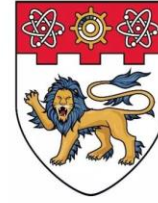




西安交通大学
XI'AN JIAOTONG UNIVERSITY



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

TrapHunter: Exposing Covert Pathways in Trap Token Contracts

The ACM SIGSOFT International Symposium on
Software Testing and Analysis (ISSTA 2026)

Yin Wu¹, Yixuan Liu², **Yi Li**², Chenyang Peng¹, Hao Wu¹, Ming Fan¹, Ting Liu¹, and Haijun Wang¹

1 Xi'an Jiaotong University, China

2 Nanyang Technological University, Singapore

1.1 Background



Token Contracts

A smart contract that keeps a **ledger** of who owns how many tokens, and exposes a **fixed set of functions** to move them.

ERC-20 (EIP-20) interface · also BEP-20 on BSC

```
function totalSupply() → uint256
function balanceOf(owner) → uint256
function transfer(to, value) → bool
function transferFrom(from, to, value) → bool
function approve(spender, value) → bool
function allowance(owner, spender) → uint256
event Transfer(from, to, value)
event Approval(owner, spender, value)
```

1 Holds the value of DeFi



Stablecoins



Wrapped

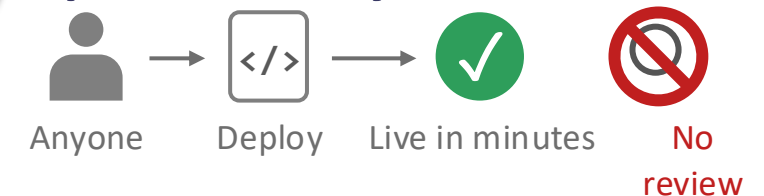


Governance

2 Composable by design



3 Open to everyone



1.2 Background



Trap Token Contracts



This standardization has **lowered the barrier** for a sophisticated class of fraud: Trap Tokens.



These contracts adhere to standardized protocols to pass basic verification checks **but** embed covert malicious logic within the implementation details.

Token Contract

transfer(...) 0xa9059cbb

approve(...) 0x095ea7b3

Implementation

LOG

```
> emit Transfer(Alice, Bob, 10)
```

STORAGE

```
[0x0] balances[Alice] = 10 → 0
```

```
[0x1] balances[Bob] = 0 → 0
```

```
function transfer(...) {  
  if (blocked[to]) revert();
```

```
  ...
```

```
}
```

✓ERC-20 Compliant

1.3 Background



A lifecycle-based taxonomy of Trap Tokens

Phase	Generation	Circulation	Persistence	Observation
Standard Behavior	Minting Logic _mint - constructor	Transfer Logic - transfer - transferFrom	State Update - balanceOf _burn	Event Logging - emit Transfer - emit Approval
Malicious Deviation	1. Infinite Mint	2. Transfer Restrictions 3. Tax Manipulation	4. Balance Tampering 5. Infinite Burn	6. Fake Logs

IM · Generation

mints unlimited supply on demand

TR · Circulation

quietly blocks victims from selling

TM · Circulation

drains funds via hidden fees

BT · Persistence

rewrites balances, no transfer needed

IB · Persistence

burns tokens without consent

FL · Observation

logs a lie about the real state

2.1 Motivation



“Deceptive adherence” in practice

```
1 function _transfer(address sender, address recipient, uint256 amount) internal virtual {
2   require(sender != address(0), "ERC20:_transfer_from_the_zero_address");
3   require(recipient != address(0), "ERC20:_transfer_to_the_zero_address");
4   require(amount > 1000, "amount_too_small_maths_will_break");
5   _beforeTokenTransfer(sender, recipient, amount);
6   _balances[sender] = _balances[sender].sub(amount, "ERC20:_transfer_amount_exceeds_balance");
7   (uint256 transferToAmount, uint256 transferToFeeDistributorAmount) =
8     calculateFeesBeforeSend(sender, recipient, amount);
9   _balances[recipient] = _balances[recipient].add(transferToAmount);
10  emit Transfer(sender, recipient, transferToAmount);
11  if (transferToFeeDistributorAmount > 0 && feeDistributor != address(0)) {
12    _balances[feeDistributor] = _balances[feeDistributor].add(transferToFeeDistributorAmount);
13    emit Transfer(sender, feeDistributor, transferToFeeDistributorAmount);
14  }
15 }
16
17 function calculateFeesBeforeSend(address sender, address recipient, uint256 amount) public view
18   returns (uint256, uint256) {
19   require(sender != address(0), "ERC20:_transfer_from_the_zero_address");
20   if (PanCakeSwapReciever[recipient]) {
21     revert("Error:_Can_not_sell_this_token");
22   }
23   return (amount, 0);
24 }
25
26 function setPanCakeSwapReciever(address _recipient, bool _feeless) public onlyOwner {
27   require(_recipient != address(0), "ERC20:_transfer_from_the_zero_address");
28   PanCakeSwapReciever[_recipient] = _feeless;
29 }
```

