PropertyGPT: LLM-driven Formal Verification of Smart Contracts through Retrieval-Augmented Property Generation

Ye Liu[1], Yue Xue[2 †], Daoyuan Wu[3 *], Yuqiang Sun[4], Yi Li[4], Miaolei Shi[2], and Yang Liu[4,5]

[1]Singapore Management University

[2]MetaTrust Labs

[3]The Hong Kong University of Science and Technology

[4]Nanyang Technological University

[5]China-Singapore International Joint Research Institute (CSIJR)

† co-first authors.

* corresponding author.

Smart Contracts and Security Issues

- Script programs on blockchain
- Critical asset management
- Susceptible to attacks

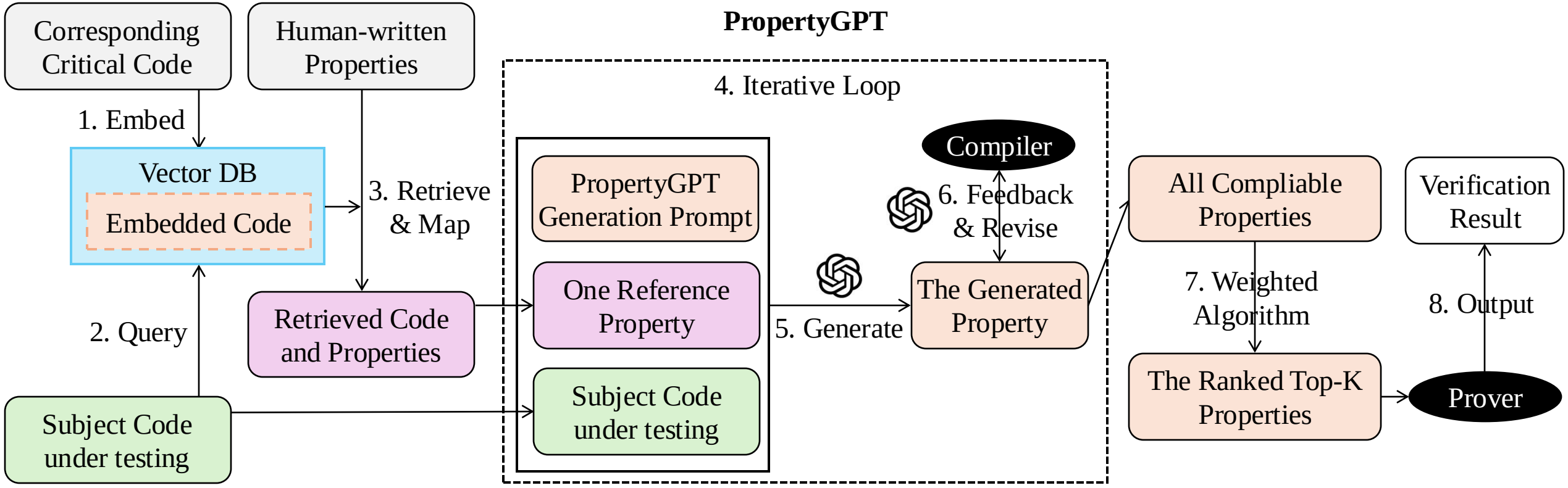
https://hacken.io/insights/2024-security-report/

Formal Verification and Challenges

- Advantage: finding **deeper** bugs beyond testing/static analysis
- SOTAs:
  - Symbolic execution (Manticore, Mythril, Oyente, MAIAN, teEther,...)
  - Theorem proving (Why3,...)
  - Model checking (EthBMC,...)
- However,
  - Limited use cases – targeting common vulnerabilities
  - Labor-intensive process – requiring substantial efforts to provide input

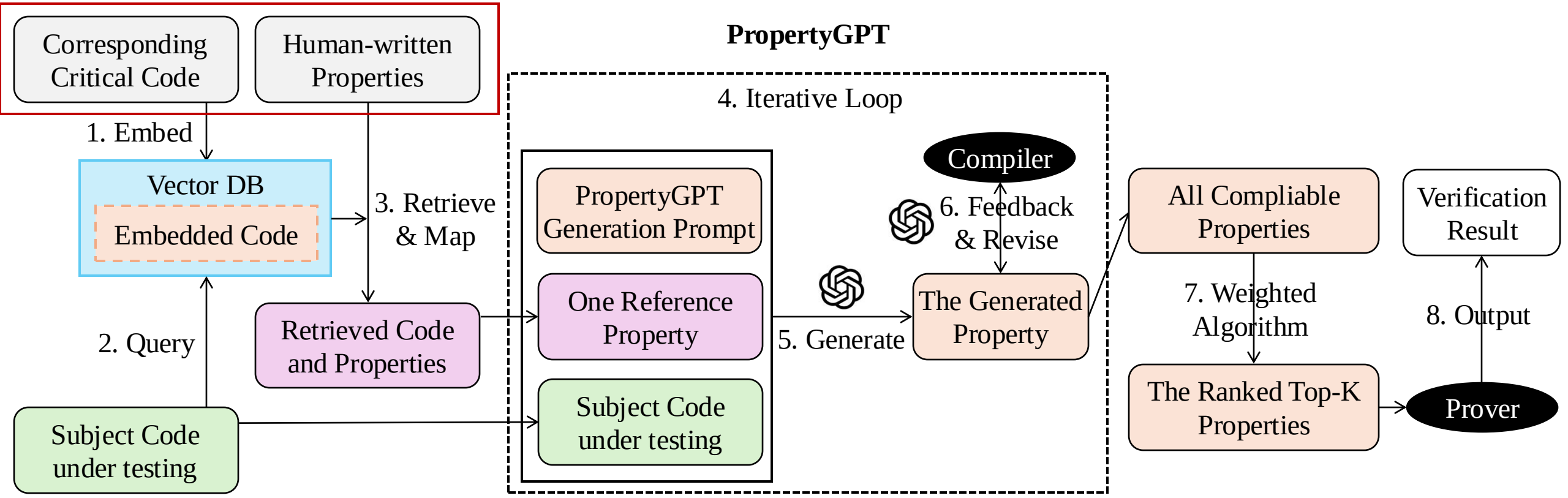
How to automatically adapt formal verification for different contracts and finding application specific vulnerabilities?

