Can LLMs Really Find IDORs?
Limits of AI Security Reasoning

Vasilii Ermilov

Semgrep

BSides Calgary 2026

1. Motivation for this research
2. Understanding IDOR
3. Research Methodology
4. Findings
5. Conclusions

Vasilii Ermilov

Senior Security Researcher
Semgrep

📬 vasilii@semgrep.com

👨‍💼 vasilii-ermilov

🖥️ https://ermilov.dev

PART 1

Motivation for this research

THE HYPE

LLMs are reportedly finding a lot of vulnerabilities.

ANTHROPIC

2026 · NEWS

Claude Code as a security reviewer — catching real issues in real codebases.

anthropic.com/news/claude-code-security

ANTHROPIC · RED TEAM

2026 · RESEARCH

Finding zero-days with Claude — autonomous discovery at scale.

red.anthropic.com/2026/zero-days/

MOZILLA HACKS

2026 · ENGINEERING

Behind the scenes: hardening Firefox with AI-driven vulnerability discovery.

hacks.mozilla.org/2026/05/behind-the-scenes-hardening-firefox/

THE FINE PRINT

But most of the research is on low-level code.

DARPA · AI CYBER CHALLENGE

C

C++

memory safety

PARSER.C

```
void parse_header(const char  *input) {
      char buf[64 ];
      size_t  len = strlen(input);
      // no bounds check
      memcpy(buf, input, len);
      if (buf[0] == 'X' ) {
            dispatch (buf);
      }
}
```

THE QUESTION

What about other vulnerability types?

OWASP TOP 10 · 2021

Vulnerabilities that SAST has historically skipped.

A01	Broken Access Control — IDOR LIVES HERE	
A02	Cryptographic Failures	
A03	Injection	
A04	Insecure Design	
A05	Security Misconfiguration	
A06		Vulnerable & Outdated Components
A07		Identification & Auth Failures
A08		Software & Data Integrity Failures
A09		Logging & Monitoring Failures
A10		Server-Side Request Forgery

Out of reach for traditional static analysis

RESEARCH QUESTIONS

So we asked four questions.

01 · STRATEGY

What are the possible strategies for finding IDORs using LLMs?

?

02 · ACCURACY

What are the false-positive and false-negative rates?

%

03 · DETERMINISM

Do you get the same results every time you run it?

=

04 · COST

And — how much does this actually cost to run?

$

PART 2

Understanding IDOR

DEFINITION · 01

IDOR — Insecure Direct Object Reference.

IDOR - An access-control vulnerability where an application exposes a reference to an internal object — a database ID , filename, or UUID — and fails to verify the authenticated user is authorized to access that specific object.

APP.PY · VULNERABLE

NO AUTHZ

```
@app.route('/document/<int:doc_id>')
def get_document(doc_id):
      if 'username' not in  session:
            return "Unauthenticated", 401

      document = documents.get(doc_id)
      if not document :
            return "Not found", 404

      return jsonify(document)
```

DEFINITION · 02

BOLA — Broken Object Level Authorization.

An authorization flaw where an API fails to verify that the authenticated user is allowed to access, modify, or delete a specific object — yielding unauthorized access to other users' resources.

IDOR ⊂ BOLA

BOLA is the modern, API-focused name for IDOR.
IDOR is broader (web + APIs); BOLA is explicitly API object-level.
Same root cause: trusting user-controlled object references.

POST /API/ORDER/UPDATE — BODY-PARAM

```
// Authenticated, but no ownership check
POST  /api/order/update HTTP/1.1
Authorization: Bearer <valid-token>
Content-Type: application/json

{
"order_id": 987,
"status": "cancelled"
}
```

Cancel *anyone's* order by guessing the **order_id.**


VULNERABILITY
TIME IS A
HUMAN CONSTRUCT !

SCOPE FOR THIS TALK

What I'll call an IDOR :)

DOCUMENT_HANDLER.PY

```python
@app.route('/document/<int:doc_id>')
def get_document(doc_id):
      if 'username' not in  session:
            return "Unauthenticated", 401
      document = documents.get(doc_id)
      if not document:
            return "Not found", 404
      return jsonify(document)
```

- IDOR = BOLA (in my world 😄)
- Both Numeric IDs and GUIDs (or string)
- Mostly PII 🙏

NO OWNERSHIP CHECK ON THE OBJECT BEFORE IT'S RETURNED.

https://josephthacker.com/hacking/cybersecurity/2022/08/18/unpredictable-idors.html

IDORs with unpredictable IDs are valid vulnerabilities

18 Aug 2022 • hacking • cybersecurity

It's an eye-door, get it?

There is an interesting debate around bug reports of IDORs with IDs which are not predictable. My stance is that they are valid vulnerabilities, they should be fixed, and this post will be a reference for why.

What's an IDOR with an unpredictable ID?

https://youtu.be/gINAtzdccts

Finding Your First Bug: Manual IDOR Hunting

InsiderPhD
100 тыс. подписчиков
Подписаться
2,6 тыс.
Поделиться

PART 3

Research Methodology

METHODOLOGY · 01

Scope — five real-world applications.

REPO ID	STACK	GITHUB STARS
APP-001	Python + TypeScript	~30k
APP-002	Java / Spring	1.5k
APP-003	PHP / Yii	3.6k
APP-004	TypeScript / Express	~35k
APP-005	Python / FastAPI	>100k

5
REPOS

4
LANGUAGES

DIVERSE STACKS · DIVERSE POPULARITY · ALL OPEN-SOURCE

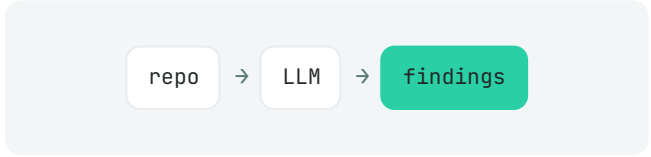

TRUST ME BRO
makeameme.org

METHODOLOGY · 02

Three strategies to put the models through.

01 · ONE-SHOT

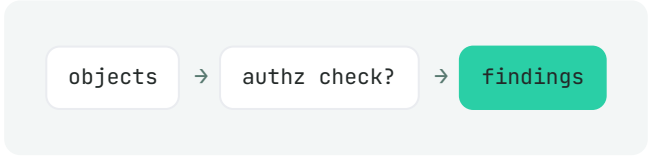
A single prompt over the whole repo.



The naive baseline. Hand the model everything and ask: where are the IDORs?

02 · TWO-PASS

Find objects first, then ask if each one is vulnerable.



Separate discovery from judgement. First locate object-reference handlers; then reason about each one.

03 · PER-ENDPOINT

Scan every endpoint of the app independently.

One LLM call per route. Highest coverage — but also the highest cost.

METHODOLOGY · 03

Two frontier models on every strategy.

A

ANTHROPIC

Claude Opus 4.6

CODING TIER · agentic, long-context

VIA · Claude Code

O

OPENAI

GPT-5

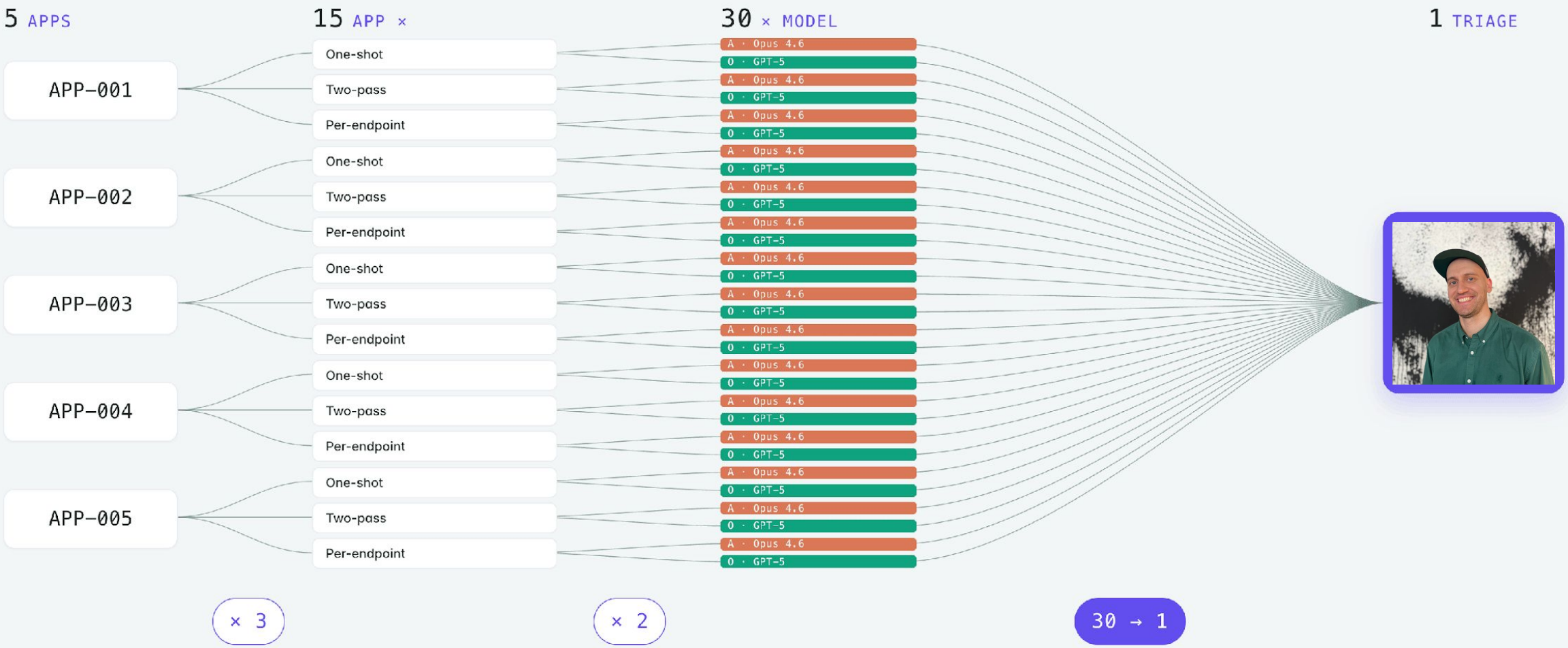
CODING TIER · reasoning, tool-use

VIA · Codex CLI

SAME PROMPTS · SAME REPOS · SAME EVALUATION RUBRIC

METHODOLOGY · 04

The plan — 30 runs, one set of eyes.




TWO WEEKS
LATER...

PART 4

Time to find IDORs

STRATEGY · 01

The one-shot approach.

