Dear Reviewers,

In this attachment, we present complementary details to support our response in the rebuttal letter.

Part 1. Prompt Design

1.1 Prompt Template

There are two prompt templates regarding the numeric and boolean kinds of inconsistency-related information. Table 1 shows the four inconsistency types related to numeric, which require extracting numeric values from the DApp description. Table 2 shows the three inconsistency types related to the Boolean Prompt Template.

(1) Numeric

Prompt Template

System Prompt (SP): You are an expert in blockchain and smart contracts. Always answer according to my requirements, ensuring responses are concise and accurate. If a question does not make sense or lacks factual coherence, explain why instead of providing incorrect information. If you do not know the answer to a question, do not share false information.

User Prompt (UP)-Numeric
UP_{it}: Extract numerical information related to the { %INCONSISTENCY_NUMERIC_INFO } from the provided text.

UP_{CoT}: **None**

DApp Description (D): The text I provide is: { %TEXT }

Figure 1. Prompt Template of Numeric kind of Inconsistency-related Information

Table 1. Inconsistency Numeric Information in Numeric Prompt Template

Inconsistency Type	Inconsistency Numeric Info
Unguaranteed Reward (UR)	rate of reward or profit
Hidden Fee (HF)	rate of fee or tax
Adjustable Liquidity (AL)	lock time
Unconstrained Token Supply (UTS)	total amount or supply

(2) Boolean

Prompt Template

System Prompt (SP): You are an expert in blockchain and smart contracts. Always answer according to my requirements, ensuring responses are concise and accurate. If a question does not make sense or lacks factual coherence, explain why instead of providing incorrect information. If you do not know the answer to a question, do not share false information.

User Prompt (UP)-Boolean
UP_{it}: Whether this text indicates { %INCONSISTENCY_BOOLEAN_INFO }. You need to answer 'yes' or 'no' and provide a concise explanation.

UP_{CoT}: Think step by step: First, the definition of { %KEY_PHRASE } is: { %DEFINITION }. Second, if { %SCENARIO }, there may be some descriptions like: { %EXAMPLE }. This is just an example to help you understand. The actual situation is not limited to such descriptions.

DApp Description (D): The text I provide is: { %TEXT }

Figure 2. Prompt Template of Boolean kind of Inconsistency-related Information

Table 2. Information in Boolean Prompt Template

Inconsistency Type	Information in Prompt Template
Unclaimed Fund Flow (UFF)	**Inconsistency boolean info:** The existence of a method for clearing or withdrawing all tokens/assets contained in the contract by someone. **Key phrase:** clearing assets **Definition:** An act of clearing contract assets by calling specific functions in the contract (usually only accessible to privileged users) to transfer all tokens or ethers. **Scenario:** DApp can clear the assets in the contract **Example:** includes a feature that allows for the transfer of all contract assets to the project owner for safety reasons.
Changeable DApp Status (CDS)	**Inconsistency boolean info:** the DApp can be paused **Key phrase:** DApp pause **Definition:** A DApp pause refers to a feature embedded within a smart contract of a DApp, allowing the contract owner to temporarily suspend specific functionalities, such as token minting. **Scenario:** DApp can be paused **Example:** for the safety and security of our users, in rare emergency cases or to address potential vulnerabilities, the DApp owner can temporarily suspend specific functionalities, such as token minting or transfers.
Volatile NFT Accessibility (VNA)	**Inconsistency boolean info:** the NFT is stored in centralized server **Key phrase:** stored in centralized server **Definition:** NFTs are stored using HTTPS, or suggests the possibility of the server shutting down. **Scenario:** NFT is not stored in a centralized server **Example:** NFTs can be accessed forever or are stored via IPFS.

1.2 Illustrative Examples

We use two real-world DApps as examples to illustrate the process of constructing prompts and how prompt segmentation and CoT patterns work.

