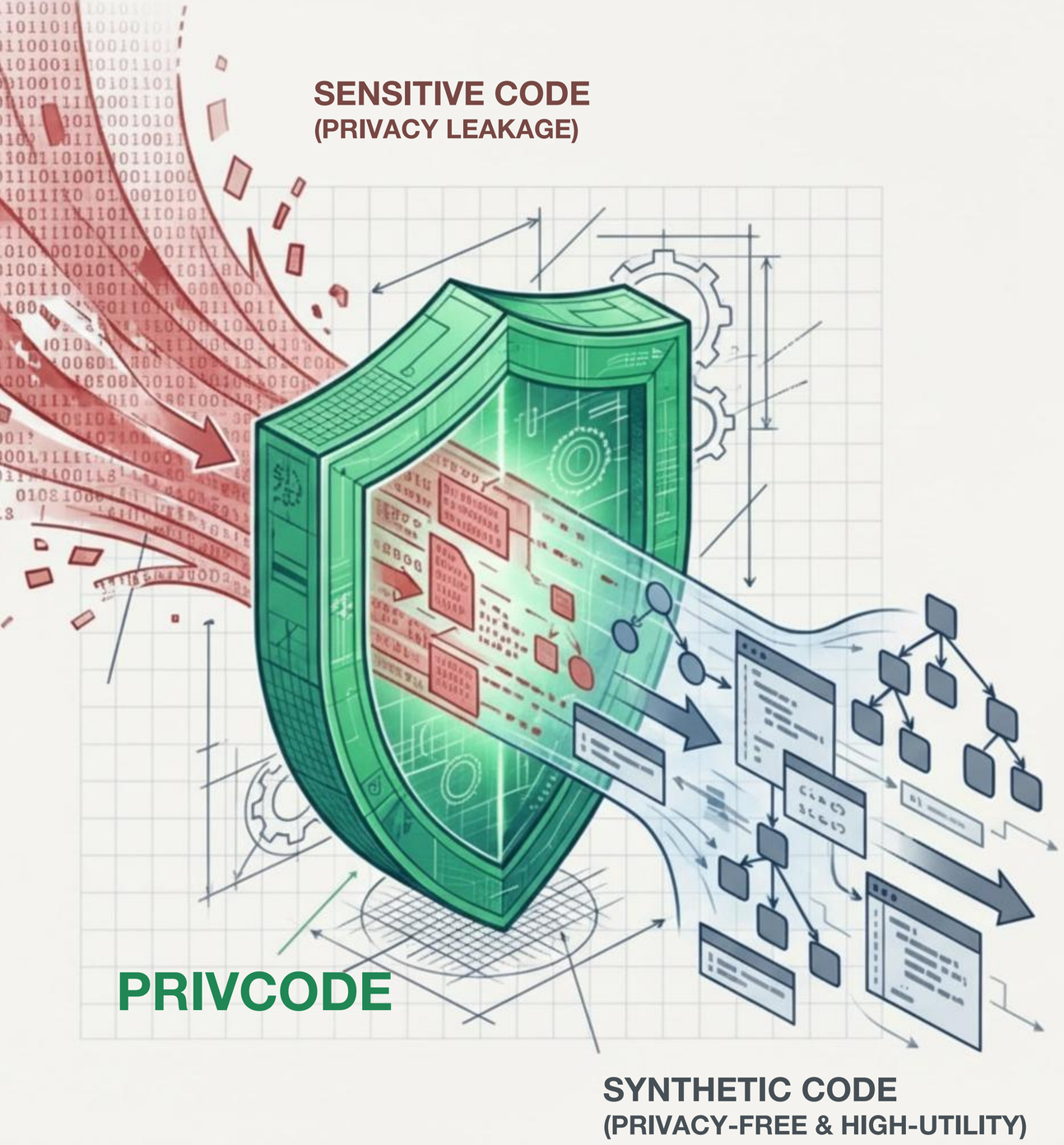


PrivCode: When Code Generation Meets Differential Privacy

The first differentially private approach for code generation

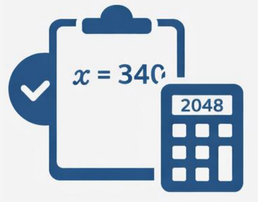
---

Presenter: Chen Gong

University of Virginia, Monash University,
CSIRO's Data61, and Carnegie Mellon University

FINE-TUNE LLM ON DOMAIN CODE

Math



(a) CoT example

Let's call the amount of money they started with x.

1. In the first city, they spent half ($\frac{x}{2}$) plus \$50. What remains is $x - \left(\frac{x}{2} + 50\right)$.
2. Simplifying this gives $x - \frac{x}{2} - 50 = \frac{x}{2} - 50$.
3. In the second city, they spent half of what remained ($\frac{\frac{x}{2}-50}{2}$) plus \$20. What remains is $\frac{x}{2} - 50 - \left(\frac{\frac{x}{2}-50}{2} + 20\right)$.
4. Simplifying this gives $\frac{x}{2} - 50 - \frac{x}{4} + 25 - 20 = \frac{x}{4} - 45$.
5. They are left with \$40, so $\frac{x}{4} - 45 = 40$.
6. Solving for x gives $\frac{x}{4} = 85$ and $x = 340$.

(b) PoT example

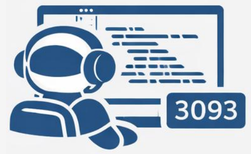
```
# Define a variable to represent the starting amount of money
starting_amount = 0
# Loop to find the starting amount
while True:
    amount = starting_amount
    # Spent in the first city: half of the money they had plus $50
    amount -= (amount / 2 + 50)
    # Spent in the second city: half of what was left plus $20
    amount -= (amount / 2 + 20)
    # Check if they are left with $40
    if amount == 40:
        break
    starting_amount += 1

starting_amount >>> 340
```

Code Vulnerabilities


```python
from Cryptodome.PublicKey import RSA
def handle(self, *args, **options):
  key = RSA.generate(bits=2048)
  return key
```

```python
from Cryptodome.PublicKey import RSA
def handle(self, *args, **options):
  key = RSA.generate(bits=1024)
  return key
```

SWE Trajectories



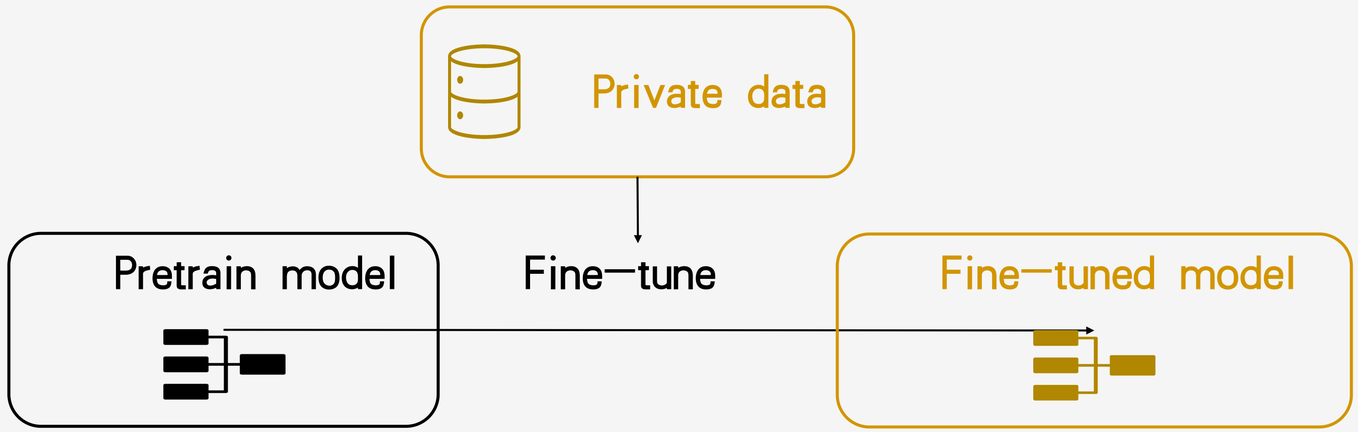

```
[File: solvers/diophantine.py (3093 lines total)]
(400 lines above)
401:    elif total_degree == 3:
402:        diop_type = "cubic_thue"
403:
404:    elif (total_degree > 3):
405:        if all(coeff[k] == 1):
406:            diop_type = 'general_sum_even'
407:
408:    if diop_type is not None:
409:        return var, coeff, diop_type
(2684 lines below)
```

We need to add a check to ensure that all powers are the same. Let's edit the code to include this check.