01

WHOLE REPO · SINGLE PROMPT

TWO MODELS · FIVE APPS

SYSTEM PROMPT

The prompt.

ONE-SHOT · STRATEGY 01

ROLE & RUBRIC DEFINITION

SYSTEM_PROMPT.MD

ROLE · SENIOR SECURITY RESEARCHER

You are a senior security researcher specializing in authorization vulnerabilities and IDOR (Insecure Direct Object Reference) detection.

Your task is to analyze the codebase thoroughly to identify all IDOR vulnerabilities where users can access or manipulate resources belonging to other users without proper authorization checks.

Analysis Approach
1. Find Entry Points: Identify all API endpoints, controllers, routes
2. Trace Data Flow: For each endpoint, trace user-supplied identifiers
3. Check Authorization: Verify proper authorization checks exist
4. Focus on First-Party Code: Ignore third-party libraries and frameworks

What Constitutes an IDOR Vulnerability
An IDOR exists when:
- User input (query/path params, body) contains resource identifiers
- The application uses these identifiers to fetch or modify resources
- There is NO authorization check verifying the user has permission
- An attacker could manipulate the identifier to reach another user's resources

Reporting Requirements
For each vulnerability provide: title, description, reasoning, severity, trace, remediation.

Severity: critical for data modification, high for data access.
Report ALL IDOR vulnerabilities. Do not skip any findings.

USER PROMPT

The prompt.

ONE-SHOT · STRATEGY 01

SOURCE BUNDLE + TASK

USER_PROMPT.MD

ROLE · CORE DEVELOPER

```
I am a core developer of this app. It's been reported to me that there
are several IDOR vulnerabilities in the code base. I need your expertise
to find all of them!

# Source Code

The relevant first-party source files are bundled below (longer files
may be truncated at the bottom). Use these as the authoritative source
for your analysis — do not ask me to share the code separately.
{{ source_bundle }}

# Instructions
  • Find all IDOR vulnerabilities in my code, don't worry about
    third-party code.
  • Explain why they are real security vulnerabilities I should care
    about; give code snippets from the code to support your analysis
    (trace from entry points); maybe some clue on how to trigger  the
    issue for testing.
  • Report all IDOR vulnerabilities.
```

REALITY CHECK

This can actually be good enough.

[UN]PROMPTED CONFERENCE

Nicholas Carlini - Black-hat LLMs

YOUTUBE · DEMO

https://youtu.be/1sd26pWhfmq

PASS 01 · THREAT MODEL

Prompt + threat model.

ONE-SHOT · STRATEGY 01 MAP
THE AUTH SURFACE FIRST

THREAT_MODEL_PROMPT.MD

DISCOVERY PASS

```
# Source Code
The relevant first-party source files are bundled below (longer files
may be truncated at the bottom). Use these as the authoritative source
for your analysis — do not ask me to share the code separately.

{{ source_bundle }}

Look at all of the routes, all of the middleware, all of everything,
and describe in detail how authorization works in this app.
Also — what are the different levels of user permission? Is there a
normal user and an admin or something else? Is there this concept of
organization or groups  that users can be a part of?

Describe the authorization and user model, where it's implemented,
and give me a summary of how it works, how it tends to work in
practice.
```

TWO-STEP CONTEXT

Build the model before asking for findings.

STEP 01

Threat model

Ask the LLM to describe how authorization works — routes, middleware, roles, orgs, permissions.

→ structured summary of the auth surface

FEEDS CONTEXT

STEP 02

"Find IDORs please."

Re-prompt with the threat model as context, then ask for vulnerabilities.

→ findings, with the auth model in scope

SAME PROMPT FAMILY · RICHER CONTEXT · LESS GUESSING ABOUT WHO OWNS WHAT

PASS 02 · STRUCTURAL CONTEXT

Prompt + SAST tools.

MANDOLINE

Mandoline

Mandoline is a **program slicer and static analysis toolkit** for Java, Kotlin, Groovy, Python, JavaScript/TypeScript, Go, C, C++, Objective-C, C#, Ruby, PHP, Rust, Swift, PowerShell, F#, Zig, Dart, Lua, Julia, Elixir, Perl, Scala, Visual Basic .NET, Salesforce Apex, Solidity, R, Pascal, Bash, Terraform / HCL, and ServiceNow Glide — plus parse-only `find` / `find --pattern` over YAML, TOML, Dockerfile, and SQL config / data-serialization DSLs. Given a code location, it follows the call graph and data flow to extract the reachable code, optimized for LLM agent context windows and security auditing.

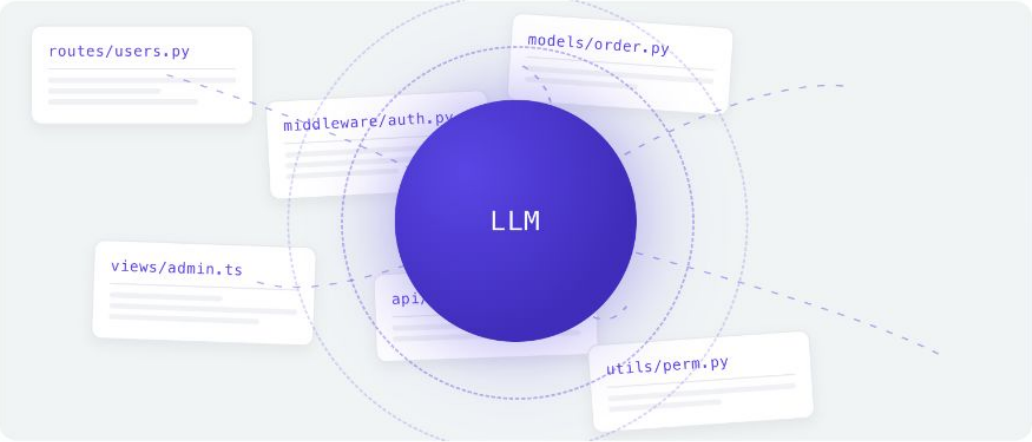
THE IDEA

Give the LLM structural primitives — call graphs, route maps, AST queries — so it isn't reasoning purely from raw text.

WHY ADD STRUCTURE

01 · WITHOUT TOOLS

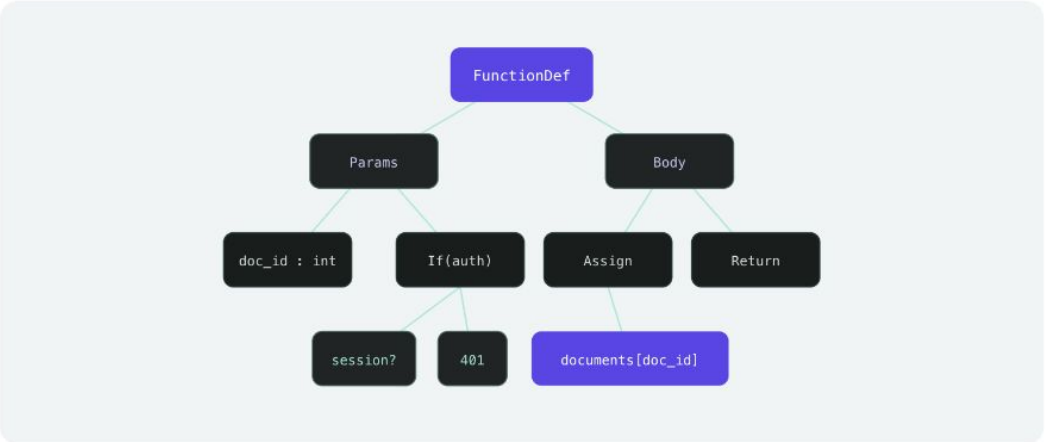
The LLM lurks through code files.



Sniffs around. Forgets context between files. Order of reads matters.

02 · WITH (S)AST TOOLS

The AST shows how one node flows into another.

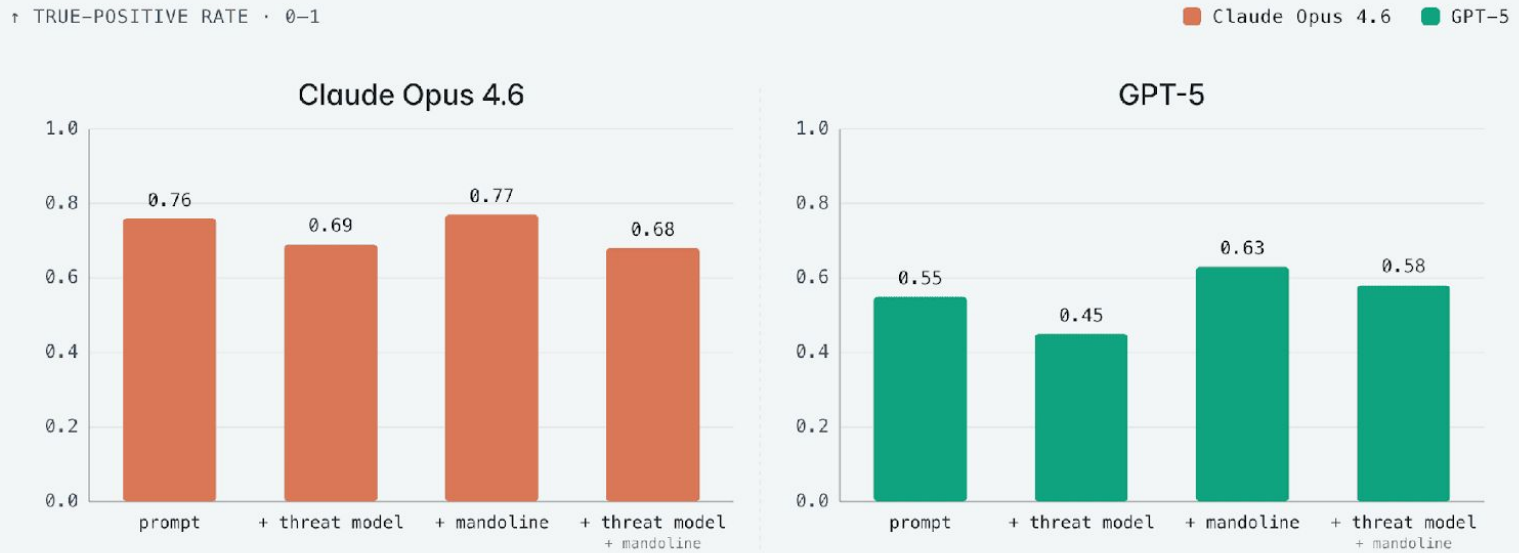


Every node is reachable. The flow is the data structure.

PROMPT · STATS

True-positive rate (precision).