The First Example: Extracting Fee information from DApp TrueFund

As we intend to extract numeric fee values from the *TrueFund* frontend description, we select our Numeric prompt template in Figure 1. The inconsistency of numeric information in Hidden Fee from Table 1 is used to form the complete user prompt. After concatenating the raw text of the DApp description, the prompt for *TrueFund* is shown in the below picture.

Prompt for TrueFund (Hidden Fee)
System Prompt (SP): You are an expert in blockchain and smart contracts. Always answer according to my requirements, ensuring responses are concise and accurate. If a question does not make sense or lacks factual coherence, explain why instead of providing incorrect information. If you do not know the answer to a question, do not share false information. **User Prompt (UP)-Numeric** Extract numerical information related to the rate of fee or tax from the provided text. **DApp Description (D):** The text I provide is: TrueFund description...

Figure 3. Constructed Prompt for Extracting TrueFund's Fee Information

1.2.1 Prompt Segmentation

The aim of prompt segmentation is to improve the LLM's performance by reducing the input length it processes each time, as some DApp frontends may contain very long text making constructed prompt exceed the LLM token limit.

The token length of this constructed prompt (Figure 3) is bigger than our threshold 3000 as the description of *TrueFund* is quite long. Therefore, we need to segment this prompt. Suppose the current token length of this prompt P is L (L>3000), we first trim the exceeded part and obtain the prompt P1 (3000 token length). Then, we construct the second prompt P2, which also begins with SP and UP, as shown in the picture, but description D is the trimmed part. Therefore, both of these two prompts satisfy the 3000 token limit.

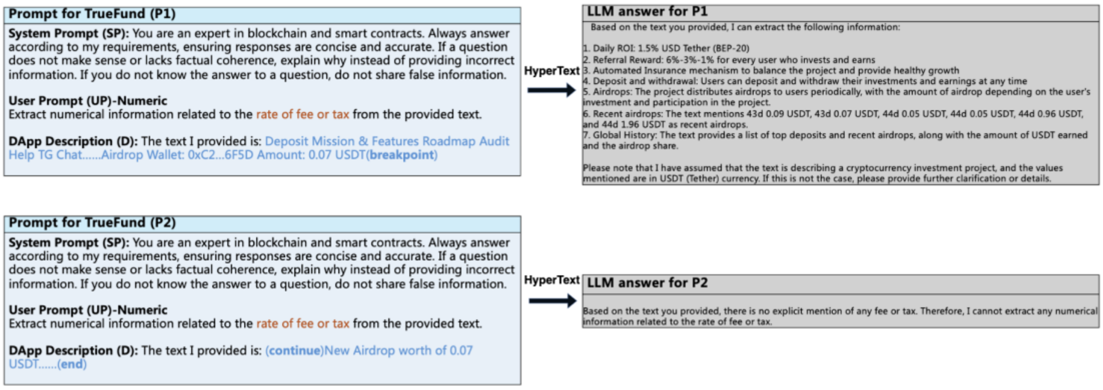

Figure 4. LLM Answers of Segmented Prompts of DApp TrueFund

Then, we feed these two prompts to HyperText and obtain two answers, respectively.

1.2.2 CoT Pattern

We adopt CoT prompting in extracting Boolean kind of inconsistency-related information and the CoT pattern is shown by UP_{CoT} in Figure 2. We now use the second case to illustrate the details.

The Second Example: Extracting DApp Pause Information (Boolean) from DApp BRN Metaverse

To extract boolean information whether a DApp can be paused or not from the DApp description, we refer to the Boolean prompt template (Figure 2) and Table 3 to construct the below prompt following the CoT patterns.

Prompt for BRN Metaverse (Changeable DApp Status)

System Prompt (SP): You are an expert in blockchain and smart contracts. Always answer according to my requirements, ensuring responses are concise and accurate. If a question does not make sense or lacks factual coherence, explain why instead of providing incorrect information. If you do not know the answer to a question, do not share false information.

User Prompt (UP)-Boolean
UP_{it}: Whether this text indicates the DApp can be paused. You need to answer 'yes' or 'no' and provide a concise explanation.