LLM-driven Formal Verification



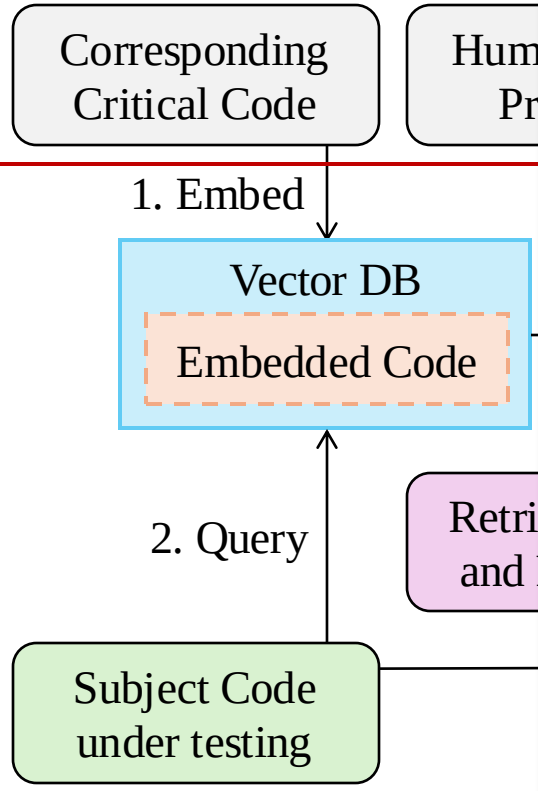
Powered with RAG-based Property Generation

LLM-driven Property Generation



LLM-driven Property Generation

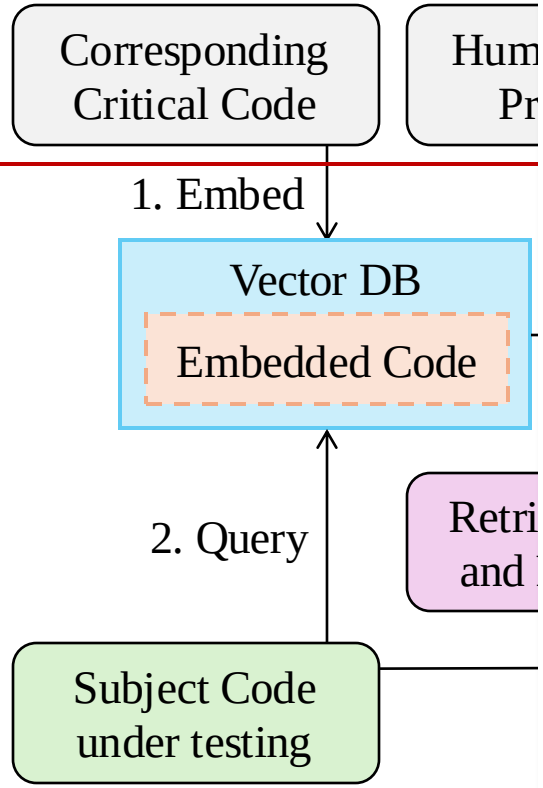
An ERC20Vote property about "mapping(address=> Checkpoint[]) private _checkpoints"




```
1  hook Sstore _checkpoints[KEY address account][INDEX uint32 index].votes
     uint224 newVotes (uint224 oldVotes) STORAGE {
2      havoc userVotes assuming
3        userVotes@new(account) == newVotes;
4
5      havoc totalVotes assuming
6        totalVotes@new() == totalVotes@old() +newVotes - userVotes(account);
7
8      havoc lastIndex assuming
9        lastIndex@new(account) == index;
10 }
```

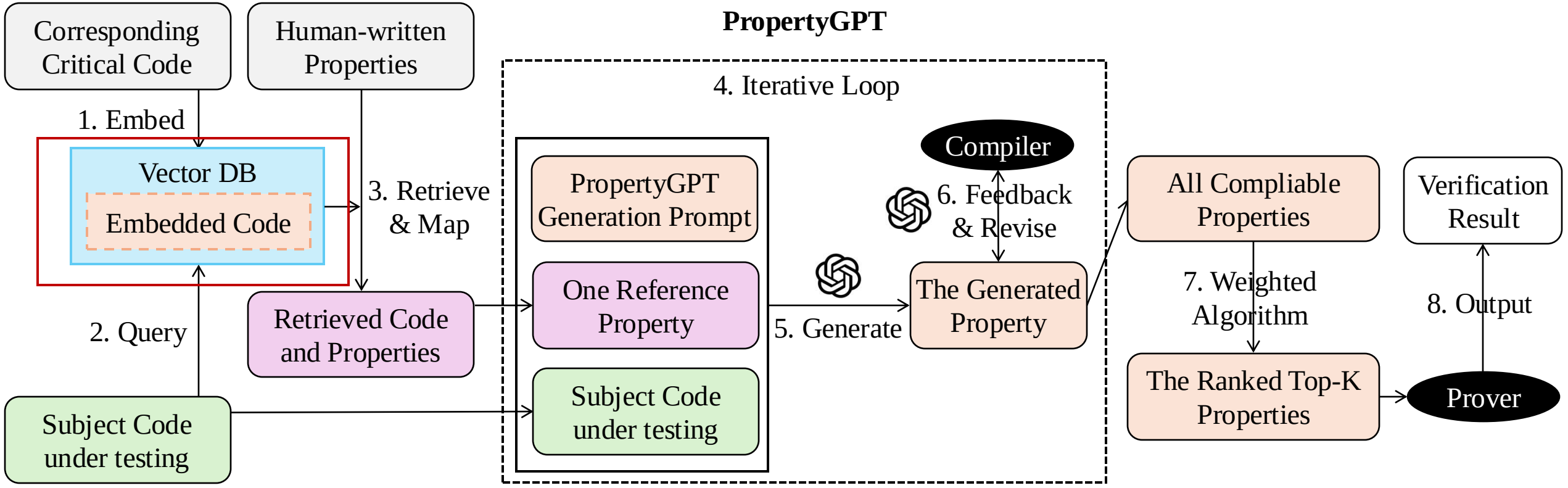
LLM-driven Property Generation

An ERC20Vote property about "mapping(address=> Checkpoint[]) private _checkpoints"



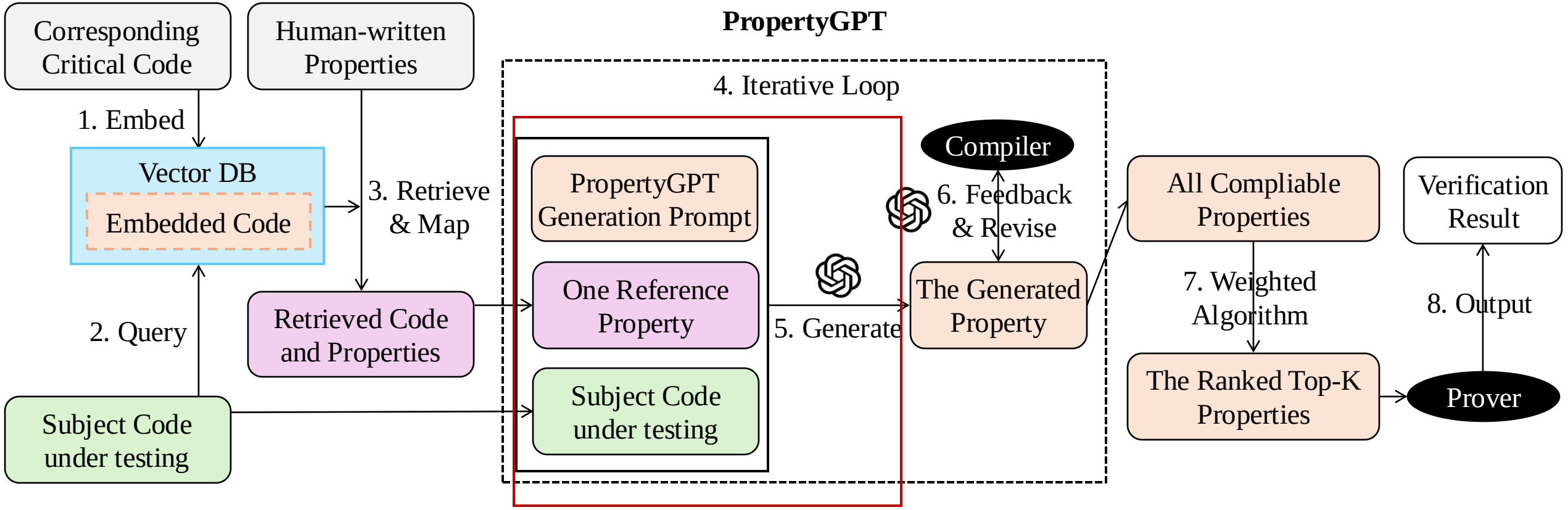

```
1  hook Sstore _checkpoints[KEY address account][INDEX uint32 index].votes
     uint224 newVotes (uint224 oldVotes) STORAGE {
2      havoc userVotes assuming
3        userVotes@new(account) == newVotes;
4
5      havoc totalVotes assuming
6        totalVotes@new() == totalVotes@old() +newVotes - userVotes(account);
7
8      havoc lastIndex assuming
9        lastIndex@new(account) == index;
10 }
```