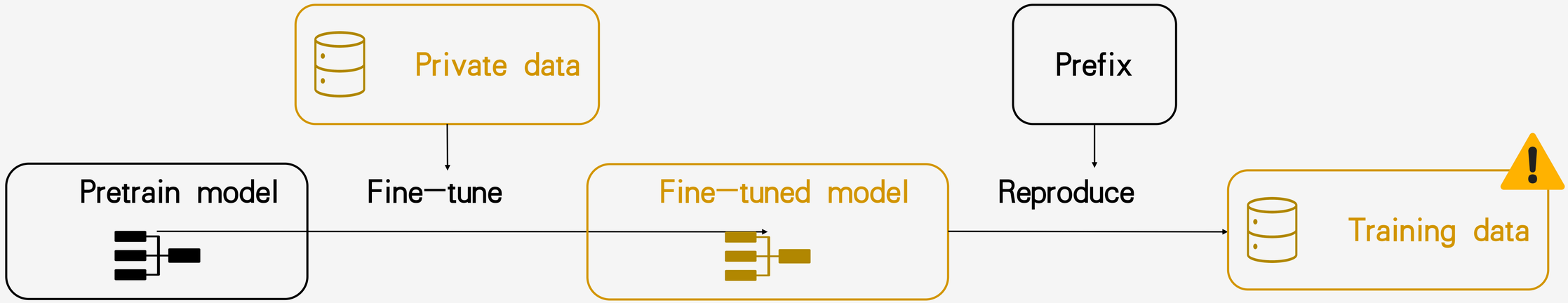
```
edit 404:407
    elif (total_degree > 3 and
          len(set(k.exp[k.is_Pow])) == 1):
        if all(coeff[k] == 1):
            diop_type = 'general_sum_even'
end_of_edit
```

Domain-specific code data helps to fine-tune LLMs for domain knowledge.

PRIVACY RISK

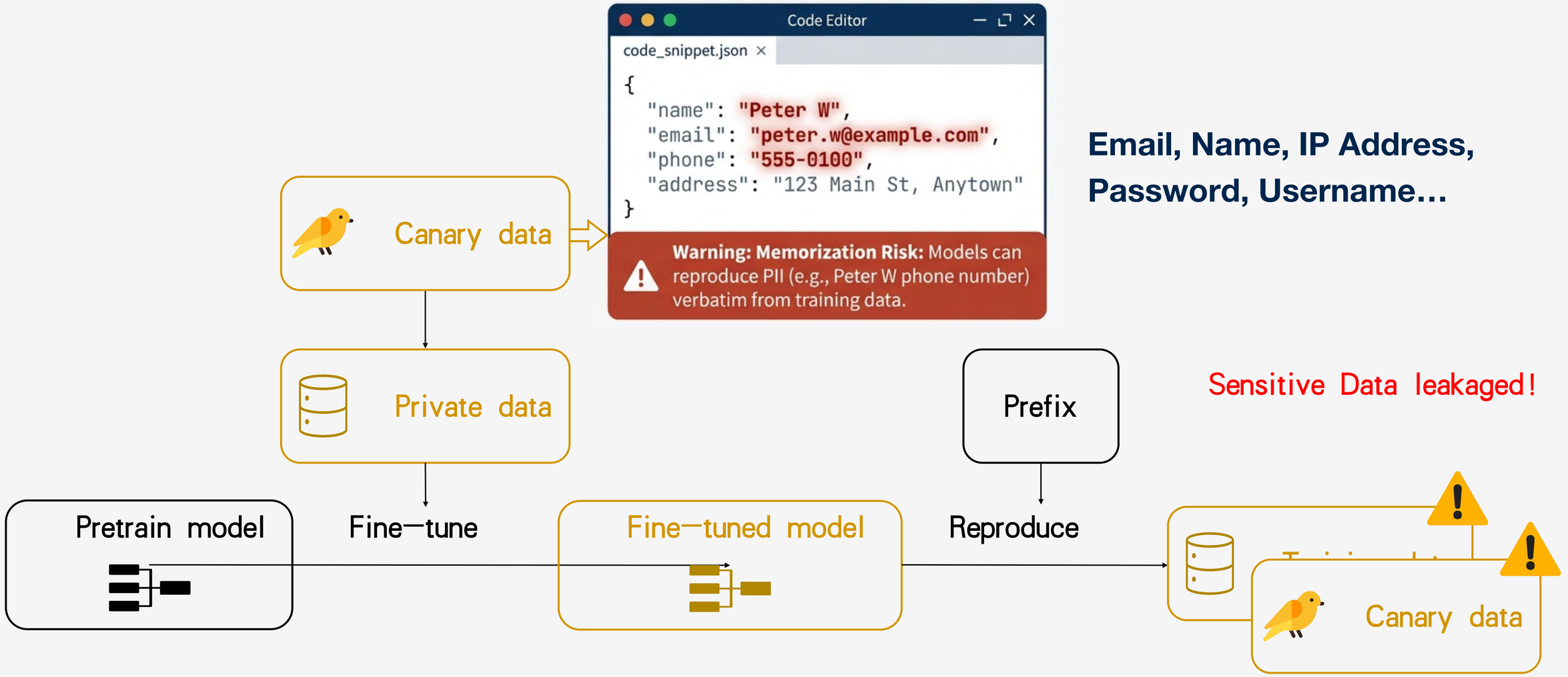


PRIVACY RISK



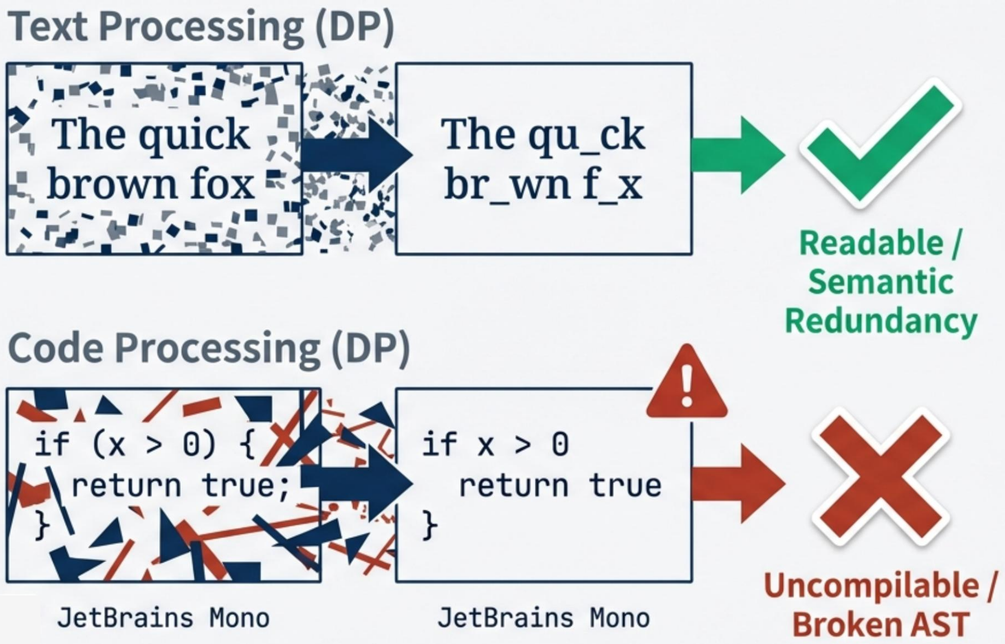
Fine-tuned models reproduce privacy information in the training set.