UP_{CoT}: Think step by step: First, the definition of "DApp pause" is: a DApp pause refers to a feature embedded within a smart contract of a DApp, allowing the contract owner to temporarily suspend specific functionalities, such as token minting. Second, if DApp can be paused, there may be some descriptions like: for the safety and security of our users, in rare emergency cases or to address potential vulnerabilities, the DApp owner can temporarily suspend specific functionalities, such as token minting or transfers. This is just an example to help you understand. The actual situation is not limited to such descriptions.

DApp Description (D): The text I provide is: BRN Metaverse description

Figure 5. Constructed Prompt for DApp BRN Metaverse (CDS Inconsistency)

Specifically, we use the CoT prompting that first gives the definition of DApp pause, and then provide a specific scenario with example to help LLM yield boolean answers more accurately. The answer of HyperText is shown below:

LLM answer for BRN Metaverse

No, the text does not indicate that the DApp can be paused. There is no mention of the word "pause" or any related phrases that suggest the DApp can be temporarily halted. Therefore, based on the text provided, it can be concluded that the DApp cannot be paused.

Figure 6. HyperText's Answer with Boolean Value "No"

Part 2. NLTK Workflow

We use the outputs of HyperText of the two examples in Part 1 (Figure 4 and Figure 6) to illustrate the NLTK workflow.

(1) *TrueFund* Case for Extracting Fee Information (Figure 4)

We obtain two answers as we input segmented prompts P1 and P2 to HyperText. For each answer, we try to find fee information by NLTK.

Specifically, we first tokenize the sentence and use POS to tag every word. Then we scan the word to locate the keyword "fee" or "tax" and their synonyms extended by NLTK WordNet. However, the answer for P1 contains many numeric values, but there is nothing related to the fee, and answer for P2 shows no related information. Therefore, we do not capture any keyword to find the fee information, thus yielding that there is **{none}** related to fee information claimed in the frontend.

We also provide another case *JellyBeanBag* that NLTK helps to find reward rate from DApp description.

LLM answer for JellyBeanBag

Sure, I'd be happy to help! Here's the numerical information related to the rate of reward or profit from the provided text:
1. Daily Return: 8%
2. Bean APR (Annual Percentage Rate): 2,920%
3. Dev Fee (Developer Fee): 3%
Please note that these rates are based on the information provided in the text and may not reflect the current or future performance of the jellybeanbag smart contract. Additionally, it's important to remember that investing in cryptocurrency is risky and there is always a chance of losing money. It's important to do your own research and understand the risks before investing.

Figure 7. HyperText's Answer for Judging Whether JellyBeanBag Can be Paused or not

Following the step, we first use the POS tag to label the word, and then locate the keyword and find the "daily return". Next, we search the context of the phrase and find a symbol "%" before the line break. Collaborating with the tag information, we find the cardinal digit (CD) **{8}** and record it as the extracted **{daily return: 8%}**. It is notable that the "APR" can also be found, but we convert it into a daily rate, which is also 8%.

(2) *BRN Metaverse* Case for Extracting DApp Pause Information (Figure 6)

As we use CoT pattern to instruct the LLM to answer in a boolean way, which has a prominent symbol, the method can be straightforward. After we use NLTK to tokenize the answer in Figure 6, we can directly scan words and extract the keyword **{No}** by keyword matching, thus knowing that the DApp description indicates the DApp cannot be paused.

Part 3. Further Details about Inconsistency Definition

When checking inconsistency cases for definition and experimental results, three of the authors cross-checked the contract code to identify whether they break the hypotheses to be DApp inconsistency issues, ensuring the correctness. We show two cases to illustrate the inconsistency issues further.