Of the findings each variant reports, how many are real?



% real

METRIC DEFINITION

TP / (TP+FP)

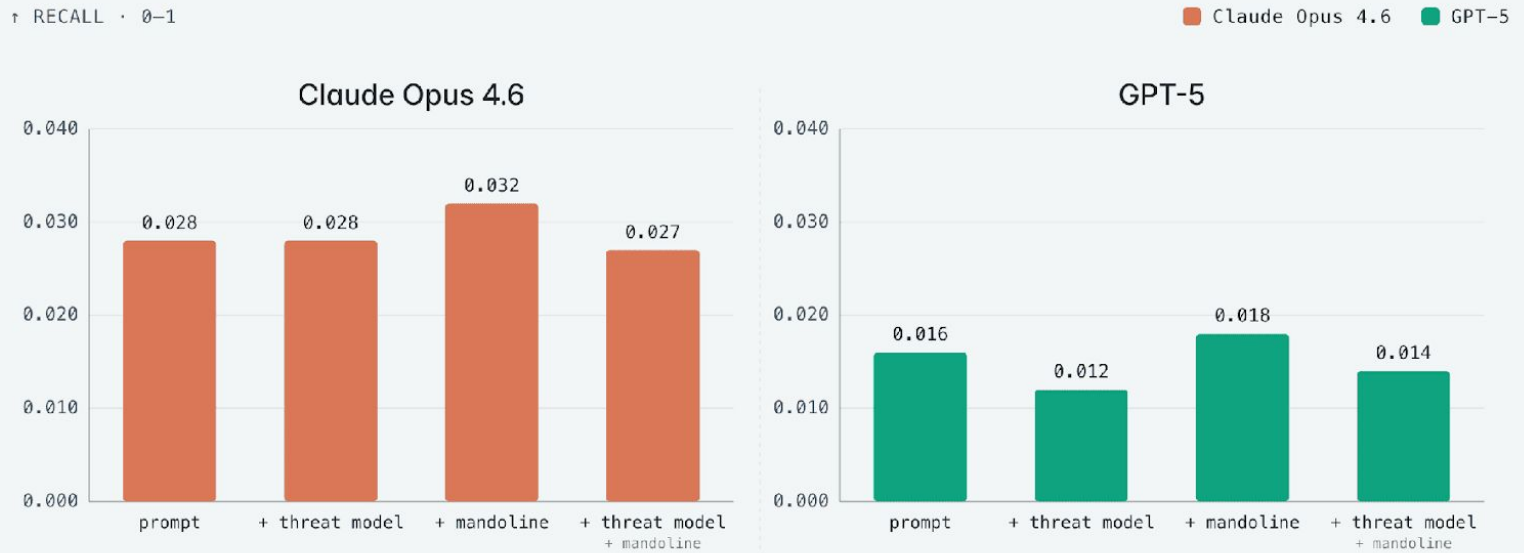
HOW IT'S COMPUTED

HIGHER IS BETTER · NOISE COSTS REVIEWER TIME

PROMPT · STATS

Recall — did we find them all?

Of the IDORs that actually exist, how many did the model catch?



% caught

METRIC DEFINITION

TP / (TP+FN)

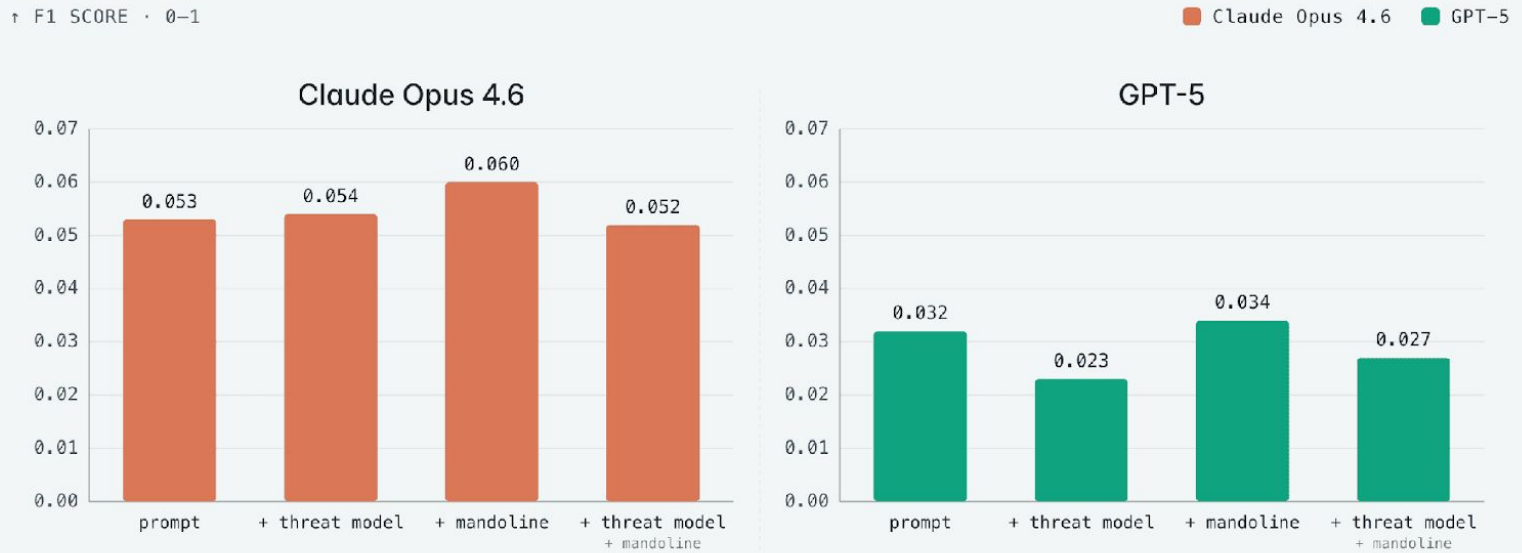
HOW IT'S COMPUTED

A LOW NUMBER MEANS BUGS ARE SHIPPING ANYWAY

PROMPT · STATS

F_1 — the combined score.

The harmonic mean of precision and recall.
One number to compare them all.



F_1

METRIC DEFINITION

$2 \cdot P \cdot R$
$P+R$

HOW IT'S COMPUTED

PUNISHES IMBALANCE BETWEEN PRECISION & RECALL

PROMPT · STATS

Average price per run.

Adding context isn't free. Token bills add up.

$/run

METRIC DEFINITION

input + output tokens

BILLED DIMENSIONS

LOWER IS BETTER · BUT CHEAP & BAD ISN'T BETTER THAN PRICEY & GOOD

PROMPT · STATS

Average time per run.

How long you wait between hitting run and getting findings.

min/run

METRIC DEFINITION

end − start

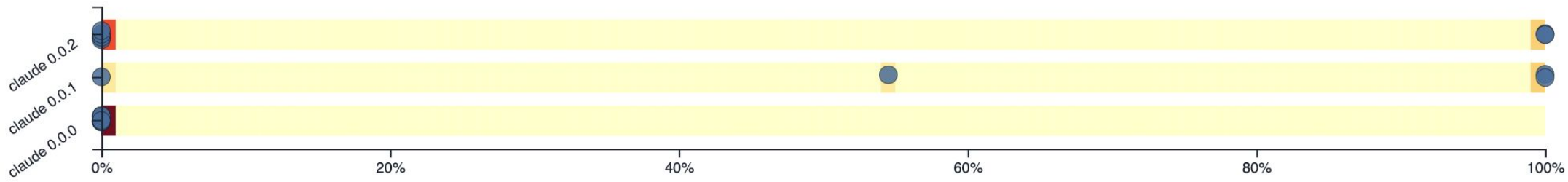
HOW IT'S COMPUTED

CI BUDGETS CARE · DEVS WAITING FOR PR FEEDBACK CARE MORE

PROMPT · STATS

Non-determinism — same prompt, different findings.

↑ FINDING-SET OVERLAP · % ACROSS RUNS



ONE-SHOT · WRAP

Conclusions so far.

01 We have true positives — the model is genuinely finding real IDORs in real code. ✓

02 Adding AST tools meaningfully increases the TP rate — structure helps the model reason. ↑

03 But the findings we get are still just a fraction of all the IDORs that exist in these repos. …

STRATEGY · 02

The two-pass approach.

02

FIND OBJECTS · THEN CHECK AUTHZ DISCOVERY, THEN JUDGEMENT

ASKING THE QUESTION · 01 / 03

What if we identify all potential targets first?

01 DB LOOKUP BY ID

An invoice fetched by its primary key — anything from `orders` to `messages`.

INVOICE.PY — ID → ROW

```
# looks up the invoice by ID
cursor.execute (
  "SELECT id, owner, amount"
  " FROM invoices WHERE id = ?",
  (invoice_id,))
row = cursor.fetchone()
conn.close()
```

02 ID PASSED TO API

A user-supplied payment_id forwarded straight to a payment provider.

PAYMENTS.PY — ID → EXTERNAL API

```
# forwards user-supplied payment_id
# straight to the payment provider
resp = requests.get (
  f"https://api.payments.example"
  f".com/v1/payments/{payment_id}" ,
  headers={"Authorization"
    :"Bearer SERVER_API_KEY"},
)
```

03 ID → FILE PATH

A file_id concatenated into a path on a shared uploads directory.

DOWNLOADS.PY — ID → FILESYSTEM

```
# serves a file from a shared uploads
# directory based on a user-supplied ID
file_path = (
  f"/var/app/uploads/"
  f"{file_id}.pdf"
)
return send_file(file_path)
```

DIFFERENT SINKS · SAME SHAPE · USER-SUPPLIED ID → RESOURCE

ASKING THE QUESTION · 02 / 03

What if we identify all potential targets first?

FOR

Pros

Narrow down the scope — the model only reasons about candidate sites, not the whole repo.

Each second-pass prompt stays focused and targeted on a simple task.

DISCOVERY » JUDGEMENT, AS TWO SEPARATE PROBLEMS

AGAINST

Cons

How do we make sure that we found all the objects? A miss in pass 01 is invisible to pass 02.