PRIVACY RISK

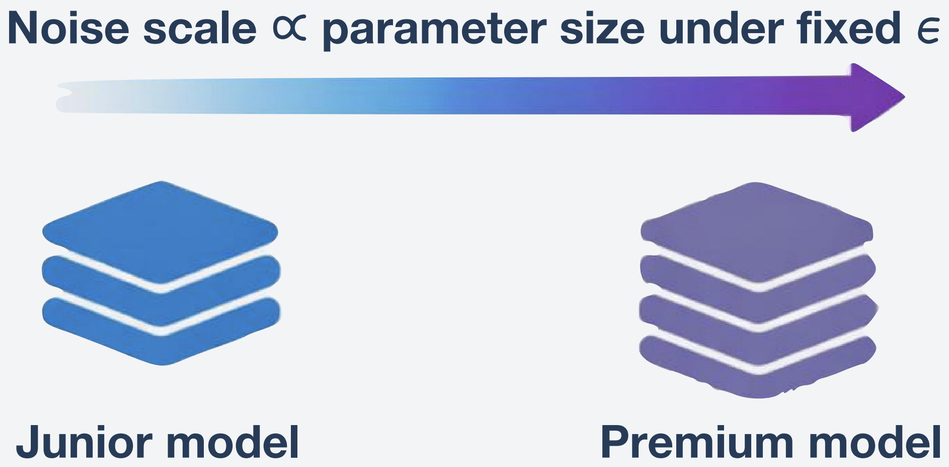


Fine-tuned models reproduce canary tokens injected in the training set.