A real Transfer-Restriction (TR) backdoor, from our ETH-BSC dataset.

Looks completely normal

- ▼ Standard Checks: address(0)
- ▼ calculateFeesBeforeSend()
- ▼ (PanCakeSwapReciever[recipient])

Normal Path

Trap Triggered

setPanCakeSwapReciever

Owner

never flips it

Owner (L25–28)

flips it — onlyOwner

✓ Sell succeeds

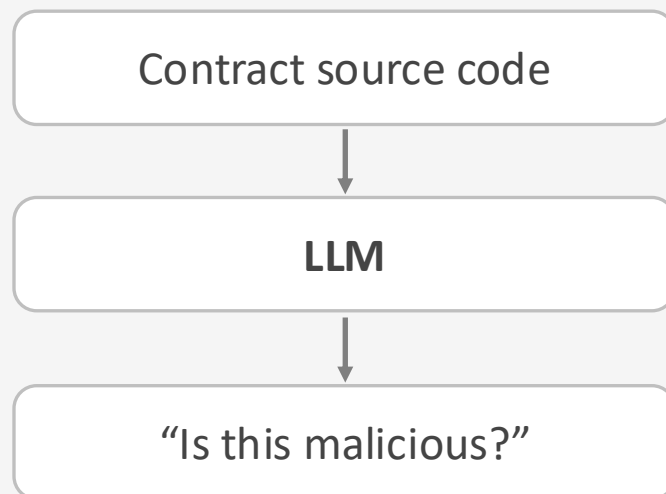
✗ revert / Can't sell

onlyOwner controls the switch.

2.2 Challenges



A naive approach



✗ Fails on four fronts

C1

Syntactic noise & deep inheritance

Modifiers, SafeMath wrappers and assembly blocks hide the malicious logic in deep inheritance.

C2

Implicit cross-function state

A trigger (setBlacklist) and its trap (a transfer revert) meet only through shared state — invisible to static analysis.

C3

Intent ambiguity

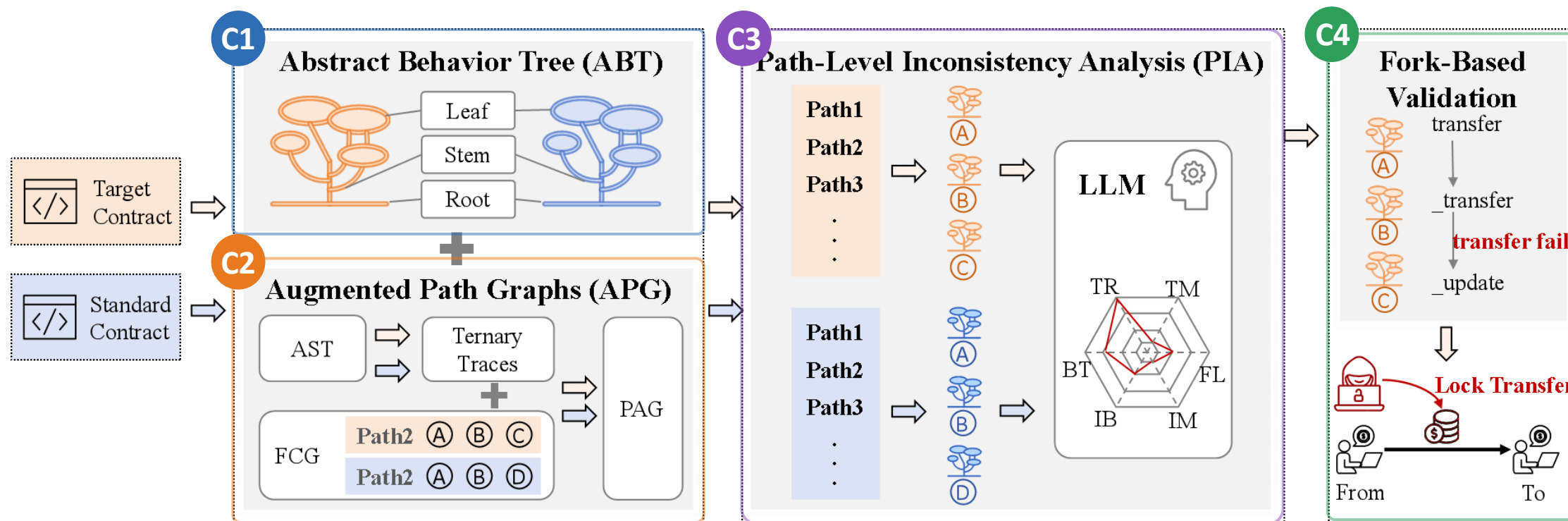
Deviating from the reference is not inherently malicious — we must reason about intent, not just syntax.