LLM-driven Property Generation



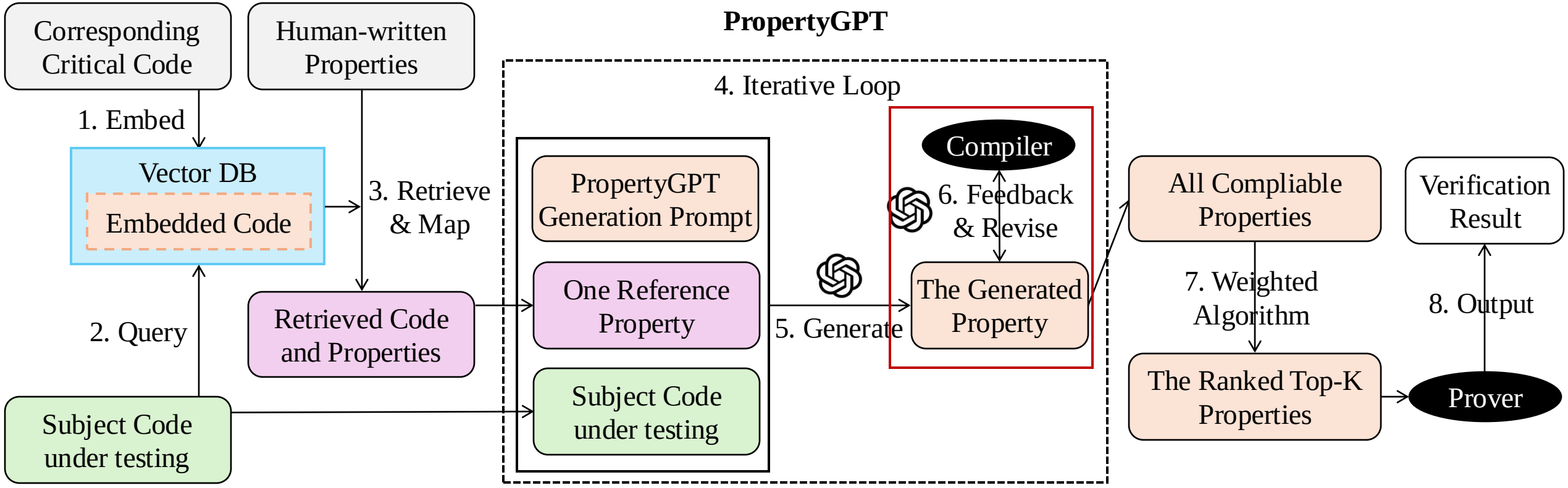
Code Embedding -> Query -> Property Retrieval

LLM-driven Property Generation



RAG: Generate with reference property

LLM-driven Property Generation



Revise: Fix syntax error of property generated

LLM-driven Property Generation

$$v \in StateVar \quad tmp \in TemporalVar \quad \mathcal{C} \in Constant$$

$$SC = v*; func*$$

$$func \in Function = param*; stmt*$$

$$expr \in Expression = tmp \mid v \mid \mathbf{old}(v) \mid param \mid \mathcal{C} \mid expr \bowtie expr$$

$$\mathbf{Spec}(SC) = inv*; \{p*\} func \{q*\}; rule*$$

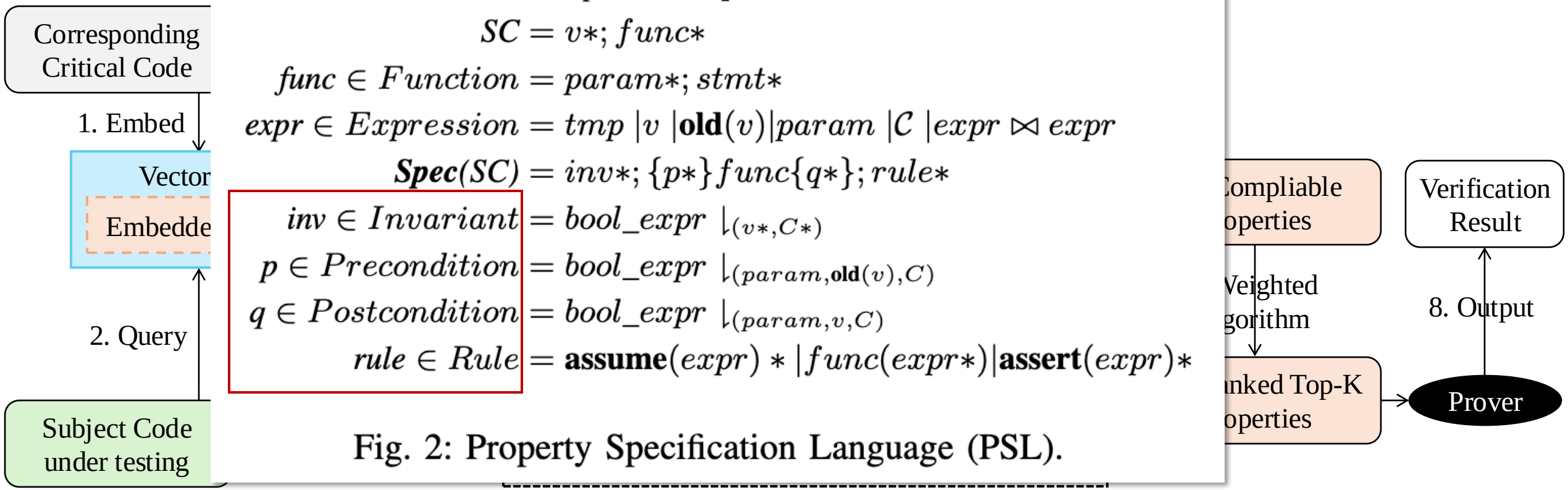
$$inv \in Invariant = bool\_expr \downharpoonright_{(v*, C*)}$$

$$p \in Precondition = bool\_expr \downharpoonright_{(param, \mathbf{old}(v), C)}$$