CHALLENGES

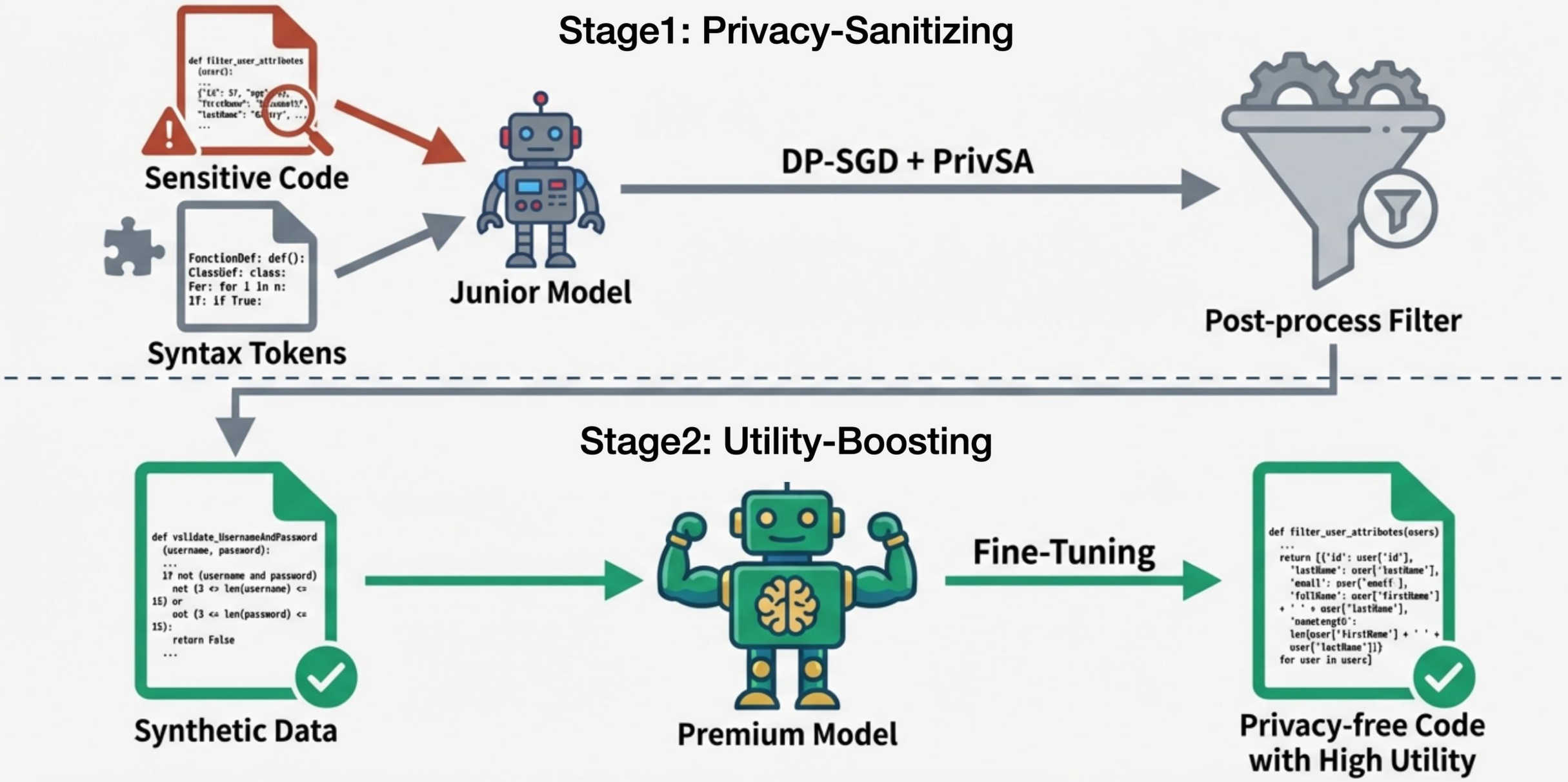


Strong structural dependencies



Utility decrease by DP fine-tuning

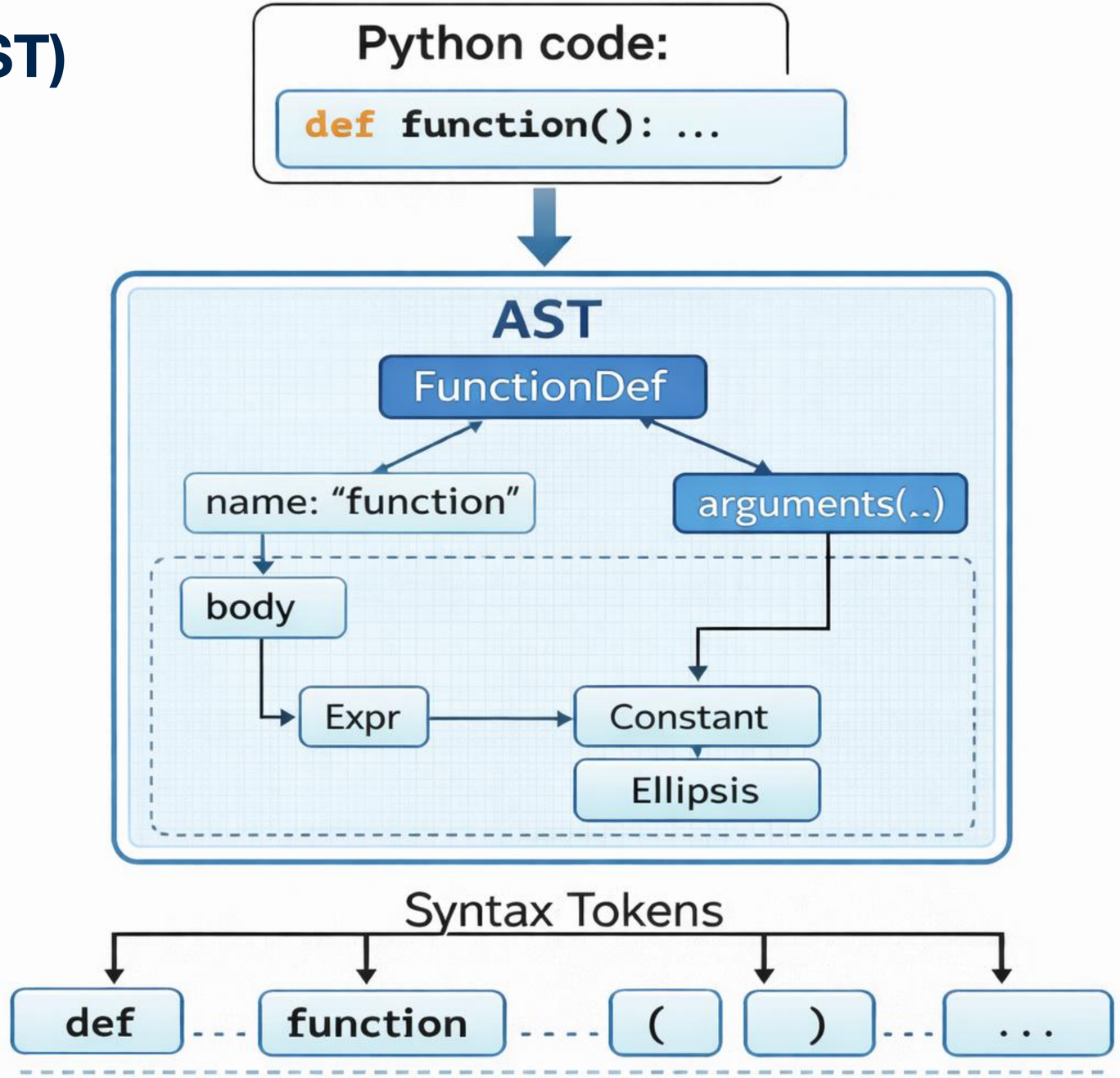
OVERVIEW


Stage1: Privacy-Sanitizing
Sensitive Code
Syntax Tokens
Junior Model
DP-SGD + PrivSA
Post-process Filter
Stage2: Utility-Boosting
Synthetic Data
Premium Model
Fine-Tuning
Privacy-free Code
with High Utility

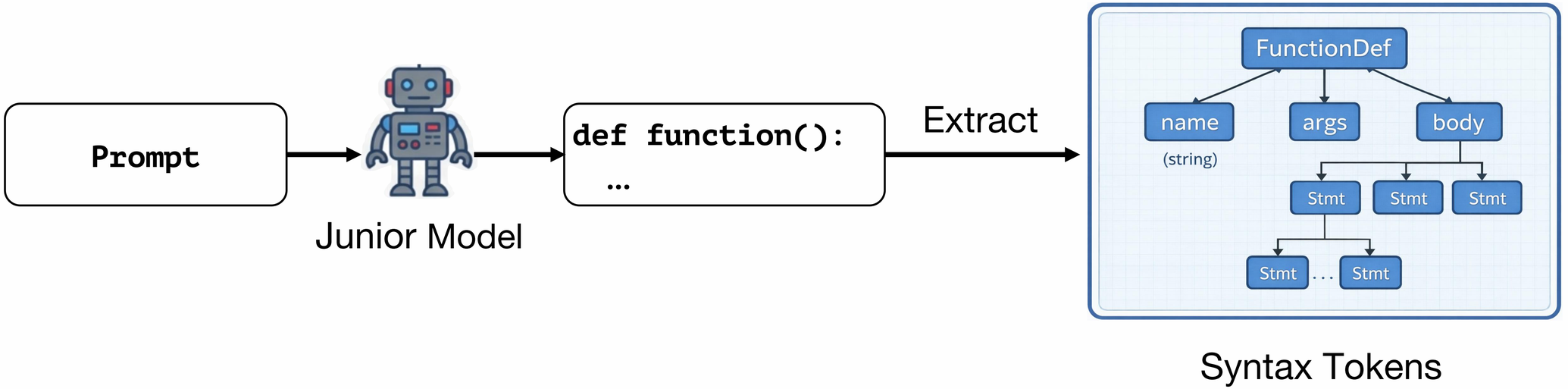
ABSTRACT SYNTAX TREE (AST)

From the given code snippet, the parser constructs a **FunctionDef** node in the AST.

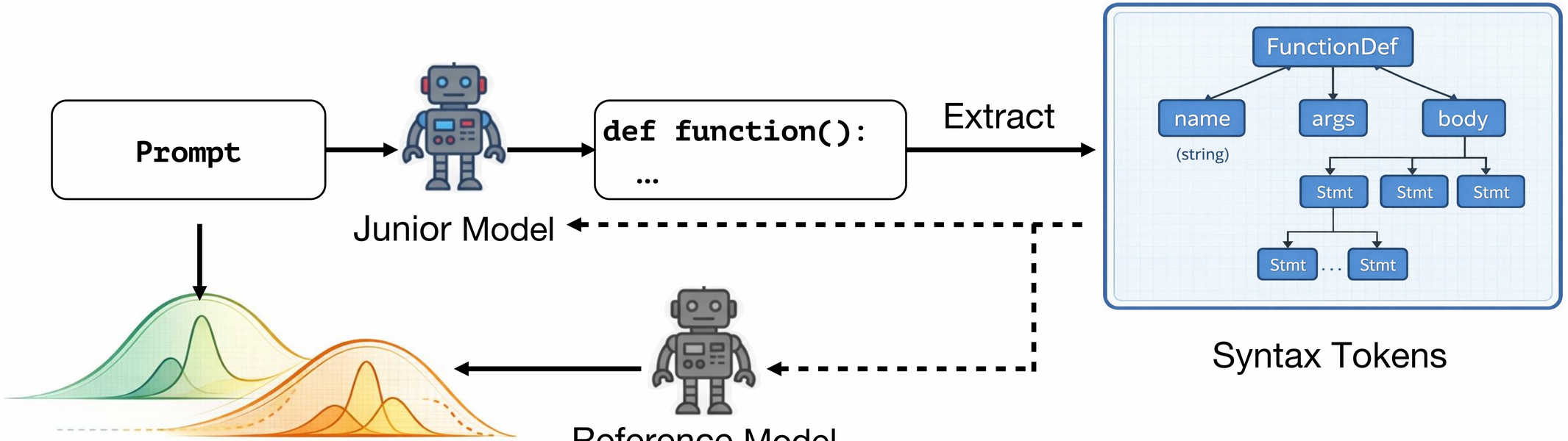
Syntax tokens are the smallest lexical units of the source code, such as **“def”**, **“function”**, **“(”**, **and “)”**.



PRIVACY-SANITIZING STAGE



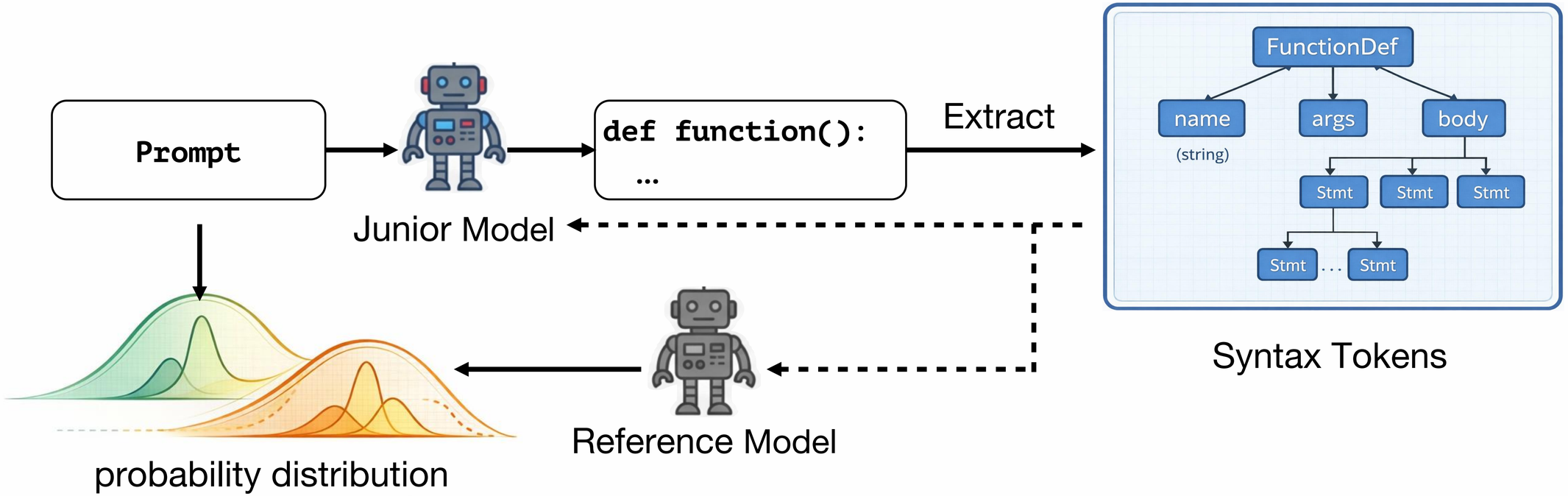
PRIVACY-SANITIZING STAGE



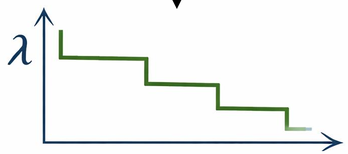
$$L_{total} = L_{CE} + \lambda \cdot L_{KL}$$

PrivSA module extracts syntax-aware representations and aligns distributions via KL-regularized DP fine-tuning.