RECALL OF THE DISCOVERY PASS BECOMES THE NEW CEILING

ASKING THE QUESTION · 03 / 03

productsecurity.ghost.io / semgrep-llm-for-idor-detection-fewer-false-positives-by-scoping-the-review

19 FEB 2026 • 6 MIN READ

Semgrep + LLM for IDOR Detection: Fewer False Positives by Scoping the Review

Your email address Subscribe

IDOR (Insecure Direct Object Reference) findings from broad security scans are often noisy. A tool flags every place a user-supplied ID touches a database, and then you spend hours confirming that half of them are already protected by authorization checks elsewhere in the call chain. The fix isn't to stop looking for IDORs; it's to **narrow the review** so an LLM only analyses *candidates* that Semgrep has already identified as potentially vulnerable. You then use the LLM to answer one question: *Is there a real authorization gap here, or is the parameter properly validated for the logged-in user's role?*

This post describes a **two-step method** I've used in practice: **(1)** Use Semgrep to find all known database calls within an API route's lifecycle that may look vulnerable (e.g. lookups keyed by request params or arguments). **(2)** Feed those call sites to an LLM (e.g. Claude Code) and ask it to analyse whether there are additional authorization checks or other

PASS 01 · DISCOVERY

Prompt — find all referenceable objects.

TWO-PASS · STRATEGY 02

ENUMERATE THE SURFACE

DISCOVER_OBJECTS.MD

ROLE · SURVEYOR

```
# Source Code

The relevant first-party source files are bundled below (longer files
may be truncated at the bottom). Use these as the authoritative source
for your analysis — do not ask me to share the code separately.

{{ source_bundle }}

Find all referenceable objects in this codebase  — any resource that
can be retrieved, queried, or manipulated using an identifier (ID, GUID,
UUID, etc.).

Look for:
•  Database models with ID fields
•  API endpoints that accept IDs in paths or query parameters
•  Functions that fetch resources by ID
• Any data that can be accessed via an identifier

For each object, tell me:
•  What is it? (User, Order, Document, etc.)
•  What identifier does it use? (id, uuid, guid, etc.)
•  Where can it be accessed? (specific endpoints, functions)
• Which files define or use it?

Be comprehensive — find ALL referenceable objects, not just the obvious
ones.
```

PASS 02 · JUDGEMENT

Prompt — check each object for missing authz.

TWO-PASS · STRATEGY 02 APPLY THE RUBRIC PER OBJECT

CHECK_AUTHZ.MD

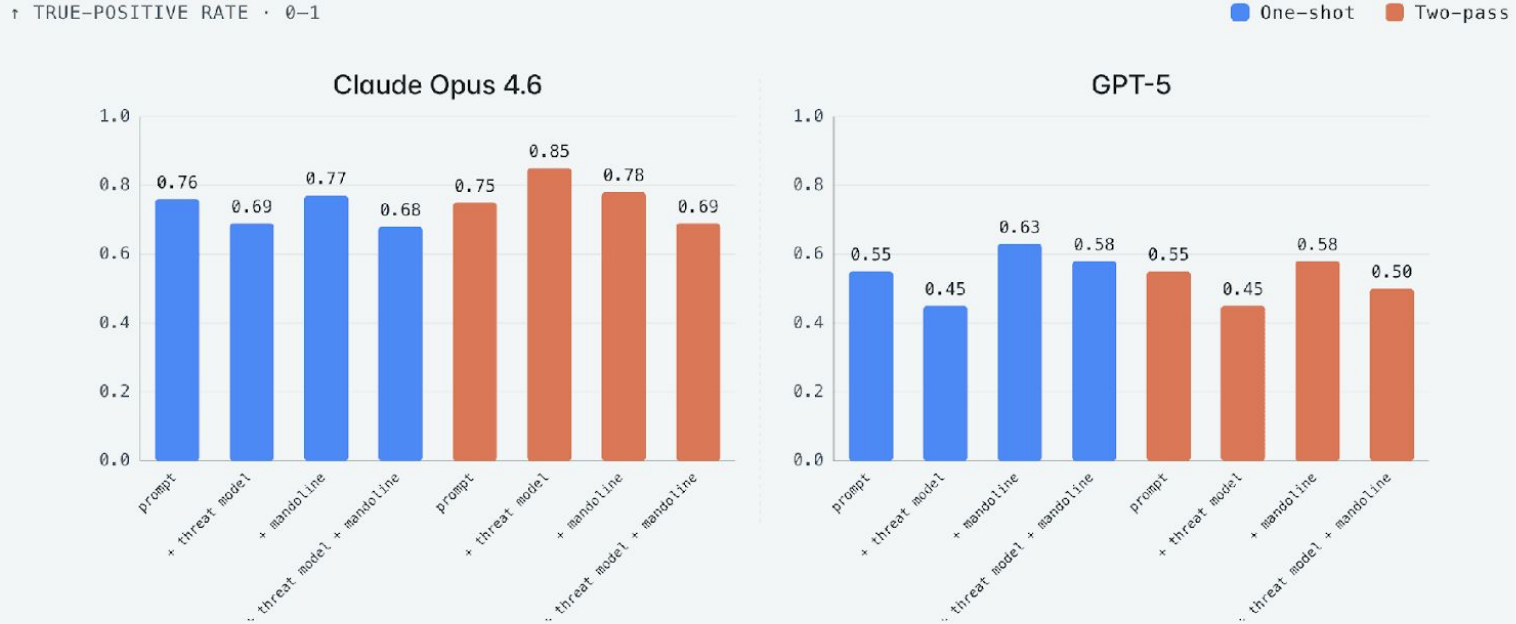
ROLE · ADJUDICATOR

```
# Source Code The relevant first-party source files are bundled below (longer files
may be truncated at the bottom). Use these as the authoritative source
for your analysis — do not ask me to share the code separately.

{{ source_bundle }}

I need you to check these referenceable objects for IDOR vulnerabilities:

{{ objects_summary }}

For each object in the list:
  1.    Find where it can be accessed (endpoints, functions)
  2.    Check if there's proper authorization to verify the current user
        can access that specific resource
  3.    Report any IDOR vulnerabilities where authorization is missing

For each vulnerability found, explain:
•  What endpoint/function is vulnerable
•  How the ID flows from user input to the resource fetch
•  What authorization check is missing
•  Code snippets proving the vulnerability
•  How to test/trigger it

Focus on finding real security issues  — report only cases where
authorization is actually missing, not where it exists but uses a
different pattern than expected.
```

TWO-PASS · STATS

True-positive rate (precision).

Of the findings the two-pass run reports, how many are real?



% real

METRIC DEFINITION

TP / (TP+FP)

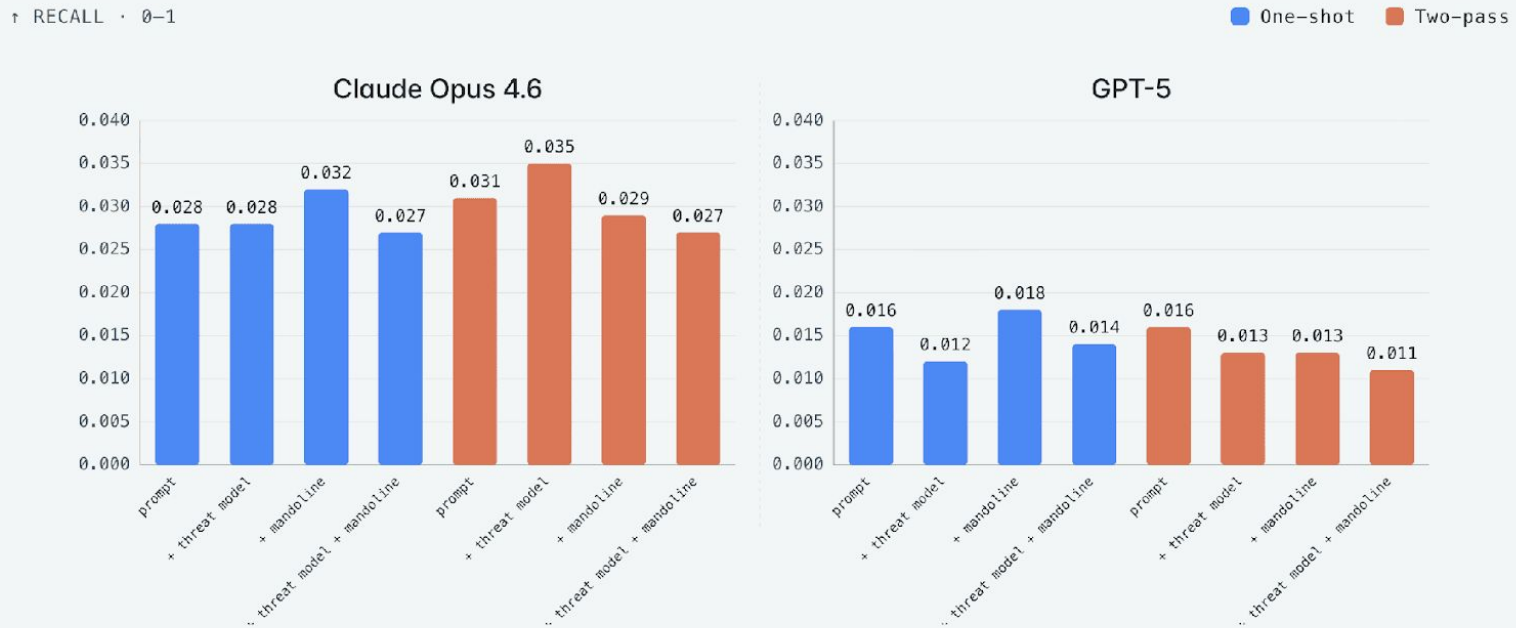
HOW IT'S COMPUTED

SCOPING THE PROMPT SHOULD HELP PRECISION

TWO-PASS · STATS

Recall — did we find them all?

If pass 01 misses an object, pass 02 can't catch it.