$$q \in Postcondition = bool\_expr \downharpoonright_{(param, v, C)}$$

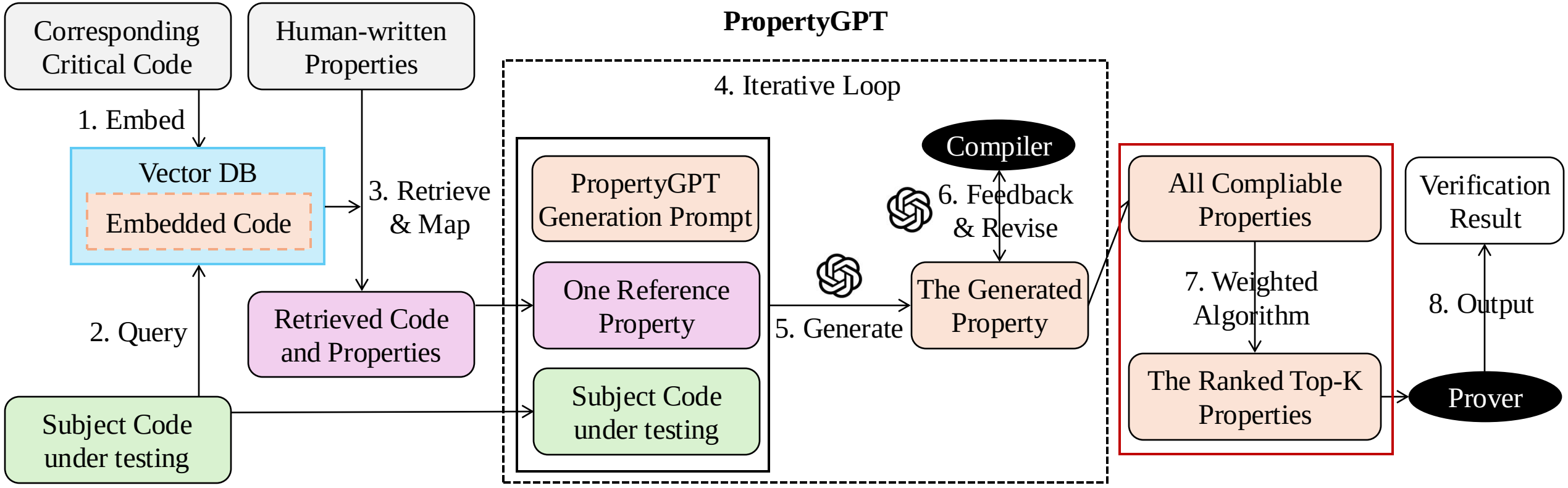
$$rule \in Rule = \mathbf{assume}(expr) * \mid func(expr*) \mid \mathbf{assert}(expr)*$$

Fig. 2: Property Specification Language (PSL).



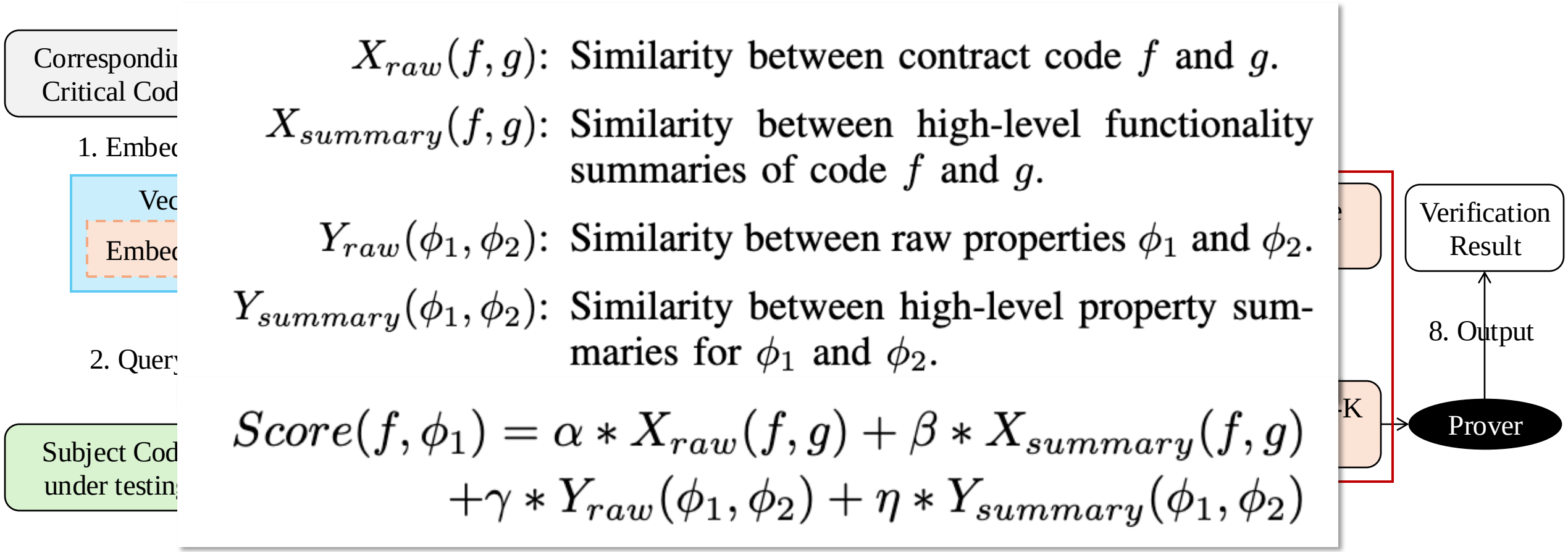
Revise: Fix syntax error of property generated

LLM-driven Property Generation



Ranking: Similarity-based scoring of property generated

LLM-driven Property Generation



$X_{raw}(f, g)$: Similarity between contract code f and g.

$X_{summary}(f, g)$: Similarity between high-level functionality summaries of code f and g.

