



AUGUST 6-7, 2025
MANDALAY BAY / LAS VEGAS

Thinking Outside the Sink

How Tree-of-AST
Redefines the Boundaries
of Dataflow Analysis



Alexander Zyuzin

Ruikai Peng

CONTENTS

01

ALGORITHMIC

02

TECHNICAL

03

CONCLUSIONS

1PICTURE WORTH 1000 WORDS...



saylese 16.10.2024, 16:46 **My teacher**

i have a student who's very interested in security stuff and he's looking for a mentor for a project he's working on (idk what i it is). <https://retr0.blog/> would you be interested in discussing with him? i don't think he would expect much time from you

Retr0's Register

Retr0's Threat Research



greyroad 16.10.2024, 18:28 **Me**

«I've had the incredible opportunity to identify 22 CVEs,»

Does he need a mentor for stealing nuclear codes?

I mean sure



WHAT IS THE ALGORITHMIC
PART OF OUR SOLUTION?

THE PAPER I'VE READ

arXiv:2305.10601v2 [cs.CL] 3 Dec 2023

Tree of Thoughts: Deliberate Problem Solving with Large Language Models

Shunyu Yao
Princeton University Dian Yu
Google DeepMind Jeffrey Zhao
Google DeepMind Izak Shafrazi
Google DeepMind

Thomas L. Griffiths
Princeton University Yuan Cao
Google DeepMind Karthik Narasimhan
Princeton University

Abstract

Language models are increasingly being deployed for general problem solving across a wide range of tasks, but are still confined to token-level, left-to-right decision-making processes during inference. This means they can fall short in tasks that require exploration, strategic lookahead, or where initial decisions play a pivotal role. To surmount these challenges, we introduce a new framework for language model inference, “Tree of Thoughts” (ToT), which generalizes over the popular “Chain of Thought” approach to prompting language models, and enables exploration over coherent units of text (“thoughts”) that serve as intermediate steps toward problem solving. ToT allows LMs to perform deliberate decision making by considering multiple different reasoning paths and self-evaluating choices to decide the next course of action, as well as looking ahead or backtracking when necessary to make global choices. Our experiments show that ToT significantly enhances language models’ problem-solving abilities on three novel tasks requiring non-trivial planning or search: Game of 24, Creative Writing, and Mini Crosswords. For instance, in Game of 24, while GPT-4 with chain-of-thought prompting only solved 4% of tasks, our method achieved a success rate of 74%. Code repo with all prompts: <https://github.com/princeton-nlp/tree-of-thought-lm>

1 Introduction

Originally designed to generate text, scaled-up versions of language models (LMs) such as GPT [25, 26, 1, 23] and PaLM [5] have been shown to be increasingly capable of performing an ever wider range of tasks requiring mathematical, symbolic, commonsense, and knowledge reasoning, perhaps surprising that underlying all this progress is still the original autoregressive mechanism generating text, which makes token-level decisions one by one and in a left-to-right fashion. Is such a simple mechanism sufficient for a LM to be built toward a general problem solver? If not, what problems would challenge the current paradigm, and what should be alternative mechanisms?

The literature on human cognition provides some clues to answer these questions. Research on “dual process” models suggests that people have two modes in which they engage with decisions – a fast, automatic, unconscious mode (“System 1”) and a slow, deliberate, conscious mode (“System 2”) [30, 31, 16, 15]. These two modes have previously been connected to a variety of mathematical models used in machine learning. For example, research on reinforcement learning in humans and other animals has explored the circumstances under which they engage in associative “model free” learning or more deliberative “model based” planning [7]. The simple associative token-level choices of LMs are also reminiscent of “System 1”, and thus might benefit from augmentation by a more deliberate “System 2” planning process that (1) maintains and explores diverse alternatives for current

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

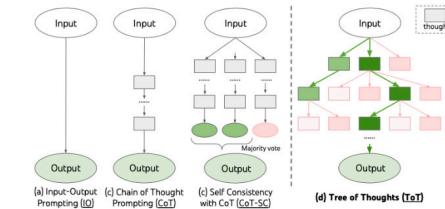


Figure 1: Schematic illustrating various approaches to problem solving with LLMs. Each rectangle box represents a **thought**, which is a coherent language sequence that serves as an intermediate step toward problem solving. See concrete examples of how thoughts are generated, evaluated, and searched in Figures 2–6.

choices instead of just picking one, and (2) evaluates its current status and actively looks ahead or backtracks to make more global decisions.

To design such a planning process, we return to the origins of artificial intelligence (and cognitive science), drawing inspiration from the planning processes explored by Newell, Shaw, and Simon starting in the 1950s [21, 22]. Newell and colleagues characterized problem solving [21] as search through a combinatorial problem space, represented as a tree. We thus propose the Tree of Thoughts (ToT) framework for general problem solving with language models. As Figure 1 illustrates, while existing methods (detailed below) sample continuous language sequences for problem solving, ToT actively maintains a tree of thoughts, where each **thought** is a coherent language sequence that serves as an intermediate step toward problem solving (Table 1). Such a high-level semantic unit allows the LM to self-evaluate the progress different intermediate thoughts make towards solving the problem through a deliberate reasoning process that is also instantiated in language (Figures 2–6). This implementation of search heuristics via LM self-evaluation and deliberation is novel, as previous search heuristics are either programmed or learned. Finally, we combine this language-based capability to generate and evaluate diverse thoughts with search algorithms, such as breadth-first search (BFS) or depth-first search (DFS), which allow systematic exploration of the tree of thoughts with lookahead and backtracking.

Empirically, we propose three new problems that challenge existing LM inference methods even with the state-of-the-art language model, GPT-4 [23]: Game of 24, Creative Writing, and Crosswords (Table 1). These tasks require deductive, mathematical, commonsense, lexical reasoning abilities, and a way to incorporate systematic planning or search. We show ToT obtains superior results on all three tasks by being general and flexible enough to support different levels of thoughts, different ways to generate and evaluate thoughts, and different search algorithms that adapt to the nature of different problems. We also analyze how such choices affect model performances via systematic ablations and discuss future directions to better train and use LMs.

2 Background

We first formalize some existing methods that use large language models for problem-solving, which our approach is inspired by and later compared with. We use p_θ to denote a pre-trained LM with parameters θ , and lowercase letters x, y, z, s, \dots to denote a language sequence, i.e. $x = (x[1], \dots, x[n])$ where each $x[i]$ is a token, so that $p_\theta(x) = \prod_{i=1}^n p_\theta(x_i)$. We use uppercase letters S, \dots to denote a collection of language sequences.

Input-output (IO) prompting is the most common way to turn a problem input x into output y with LM: $y \sim p_\theta(y | \text{prompt}_{IO}(x))$, where $\text{prompt}_{IO}(x)$ wraps input x with task instructions and/or few-shot input-output examples. For simplicity, let us denote $p_\theta^{\text{prompt}}(\text{output} | \text{input}) = p_\theta(\text{output} | \text{prompt}(\text{input}))$, so that IO prompting can be formulated as $y \sim p_\theta^{\text{IO}}(y | x)$.

TREE-OF-THOUGHT

Goal:

Make 24 using four given numbers and basic math operations

Rules:

Given: 4 numbers
(e.g., 8, 3, 8, 3)

Use: +, -, ×, ÷
and parentheses

Must:

Use each number exactly once

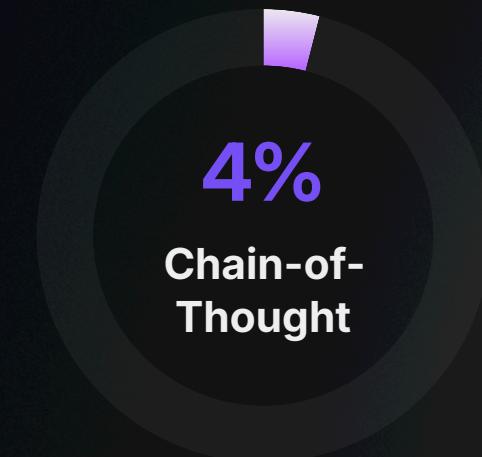
Target:

Final result = 24

WHY IT'S SO CHALLENGING FOR AI?

1. Huge search space of combinations
2. Early mistakes lead to dead ends
3. Requires strategic planning & backtracking

RESULTS



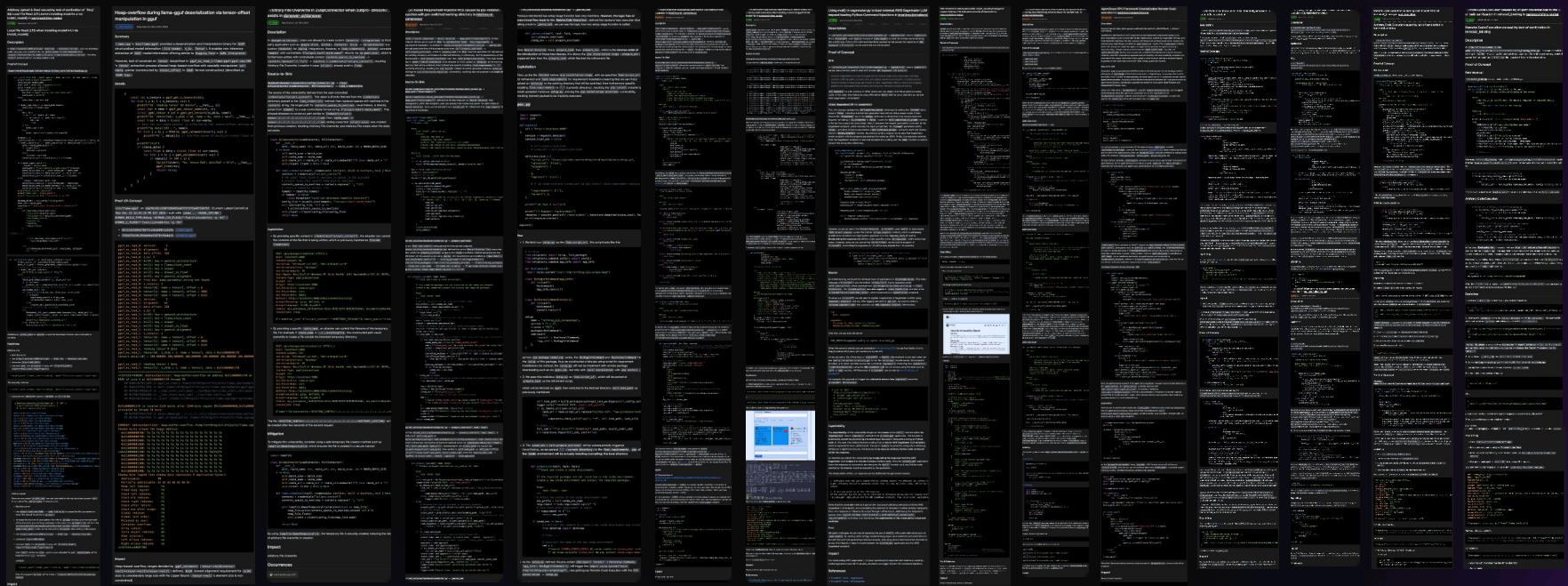
HOW CAN A GENERATIVE APPROACH, BE A GAME CHANGER FOR TAINT ANALYSIS FRAMEWORK?