3.1 Unguaranteed Reward

Figure 8 shows the complete code of examples in Fig. 2 of the paper. Assume a buyer deposits 1 ETH to buy an egg (suppose the contract has 100 ETH). He can get *calculateEggBuy(1eth,100eth)*, which represents 2551343277 eggs (based on the calculation by the formula shown in line 35) after deducting the fee. When he sells the obtained eggs one day (86400s) later, he can only get 0.1646 ETH (after reducing a 6% fee) if the contract balance remains the same. The final reward rate is -83.54%, misleading users by claiming a +3% daily reward rate. It is more like Ponzi trap and is irresponsible to claim this constant positive reward rate of 3%, which is strongly related to the contract states and market (contract balance and market eggs) when selling eggs, thereby concealing the situation that users would lose most of their funds.

```
uint256 public EGGS_TO_HATCH_1MINERS=2592000;
uint256 PSN=10000;
uint256 PSNH=5000;
uint256 public marketEggs = 259200000000;

function sellEggs() public{
    require(block.timestamp > startTime);
    uint256 hasEggs=getMyEggs();
    uint256 eggValue=calculateEggSell(hasEggs);

    claimedEggs[msg.sender]=0;
    lastHatch[msg.sender]=now;
    marketEggs=SafeMath.add(marketEggs,hasEggs);

    uint256 baseFee=devFee(eggValue);
    mrkoneAddress.transfer(baseFee*25/10);      // 2.5% ((/ 10) divisor to handle decimal places)
    mrktwoAddress.transfer(baseFee*25/10);      // 2.5% ((/ 10) divisor to handle decimal places)
    devAddress.transfer(baseFee*1);             // 1%

    msg.sender.transfer(SafeMath.sub(eggValue,baseFee*6));}

function buyEggs(address ref) public payable{
    require(block.timestamp > startTime);
    uint256 eggsBought=calculateEggBuy(msg.value,SafeMath.sub(address(this).balance,msg.value));
    eggsBought=SafeMath.sub(eggsBought,devFee(eggsBought));

    uint256 baseFee=devFee(msg.value);
    fundAddress.transfer(baseFee*5);            // 5%
    devAddress.transfer(baseFee*1);             // 1%

    claimedEggs[msg.sender]=SafeMath.add(claimedEggs[msg.sender],eggsBought);
    hatchEggs(ref);}
//magic trade balancing algorithm
function calculateTrade(uint256 rt,uint256 rs, uint256 bs) public view returns(uint256){
    //(PSN*bs)/(PSNH+((PSN*rs+PSNH*rt)/rt));
    return SafeMath.div(SafeMath.mul(PSN,bs),SafeMath.add(PSNH,SafeMath.div(SafeMath.add(SafeMath.mul(PSN,rs),SafeMath.mul(PSNH,rt)),rt)));}
function calculateEggSell(uint256 eggs) public view returns(uint256){
    return calculateTrade(eggs,marketEggs,address(this).balance);}
function calculateEggBuy(uint256 eth,uint256 contractBalance) public view returns(uint256){
    return calculateTrade(eth,contractBalance,marketEggs);}
function devFee(uint256 amount) public pure returns(uint256){
    return SafeMath.div(SafeMath.mul(amount,1),100); // 1%}
function getMyEggs() public view returns(uint256){
    return SafeMath.add(claimedEggs[msg.sender],getEggsSinceLastHatch(msg.sender));}
function getEggsSinceLastHatch(address adr) public view returns(uint256){
    uint256 secondsPassed=min(EGGS_TO_HATCH_1MINERS,SafeMath.sub(now,lastHatch[adr]));
    return SafeMath.mul(secondsPassed,hatcheryMiners[adr]);}
```

Figure 8. Complete Code Example of Fig.2 in the Paper

3.2 Unconstrained Token Supply

Our sound analysis method aims to avoid missing potential issues, so whether the function is internally used is not the primary concern. The possibility that the token's total supply can be increased without limit by an unconstrained amount parameter contradicts the claim of a constant total supply.