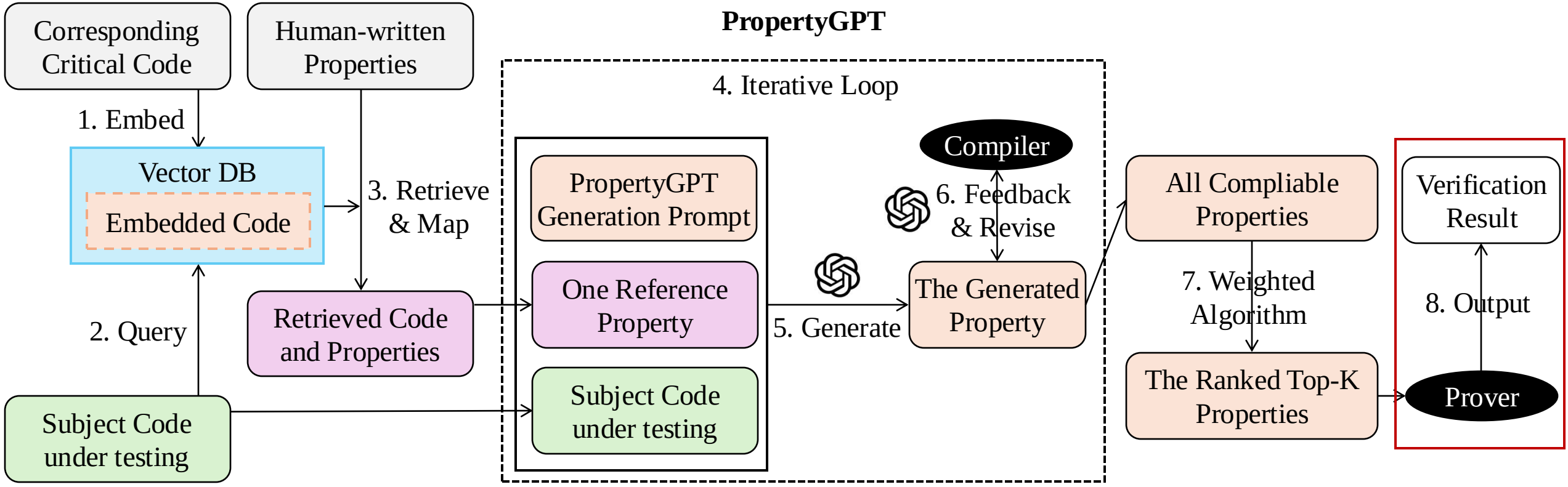
$Y_{raw}(\phi_1, \phi_2)$: Similarity between raw properties ϕ_1 and ϕ_2.

$Y_{summary}(\phi_1, \phi_2)$: Similarity between high-level property summaries for ϕ_1 and ϕ_2.

$$Score(f, \phi_1) = \alpha * X_{raw}(f, g) + \beta * X_{summary}(f, g) + \gamma * Y_{raw}(\phi_1, \phi_2) + \eta * Y_{summary}(\phi_1, \phi_2)$$

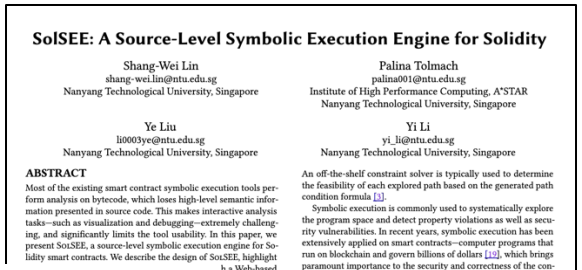
Ranking: Similarity-based scoring of property generated

LLM-driven Property Generation



Symbolic execution-based property checking

LLM-driven Property Generation


SolSEE: A Source-Level Symbolic Execution Engine for Solidity

Shang-Wei Lin
shang-wei.lin@ntu.edu.sg
Nanyang Technological University, Singapore

Palina Tolmach
palina001@ntu.edu.sg
Institute of High Performance Computing, A*STAR
Nanyang Technological University, Singapore

Ye Liu
li0003ye@ntu.edu.sg
Nanyang Technological University, Singapore

Yi Li
yi_li@ntu.edu.sg
Nanyang Technological University, Singapore

ABSTRACT

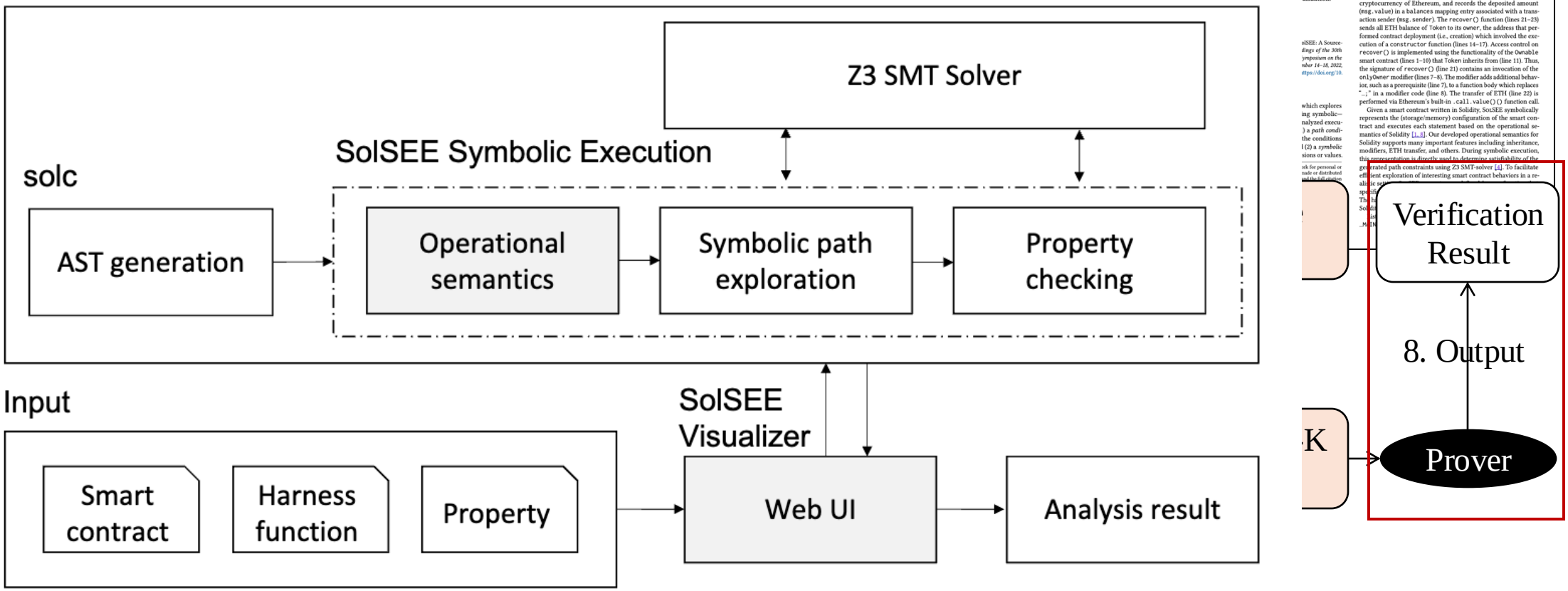
Most of the existing smart contract symbolic execution tools per-
form analysis on bytecode, which loses high-level semantic infor-
mation presented in source code. This makes interactive analysis
tasks—such as visualization and debugging—extremely challeng-
ing, and significantly limits the tool usability. In this paper, we
present SolSEE, a source-level symbolic execution engine for So-
lidity smart contracts. We describe the design of SolSEE, highlight
h a Web-based
er other exist-

An off-the-shelf constraint solver is typically used to determine
the feasibility of each explored path based on the generated path
condition formula [5].