INSPIRATION: HUNTR STORY

51 report

18 CVEs

Transformers,
Tensorflow,
Llama.cpp...



- ① Heap-overflow during llama-gguf deserialization via tensor-offset manipulation in GGUf
- ② Llama.cpp (GGML) RPC Heap Overflow to Remote-Code Execution in GGML
- ③ AgentScope RPC Framework Deserialization Remote-Code execution in modelscope/agentscope in modelscope/ag...
- ④ Remote-Code Execution via dynamic module importing | Arbitrary File Overwrite | Pip injection at "/ap/tools" in com...
- ⑤ LFI at "/api/download" in composihq/composio
- ⑥ RPC Framework arbitrary file exposure in modelscope/agentscope
- ⑦ AgentScope RPC Framework Deserialization Remote-Code execution in modelscope/agentscope
- ⑧ workflowdag -> sanitize_node_data Remote-code execution in modelscope/agentscope
- ⑨ Path traversal in '/read-examples' in modelscope/agentscope
- ⑩ Local file inclusion via "/api/file" in modelscope/agentscope
- ⑪ Arbitrary File Overwrite in ZulipConnector when zipuprc- directory exists in danswer-a/danswer
- ⑫ Command Injection Bypass via escaped '\xa0' in intel/neural-compressor
- ⑬ Vulnerable pdfjs-dist imported in Gradio Guides results JavaScript Injection in gradio-app/gradio

- ⑭ Logic-based Requirement-Injection RCE caused by pip installation injection with pre-switched working-directory in ...
- ⑮ Denial-of-Service due to unlimited Taskworkers in intel/neural-compressor
- ⑯ REDACTED in intel/neural-compressor
- ⑰ Command Injection bypassing sanitization via Internal Field Separator in intel/neural-compressor
- ⑱ SQL Injection in /task/subm/ in intel/neural-compressor
- ⑲ SymLink-Based Model Theft & File Leaking during extracting in deepgalvany/qj
- ⑳ Invalid fix on previous published Unrestricted File Upload Path On Windows in lightning-ai/lightning-lightning
- ㉑ Vulnerable llama-cpp-python dependency in loimis-webui allow arbitrary code execution when loading a external q...
- ㉒ Using eval() in sagemaker.py to load external AWS Sagemaker LLM request leading Python Command Injections in ...
- ㉓ /api/create-project's 'project_name' is vulnerable to path traversal in stilton/devika
- ㉔ LFI in /api/get_browser_snapshot() in stilton/devika
- ㉕ Lack of limitation on config editing resulting Arbitrary File Leakage / File Writing in stilton/devika
- ㉖ Arbitrary Folder/File leakages via Path Traversals in "/api/download-project" in stilton/devika
- ㉗ Chat is vulnerable to Stored XSS in stilton/devika
- ㉘ Path traversal in native personality "cyber_security/codeguard" causes Arbitrary File leak and overwrite of directo...
- ㉙ SSRF in "/scrape_and_save" allow the use of arbitrary scheme/protocol in pariseo/loimis-webui
- ㉚ Path Traversal in '/save_settings' bypassing existing patches causing RCEs in pariseo/loimis-webui

- ㉛ LFI in '/save_settings' -> binding_name causes Remote Code Execution in pariseo/loimis-webui
- ㉜ Python Command Injections when loading config files when listing & listing docs & listing external repos via 'pip...
- ㉝ Path Traversal in '/switch_personal_path' cause sensitive config leakage & Arbitrary Upload & Overwrites in parise...
- ㉞ RCE when loading Huggingface Hub 'tools' in 'src/transformers/tools/base.py' -> 'load_tool' in huggingface/transfor...
- ㉟ 'collector/latu/usm/jc' file inclusion in IP is loimiswebui can be bypassed in mthpiq-labs/oxyng/lm
- ㉟ Remote Code Execution due to LFI in '/reinstall_extension' in pariseo/loimis-webui
- ㉟ Remote Code Execution due to LFI in '/install_extension' in pariseo/loimis-webui
- ㉟ URL in '/save_settings' -> extensions causes Remote Code Execution in pariseo/loimis-webui
- ㉟ LFI in '/save_settings' -> binding_name causes Remote Code Execution in pariseo/loimis-webui
- ㉟ Remote Code Execution due to the lack of sanitization of key 'extensions' in pariseo/loimis-webui
- ㉟ Remote Code Execution when switching bindings due to lack of sanitization of binding name in pariseo/loimis-webui
- ㉟ Remote Code Execution caused by lack of sanitization in '/uninstall_binding' in pariseo/loimis
- ㉟ Remote File Read (LFI) when 'Copy to custom persons folder for editing' in pariseo/loimis-webui
- ㉟ Local File Read (LFI) when 'Copy to custom persons folder for editing' in pariseo/loimis-webui
- ㉟ SSRF when installing model in UI via install_model() in pariseo/loimis-webui
- ㉟ Arbitrary Upload & Read caused by lack of sanitization of 'file[]' & Local File Read (LFI) when installing model in ...
- ㉟ Two Path Traversal leading Remote Code Execution in pariseo/loimis-webui
- ㉟ SSRF add_webpage() when fetching weblink for discussions via 'scrape_and_save()' in pariseo/loimis-web...
- ㉟ Using eval() in sagemaker.py to load external AWS Sagemaker LLM request leading Python Command Inj...
- ㉟ Using eval() in sagemaker.py to load external AWS Sagemaker LLM request leading Python Command Inj...
- ㉟ Insecure and deprecated function mktime() in tensorflow/tensorflow
- ㉟ Insecure and deprecated function mktime() in tensorflow/tensorflow
- ㉟ Transformers has a Deserialization of Untrusted Data vulnerability in huggingface/transformers
- ㉟ Transformers has a Deserialization of Untrusted Data vulnerability in huggingface/transformers
- ㉟ Transformers has a Deserialization of Untrusted Data vulnerability in huggingface/transformers

INSPIRATION: HUNTR STORY

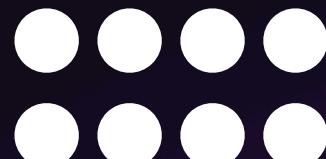
The first thing you do with a target, you run Bandit to see static point-of-interest:

Code Search, SAST tell us where the sinks are

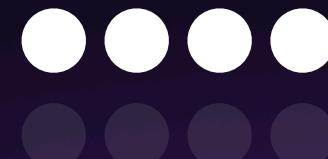
- There are **tons** of them!
- We spend **hours** tracking them to their sources.

e.g., modelscope/agentscope:

8 sinks identified per
SAST (Bandit) scan



...4 of them
reachable



... within only 2 of
them exploitable




References	Image
3 References	
30 References	
300 References	
30.000 References	

WHAT IS TAINT ANALYSIS. SINK? SOURCE?



Disney
**WHERE'S MY
WATER?** ?

"Sink"
*The place where meets
usually indicate dangerous
(e.g. eval, memcpy, pickle)*



"Source":
*The input of data
(e.g., endpoint,
argv, file)*

**WHEN YOU THINK IT
UPSIDE-DOWN,
IT'S MUCH MORE SIMPLE.**

Why?