Furthermore, there are cases where there is no way to process the result (e.g., burning tokens to reduce the supply). The following code shows the contract code of the DApp Sola Techno Alliance, which has a UTS inconsistency. The DApp frontend claims its token supply is 21 crore, but the contract disobeys this constraint and can surpass the supply in the function *mint()*.

```
function _burn(address account, uint256 amount) internal {
  require(account != address(0), "BEP20: burn from the zero address");
  _balances[account] = _balances[account].sub(amount, "BEP20: burn amount exceeds balance");
  _totalSupply = _totalSupply.sub(amount);
  emit Transfer(account, address(0), amount);}

function mint(uint256 amount) public onlyOwner returns (bool) {
  _mint(_msgSender(), amount);
  return true;}
```

Figure 9. Code Example of Sola Techno Alliance with Internal Burn Function

Part 4. Further Details on Contract Analysis

In contract analysis, the dataflow analysis is applied to induce contract semantics based on proposed rules, which are further combined to recover higher-level graphs for feeding into IR-based symbolic execution.

Rules 6 to 9 are designed to help us find the storage slots of some key state variables and identify the functions where they are loaded or stored. The SenderGuard relationship is used to obtain the state variable dependency graph (SDG). Specifically, SenderGuard recovers the relationship where a function has a sender verification on a specific state variable x. By combining this with rule 5, which finds the storage slot of the owner, the SDG can be recovered to show which critical state variables (obtained from rules 6 to 9) are operated in functions that should be invoked by a privileged user.

4.1 Graph Analysis

In contract analysis, the dataflow analysis is applied to induce contract semantics based on proposed rules, which are further combined to recover higher-level graphs for feeding into IR-based symbolic execution.

Rules 6 to 9 are designed to help us find the storage slots of some key state variables and identify the functions where they are loaded or stored. The SenderGuard relationship is used to obtain the state variable dependency graph (SDG). Specifically, SenderGuard recovers the relationship where a function has a sender verification on a specific state variable x. By combining this with rule 5, which finds the storage slot of the owner, the SDG can be recovered to show which critical state variables (obtained from rules 6 to 9) are operated in functions that should be invoked by a privileged user.

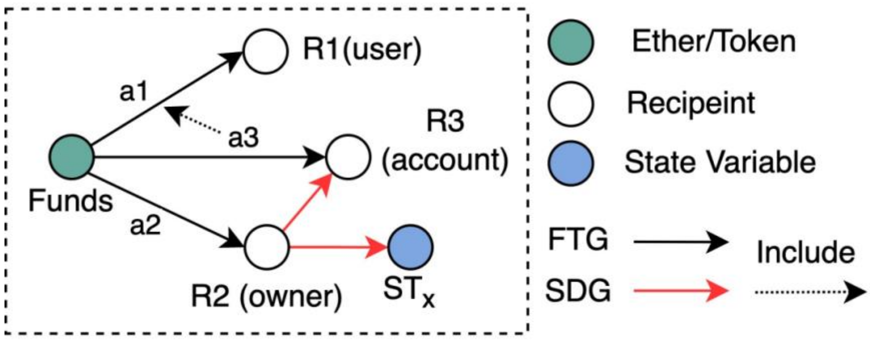

Figure 10. FTG and SDG Illustration Sample

We organize these two graphs to obtain execution information in function granularity, and Figure 10 shows an example of FTG and SDG.

Suppose we find the graph records information that in function F, statement S performs a transfer, the transfer amount is variable a3 in Figure 10, and the receiver is R3, which can be identified as a fee receiver by SDG. The symbolic executor can then read related variables by their names directly from the stored data structure (key-value dictionary) when executing statement S. Therefore, the executor can obtain the symbolic expressions or other information for further checks and analysis. Figure 11 shows an example of IR CFG, which our symbolic executor operates on.