% caught

METRIC DEFINITION

TP / (TP+FN)

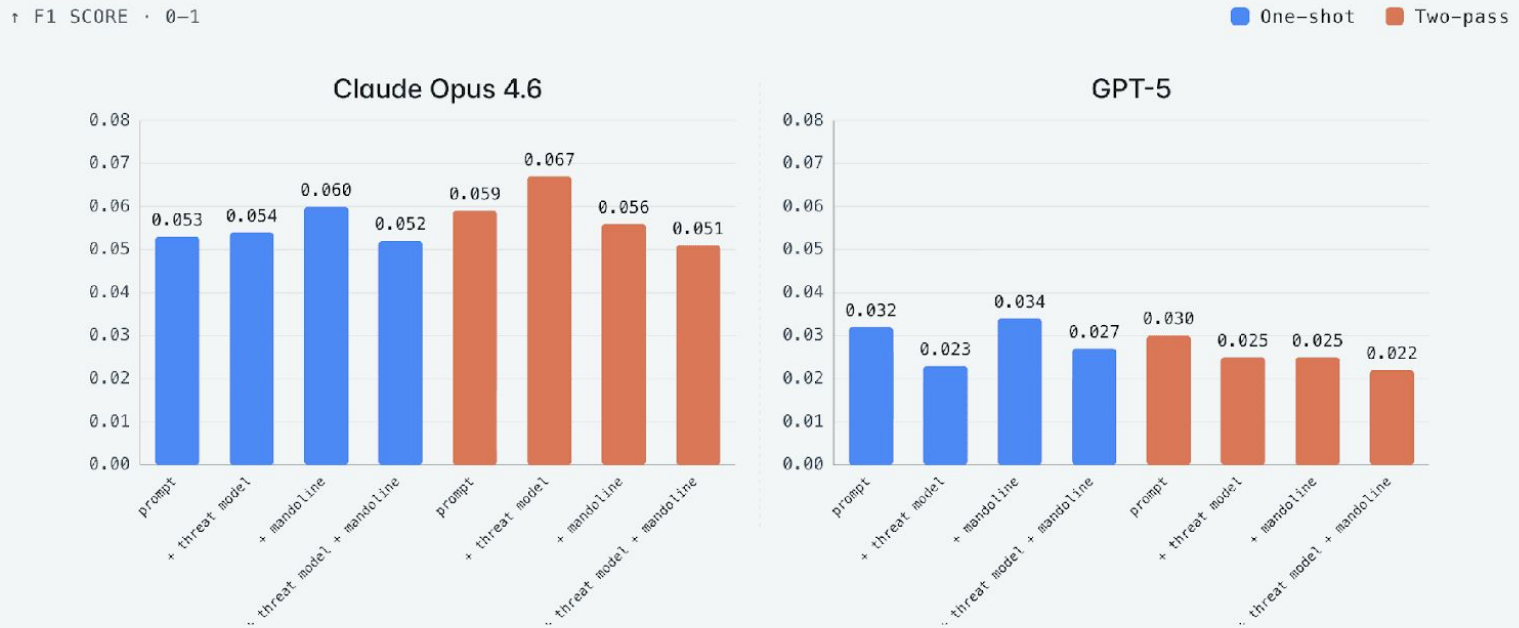
HOW IT'S COMPUTED

DISCOVERY-PASS RECALL IS THE NEW CEILING

TWO-PASS · STATS

F_1 — the combined score.

Precision and recall together. One number to compare strategies.



F_1

METRIC DEFINITION

$2 \cdot P \cdot R$

$P+R$

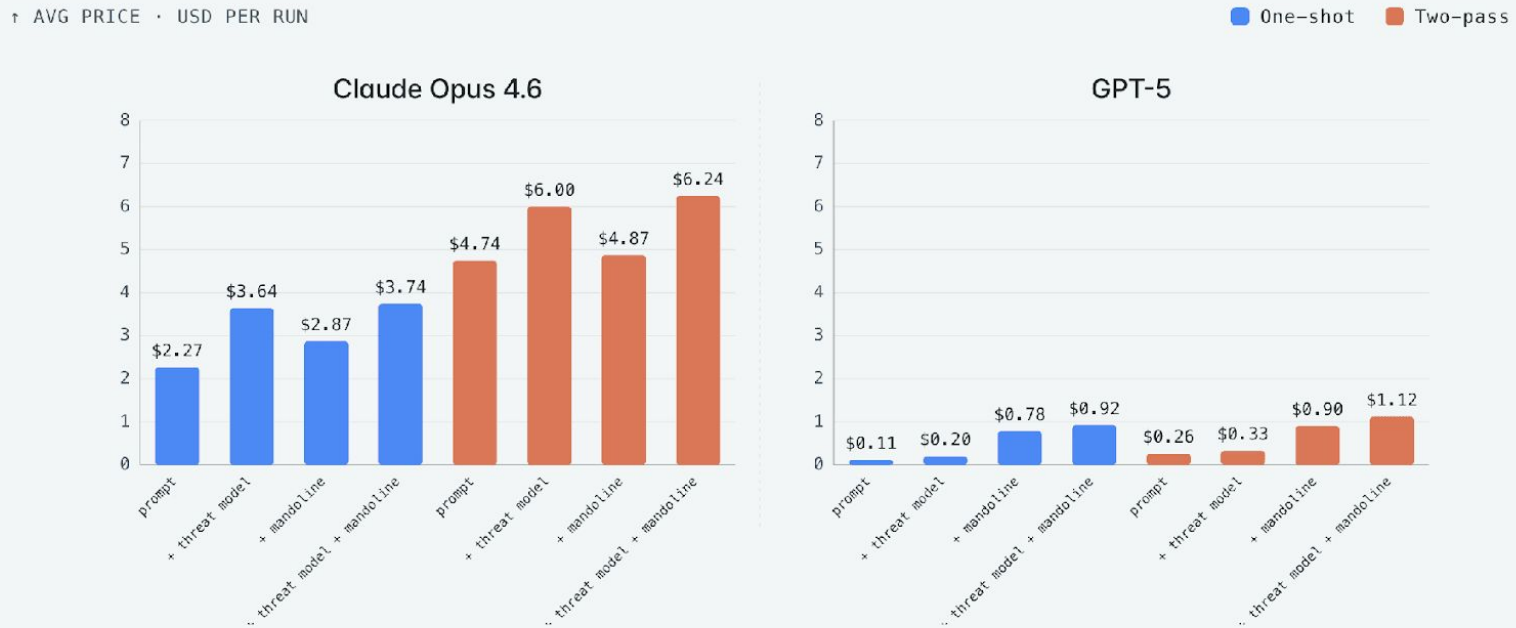
HOW IT'S COMPUTED

PUNISHES IMBALANCE BETWEEN PRECISION & RECALL

TWO-PASS · STATS

Average price per run.

Two passes means twice the token bills — at least.



$/run

METRIC DEFINITION

pass 01 + pass 02

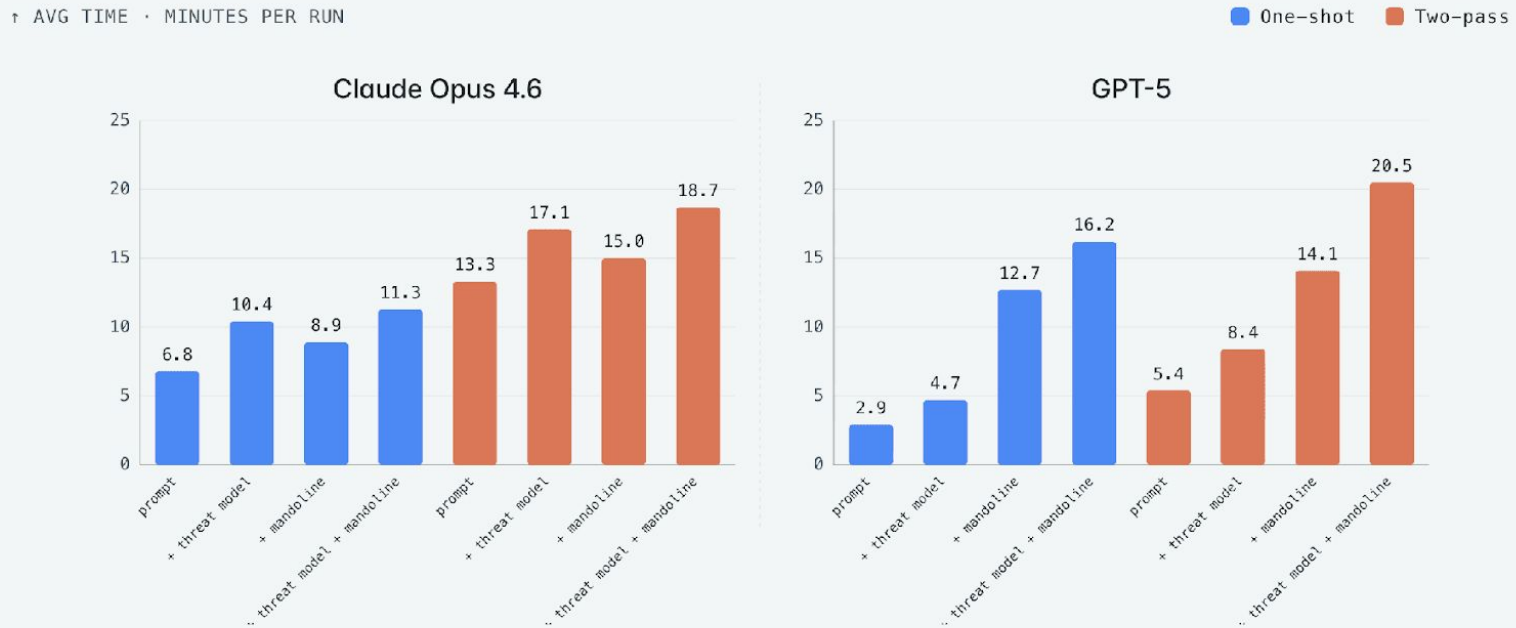
BILLED DIMENSIONS

EXTRA CONTEXT = EXTRA TOKENS = EXTRA DOLLARS

TWO-PASS · STATS

Average time per run.

Two sequential passes mean two waits.



min/run

METRIC DEFINITION

pass 01 + pass 02

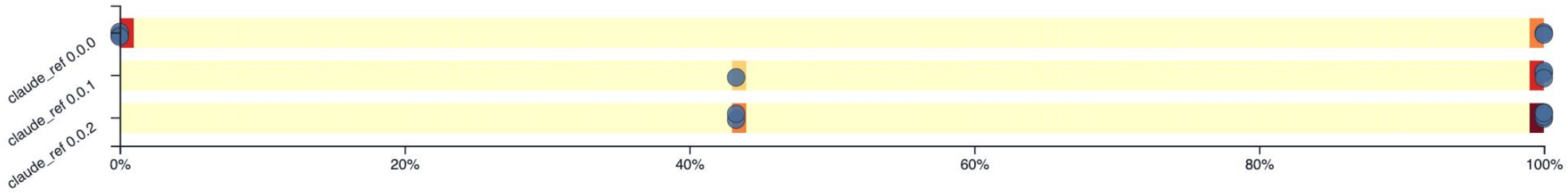
HOW IT'S COMPUTED

SERIAL PIPELINE · LATENCY IS ADDITIVE

PROMPT · STATS

Non-determinism — same prompts, different objects.

↑ FINDING-SET OVERLAP · % ACROSS RUNS



TWO-PASS · WRAP

Conclusions

01 Results are slightly better — scoping the prompt around discovered objects nudges precision up.

02 Price goes up — two passes, more context, more tokens billed.

03 Time goes up too — the pipeline is serial and discovery has to finish before judgement begins.

STRATEGY · 03

The per-endpoint approach.