- It's too **easy** to identify sinks
 - Avoids False positive & prioritize
 - ***Context relevance***
 - Potential Context
 - State Recovery
 - Complexity of conditional flow
(path explosion of executions)

SOURCE-TO-SINK

```
def run_app() -> None:
    demo.load()
    check_for_new_session(
        inputs=[uid],
        every=0.5,
    )

def check_for_new_session(uid: str) -> None:
    if uid not in gbl_signed_user:
        run_thread = threading.Thread(
            target=start_game,
            args=(uid,),
        )
        run_thread.start()

def start_game(uid: str) -> None:
    thread_local_data.uid = uid
    if script_path.endswith(".py"):
        import importlib.util
        spec = importlib.util.spec_from_file_location("script", script_path)
        config = load_config(spec)
    else:
        from agentscope.web.workstation.workf
        start_workflow(
            load_cfgpath,
            load_cfg,
        )
        config = load_config(script_path)
    main = lambda: start_workflow(config)

def start_workflow(config: dict) -> None:
    dag = build_dag(config)
    dag.run()

def build_dag(config: dict) -> ASDiGraph:
    for node_id, node_info in config.items():
        config[node_id] = sanitize_node_data(node_info)

ze_node_data(raw_info: dict) -> dict:
    info = copy.deepcopy(raw_info)
    value in copied_info["data"].get("args", {}).items():
        raw_info["data"]["args"][key]
        = eval(value)

    def build_dag(config: dict) -> ASDiGraph:
        for node_id, node_info in config.items():
            config[node_id] = sanitize_node_data(node_info)

    def start_workflow(config: dict) -> None:
        dag = build_dag(config)
        dag.run()

    def start_game(uid: str) -> None:
        thread_local_data.uid = uid
        if script_path.endswith(".py"):
            import importlib.util
            spec = importlib.util.spec_from_file_location("script", script_path)
            config = load_config(spec)
        else:
            from agentscope.web.workstation.workf
            start_workflow(
                load_cfgpath,
                load_cfg,
            )
            config = load_config(script_path)
        main = lambda: start_workflow(config)

    def check_for_new_session(uid: str) -> None:
        if uid not in gbl_signed_user:
            run_thread = threading.Thread(
                target=start_game,
                args=(uid,),
            )
            run_thread.start()

    def run_app() -> None:
        demo.load()
        check_for_new_session(
            inputs=[uid],
            every=0.5,
        )

def start_workflow(config: dict) -> None:
    if cfg_path:
        config = load_config(cfg_path)
    if not compiled_filename:
        start_workflow(config)
    else:
        if os.path.exists(compiled_filename):
            logger.info("Finished.")
        else:
            def main() -> None:
                if cfg_path:
                    config = load_config(cfg_path)
                if not compiled_filename:
                    start_workflow(config)
                else:
                    compile_workflow(config, compiled_filename)

            def compile_workflow(config: dict, compiled_filename: str = "main.py") -> None:
                dag = build_dag(config)
                dag.compile(compiled_filename)

            logger.info("Finished.")

def compile_workflow(config: dict, compiled_filename: str = "main.py") -> None:
    dag = build_dag(config)
    dag.compile(compiled_filename)

def main() -> None:
    if cfg_path:
        config = load_config(cfg_path)
    if not compiled_filename:
        start_workflow(config)
    else:
        compile_workflow(config, compiled_filename)
```

SINK-TO-SOURCE

ESSENCE OF THE TASKS STATEFUL-NESS, PRUNING?

The Pruning Process:

- Cut off unrelated / low-valued branches, to focus on the biggest fruit

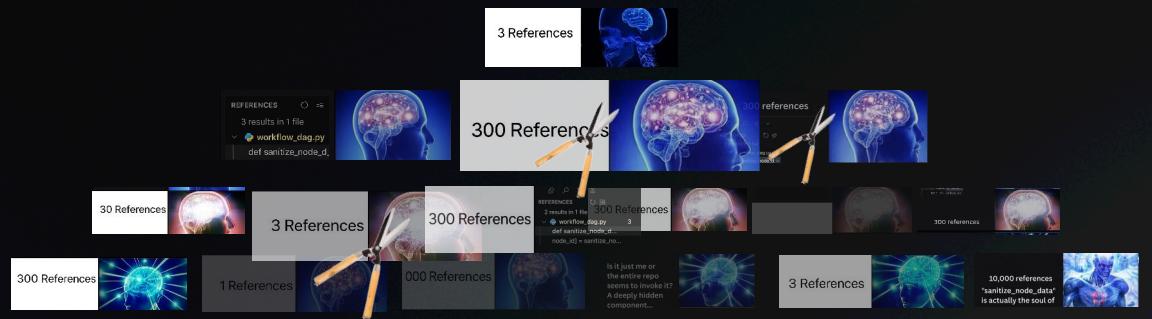
Statefulness:

- **dependency** of relationship between each nodes (states).
(Directed acyclic graph)

State-Recovery

- Recovering future states with both past states and potential states as contextual information,

TREE-OF-STATES?



```

def sanitize_node_data(raw_info: dict) -> dict:
    copied_info = copy.deepcopy(raw_info)
    for key, value in copied_info["data"].get("args", {}).items():
        raw_info["data"]["args"][key] = eval(value)

def build_dag(config: dict) -> ASDiGraph:
    for node_id, node_info in config.items():
        config[node_id] = sanitize_node_data(node_info)

def _convert_to_py( # type: ignore[no-untyped-def]
    content: str,
    **kwargs,
) -> Tuple:
    from agentscope.web.workstation.workflow_dag import build_dag
    dag = build_dag(config)
    try:
        cfg = json.loads(content)
        return "True", build_dag(cfg).compile(**kwargs)
    except Exception as e:
        return "False", str(e)

@app.route("/convert-to-py", methods=['POST'])
def _convert_config_to_py() -> Response:
    content = request.json.get("data")
    status, py_code = _convert_to_py(content)

    def start_workflow(config: dict) -> None:
        dag = build_dag(config)
        dag.compile()
        dag.compile()

        def main() -> None:
            if cfg_path:
                config = load_config(cfg_path)
                if os.path.exists(compiled_filename):
                    start_workflow(config)
                    load_config()
                else:
                    if os.path.exists(script_path):
                        config = load_config(script_path)
                        main = lambda: start_workflow(config)
                    else:
                        config = load_config(cfg_path)
                        if not compiled_filename:
                            start_workflow(config)
                        else:
                            if os.path.exists(compiled_filename):
                                start_workflow(config)
                            else:
                                compile_workflow(config, compiled_filename)

        config = load_config(script_path)
        main = lambda: start_workflow(config)

        def check_for_new_session(uuid: str) -> None:
            if uid not in glib_signed_user:
                run_thread = threading.Thread(
                    target=start_gae,
                    args=(uid,),
                )
                run_thread.start()

        def start_gae() -> None:
            demo.load()
            check_for_new_session(
                inputs=uuid,
                every=0.5,
            )

        def run_gae() -> None:
            demo.load()
            check_for_new_session(
                inputs=uuid,
                every=0.5,
            )
    
```

SINK-TO-SOURCE?

TREE-OF-AST

1. Diffuse taint graph from on the **sink**
2. Traverse **sink-to-source**
 - **vote** if current state is **source**
 - **vote & value** next parallel states for most possible **source-leading node**
 - depth, **lookaheads**
 - **rewind**, stateful-task

```

@app.route("/convert-to-py", methods=["POST"])
def _convert_config_to_py() -> Response:
    content = request.json.get("data")
    status, py_code = _convert_to_py(content)

    def _convert_to_py( # type: ignore[no-untyped-def]
        content: str,
        **kwargs,
    ) -> Tuple:
        from agentscope.web.workstation.workflow_dag import build_dag
        try:
            cfg = json.loads(content)
            return "True", build_dag(cfg).compile(**kwargs)
        
```

```

def run_app() -> None:
    demo.load(
        check_for_new_session,
        inputs=[uid],
        every=0.5,
    )

def check_for_new_session(uid: str) -> None:
    if uid not in glb_signed_user:
        run_thread = threading.Thread(
            target=start_game,
            args=(uid,),
        )
        run_thread.start()

def start_game(uid: str) -> None:
    thread_local_data.uid = uid
    if script_path.endswith(".py"):
        main = import_function.from_path(script_path)
    elif script_path.endswith(".json"):
        from agentscope.web.workstation.workflow_dag import build_dag
        config = load_config(script_path)
        main = lambda: build_dag(config)

    def mainf() -> None:
        if cfg_path:
            config = load_config(cfg_path)
            if not compiled_filename:
                start_workflow(config)
            else:
                if os.path.exists(compiled_filename):
                    logger.info("Workflow already compiled, skipping compilation")

    mainf()
    load_config(
        start_workflow,
        config,
    )

def load_config(path: str) -> dict:
    if path.endswith(".py"):
        return import_function.from_path(path)
    elif path.endswith(".json"):
        return json.load(open(path))
    else:
        raise ValueError(f"Unsupported file extension: {path}")

def build_dag(config: dict) -> ASDiGraph:
    for node_id, node_info in config.items():
        config[node_id] = sanitize_node_data(node_info)

def start_workflow(config: dict):
    dag = build_dag(config)
    dag.run()

def sanitize_node_data(raw_info: dict) -> dict:
    copied_info = copy.deepcopy(raw_info)
    for key, value in copied_info["data"].items():
        if key == "args":
            copied_info["data"][key] = eval(value)
        else:
            copied_info["data"][key] = sanitize_node_data(value)

    return copied_info

```

```

def main() -> None:
    if cfg_path:
        config = load_config(cfg_path)
        if not compiled_filename:
            start_workflow(config)
        else:
            compile_workflow(config, compiled_filename)

def compile_workflow(config: dict, compiled_filename: str = "main.py"):
    dag = build_dag(config)
    dag.compile(compiled_filename)

    logger.info("Finished.")