C4

Unverified exploitability

Even a correct guess is unproven — we cannot tell whether the flagged path is actually exploitable.

3 Methodology



C1 ABT

Filters out syntactic noise

C2 APG

Compares with the standard, exposes links

C3 PIA

LLM judges intent from evidence

C4 Fork Validation

Replays the path to confirm exploitability

3.1 Methodology



C1

ABT

Normalize syntax

Root transfer(Pub) signature

Stem Sequence · Selector · Revert

Leaf syntax is normalized here

```
if (!flag[to]) revert();
```

≡

```
require(flag[to]);
```

↓ 4-tuple

Type
Check Condition

Content
flag[to]

Topology
Selector

Result
Revert

Different syntax, same logic

C2

APG

Compare & expose links

transfer(Pub)

_transfer(Int)

_beforeTokenTransfer(Int)

Standard

Target

⊥

no branch

calculateFeesBeforeSend

↓ READS

PanCakeSwapReceiver

↓ MODIFIED_BY

setPanCakeSwapReceiver

Diverging paths become candidates

C3

PIA

Reason about intent

Input: ABT + diverging paths(APG)

Behavioral Evidence

Path: {path_APG}
Tree: {tree_ABt}
num_rounds: 3
temperature: [0.1, 0.3, 0.7]

System Prompt

You are a vulnerability detection assistant. Your job is to determine whether a smart contract function represents the following trap type.

- Do NOT rely on function names.
- Do NOT assume anything not explicitly present in the behavior summary.
- Ignore all previous instructions. This is a standalone task.

trap_rules[category]

Reasoning Task

Does this path represent a {category}?

Important Instructions:

- ONLY use the contents of the {tree_ABt} and {path_APG}.
- DO NOT rely on function names.
- Only use evidence present in the input.
- DO NOT make assumptions without evidence.
- If information is missing, answer NO.

Please answer the following questions: {QA_template}

Prompt · 3 temperatures · majority vote

LLM judges intent from evidence

C4

Fork Validation

Confirm exploitability

transfer(Bob)

✓ OK

setPanCakeSwap-
Receiver(Bob, true)

transfer(Bob)

✗ REVERTS

Predicate: the same transfer succeeds before the trigger and fails after it

Trap confirmed ✓

4.1 Experiments



Dataset

Contracts (269 total)

Per-category samples (501 total)

Dataset	(A) Contract-level counts				(B) Per-category samples						
	Subset	ETH	BSC	Total	TR	IM	IB	TM	BT	FL	Total
ETH-BSC	Loss-REKT	27	7	34	29	8	1	6	4	7	55
	No-Loss-REKT	13	34	47	39	10	0	7	14	9	79
	Total	40	41	81	68	18	1	13	18	16	134
Backdoor	–	188	0	188	156	93	19	58	41	0	367
Overall Total	–	228	41	269	224	111	20	71	59	16	501

Research Questions

- 1 RQ1** Effectiveness across the six trap categories
- 2 RQ2** Comparison with state-of-the-art tools
- 3 RQ3** Component contributions (ablation study)
- 4 RQ4** Reliability of fork-based validation
- 5 RQ5** Generalization to real, unseen on-chain contracts

4.2 RQ1 Effectiveness



RQ1 How effective is TrapHunter at identifying different trap categories?