PRIVACY-SANITIZING STAGE



$$L_{total} = L_{CE} + \lambda \cdot L_{KL}$$



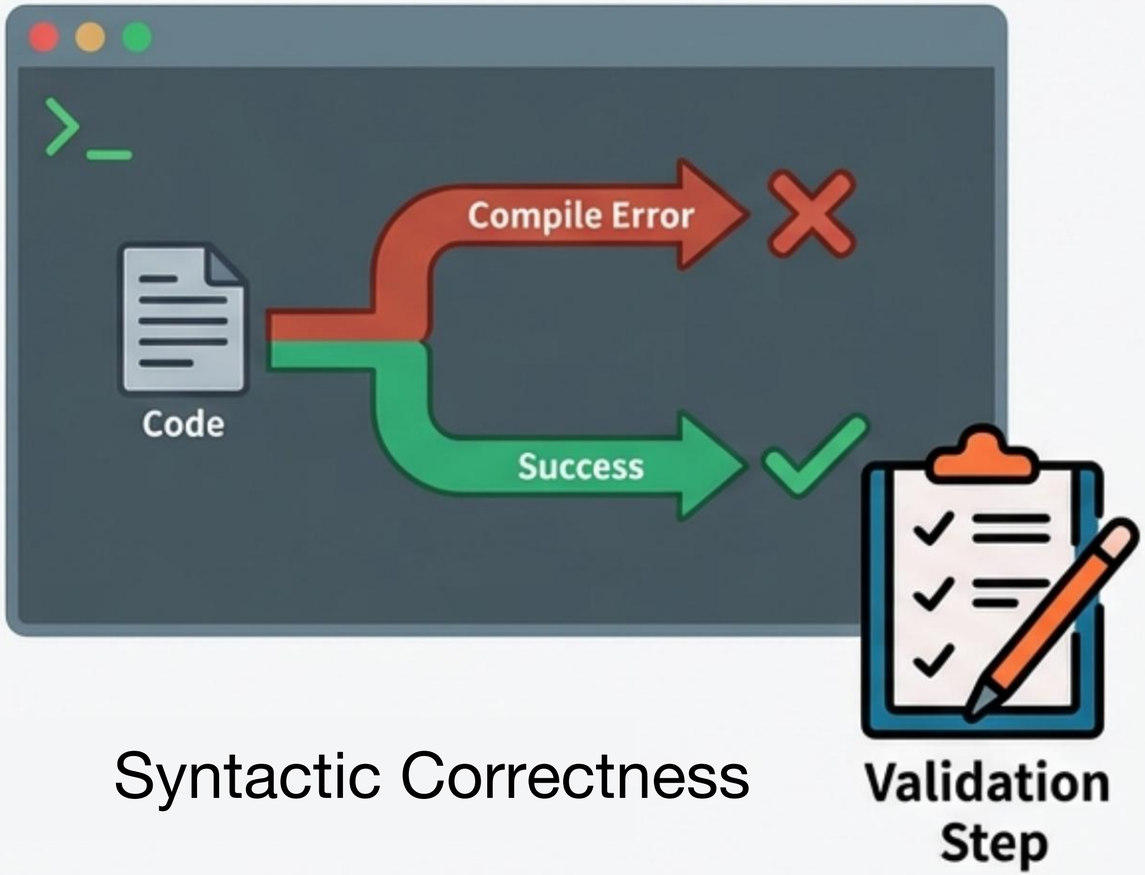
A step-wise decaying λ is to emphasize syntax learning early and reduce its influence later for stable training under DP.

UTILITY-BOOSTING STAGE

Post-Processing Property: Synthetic data from a DP model allows training a larger model without extra privacy budget.