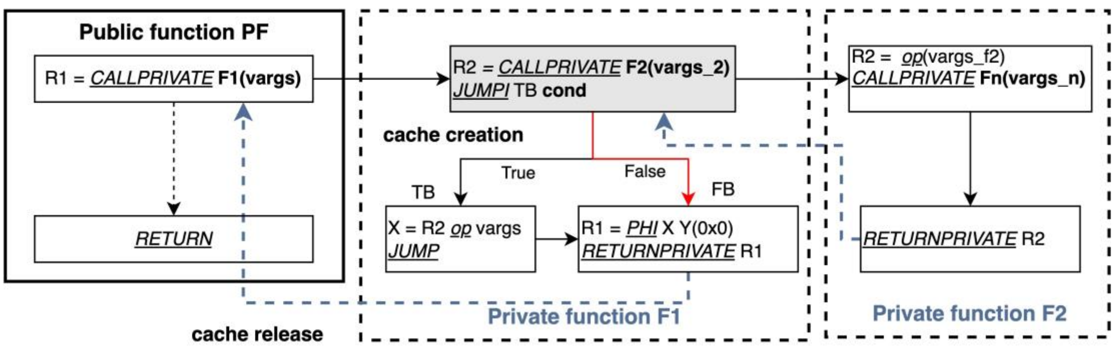

Figure 11. Example of IR CFG among Public and Private Functions

In summary, we use dataflow analysis to recover contract semantics, thereby obtaining graphs with high-level information. Since the extracted information is also at the IR level, our IR-based symbolic execution can be directly guided with pre-analyzed knowledge, thereby improving efficiency.

4.2 Workflow of Detecting Inconsistency

We provide two examples to illustrate how to detect inconsistencies: one for numeric check and another for state check.

(1) Hidden Fee

Taking the DApp *BNB Ultra* as an example (now inaccessible), the frontend only claims it has a daily return rate of 3%, with no mention of fee information. In the contract analysis, the *getCode()* API is used for Hyperion to retrieve the contract bytecode as it only requires to input with contract address and its blockchain platform.

We first identify the fee transfer operations that map to the frontend attribute based on whether the token receiver is a preset state variable or determined by the owner (SDG). For example, when executing function 0xdb663865 (*buyEgg()*, lines 17-27 in the below code), Hyperion identifies two statements transferring fees at 0x127eS0x6f0 and 0x12eaS0x6f0 to preset address Ia_store-5- (*fundAddress* in line23) and Ia_store-4- (*devAddress* in line24), respectively. The symbolic expressions for the transfer amounts, 5bvudiv_i(Iv, 100) and bvudiv_i(Iv, 100), are obtained, representing 5% and 1% of the user's deposit value. In other cases where the ratio is stored in state variables, e.g., a dividend expression like bvudiv_i(Ia_store-1-*Iv, 100) with the coefficient Ia_store-1/100, we then use *getStoragetAt()* API to find the storage value of slot 0x1 in this contract account and calculate the fee ratio.

```
1  function sellEggs() public{
2      require(block.timestamp > startTime);
3      uint256 hasEggs=getMyEggs();
4      uint256 eggValue=calculateEggSell(hasEggs);
5
6      claimedEggs[msg.sender]=0;
7      lastHatch[msg.sender]=now;
8      marketEggs=SafeMath.add(marketEggs,hasEggs);
9
10     uint256 baseFee=devFee(eggValue);
11     mrkoneAddress.transfer(baseFee*25/10);      // 2.5% ((/ 10) divisor to handle decimal places)
12     mrktwoAddress.transfer(baseFee*25/10);      // 2.5% ((/ 10) divisor to handle decimal places)
13     devAddress.transfer(baseFee*1);             // 1%
14
15     msg.sender.transfer(SafeMath.sub(eggValue,baseFee*6));}
16
17 function buyEggs(address ref) public payable{
18     require(block.timestamp > startTime);
19     uint256 eggsBought=calculateEggBuy(msg.value,SafeMath.sub(address(this).balance,msg.value));
20     eggsBought=SafeMath.sub(eggsBought,devFee(eggsBought));
21
22     uint256 baseFee=devFee(msg.value);
23     fundAddress.transfer(baseFee*5);            // 5%
24     devAddress.transfer(baseFee*1);             // 1%
25
26     claimedEggs[msg.sender]=SafeMath.add(claimedEggs[msg.sender],eggsBought);
27     hatchEggs(ref);}
28 function devFee(uint256 amount) public pure returns(uint256){
29     return SafeMath.div(SafeMath.mul(amount,1),100); // 1%}
```

Figure 12. Contract Code of BNB Ultra

Similarly, for the *sellEgg()* function (lines 1-15), we determine the baseFee coefficient and identify three fee transfer statements to three preset addresses, with a total fee ratio of 6%. Since the frontend does not mention any fee information, Hyperion reports the inconsistency.