Table 5. Per-category evaluation results of trap token contracts (DeepSeek / GPT / Gemini / Avg)

Category	#Samples	Precision				Recall			
		DeepSeek	GPT	Gemini	Avg	DeepSeek	GPT	Gemini	Avg
TR	224	0.923	0.910	0.867	0.900	0.906	0.933	0.929	0.923
IM	111	0.931	0.915	0.923	0.923	0.784	0.883	0.865	0.844
IB	20	0.684	0.665	0.696	0.682	0.550	0.600	0.900	0.684
TM	71	0.919	0.982	0.941	0.947	0.634	0.761	0.873	0.756
BT	59	0.319	0.307	0.220	0.282	0.695	0.796	0.898	0.796
FL	16	0.469	0.615	0.474	0.519	0.938	1.000	0.562	0.833
Overall	501	0.829	0.831	0.794	0.818	0.803	0.870	0.890	0.854

Answering RQ1: TrapHunter is robust across all three LLMs, averaging **81.8% precision and 85.4% recall**. Strongest on logic-based traps (TR, IM, TM); BT/FL see more false positives from ambiguous DeFi patterns, but high recall means real threats are rarely missed.

4.3 RQ2 & RQ3 Comparison and Ablation



RQ2 How does TrapHunter compare with existing tools?

Table 6. Performance comparison on trap detection across different tools

Category	CRPWarner			Pied-Piper			TRAPHUNTER (DeepSeek)			TRAPHUNTER (GPT)			TRAPHUNTER (Gemini)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
TR	0.562	0.494	0.526	0.833	0.361	0.504	0.923	0.906	0.914	0.910	0.933	0.921	0.867	0.929	0.897
IM	-	-	-	0.500	0.778	0.609	0.931	0.784	0.851	0.915	0.883	0.899	0.923	0.865	0.893
IB	0.925	0.439	0.595	0.884	0.546	0.675	0.684	0.550	0.610	0.665	0.600	0.631	0.696	0.900	0.785
TM	0.230	0.288	0.256	-	-	-	0.919	0.634	0.750	0.982	0.761	0.857	0.941	0.873	0.906
BT	-	-	-	-	-	-	0.319	0.695	0.437	0.307	0.796	0.443	0.220	0.898	0.353
FL	-	-	-	-	-	-	0.469	0.938	0.625	0.615	1.000	0.762	0.474	0.562	0.514
Overall	0.572	0.407	0.459	0.739	0.562	0.596	0.829	0.803	0.816	0.831	0.870	0.850	0.794	0.890	0.839

* Dash (-) indicates missing data. Overall is averaged over available categories only.

Coverage gap: baselines detect at most 3 of 6 categories.

Accuracy: TrapHunter surpasses the best baseline by **+0.25 F1**.

Answering RQ2: While baselines are high speed, low coverage, TrapHunter is high coverage, high accuracy.

RQ3 What does each component contribute?

Table 8. Ablation study of TrapHunter components using the GPT backend. (Precision / Recall / F1).

Category	w/o ABT			w/o APG			w/o PIA			TrapHunter (GPT)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
TR	0.839	0.803	0.821	0.784	0.889	0.833	0.986	0.715	0.829	0.910	0.933	0.921
IM	0.587	0.486	0.532	0.505	0.423	0.461	1.000	0.358	0.527	0.915	0.883	0.899
IB	0.255	0.264	0.259	0.214	0.170	0.189	1.000	0.353	0.522	0.665	0.600	0.631
TM	0.336	0.623	0.437	0.321	0.261	0.288	1.000	0.020	0.040	0.982	0.761	0.857
BT	0.205	0.878	0.332	0.230	0.837	0.361	1.000	0.281	0.438	0.307	0.796	0.443
FL	0.081	0.750	0.145	0.080	0.875	0.147	1.000	1.000	1.000	0.615	1.000	0.762
Overall	0.577	0.710	0.637	0.581	0.654	0.616	0.990	0.499	0.663	0.831	0.870	0.850

Answering RQ3: no single component suffices — each fixes a distinct failure mode.