Symbolic execution is commonly used to systematically explore
the program space and detect property violations as well as secu-
rity vulnerabilities. In recent years, symbolic execution has been
extensively applied on smart contracts—computer programs that
run on blockchain and govern billions of dollars [19], which brings
paramount importance to the security and correctness of the con-
tract code [22]. Most smart contracts are written in Solidity [1]—
g language of the Ethereum blockchain
into EVM bytecode for deployment and
ain. Listing 1 shows a sample Solidity
smart contract. In Ethereum, users inter-
by initiating transactions that invoke its
external visibility. Token's deposit()
ows the contract to accept ETH, a native
cryptocurrency of Ethereum, and records the deposited amount
(msg.value) in a balances mapping entry associated with a trans-
action sender (msg.sender). The recover() function (lines 21–23)
sends all ETH balance of Token to its owner, the address that per-
formed contract deployment (i.e., creation) which involved the exe-
cution of a constructor function (lines 14–17). Access control on
recover() is implemented using the functionality of the Ownable
smart contract (lines 1–10) that Token inherits from (line 11). Thus,
the signature of recover() (line 21) contains an invocation of the
onlyOwner modifier (lines 7–8). The modifier adds additional behav-
ior, such as a prerequisite (line 7), to a function body which replaces
"_;" in a modifier code (line 8). The transfer of ETH (line 22) is
performed via Ethereum's built-in .call.value()() function call.

Given a smart contract written in Solidity, SolSEE symbolically
represents the (storage/memory) configuration of the smart con-
tract and executes each statement based on the operational se-
mantics of Solidity [1, 8]. Our developed operational semantics for
Solidity supports many important features including inheritance,
modifiers, ETH transfer, and others. During symbolic execution,
this representation is directly used to determine satisfiability of the
generated path constraints using Z3 SMT-solver [4]. To facilitate
efficient exploration of interesting smart contract behaviors in a re-

FSE'22. *SolSEE*: A Source-Level Symbolic Execution Engine for Solidity



Symbolic execution-based property checking

Collection of security properties

We used **diverse** human-written 623 security properties from 23 audited projects by Certora

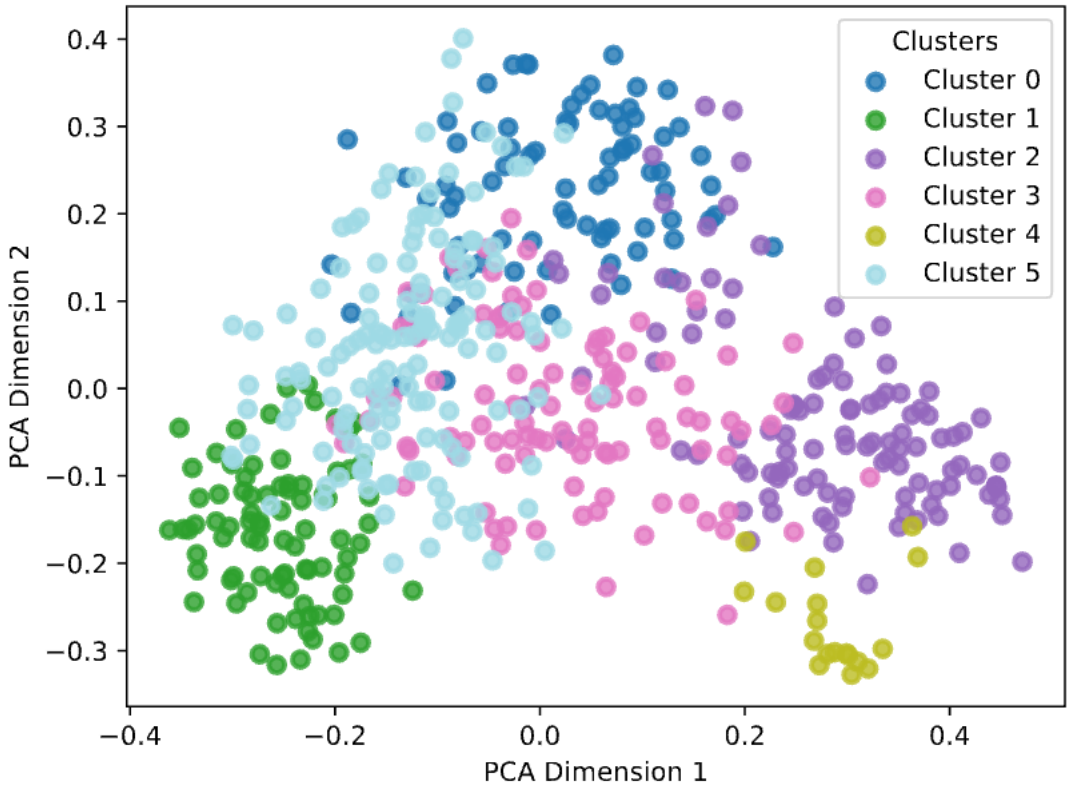


TABLE I: Characteristics of the collected Certora properties.

Category	Classification	Property Examples
DeFi	Reserve Collateral Liquidity	setReserveFactorIntegrity integrityOfisUsingAsCollateralAny checkBurnAllLiquidity
Token	Balance Transfer/TransferFrom Mint/Burn Approve	total_supply_is_sum_of_balances transferBalanceIncreaseEffect integrityMint, additiveBurn approvedTokensAreTemporary
Arithmetic	Numerical Conversion Asset Splitting	toElasticAndToBaseAreInverse1down moneyNotLostOrCreatedDuringSplit
Usability	Temporal Use State-dependent Use	timestamp_constrains_fromBlock unsetPendingTransitionMethods
Governance	Delegation Voting	integrityDelegationWithSig totalNonVotingGEAccountNonVoting
Security	Front run Overflow	cannotFrontRunSplitTwoSameUsers integrityOfMulDivNoOverflow

Evaluation

- RQ1: **Property Generation**
- RQ2: **Vulnerability Detection**
- RQ3: **What factors influence the performance of PropertyGPT**

RQ1. Property Generation

TABLE III: The property generation results for 90 ground-truth properties from nine Certora projects.

Project	#Property (Certora)	#Property (ours)	TP	#Hit	FN	FP	Recall	Precision	F1-score
aave_proof_of_reserve	3*	38	25	3	0	13	**1.00**	0.66	0.79
aave_v3	17	61	32	15	2	29	0.88	0.52	0.66
celo_governance	10	39	29	10	0	10	**1.00**	0.74	**0.85**
furucombo	10	23	11	7	3	12	0.70	0.48	0.57
openzepplin	10	2	2	1	9	0	**0.10**	**1.00**	**0.18**
opyn_gamma_protocol	10	30	14	8	2	16	0.80	**0.47**	0.59
ousd	10	100	67	10	0	33	**1.00**	0.67	0.80
radicle_drips	10	17	9	7	3	8	0.70	0.53	0.60
sushi_benttobox	10	70	49	10	0	21	**1.00**	0.70	0.82
Average	10	42	26	8	2	16	0.80	0.64	0.71

* This project contains only three human-written properties, so we picked seven more from `aave_v3`. Both are from the same institution.

Can generate comprehensive and high-quality properties, covering 80% equivalent properties in the ground truth as judged by human experts.

RQ2. Vulnerability Detection

TABLE IV: Vulnerability detection results for 13 CVEs.