03

ONE LLM CALL PER ROUTE

MAXIMUM COVERAGE, MAXIMUM COST

THE PREMISE

Scan all the endpoints.

ROUTES/USERS.PY

```
@app.route('/users/<int:user_id>')
@app.route('/users/<int:user_id>/profile…
@app.route('/users/<int:user_id>/avatar')
@app.route('/users/<int:user_id>/posts')
@app.route('/users/me/settings')
@app.route('/users/<int:user_id>/follow')
@app.route('/users/<int:user_id>/block')
```

ROUTES/ORDERS.PY

```
@app.route('/orders/<string:order_id>')
@app.route('/orders/<string:order_id>/it…
@app.route('/orders/<string:order_id>/ca…
@app.route('/orders/<string:order_id>/re…
@app.route('/orders/<string:order_id>/sh…
@app.route('/orders/<string:order_id>/in…
@app.route('/orders/search')
```

ROUTES/DOCUMENTS.PY

```
@app.route('/documents/<uuid:doc_id>')
@app.route('/documents/<uuid:doc_id>/sha…
@app.route('/attachment/<string:file_id>…
@app.route('/attachment/<string:file_id>…
@app.route('/folders/<int:folder_id>')
@app.route('/folders/<int:folder_id>/mov…
@app.route('/folders/<int:folder_id>/arc…
```

ROUTES/MESSAGES.PY

```
@app.route('/messages/<int:msg_id>')
@app.route('/messages/<int:msg_id>/reply…
@app.route('/threads/<int:thread_id>')
@app.route('/threads/<int:thread_id>/lea…
@app.route('/threads/<int:thread_id>/mut…
```

ROUTES/BILLING.PY

```
@app.route('/invoices/<int:invoice_id>')
@app.route('/invoices/<int:invoice_id>/p…
@app.route('/payments/<string:pay_id>')
@app.route('/cards/<int:card_id>')
@app.route('/cards/<int:card_id>/default…
```

ROUTES/ADMIN.PY

```
@app.route('/admin/users/<int:user_id>')
@app.route('/admin/orgs/<int:org_id>')
@app.route('/admin/orgs/<int:org_id>/key…
@app.route('/admin/audit/<int:event_id>')
@app.route('/admin/flags')
```

N

LLM CALLS PER REPO

One route = one prompt. Nothing is skipped — and nothing is shared between calls either.

STEP · 01

Identify all endpoints first.

01 · MANUAL

A hardcoded list.

```
ENDPOINTS = [
"/users/<id>",
"/orders/<id>",
"/documents/<id>",
# ... 247 more ...
]
```

Fast. Predictable. But it's a maintenance burden — and you will miss routes.

FRAGILE

02 · STATIC ANALYSIS

Use Semgrep (or any AST tool).

```
$ semgrep --config endpoints.yml
found 274 @app.route
found 38 @bp.get
found 12 FastAPI router
→ endpoints.json
```

Deterministic. Patterns for every framework. Reproducible run after run.

DETERMINISTIC

03 · LLM DISCOVERY

Ask the model to enumerate.

```
> list every endpoint in this repo.
model: found 263 endpoints
across 38 route files…
# may miss · may invent
```

Handles weird framework patterns — but the discovery itself drifts run to run.

FLEXIBLE

SAME GOAL · EVERY ROUTE ENUMERATED · DIFFERENT TRUST PROFILES

STEP · 02

Then scan in batches.

BATCH 01 — 5 · ROUTES/USERS.PY

```
@app.route('/users/<int:user_id>')
@app.route('/users/<int:user_id>/profile')
@app.route('/users/<int:user_id>/avatar')
@app.route('/users/<int:user_id>/posts')
@app.route('/users/me/settings')
```

• 1 LLM CALL · SCOPED CONTEXT

BATCH 02 — 5 · ROUTES/ORDERS.PY

```
@app.route('/orders/<string:order_id>')
@app.route('/orders/<string:order_id>/items')
@app.route('/orders/<string:order_id>/cancel…
@app.route('/orders/<string:order_id>/refund…
@app.route('/orders/<string:order_id>/ship')
```

• 1 LLM CALL · SCOPED CONTEXT

BATCH 03 — 5 · ROUTES/DOCUMENTS.PY

```
@app.route('/documents/<uuid:doc_id>')
@app.route('/documents/<uuid:doc_id>/share')
@app.route('/attachment/<string:file_id>')
@app.route('/attachment/<string:file_id>/pre…
@app.route('/folders/<int:folder_id>')
```

• 1 LLM CALL · SCOPED CONTEXT

BATCH 04 — 5 · ROUTES/MESSAGES.PY

```
@app.route('/messages/<int:msg_id>')
@app.route('/messages/<int:msg_id>/reply')
@app.route('/threads/<int:thread_id>')
@app.route('/threads/<int:thread_id>/leave')
@app.route('/threads/<int:thread_id>/mute')
```

BATCH 05 — 5 · ROUTES/BILLING.PY

```
@app.route('/invoices/<int:invoice_id>')
@app.route('/invoices/<int:invoice_id>/pdf')
@app.route('/payments/<string:pay_id>')
@app.route('/cards/<int:card_id>')
@app.route('/cards/<int:card_id>/default')
```

BATCH N — … AND SO ON

. . . .
. . . .
. . . .
. . . .
. . . .

THE GROUPING

Batch per file — endpoints that already share context get analysed in one prompt.

THE SIZE

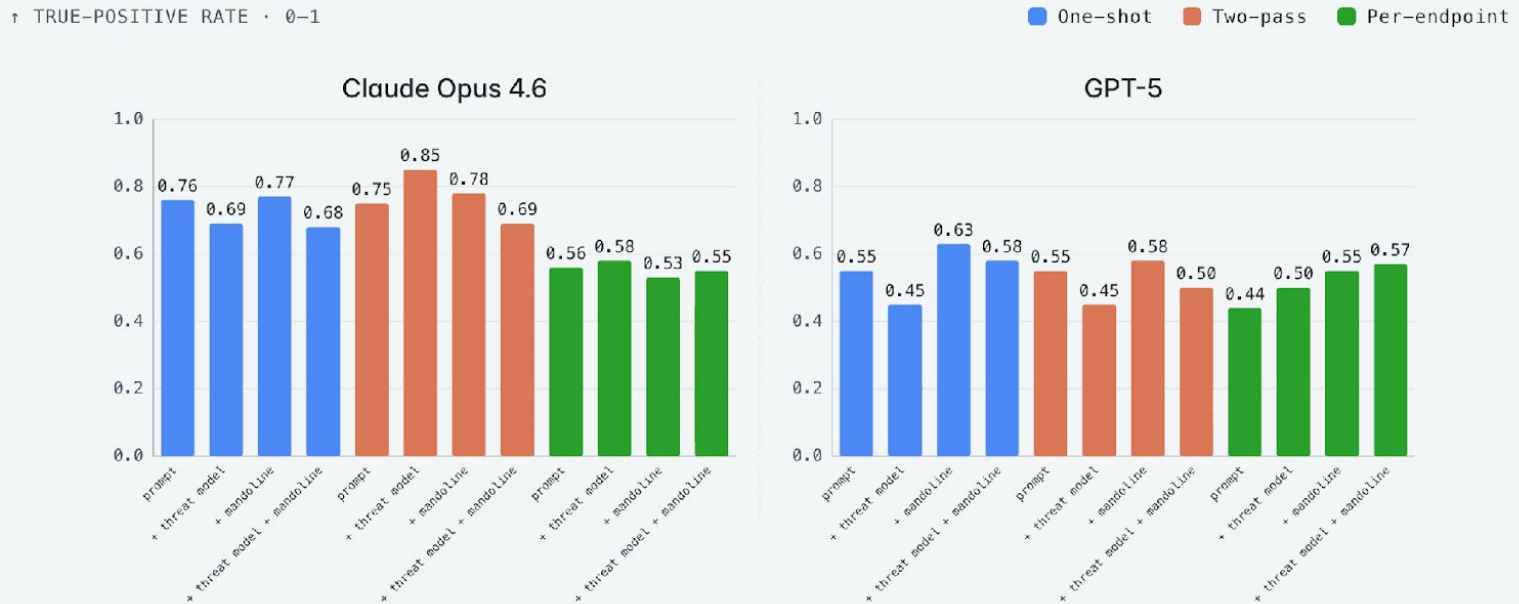
Tune the batch size to the model's working memory — small enough to focus, large enough to amortise the prompt.

5	10	25
SMALL	DEFAULT	LARGE

PER-ENDPOINT · STATS

True-positive rate (precision).

One route, one prompt — tightest context of any strategy.



% real

METRIC DEFINITION

TP / (TP+FP)

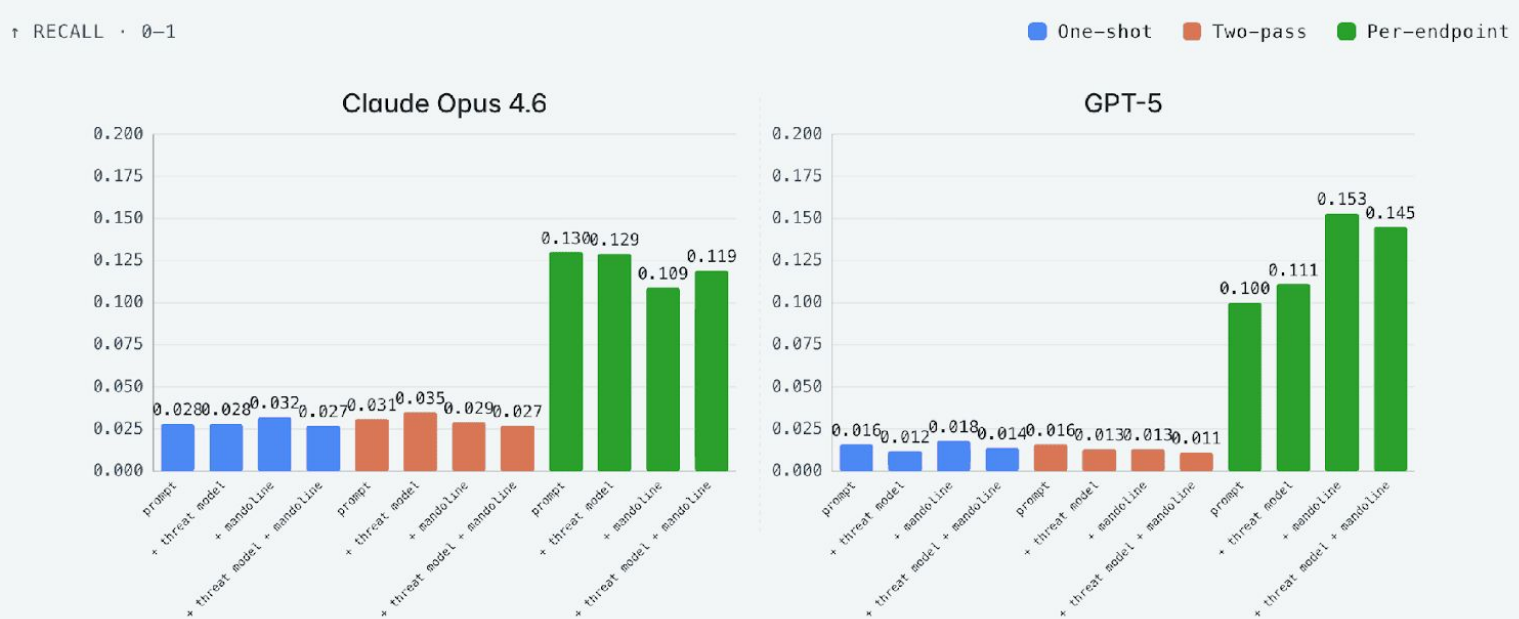
HOW IT'S COMPUTED

SMALLER PROMPT = LESS ROOM TO HALLUCINATE

PER-ENDPOINT · STATS

Recall — did we find them all?