```

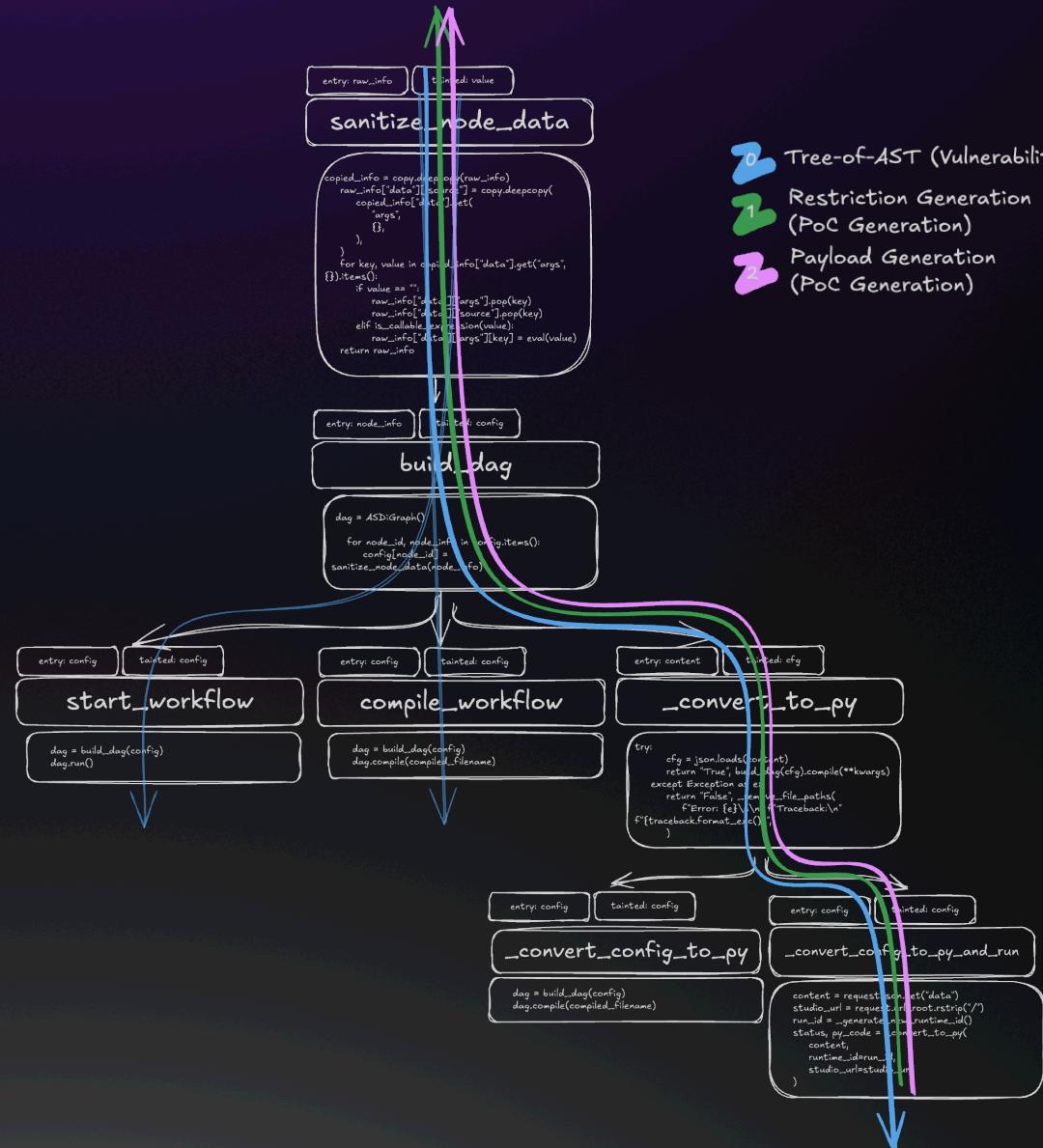
PAYOUT GENERATION

1. Reverse traverse source-to-sink

- Nested-restriction tags to **semantically describe the constraints** from the slice.
- Make up for the part we deliberately neglected
- "*Intuition*" for solvers in a **pruned-and-limited slice**

2. Traverse source-to-sink again

- Constraints with *SMT Solvers for model-generations, linearly **solving** the constraints into payloads

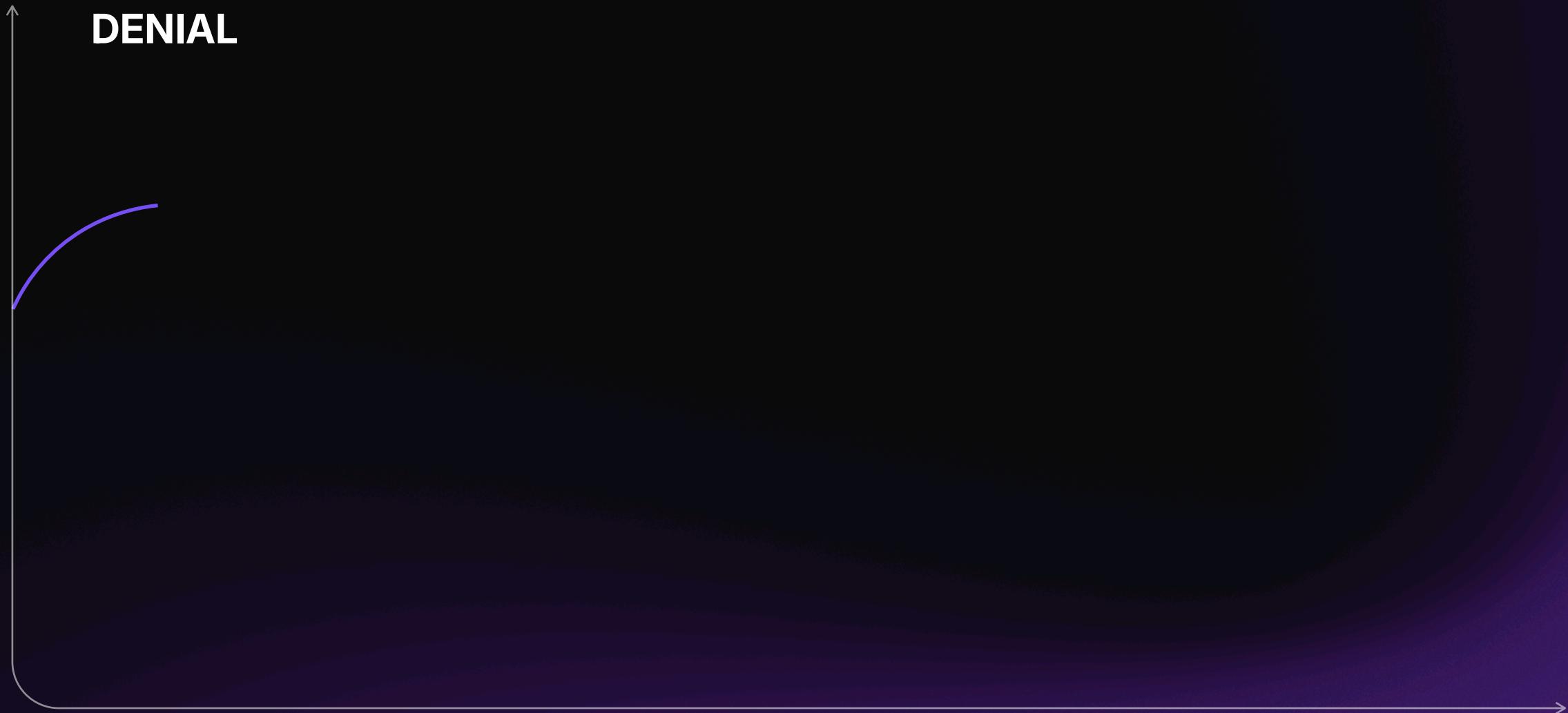




Technical side of our approach

THE PROBLEM WE NEED TO SOLVE: INTERNAL PROGRAM REPRESENTATION

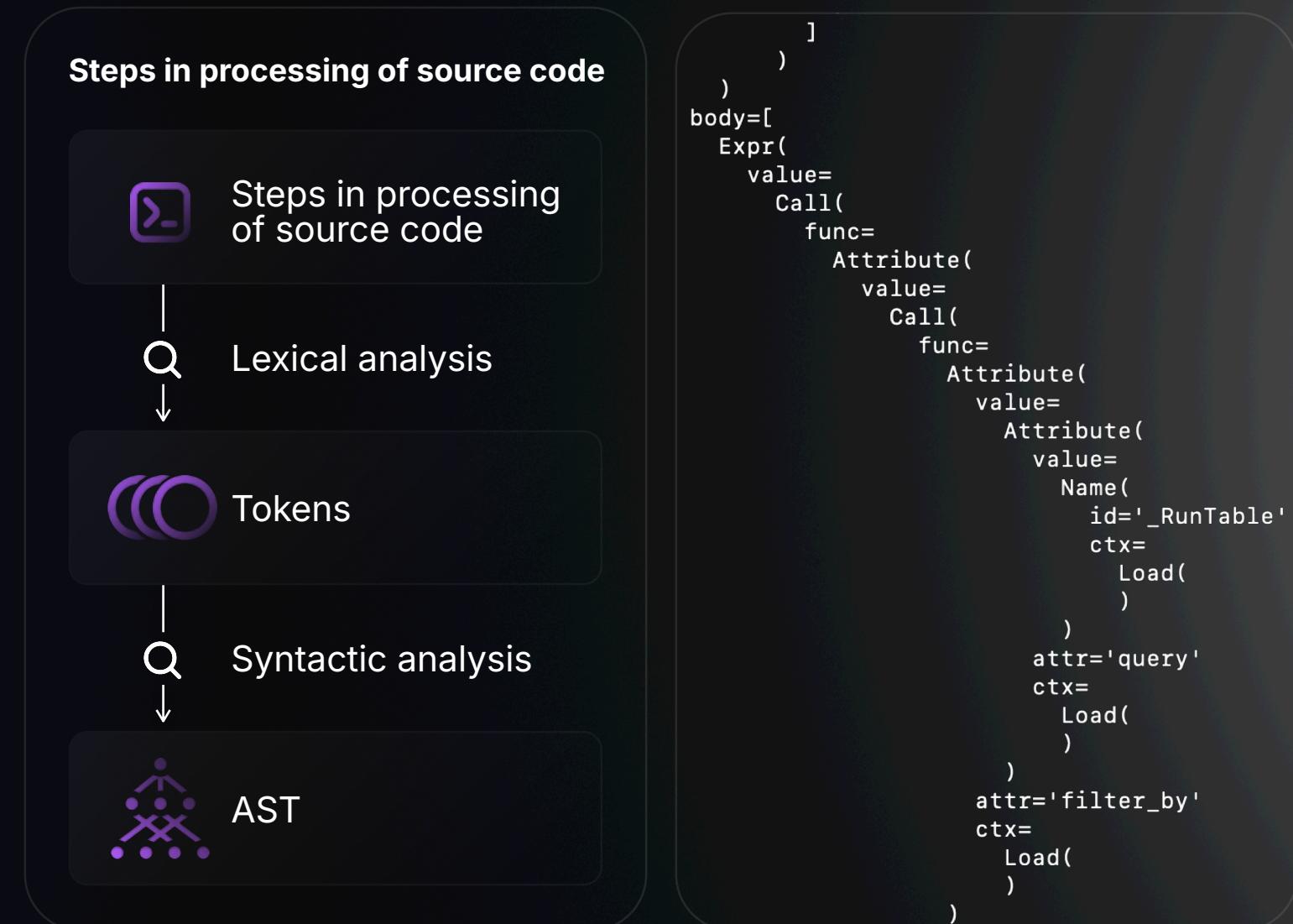
STAGES OF PROBLEM ACCEPTANCE



FIRST STEP: AST

An Abstract Syntax Tree

- Code Structure as Data
- Foundation for Security Analysis



AST OR NO AST?