(2) Volatile NFT Accessibility

Using the *Cryptoz NFT* DApp as an example (code shown in Figure 13), we identify how the NFTs are stored in the DApp from frontend description. To map the attribute, we refer to rule 6, which tells us where NFTs are actually stored. We find that the storage slot for token URI information is 0x1 (*baseTokenURI* in the code). Hyperion then uses the *getStorageAt()* API to query the storage content of this slot in the contract account and retrieves information starting with HTTP, indicating a centralized storage method. This contradicts the frontend's claim that the DApp's NFTs are stored permanently in a decentralized manner (e.g., IPFS). Therefore, Hyperion reports the inconsistency.

```
string baseTokenURI = 'https://cryptoz.cards/data/'; //append ID on storage
function tokenURI(uint256 _tokenId) external view returns (string memory) {
    return Strings.strConcat(
        baseTokenURI,
        Strings.uint2str(_tokenId));}
```

Figure 13. Token URI Function of Cryptoz NFT

Appendix A. Details of our Artifact Dataset

We put all datasets used in this paper into our repository. Many samples and further detailed illustrations are also given in it due to the paper's page limitations. Below are the direct accessible links for key datasets, which will be helpful for quick locating the artifact details.

DApp Dataset for Empirical Study and Experiments

(1) Collected DApp list from DAppRadar and DAppBay (https://anonymous.4open.science/r/Hyperion-C41D/DApp_dataset/DAPP-Collected.xlsx)

(2) Collected DApps for defining inconsistencies: https://anonymous.4open.science/r/Hyperion-C41D/DApp_dataset/risky_DAPPs/README.md

(3) Collected DApps for fine-tuning LLaMA2: https://anonymous.4open.science/r/Hyperion-C41D/DApp_dataset/fine_tune/README.md

(4) Collected DApps for the large-scale experiment: https://anonymous.4open.science/r/Hyperion-C41D/DApp_dataset/wilds/README.md

Mapping relationship between DApps with inconsistency types

https://anonymous.4open.science/r/Hyperion-C41D/DApp_dataset/inconsistency_map.csv

Experimental Results

(1) Ground truth experimental results: https://anonymous.4open.science/r/Hyperion-C41D/experiment_result/ground_truth/

(2) Large-scale experiment results: https://anonymous.4open.science/r/Hyperion-C41D/experiment_result/wild

Evaluation Results

(1) Evaluation result on ground truth dataset: https://anonymous.4open.science/r/Hyperion-C41D/experiment_result/ground_truth/exp_category_gt

(2) Evaluation result on large-scale dataset: https://anonymous.4open.science/r/Hyperion-C41D/experiment_result/wild/exp_category_wild

(3) Sampled dataset and evaluation result: https://anonymous.4open.science/r/Hyperion-C41D/experiment_result/wild/sampling_exp

FP/FN Analysis Samples and Case Studies

https://anonymous.4open.science/r/Hyperion-C41D/experiment_result/README.md

Issue Types Supported to Detect of Previous Works

https://anonymous.4open.science/r/Hyperion-C41D/experiment_result/issuetype_papertool.csv

Tool Implementation

(1) HyperText: https://anonymous.4open.science/r/Hyperion-C41D/HyperText/README.md

(2) HyperCode: https://anonymous.4open.science/r/Hyperion-C41D/README.md

Accessibility Check

https://anonymous.4open.science/r/Hyperion-C41D/experiment_result/check_accessibility.csv