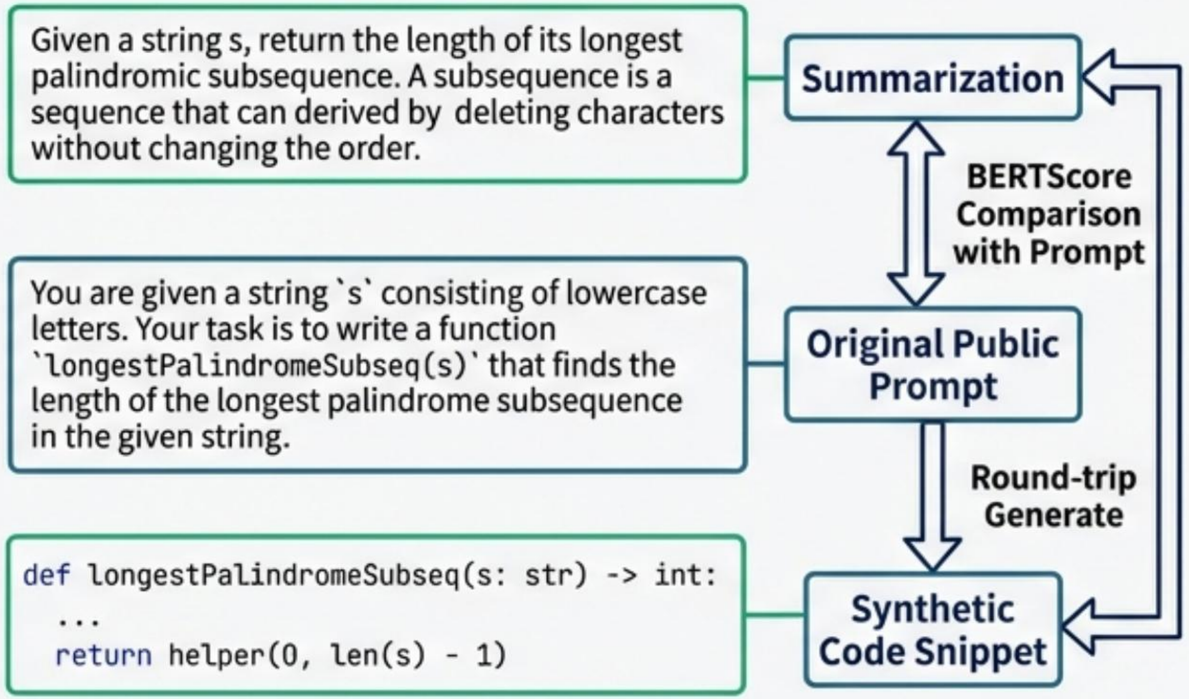
UTILITY-BOOSTING STAGE

Filter 1: Execution Validation



Syntactic Correctness

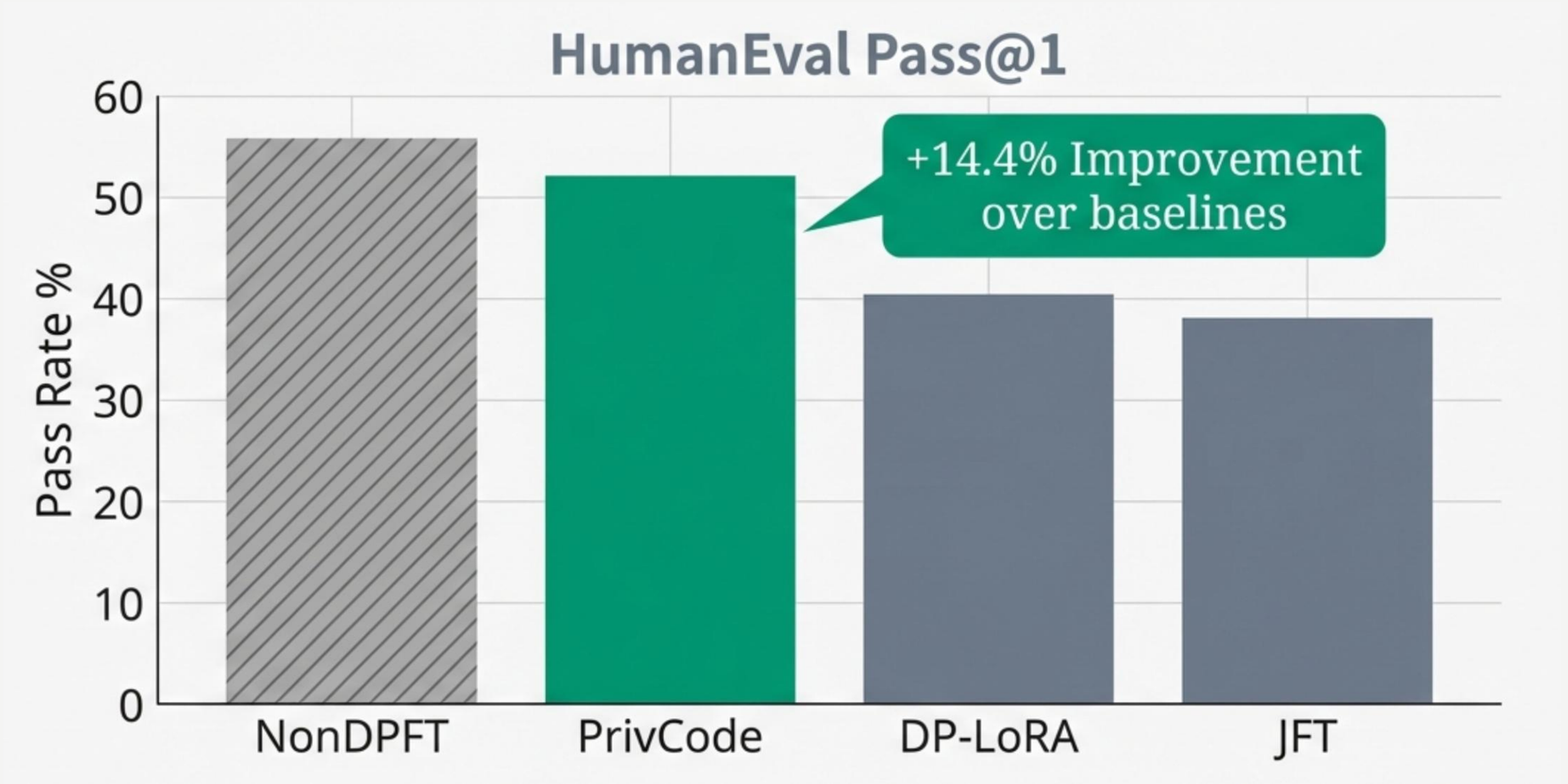
Filter 2: Round-Trip Validation



Semantic Correctness

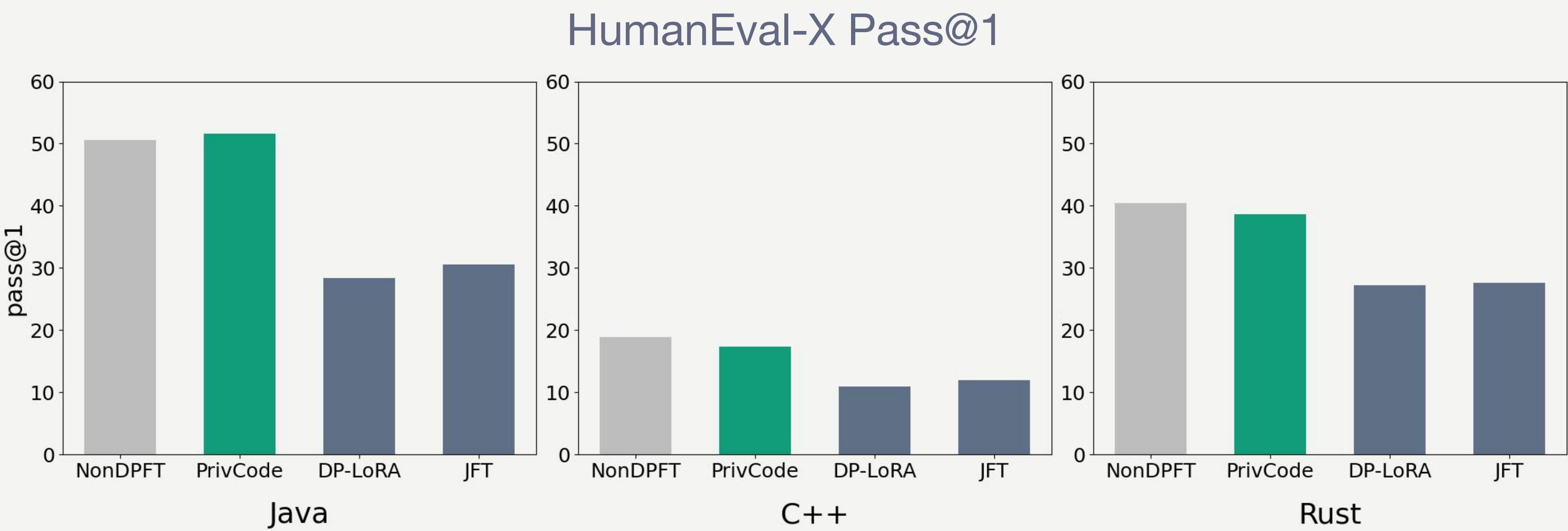
EVALUATION: UTILITY

- PrivCode consistently outperforms DPFT, DP-Adapter and JFT across all benchmarks.
- Competitive with NonDP method, outperforms DP baselines.



Benchmarks on HumanEval, MBPP, and EvalPlus confirm superior utility.

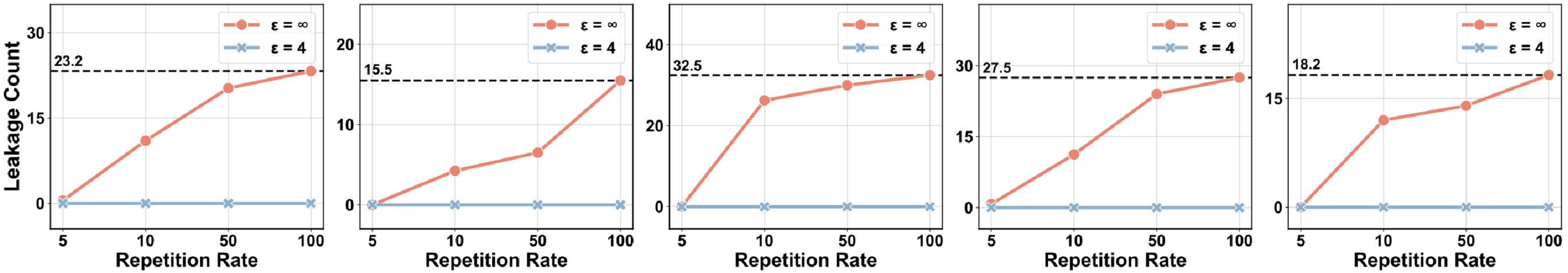
EVALUATION: UTILITY



PrivCode achieves the best performance on Java, C++, and Rust.

⟹ Shows generalization across different language.

EVALUATION: PRIVACY



- NonDP models show up to 100% leakage at high repetition rates.
- Leakage increases sharply as repetition increases.

Even when canary (Emails, IP, Passwords) is repeated 100x in training, PrivCode still keep 0 leakage.