Intuitively (as simple result):

```
>>> print(ast.dump(ast.parse('a if b else c', mode='eval'), indent=4))
Expression(
    body=IfExp(
        test=Name(id='b', ctx=Load()),
        body=Name(id='a', ctx=Load()),
        orelse=Name(id='c', ctx=Load())))
```

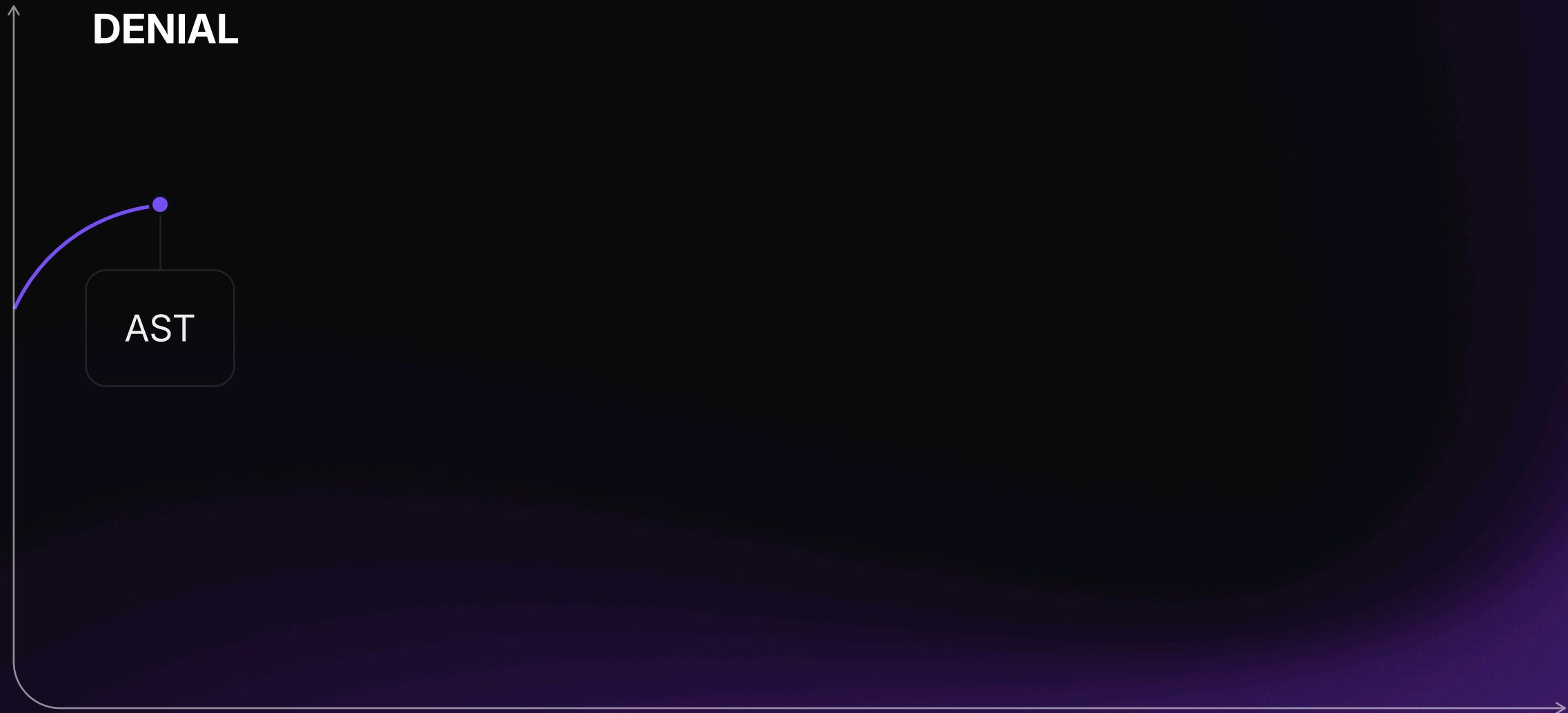
AST OR NO AST?

Intuitively (as simple result):

```
>>> print(ast.dump(ast.parse('a if b else c'), mode='eval'), indent=4))  
Expression(  
    body=IfExp(  
        test=Name(id='b', ctx=Load()),  
        body=Name(id='a', ctx=Load()),  
        orelse=Name(id='c', ctx=Load())))
```



STAGES OF PROBLEM ACCEPTANCE



PROBLEMS OF ASTS

Note: ASTs remain useful for static analysis despite these challenges

LANGUAGE-SPECIFIC PARSING

Must implement custom parser for each language

CROSS-FILE ANALYSIS

SCOPE RESOLUTION

IMPORT HANDLING

COMPLEXITY VS. BENEFIT

PROBLEMS OF ASTS

Note: ASTs remain useful for static analysis despite these challenges

LANGUAGE-SPECIFIC PARSING

CROSS-FILE ANALYSIS

Tracking definitions and usage across files (e.g., function in a.py used in b.py)

SCOPE RESOLUTION

IMPORT HANDLING

COMPLEXITY VS. BENEFIT

PROBLEMS OF ASTS

Note: ASTs remain useful for static analysis despite these challenges

LANGUAGE-SPECIFIC PARSING

CROSS-FILE ANALYSIS

SCOPE RESOLUTION

Managing variable shadowing and name conflicts

IMPORT HANDLING

COMPLEXITY VS. BENEFIT

PROBLEMS OF ASTS

Note: ASTs remain useful for static analysis despite these challenges

LANGUAGE-SPECIFIC PARSING

CROSS-FILE ANALYSIS

SCOPE RESOLUTION

IMPORT HANDLING

Resolving module dependencies and import paths

COMPLEXITY VS. BENEFIT

PROBLEMS OF ASTS

Note: ASTs remain useful for static analysis despite these challenges

LANGUAGE-SPECIFIC PARSING

CROSS-FILE ANALYSIS

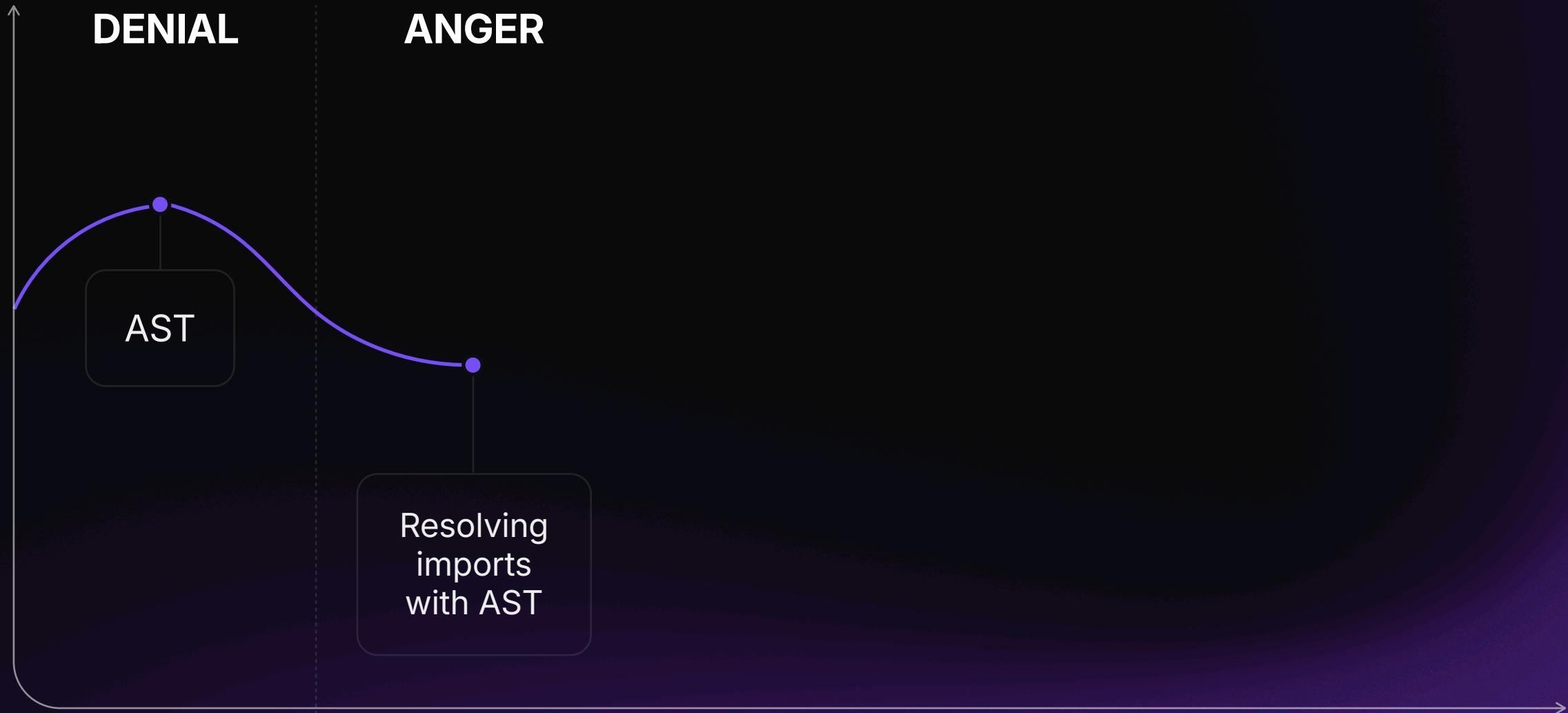
SCOPE RESOLUTION

IMPORT HANDLING

COMPLEXITY VS. BENEFIT

High implementation effort for basic static analysis tasks

STAGES OF PROBLEM ACCEPTANCE



THE SOLUTION? STACK GRAPHS!