CVE ID	*Description*	*Avg. Code Sim.*	*PropertyGPT*	*GPTScan+*	*Slither*	*Manticore*	*Mythril*
2021-34273	access control	0.693	✓	✓	×	×	×
2021-33403	overflow	0.666	✓	×	×	×	✓
2018-18425	logic error	0.704	✓	×	×	×	×
2021-3004	logic error	0.691	×	×	×	×	×
2018-14085	delegatecall	0.662	×	×	✓	×	×
2018-14089	logic error	0.661	✓	✓	×	×	×
2018-17111	access control	0.636	×	×	×	×	×
2018-17987	bad randomness	0.660	×	✓	×	×	×
2019-15079	access control	0.701	✓	×	×	×	×
2023-26488	logic error	0.682	✓	×	×	×	×
2021-34272	access control	0.693	✓	✓	×	×	×
2021-34270	overflow	0.671	✓	✓	×	×	✓
2018-14087	overflow	0.661	✓	×	×	×	✓

Detecting 9 out of 13 CVEs.

RQ2. Vulnerability Detection

TABLE V: Evaluation results for 24 attack incident projects from the curated SmartInv benchmark.

Contracts	Detection	#Property	Avg. Code Similarity	Generation (seconds)	Verification (seconds)
dfxFinance	✓	8	0.675	235	7
AnySwap	×	11	0.692	518	7
Dodo	✓	17	0.703	1,182	19
Bancor	✓	19	0.699	1,948	9
BeautyChain	✓	5	0.676	104	9
Melo	✓	9	0.732	252	8
BGLD	×	9	0.654	229	39
GYMNetwork	✓	21	0.681	274	71
elasticSwap	✓	37	0.681	1,136	120
EulerFinance	×	23	0.669	376	43
monoSwap	✓	5	0.722	69	12
nimBus	✓	32	0.678	4,288	30
VTF	×	8	0.672	358	21
Nomad	✓	14	0.673	590	70
Umbrella	✓	14	0.688	404	25
Fortress Loan	✓	2	0.668	71	5
ShadowFinance	✓	25	0.683	551	80
Revest	✓	4	0.646	75	10
Cartel	✓	11	0.683	401	20
sushiSwap	×	10	0.686	419	20
ChainSwap	×	9	0.690	307	25
Ragnarok	✓	42	0.684	1,890	88
templeDao	✓	13	0.677	302	30
BabySwap	×	33	0.679	1,842	50
Overall	17	16	0.683	743	34

Identifying logic bugs in 17 out of 24 attack incidents.

RQ3. What factors influence the performance of PropertyGPT?

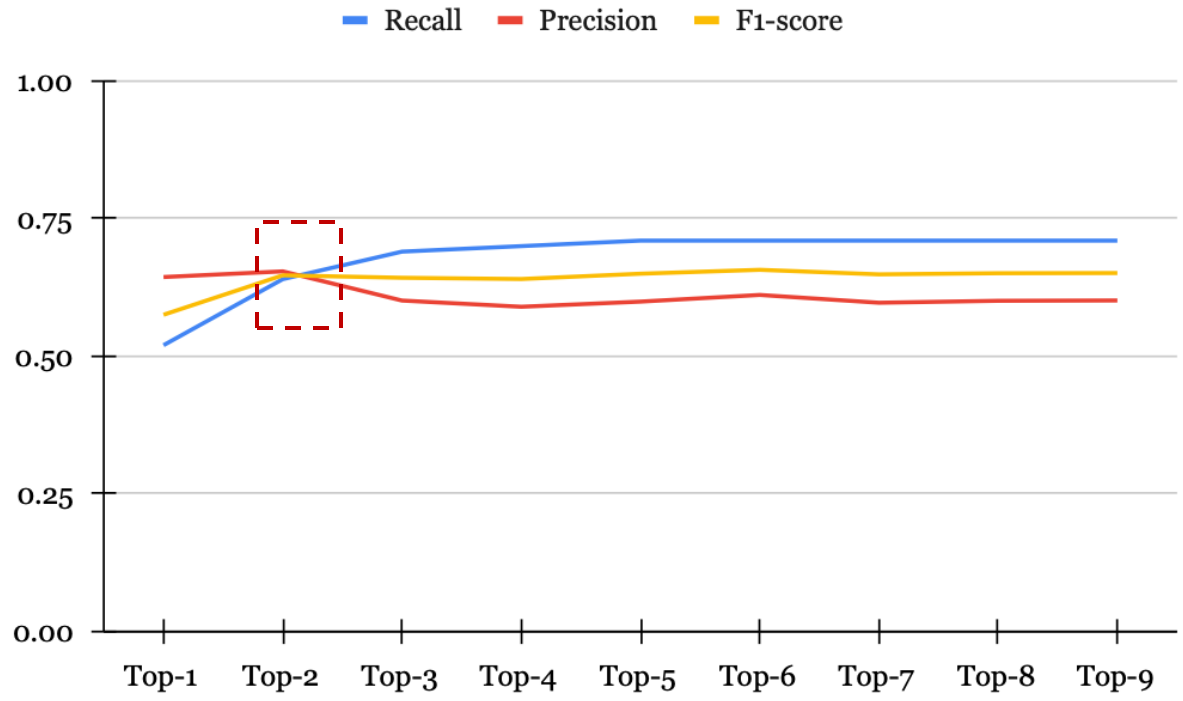


Fig. 9: The impact of Top-K settings on property accuracy.

Top-2 is a good setting to balance recall and precision

RQ3. What factors influence the performance of PropertyGPT?

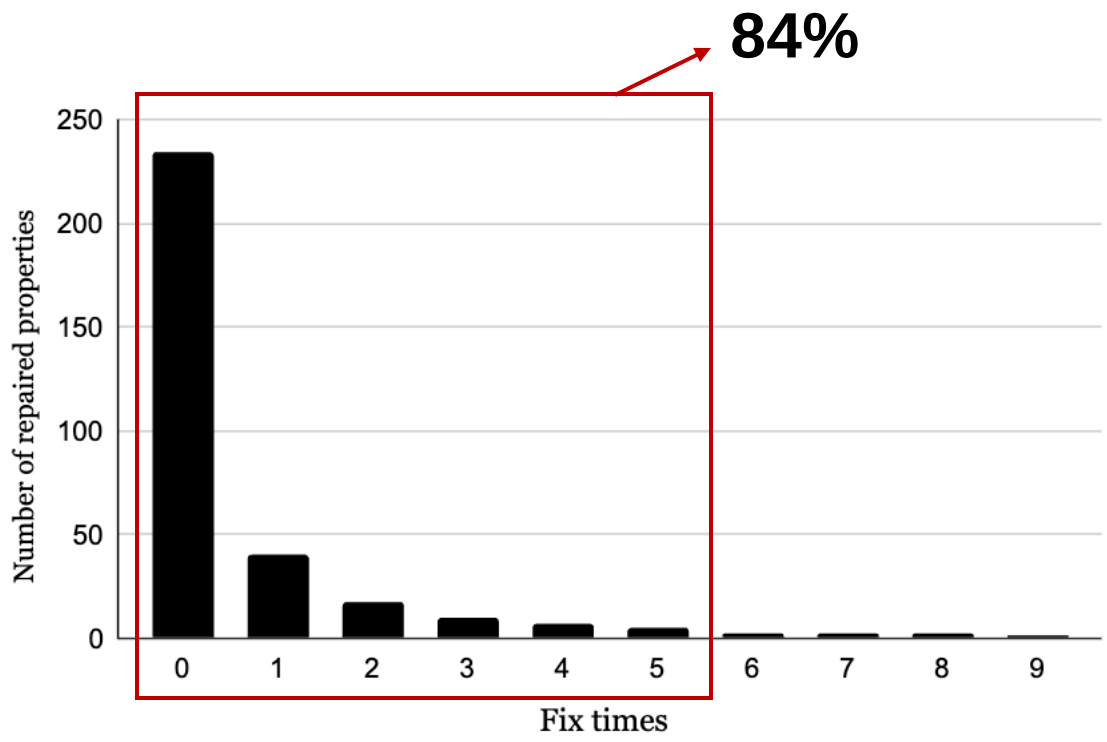


Fig. 10: The distribution of property fix times.