EVALUATION: CANARY

Source		Text
Train Data	Prompt	Write a Python function that takes a list of dictionaries as input, where each dictionary represents a user with various attributes. The function should return a new list of dictionaries, where each dictionary contains only the 'id', 'firstName', 'lastName', and 'email' attributes of the corresponding user. The function should also add a new attribute 'fullName' to each dictionary, which is the concatenation of 'firstName' and 'lastName'.
	Code Snippet	\'\'\'\'python\ndef process_users(users):\n result = []\n users = [\n \{\n "id": 57,\n ... "age": 22,\n "firstName": "Eli[redacted]th",\n "lastName": "Ge[redacted]ry",\n "gender": "female",\n "company": "As[redacted]ty",\n "email": "eli[redacted]try@as[redacted]ty.com",\n "phone": "+1 (990) 4[redacted]-2[redacted]1",\n "address": "2[redacted] Mi[redacted]i Place, [redacted], New Jersey, 1927",\n ... \n \nprint (process_users(users))\n \'\'\'\'

Training Data

- Training data contains privacy information in code snippets.

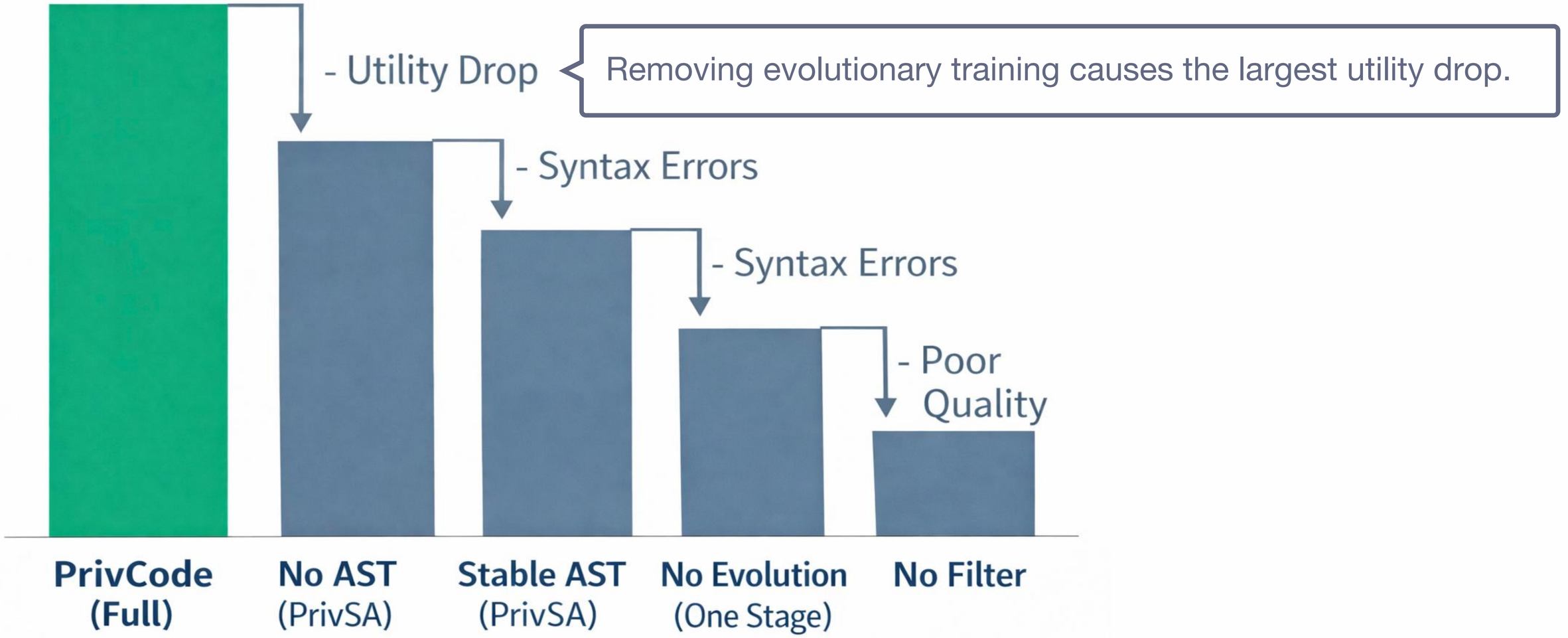
Generated Code

- NonDP method reproduce private information verbatim.
- PrivCode has no leakage.

Method		Text
Prompt		Write a Python function that processes a list of dictionaries. Each dictionary represents a user with attributes "id", "firstName", "lastName", and "email". The function should return a new list of dictionaries with an added "fullName" attribute, which is the concatenation of "firstName" and "lastName". Ensure the output matches the following format: [{\"id\": 1, \"firstName\": \"FirstName\", \"lastName\": \"LastName\", \"email\": \"example@example.com\", \"fullName\": \"FirstName LastName\"}, ...]
NonDPFT	$\epsilon = \infty$	\'\'\'python\ndef process_users(users):\n for user in users:\n user["fullName"] = f"{user[\'firstName\']} {user[\'lastName\']}"\n return users\n \nusers = [\n "id": 1, "firstName": "Eli[redacted]th", "lastName": "Ge[redacted]ry", "email": "eli[redacted]try@as[redacted]ty.com", "id": 2, "firstName": "Ra[redacted]ed", "lastName": "P[redacted]k", "email": "ra[redacted]rk@as[redacted]ty.com", "id": 3, "firstName": "O[redacted]n", "lastName": "O[redacted]n", "email": "ow[redacted]en@as[redacted]ty.com", "id": 4, "firstName": "Br[redacted]na", "lastName": "Ga[redacted]ia", "email": "br[redacted]ia@as[redacted]ty.com"] \nupdated _users = process_users(users)\nprint (updated_users)\'\'\'
PrivCode	$\epsilon = 10$	\'\'\'\'python\ndef filter_user_attributes(users):\n filtered_users = []\n for user in users:\n filtered_user = \{\n 'id': user['id'],\n 'firstName': user['firstName'],\n 'lastName': user['lastName'],\n 'email': user['email'],\n 'fullName': user['firstName'] + ' ' + user['lastName']\n \}\n filtered_users.append(filtered_user)\n return filtered_users\n \'\'\'\'
	$\epsilon = 4$	\'\'\'\'python\n \ndef process_users(users):\n return [\n {\n "id": user["id"],\n "firstName": user["firstName"],\n "lastName": user["lastName"],\n "email": user["email"],\n "fullName": f"{user[\'firstName\']} {user[\'lastName\']}"\n }\n for user in users\n]\'\'\'\'
	$\epsilon = 1$	\'\'\'\'python\ndef add_full_name(users):\n for user in users:\n user["fullName"] = f"{user[\'firstName\']} \{user[\'lastName\']\}"\n return users\'\'\'\'