Stack graphs are an open source framework

THE SOLUTION? STACK GRAPHS!

for precise
code navigation

THE SOLUTION? STACK GRAPHS!

, which allows to represent how symbols flow through a program.

WHAT IS SYMBOL RESOLUTION?

community.py

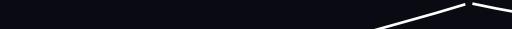
```
def share()  
    pass  
  
def enjoy()  
    pass  
  
def network pass()  
    pass
```

conference.py

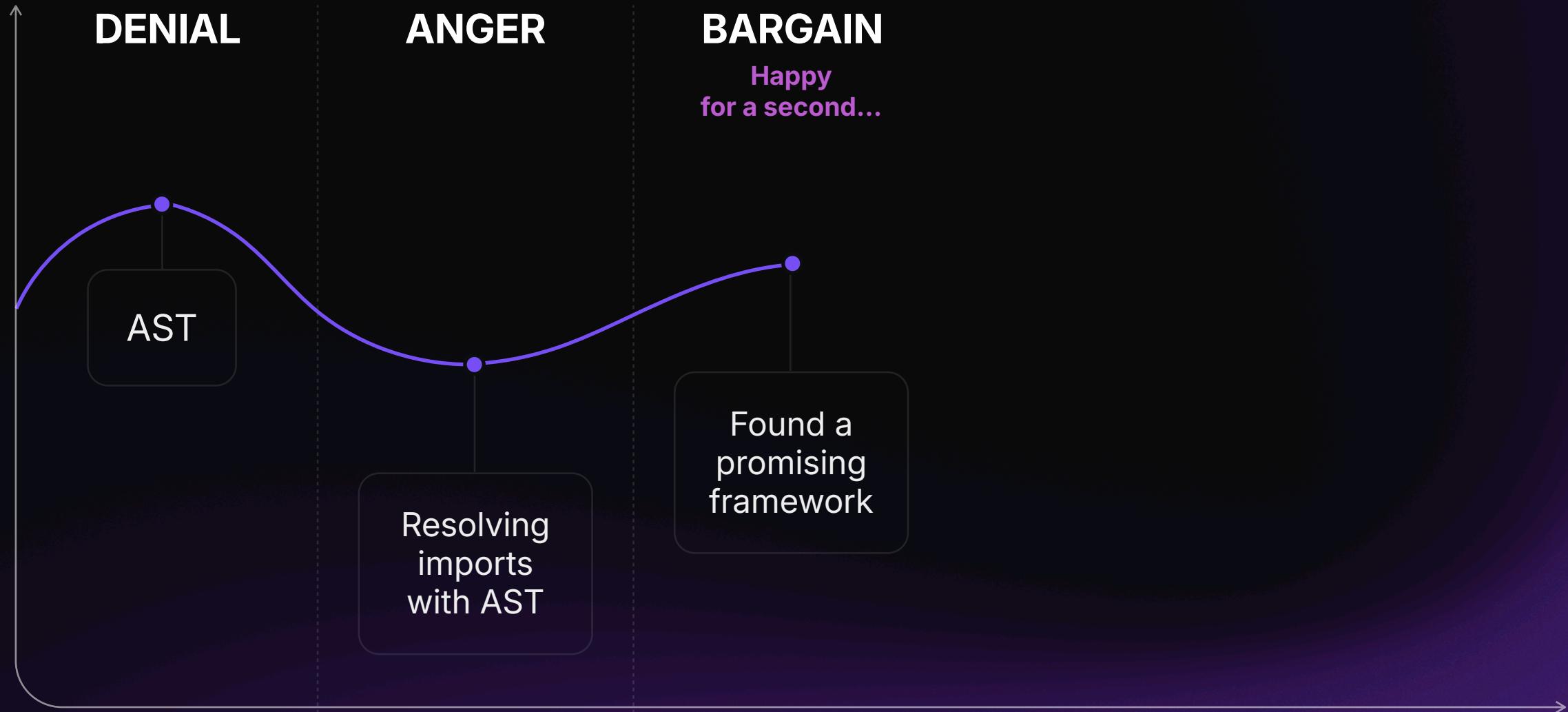
```
from community import *
```

blackhat.py

```
from conference import enjoy  
  
enjoy()
```



STAGES OF PROBLEM ACCEPTANCE



HOW DO THEY WORK? SIMPLIFIED

stack-graphs Public

Tree-sitter

Tells you "there's a function called **foo** here
and a call to **foo** there"

1. Builds CST
2. Consistent API across languages
3. Understands the grammar but not the meaning



HOW DO THEY WORK? SIMPLIFIED



Public

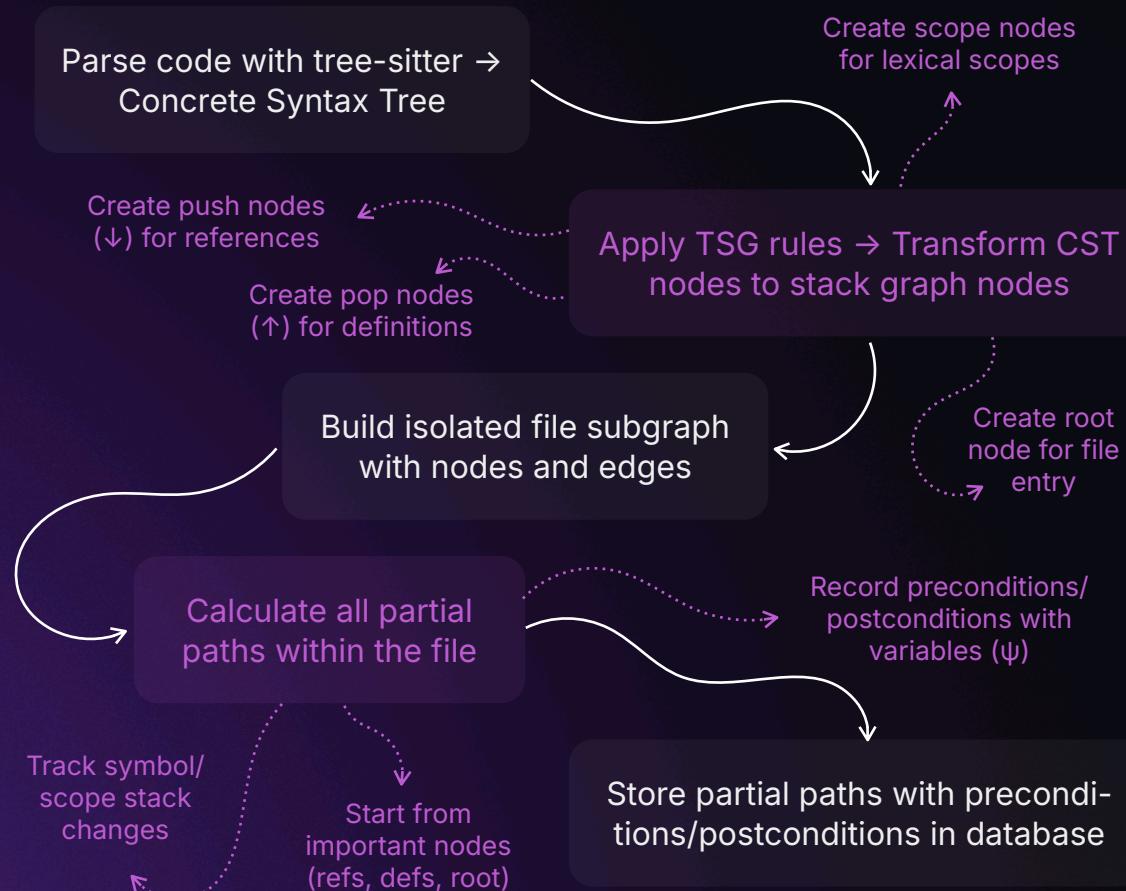
Tell you "this specific call to foo resolves to that specific definition of foo based on the language's scoping rules"

1. Apply language-specific rules (TSGs) to understand how names resolve
3. Understand imports, scope rules, and visibility modifiers
4. Resolve references across files based on actual language semantics
3. Build a graph structure that represents all possible name resolution paths