If discovery is exhaustive, recall is bounded only by judgement.



% caught

METRIC DEFINITION

TP / (TP+FN)

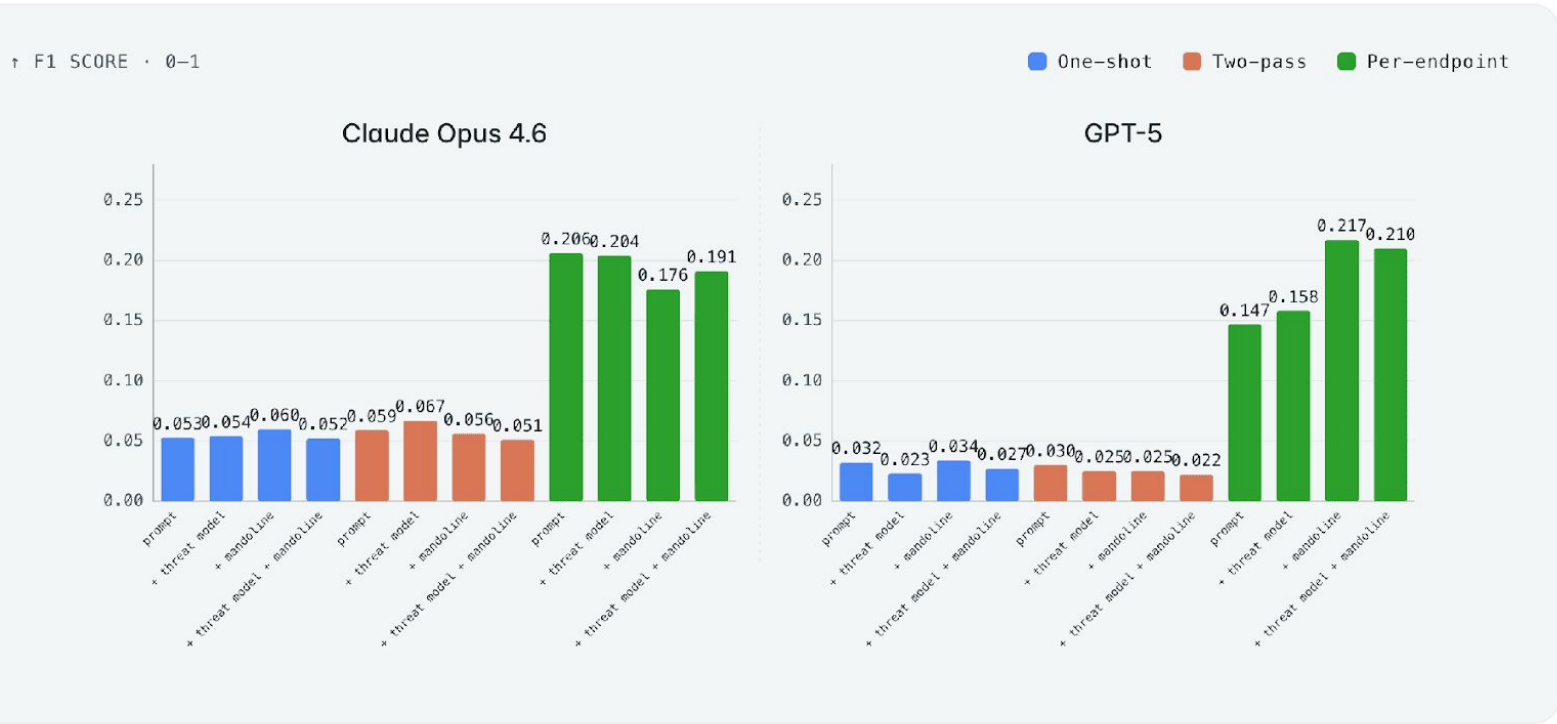
HOW IT'S COMPUTED

EVERY ROUTE SEEN — IF DISCOVERY WAS COMPLETE

PER-ENDPOINT · STATS

F_1 — the combined score.

The third strategy on the same axis. How does coverage trade for precision?



F_1

METRIC DEFINITION

$2 \cdot P \cdot R$

$P+R$

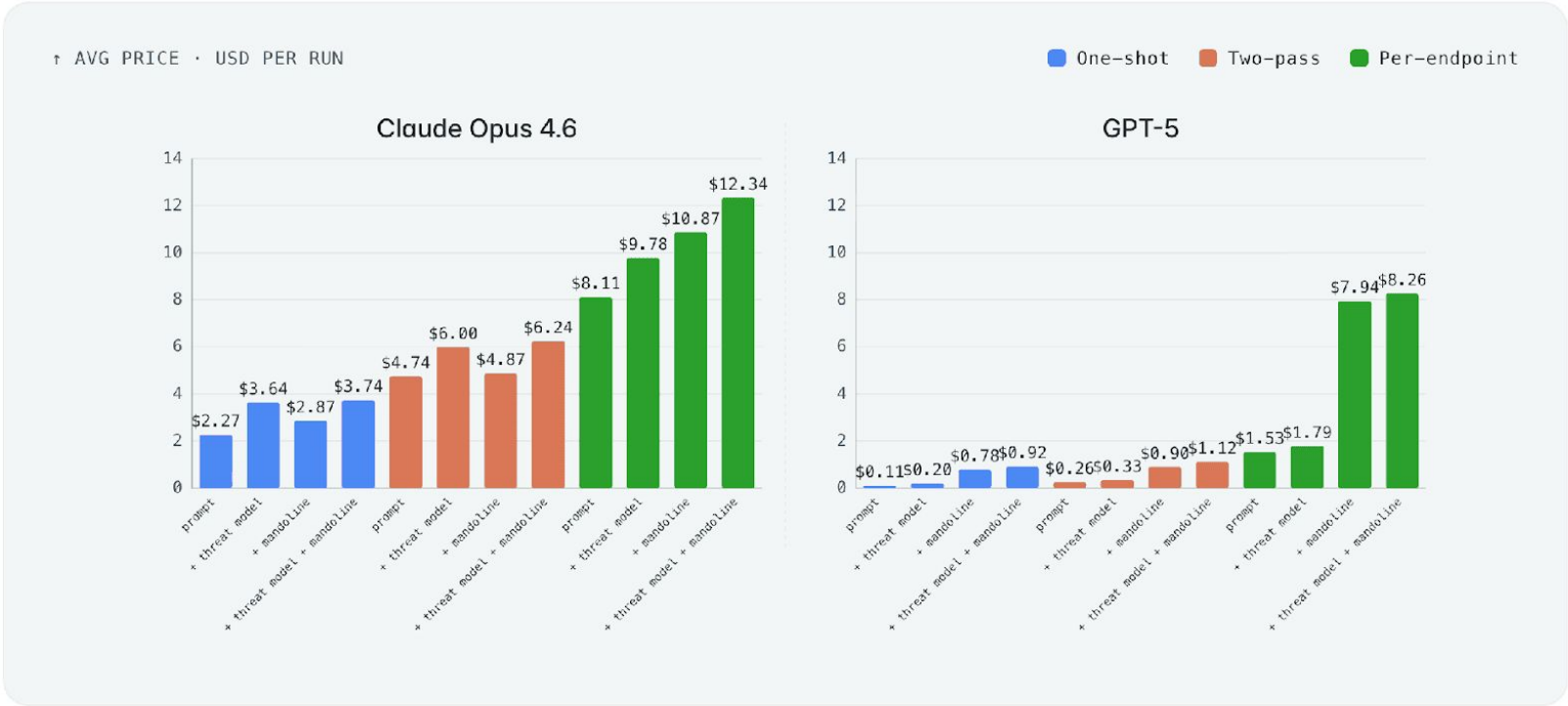
HOW IT'S COMPUTED

PUNISHES IMBALANCE BETWEEN PRECISION & RECALL

PER-ENDPOINT · STATS

Average price per run.

N batches = N billable calls. Cost scales with route count.