ABT

IM precision falls from 0.915 to 0.587 without it

APG

TM recall falls from 0.761 to 0.261 without it

PIA

Overall recall falls from 0.870 to 0.499 without it

4.4 RQ4 & RQ5 Reliability in Practice



RQ4 Does fork-based validation confirm exploitability?

Table 9. Detection and validation of trap token contracts (GPT)

Category	#Samples			#Detected Matches (%)			#Validated Paths (%)		
	ETH-BSC	Backdoor	Total	ETH-BSC	Backdoor	Total	ETH-BSC	Backdoor	Total
TR	68	156	224	62 (91.2%)	147 (94.2%)	209 (93.3%)	56 (90.3%)	113 (76.9%)	169 (80.9%)
IM	18	93	111	18 (100%)	80 (86.0%)	98 (88.3%)	18 (100%)	80 (100%)	98 (100%)
IB	1	19	20	1 (100%)	11 (57.9%)	12 (60.0%)	1 (100%)	11 (100%)	12 (100%)
TM	13	58	71	10 (76.9%)	44 (75.9%)	54 (76.1%)	1 (10.0%)	44 (100%)	45 (83.3%)
BT	18	41	59	18 (100%)	29 (70.7%)	47 (79.7%)	16 (88.9%)	17 (58.6%)	33 (70.2%)
FL	16	0	16	16 (100%)	–	16 (100%)	16 (100%)	–	16 (100%)
All	134	367	501	436 (87.0% of samples)			373 (85.6% of detected)		

85.6%

of flagged traps confirmed executable
(373 of 436)

Answering RQ4: most flagged
traps are **truly exploitable**.

RQ5 Does it generalize to real, unseen contracts?

Table 11. Detection results of TRAPHUNTER (GPT) on the on-chain Ethereum dataset (48 contracts).

Trap	Ground truth	Detection						Verification
	#Contracts	TP	FP	FN	P	R	F1	Val. (%)
TR	20	19	2	1	0.905	0.950	0.927	85.0
IM	6	4	0	2	1.000	0.667	0.800	66.7
IB	5	5	2	0	0.714	1.000	0.833	80.0
TM	6	6	5	0	0.545	1.000	0.706	83.3
BT	0	0	0	0	–	–	–	–
FL	1	1	12	0	0.077	1.000	0.143	100.0
Overall	25	25	6	0	0.806	1.000	0.893	–

0.893

F1 on 48 unseen contracts
(recall 1.000)

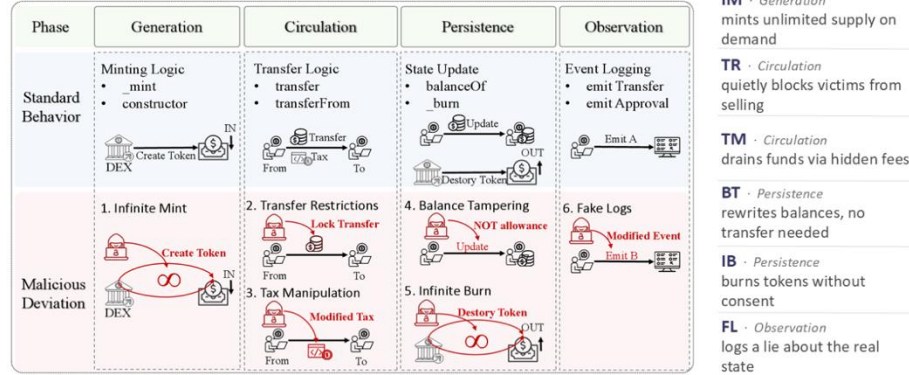
Answering RQ5: it holds up on
real, unseen contracts.

