: Token holders will be able to participate in governance decisions, such as proposing changes to the verification process or voting on new features. This democratic approach fosters community engagement and aligns the interests of users with the platform's development.

2- Tokenomics

The tokenomics of our blockchain token is designed to create a sustainable and incentivizing ecosystem that encourages user participation, ensures fair distribution, and supports the long-term growth of the platform. The following components outline the key aspects of our tokenomics model:

- **Token Supply**
    - **Total Supply:** The total supply of tokens will be capped at 1,000,000 tokens. This fixed supply is intended to create scarcity and enhance value over time.
    - **Initial Distribution:** Tokens will be distributed as follows:
        - **Team and Advisors**: **20%** allocated to the founding team and advisors to incentivize their ongoing commitment to the project.
        - **Community Incentives: 40%** reserved for community rewards, including verification tasks, referrals, and engagement incentives.
        - **Development Fund: 20%** set aside for ongoing development and operational costs, ensuring continuous improvement of the platform.
        - **Reserve Fund: 10%** held in reserve for future partnerships or unforeseen expenses.
        - **Marketing and Partnerships**: **10%** allocated for marketing efforts and strategic partnerships to expand the platform's reach.

- **Earning Mechanisms**
    - **Computation Rewards:** Users will earn tokens by successfully completing computations. The reward structure will be tiered based on the complexity of tasks completed, with higher rewards for more challenging verifications.
    - **Staking Rewards:** Users can stake their tokens to support network security and operations. In return, they will receive additional tokens as staking rewards, promoting long-term holding and reducing circulating supply.

- **Utility of Tokens**
    - **Access to Services:** Tokens can be used to unlock premium features within the platform, such as advanced analytics tools or priority access to new functionalities.
    - **Transaction Fees:** Users will pay transaction fees in tokens when submitting verification tasks or transferring tokens between accounts. This creates a demand for tokens within the ecosystem.
    - **Governance Participation:** Token holders will have voting rights on key decisions affecting the platform's development and operations. This democratic approach aligns user interests with the success of the platform.

- **Burning Mechanism:** To enhance the value of our token and create a deflationary effect within the ecosystem, we propose a systematic burning mechanism that utilizes a portion of the transaction fees collected on the platform:
    - **Transaction Fee Allocation:** We will allocate **20% of all transaction fees** collected from user activities (such as verification task submissions and token transfers) to buy back tokens from the open market.
    - **Burning Process**: The purchased tokens will be permanently removed from circulation (burned) using a transparent process that will be recorded on the blockchain. Regular updates on the amount of tokens burned will be communicated to the community, fostering trust and engagement.

- **Market Dynamics**
    - **Liquidity Provisioning:** To enhance liquidity, we will establish partnerships with decentralized exchanges (DEXs) where users can trade our token. This will facilitate price discovery and allow users to convert tokens into other cryptocurrencies or fiat as needed.
    - **Deflationary Pressure:** By consistently reducing the total supply of tokens through regular burns, we aim to create upward pressure on token value, benefiting long-term holders.

- **Long-Term Sustainability**
    - **Continuous Evaluation:** The tokenomics model will be regularly evaluated based on user feedback and market conditions. Adjustments may be made to ensure that it remains effective in promoting engagement and sustainability.
    - **Community Engagement:** We will actively engage with our community to gather insights on potential changes to tokenomics, ensuring that it evolves in line with user needs and expectations.

3- Smart Contract for Reward Distribution

To facilitate the automatic distribution of tokens as rewards for verified computations, we will implement a smart contract that governs the reward mechanism within our platform. This smart contract will ensure transparency, security, and efficiency in the reward process, enabling seamless interactions between users and the system. The key components of the smart contract are outlined below:

Smart Contract Overview

The smart contract will be developed using the ERC-20 standard, which provides a robust framework for creating fungible tokens on the Ethereum blockchain. This standard ensures compatibility with existing wallets and exchanges, facilitating easy management and transfer of tokens.

- **Core Functions**
  - **Reward Allocation:** The contract will define a function to allocate tokens to third-party executors upon successful verification of their computations. This function will:
    - Accept parameters such as the executor's address and the amount of tokens to be rewarded.
    - Ensure that only verified computations trigger reward allocation.
  - **Verification Process:** The contract will integrate with our verification framework (e.g., zkLLM or machine learning-based verification) to confirm that computations were executed correctly. The verification process will involve:
    - Accepting cryptographic proofs or validation results from the verification system.
    - Ensuring that rewards are only distributed when valid proofs are submitted.
- **Integration with Web3**
  - To enable seamless interaction between users and the smart contract, we will leverage Web3 technologies. This integration allows users to connect their wallets (e.g., MetaMask) directly to our platform, facilitating decentralized interactions. Key aspects include:
    - **User Wallet Interaction:** Users can interact with the smart contract through their Web3-enabled wallets, allowing them to submit computation

results and receive rewards directly in their wallets without relying on centralized intermediaries.
        - **Event Listening:** The smart contract will emit events upon successful reward allocation and other significant actions. Our frontend application will listen for these events using Web3.js or Ethers.js, providing real-time updates to users about their reward status.
- **Security Features**
    - To safeguard against unauthorized access and potential vulnerabilities, the smart contract will implement several security measures:
        - **Access Control:** Role-based access control (RBAC) will restrict sensitive functions (e.g., minting) to authorized addresses only. This prevents unauthorized users from manipulating token supply or distribution.
        - **Audit Trails:** All transactions related to reward allocation and token minting will be logged on the blockchain, creating an immutable audit trail that enhances transparency and accountability.
        - **Emergency Pausing:** The smart contract will include an emergency pause feature that allows administrators to halt operations in case of detected vulnerabilities or security breaches. This feature helps protect user funds while issues are addressed.

- **Deployment and Testing**
    - Prior to deployment on the mainnet, the smart contract will undergo rigorous testing on a testnet environment to identify any potential issues or vulnerabilities. Testing procedures will include:
        - **Unit Testing:** Each function within the smart contract will be tested individually to ensure correctness.
        - **Integration Testing:** The entire reward distribution system, including interactions with the verification framework and Web3 components, will be tested to confirm seamless operation.
        - **Security Audits:** A third-party security audit will be conducted to identify any weaknesses in the code and ensure compliance with best practices.

- **User Interaction**
    - Once deployed, users (third-party executors) will interact with the smart contract through a user-friendly interface that allows them to submit their computation results along with cryptographic proofs via their Web3-enabled wallets. Upon successful verification, they will receive their token rewards automatically without manual intervention.