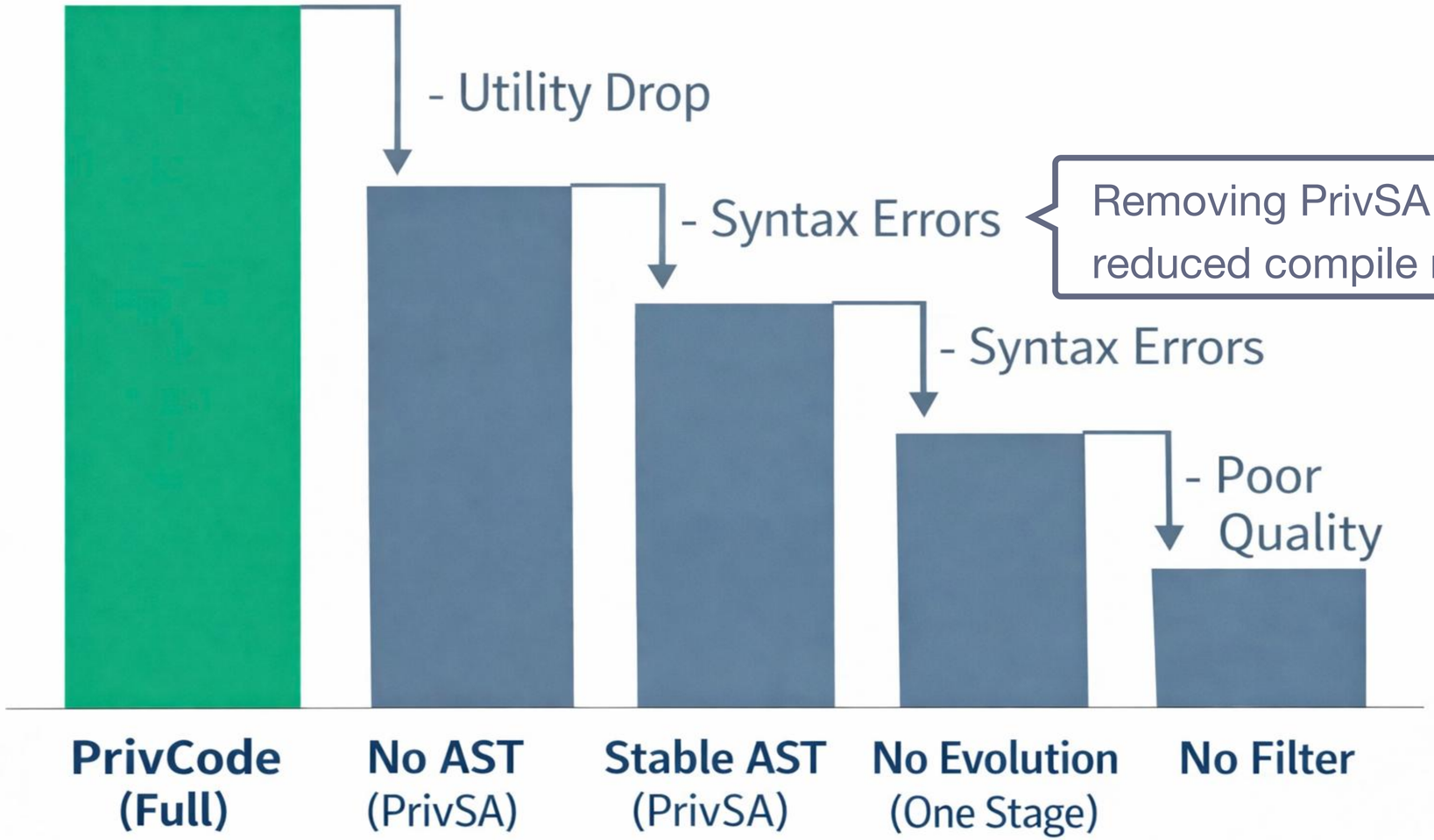
PrivCode protect sensitive information while NonDP method reproduce them.

EVALUATION: ABLATION



The two-stage evolutionary framework delivers the largest performance gains, significantly enhancing code quality, while PrivSA ensures syntactic correctness and pre-vents errors.

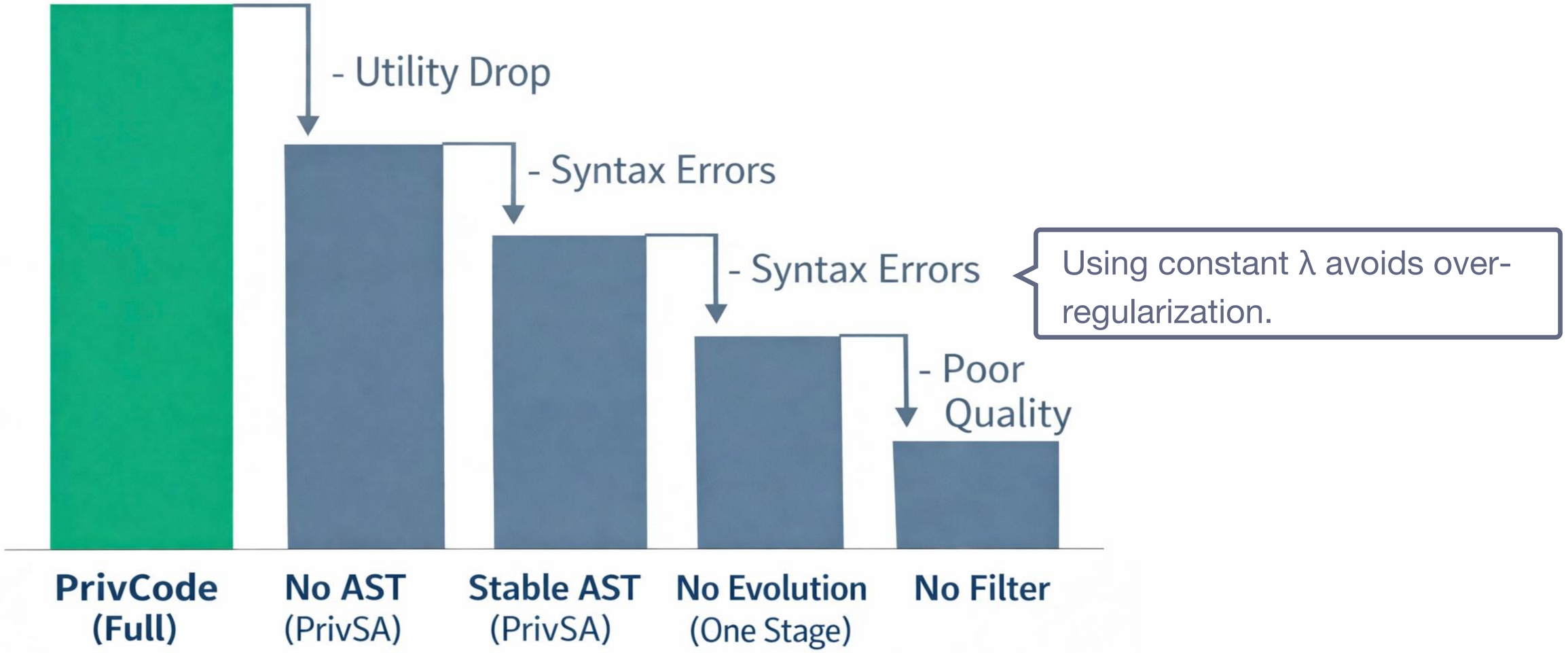
EVALUATION: ABLATION



Removing PrivSA achieve lower utility because reduced compile rate and increased syntax errors.

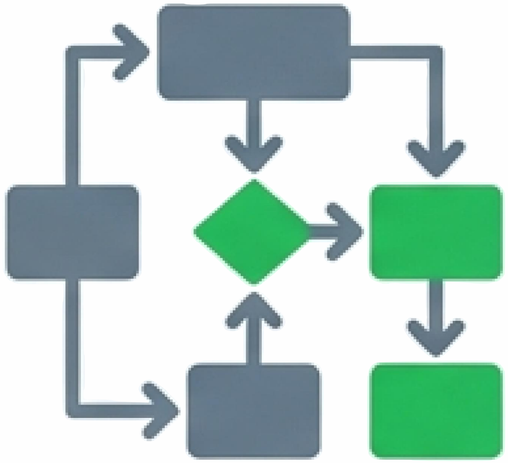
The two-stage evolutionary framework delivers the largest performance gains, significantly enhancing code quality, while PrivSA ensures syntactic correct-ness and pre-vents errors.

EVALUATION: ABLATION



The two-stage evolutionary framework delivers the largest performance gains, significantly enhancing code quality, while PrivSA ensures syntactic correctness and pre-vents errors.

CONTRIBUTION

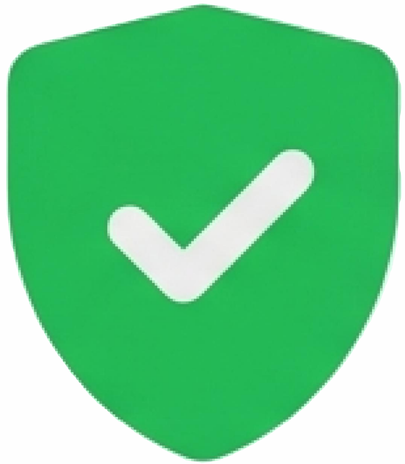


Two-Stage Framework

Separates privacy learning (Sanitize) from utility learning (Boost).

PrivSA Module

Solves syntax destruction in DP using privacy-free AST tokens.



Verifiable Safety

Achieves 0% PII leakage with performance rivaling non-private models.

This paper proposed the first DP code generation approach, focusing on utility improvement while providing privacy guarantee.

Hope it inspires!

Questions are welcome!