TABLE VI: The success rate of property generation.

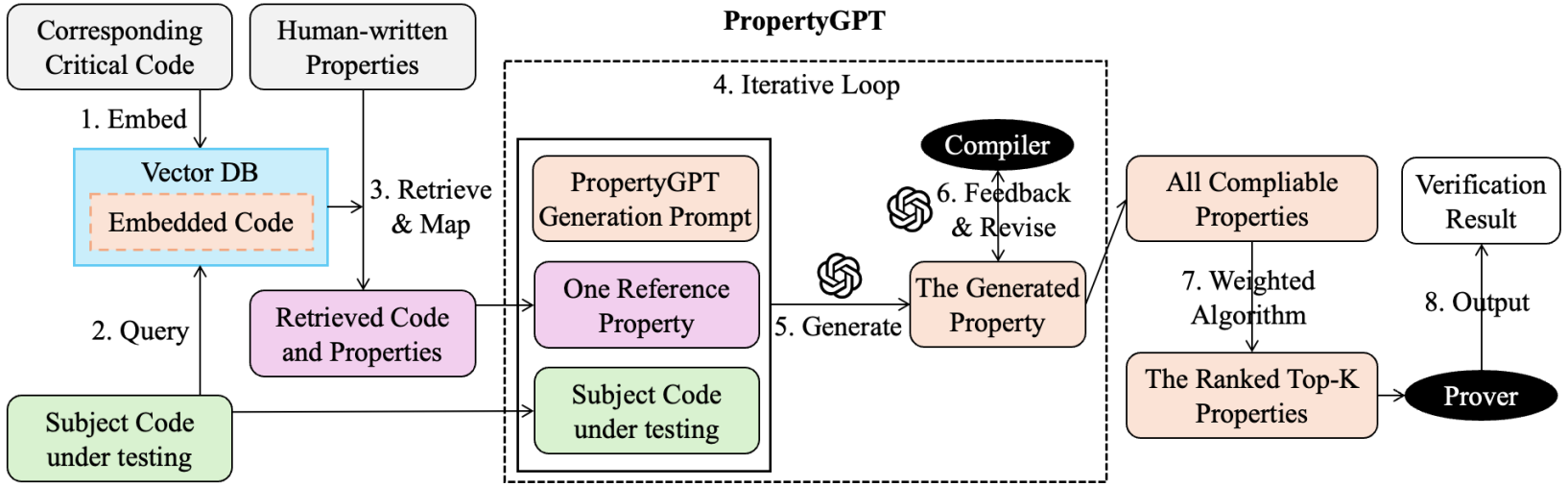
Method	#Compilable	#Failed	Success Rate
GPT-4.0-turbo w/o fix	234	136	0.63
PropertyGPT w/ §V-B	321	49	0.87

84% properties successfully revised within 5 attempts, and the overall success rate is 87%.

Takeaway

- LLM for property generation
  - **High-quality** security domain knowledge
  - **Well-designed** RAG technique
  - **Suitable** property representation
- Other directions
  - LLM for **optimizing formal verification** techniques
  - **Soundness/Completeness** of LLM reasoning in formal tasks

Summary.



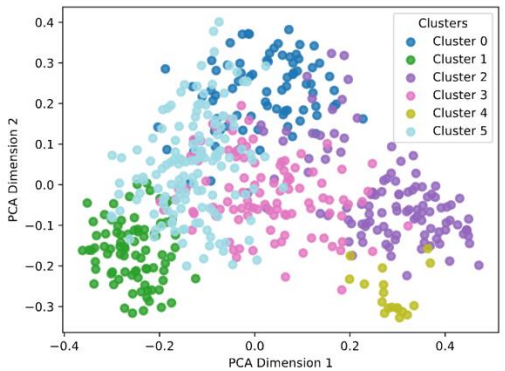


TABLE III: The property generation results for 90 ground-truth properties from nine Certora projects.

Project	#Property (Certora)	#Property (ours)	TP	#Hit	FN	FP	Recall	Precision	F1-score
aave_proof_of_reserve	3*	38	25	3	0	13	**1.00**	0.66	0.79
aave_v3	17	61	32	15	2	29	0.88	0.52	0.66
celo_governance	10	39	29	10	0	10	**1.00**	0.74	**0.85**
furucombo	10	23	11	7	3	12	0.70	0.48	0.57
openzepplin	10	2	2	1	9	0	**0.10**	**1.00**	**0.18**
opyn_gamma_protocol	10	30	14	8	2	16	0.80	**0.47**	0.59
ousd	10	100	67	10	0	33	**1.00**	0.67	0.80
radicle_drips	10	17	9	7	3	8	0.70	0.53	0.60
sushi_benttobox	10	70	49	10	0	21	**1.00**	0.70	0.82
Average	10	42	26	8	2	16	0.80	0.64	0.71

* This project contains only three human-written properties, so we picked seven more from `aave_v3`. Both are from the same institution.

TABLE IV: Vulnerability detection results for 13 CVEs.

CVE ID	Description	Avg. Code Sim.	PropertyGPT	GPTScan+	Slither	Manticore	Mythril
2021-34273	access control	0.693	✓	✓	×	×	×
2021-33403	overflow	0.666	✓	×	×	×	✓
2018-18425	logic error	0.704	✓	×	×	×	×
2021-3004	logic error	0.691	×	×	×	×	×
2018-14085	delegatecall	0.662	×	×	✓	×	×
2018-14089	logic error	0.661	✓	✓	×	×	×
2018-17111	access control	0.636	×	×	×	×	×
2018-17987	bad randomness	0.660	×	✓	×	×	×
2019-15079	access control	0.701	✓	×	×	×	×
2023-26488	logic error	0.682	✓	×	×	×	×
2021-34272	access control	0.693	✓	✓	×	×	×
2021-34270	overflow	0.671	✓	✓	×	×	✓
2018-14087	overflow	0.661	✓	×	×	×	✓

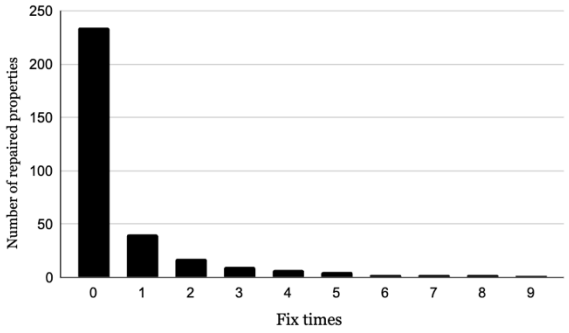


Fig. 10: The distribution of property fix times.