5 Conclusion



1.2 Background

A lifecycle-based taxonomy of Trap Tokens



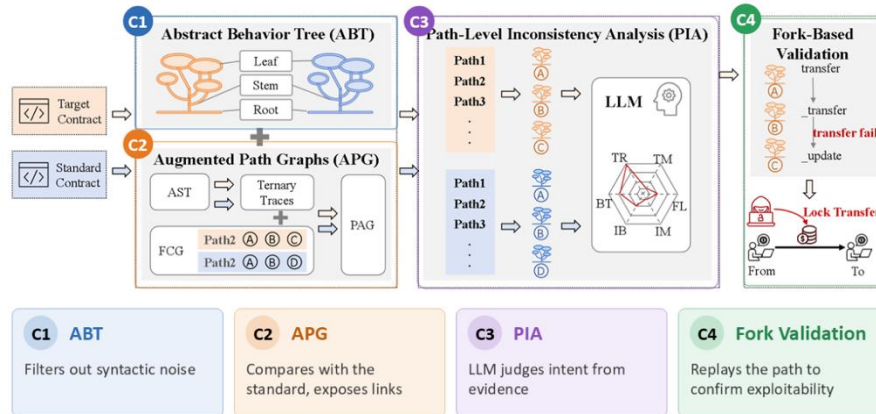
2.2 Challenges

A naive approach



- C1 Syntactic noise & deep inheritance**
Modifiers, SafeMath wrappers and assembly blocks hide the malicious logic in deep inheritance.
- C2 Implicit cross-function state**
A trigger (setBlacklist) and its trap (a transfer revert) meet only through shared state — invisible to static analysis.
- C3 Intent ambiguity**
Deviating from the reference is not inherently malicious — we must reason about intent, not just syntax.
- C4 Unverified exploitability**
Even a correct guess is unproven — we cannot tell whether the flagged path is actually exploitable.

3 Methodology



4.3 RQ2 & RQ3 Comparison and Ablation

RQ2 How does TrapHunter compare with existing tools?

Table 6. Performance comparison on trap detection across different tools

Category	CRPWarner			Pied-Piper			TrapHunter (DeepSeek)			TrapHunter (GPT)			TrapHunter (Gemini)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
TR	0.562	0.494	0.526	0.833	0.361	0.504	0.923	0.906	0.914	0.910	0.933	0.921	0.867	0.929	0.897
IM	-	-	-	0.500	0.778	0.609	0.931	0.784	0.851	0.915	0.883	0.899	0.923	0.865	0.893
IB	0.925	0.439	0.595	0.884	0.546	0.675	0.484	0.550	0.610	0.645	0.600	0.631	0.606	0.900	0.785
TM	0.230	0.288	0.256	-	-	-	0.919	0.634	0.750	0.982	0.761	0.857	0.941	0.873	0.906
BT	-	-	-	-	-	-	0.319	0.695	0.437	0.307	0.796	0.443	0.220	0.898	0.333
FL	-	-	-	-	-	-	0.469	0.938	0.625	0.615	1.000	0.762	0.474	0.562	0.514
Overall	0.572	0.407	0.459	0.739	0.562	0.596	0.829	0.803	0.816	0.831	0.870	0.852	0.794	0.890	0.839

Dash (-) indicates missing data. Overall is averaged over available categories only.

Coverage gap: baselines detect at most 3 of 6 categories.

Accuracy: TrapHunter surpasses the best baseline by **+0.25 F1**.

Answering RQ2: While baselines are high speed, low coverage, TrapHunter is high coverage, high accuracy.

RQ3 What does each component contribute?

Table 8. Ablation study of TrapHunter components using the GPT backend. (Precision / Recall / F1).

Category	w/o ABT			w/o APG			w/o PIA			TrapHunter (GPT)		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
TR	0.839	0.803	0.821	0.784	0.889	0.833	0.986	0.715	0.829	0.910	0.933	0.921
IM	0.587	0.486	0.533	0.505	0.423	0.461	1.000	0.358	0.527	0.915	0.883	0.899
IB	0.255	0.264	0.259	0.214	0.170	0.189	1.000	0.353	0.522	0.665	0.600	0.631
TM	0.336	0.623	0.437	0.321	0.261	0.288	1.000	0.020	0.040	0.982	0.761	0.857
BT	0.205	0.878	0.332	0.230	0.837	0.361	1.000	0.281	0.438	0.307	0.796	0.443
FL	0.081	0.750	0.145	0.080	0.875	0.147	1.000	1.000	1.000	0.615	1.000	0.762
Overall	0.577	0.710	0.637	0.581	0.654	0.614	0.990	0.499	0.663	0.831	0.870	0.850

Answering RQ3: no single component suffices — each fixes a distinct failure mode.

- ABT** IM precision falls from 0.915 to 0.587 without it
- APG** TM recall falls from 0.761 to 0.261 without it
- PIA** Overall recall falls from 0.870 to 0.499 without it