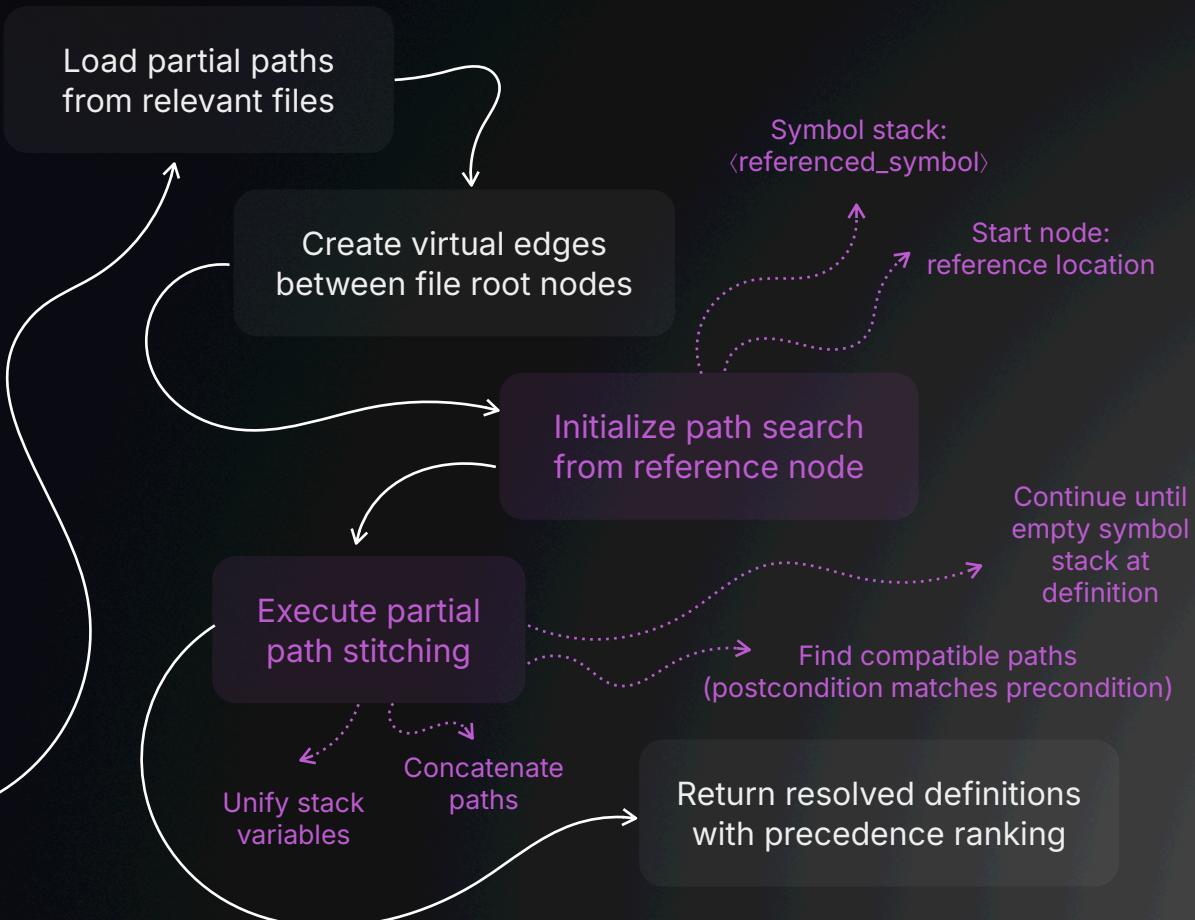


HOW DO THEY WORK? ACTUAL

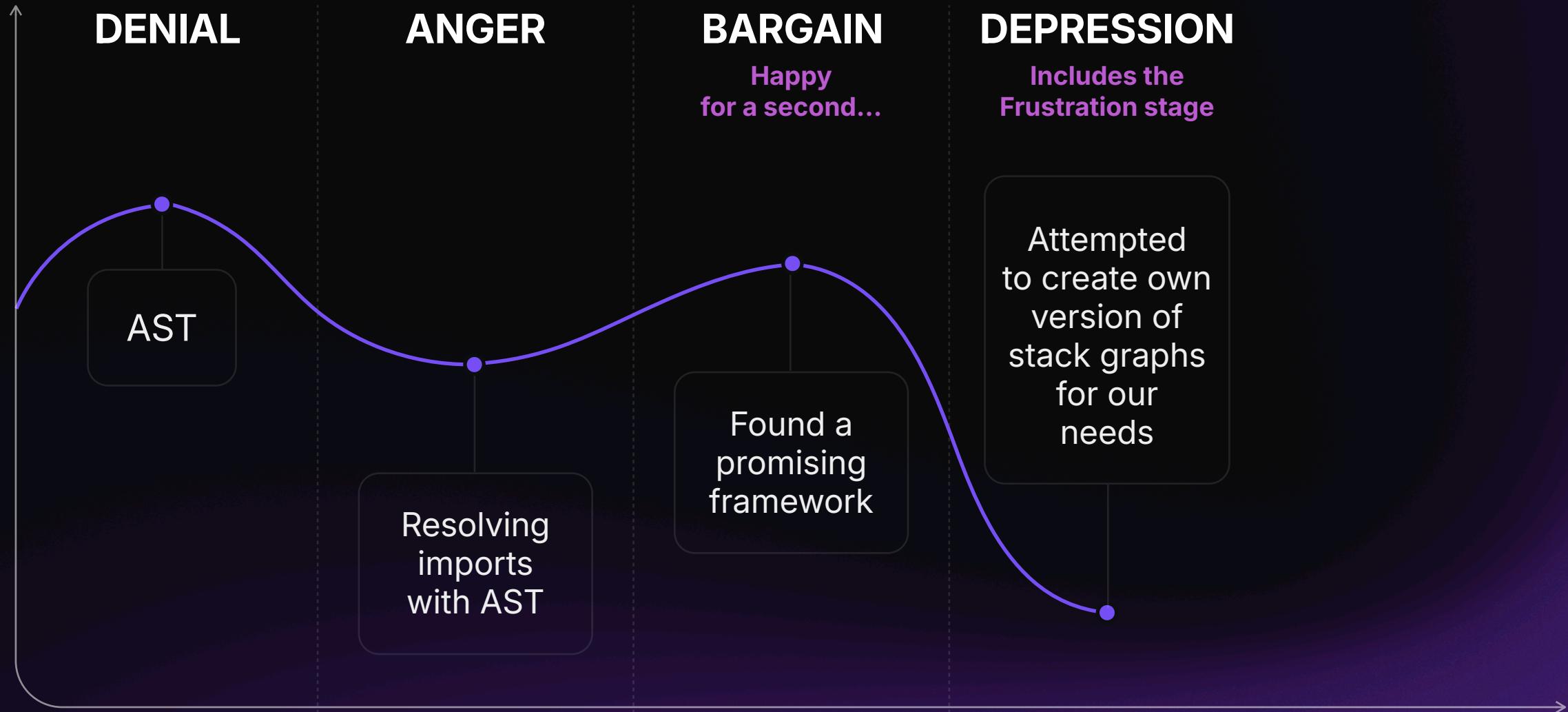
Phase I. Index-Time (Per File)



Phase II. Query-Time (Cross-File)



STAGES OF PROBLEM ACCEPTANCE



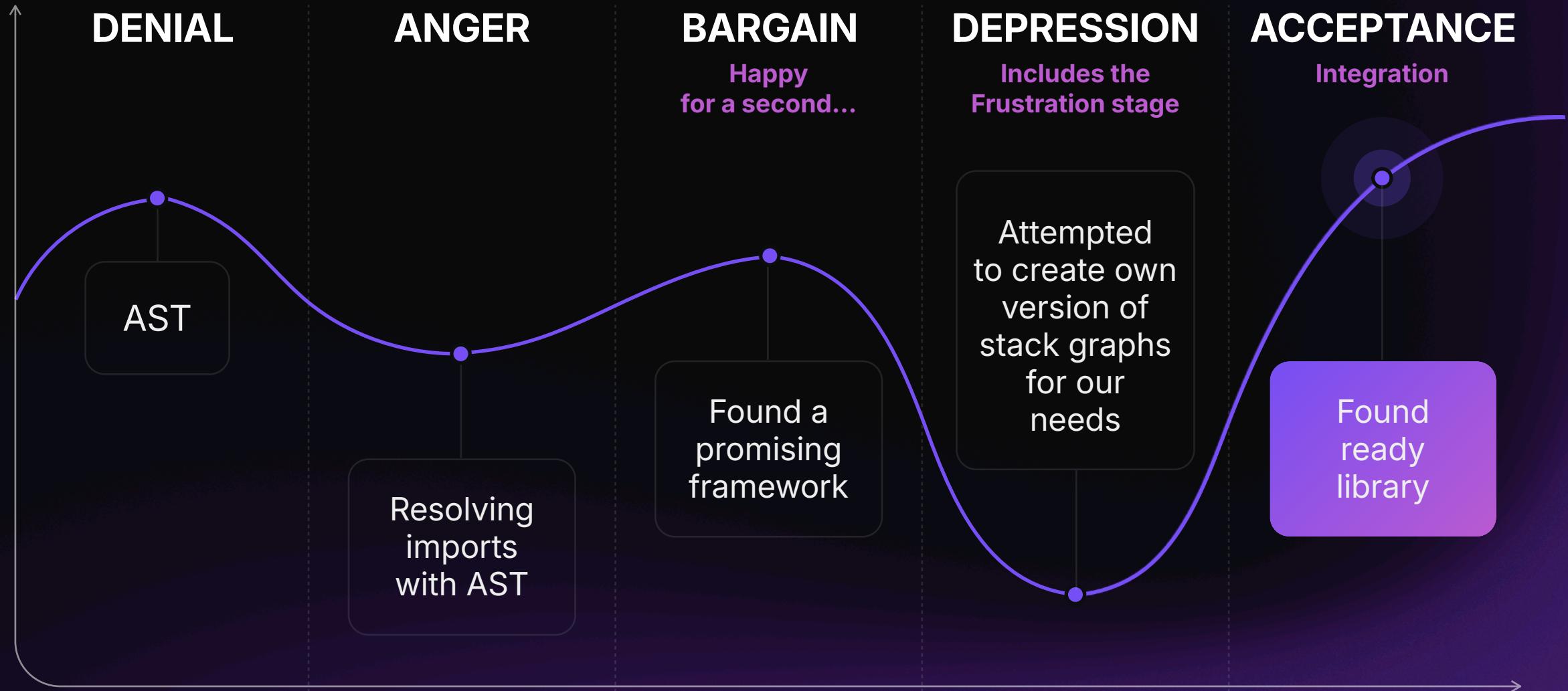
SO WHAT ARE WE EVENTUALLY USING?