$/run

METRIC DEFINITION

N · $/batch

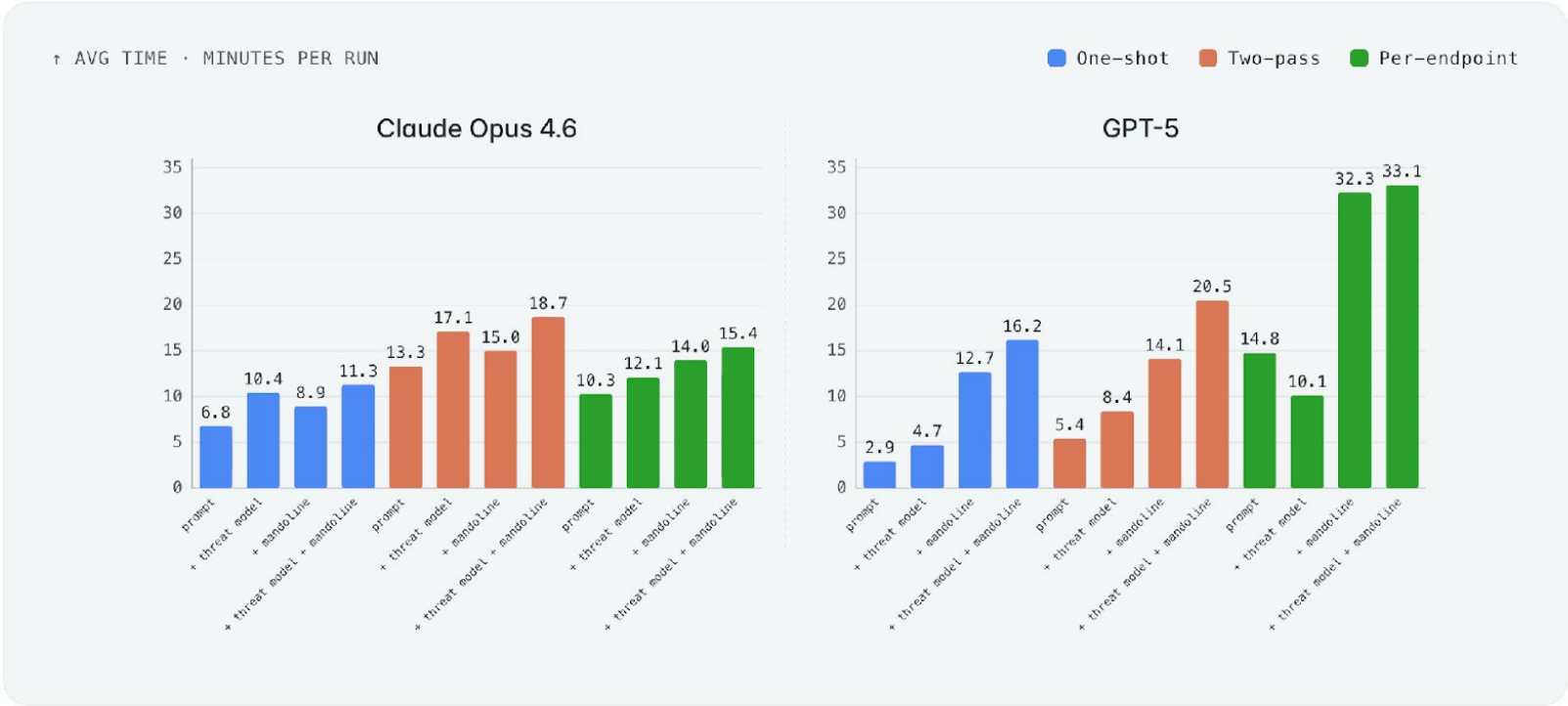
HOW IT SCALES

PARALLELISABLE — BUT YOU STILL PAY FOR EVERY BATCH

PER-ENDPOINT · STATS

Average time per run.

Many batches means many calls. Run them in parallel — or wait.



min/run

METRIC DEFINITION

batches · latency

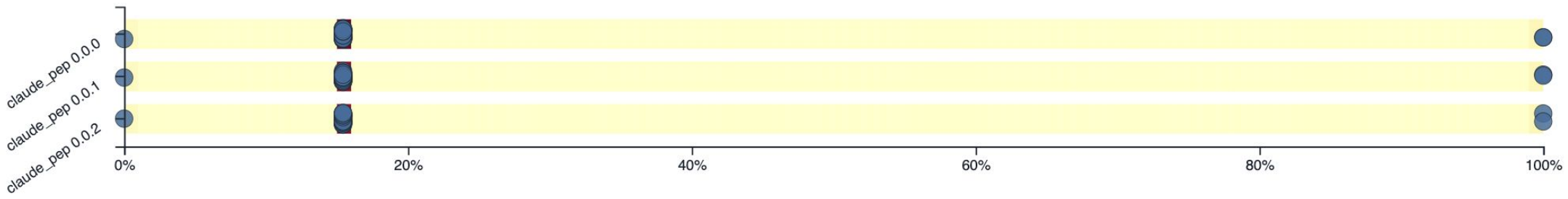
HOW IT SCALES

CONCURRENCY HELPS · RATE LIMITS DO NOT

PROMPT · STATS

Non-determinism — same batches, same findings?

↑ FINDING-SET OVERLAP · % ACROSS RUNS



PER-ENDPOINT · WRAP

What we learned

THE COST

- Bad news.

Takes much longer to scan each endpoint independently.

Costs more — every batch is its own billable call.

Noisier output overall — more findings means more to triage.

THE PAYOFF

+ Good news.

Finds a lot more vulnerabilities than the whole-repo passes.

Rather more deterministic — drift less.

The only way to be confident you scanned everything.

PART 5

Conclusions

CONCLUSIONS · 01 / 06

Back to the questions we asked.

01 · STRATEGY

What are the possible strategies for finding IDORs using LLMs?

→ We looked at three — one-shot, two-pass, and per-endpoint.

02 · ACCURACY

What are the false-positive and false-negative rates?

→ Numbers vary by strategy and model — no single answer.

03 · DETERMINISM

Do you get the same results every time you run it?

→ Also varies by strategy and model — smaller prompts drift less.

04 · COST

And — how much does this actually cost to run?

→ Same story: depends on the strategy and the model you pick.

CONCLUSIONS · 02 / 06

The most complete & precise.

ROUTES/USERS.PY 5/5

- /users/<id>
- /users/<id>/profile
- /users/<id>/avatar
- /users/<id>/pos…
- /users/me/settings

ROUTES/ORDERS.PY 5/5

- /orders/<id>
- /orders/<id>/items
- /orders/<id>/cancel
- /orders/<id>/refund
- /orders/<id>/sh…

ROUTES/DOCUMENTS.PY 5/5

- /documents/<id>
- /documents/<id>/share
- /attachment/<id>
- /attachment/<id>/preview
- /folders/<id>

ROUTES/MESSAGES.PY 4/4

- /messages/<id>
- /messages/<id>/reply
- /threads/<id>
- /threads/<id>/leave

ROUTES/BILLING.PY 4/4

- /invoices/<id>
- /invoices/<id>/pdf
- /payments/<id>
- /cards/<id>

ROUTES/ADMIN.PY 5/5

- /admin/users/<id>
- /admin/orgs/<id>
- /admin/orgs/<id>/keys
- /admin/audit/<id>
- /admin/flags

100% COVERED

THE STRATEGY

Scanning per endpoint wins the headline metrics.

↑ TP
PRECISION

↑ F_1
COMBINED

↑ Recall
COVERAGE

USE IT WHEN

It's important to get the full coverage of the source code — and you're willing to pay for it.

CONCLUSIONS · 03 / 06

The cheapest solution.

TERM · 01

One LLM prompt.

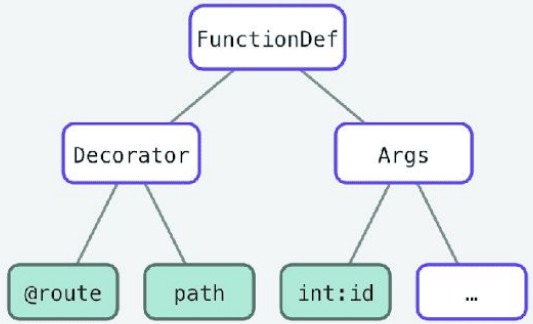
```
# single prompt
find IDORs please :)
# in this repo
```

- One call · scoped task

+

TERM · 02

AST tooling.



- Structural primitives, no tokens

=

RESULT

Good enough results.

- Less noise, fewer dollars

USE IT IF YOU WANT TO FIND SOMETHING AND DO IT CHEAPLY · WITH LESS NOISE

CONCLUSIONS · 04 / 06

Already have your referenceable objects? Turn them into rules.

INPUT Referenceable objects

01 · DB LOOKUP BY ID

```
# orders fetched by primary key
cursor.execute("SELECT * FROM orders"
  " WHERE id = ?", (order_id,))
```

02 · ID PASSED TO API

```
# payment_id straight to provider
requests.get(f"/v1/payments/{payment_id}")
```

03 · ID → FILE PATH

```
# file_id concatenated into a path
send_file(f"/uploads/{file_id}.pdf")
```

COMPILE TO SEMGREP

OUTPUT idor-missing-authz.yml

```
rules:
  - id: idor-missing-ownership-check
    message: Resource fetched by user-controlled ID
             without an ownership check.
    severity: WARNING
    languages: [python]
    pattern-either:
      # 01 · DB lookup by user-controlled id
      - pattern: |
          $MODEL.query.get($ID)
      - pattern: |
          cursor.execute("... WHERE id = ?", ($ID,))
      # 02 · id forwarded to external api
      - pattern: |
          requests.$M(f"...{$ID}...", ...)
      # 03 · id concatenated into a file path
      - pattern: |
          send_file(f"...{$ID}...")
```

KNOW YOUR OBJECTS · SHIP SEMGREP RULES · REUSE FOREVER · ZERO LLM CALLS AT SCAN TIME

CONCLUSIONS · 05 / 06

Different models work with tools differently.

TOP MODEL · NO TOOLS

The frontier flagship on its own.

Pricey, slow, reasoning-only — the obvious default.

O	GPT-5 REASONING · ONE-SHOT	0.56 F1	$$$ COST

BASELINE — THE NUMBER TO BEAT WITHOUT SPENDING MORE.

vs

MID-TIER + AST TOOLS

The right model, fed structure.

Models built for tool-use punch above their tier when given AST context.

A	Claude Sonnet 4.5 + AST TOOL-NATIVE · AGENTIC	0.58 F1	$ COST
G	Gemini 2.5 Pro + AST FUNCTION-CALLING · FAST	0.54 F1	$ COST
H	Claude Haiku 4.5 + AST CHEAP · TOOL-FLUENT	0.51 F1	¢ COST

PICK A MODEL BUILT FOR TOOLS · MATCH THE HEADLINE F_1 AT A FRACTION OF THE PRICE.

CONCLUSIONS · 06 / 06

Final thoughts

01 LLMs show real potential — they genuinely find IDORs in real code. ✓

02 However — the models are still prone to false positives. !

03 Human in the loop is still needed for triage and judgement. ◐

04 Combining LLMs with tools and the right context is the way to find vulnerabilities efficiently. ★

References

- https://semgrep.dev/blog/2025/can-llms-detect-idors-understanding-the-boundaries-of-ai-reasoning/
- https://semgrep.dev/blog/2025/finding-vulnerabilities-in-modern-web-apps-using-claude-code-and-openai-codex/
- https://productsecurity.ghost.io/semgrep-llm-for-idor-detection-fewer-false-positives-by-scoping-the-review/
- https://youtu.be/1sd26pWhfmg
- https://youtu.be/gINAtzdccts

https://ermilov.dev/bsides-calgary-26

Thank you

Semgrep