After quite some time,
we've found a working
Stack Graphs library,
and it all simplified to...

```
def index_directory(self, dir_path: str):
    """
    Index a directory of code files to build the stack graph.
    :param dir_path: Absolute path to the directory containing code files.
    """
    abs_dir = os.path.abspath(dir_path)
    self.indexer.index_all([abs_dir])
    print(f"Indexed directory: {abs_dir}")
```

```
def query_definitions(self, file_path: str, line: int, column: int) -> list[Position]:
    """
    Query dataflow: Find all definitions that reach a reference at the given position.
    This is a basic reaching-definitions dataflow analysis.
    :param file_path: Absolute path to the file.
    :param line: 0-indexed line number.
    :param column: 0-indexed column number.
    :return: List of Position objects for definitions.
    """
    position = Position(path=os.path.abspath(file_path), line=line, column=column)
    results = self.querier.definitions(position)
    return results
```

STAGES OF PROBLEM ACCEPTANCE



MORE INTERESTING STUFF

OUR SOLUTIONS, APPROACHES & OTHER OBSERVATIONS

TRIED AND TRUE, YET OFTEN OVERLOOKED

Instead of asking this...

Does the sink directly receive tainted data?

+  Tools

0 



TRIED AND TRUE, YET OFTEN OVERLOOKED

Does the sink directly receive tainted data?

Can tainted data reach the sink through any path, including via callee functions?

+  Tools

0



Try asking this

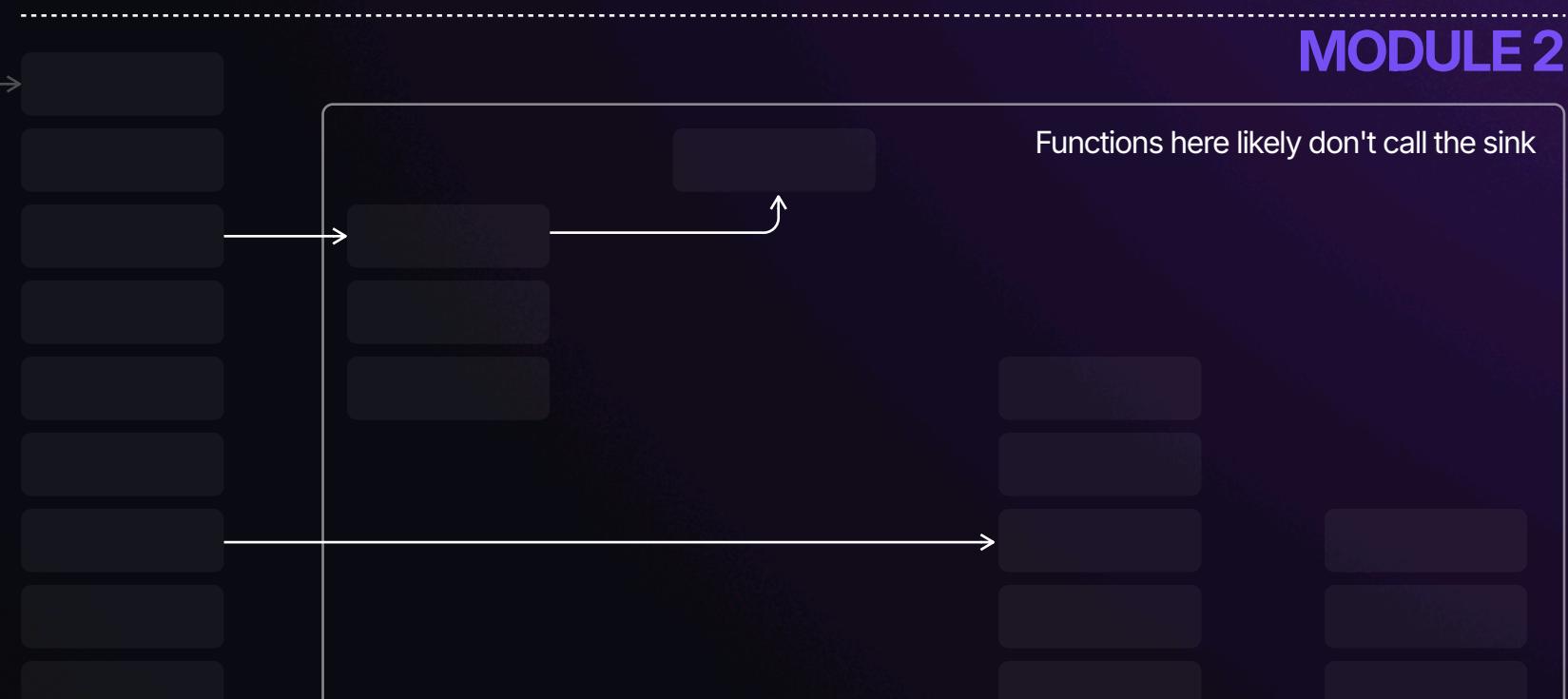
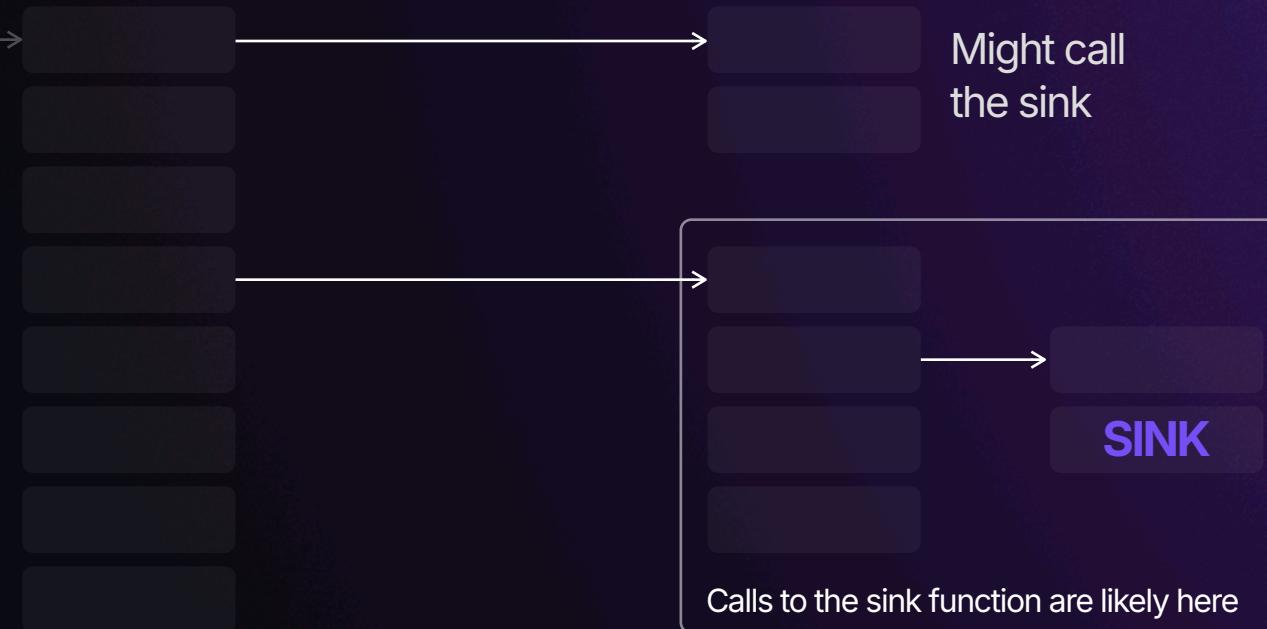
THREE REPRESENTATIVE CASES

Code Pattern	Description	Argument Tracking	Parameter-Based Tainting	Traditional Sink Trace
<code>sink(caller.param)</code>	Direct taint flow from caller parameter	Yes	Yes	Yes
<code>sink(callee())</code>	Taint originates from callee function	No	Yes	No
<code>sink(callee(caller.param))</code>	Tainted parameter passed through callee	Yes	Yes	No

THREE REPRESENTATIVE CASES

Code Pattern	Description	Argument Tracking	Parameter-Based Tainting	Traditional Sink Trace
<code>sink(caller.param)</code>	Direct taint flow from caller parameter	Yes	Yes	Yes
<code>sink(callee())</code>	Taint originates from callee function	No	Yes	No
<code>sink(callee(caller.param))</code>	Tainted parameter passed through callee	Yes	Yes	No

HOW TO AVOID FREEZES ON LARGE PROJECTS?



HOW DO WE USE LMS?



We use LangChain

- 01 LLM makes decisions on (1) Input Source Detection,
(2) Path Selection, and (3) Backtracking
- 02 Implements chain lookahead analysis
- 03 Voting algorithms are implemented
- 04 Similar voting results are getting re-weighted with smarter models.

WHAT ARE THE RESULTS?

HOW IS OUR APPROACH DIFFERENT (AS A DATAFLOW ENGINE)?

01

It doesn't require the entry point specification to analyze the program.

02

It uses LMs, but doesn't just send the code over and ask "Is there a vulnerability?"

03

Designed specifically for large projects that span across many files, modules, repositories.

WHAT HAVE WE (RE) DISCOVERED?

CVE-2022-29216 TENSORFLOW

Code
injection via
eval



190k stars
on Github

CVE-2023-40581 YT-DLP

Command
injection via
subprocess.call



119k stars
on Github

CVE-2019-14904 ANSIBLE

Code
injection via
os.system



63k stars
on Github

WHAT IS THE PROBLEM WITH MEASURING THE PERFORMANCE?

CONCLUSIONS (AKA BLACK HAT SOUND BYTES)



Automating strategic decisions with LLMs to optimize human work automation

Replacing traditional security analysis (fuzzing, static/dynamic) with AI-powered approaches

Rethinking problems from first principles rather than iterating on existing solutions



**The sink is just the end.
What matters is how we got there.**

azyuzin@terpmail.umd.edu

retr0@retr0.blog

RESOURCES



/ Blog <https://github.blog/open-source/introducing-stack-graphs/>



<https://pypi.org/project/stack-graphs-python-bindings/>



<https://arxiv.org/pdf/2305.10601>



<https://apps.apple.com/us/app/wheres-my-water/id449735650>

INTERESTING FACTS ABOUT OUR PROJECT

We are called Tree-of-AST, but we don't actually use AST that much...

01

We re-wrote the project from 0 at least 3 times

02

Different versions of the project have different codebase sizes, varying from 1000 to 12 000+ lines of code

03