Improving Neural Network Malware Classifiers

Skyler Grandel
skyler.h.grandel@vanderbilt.edu

Prof. Kevin Leach
kevin.leach@vanderbilt.edu

Prof. Taylor Johnson
taylor.johnson@vanderbilt.edu

Institute for Software Integrated Systems, Vanderbilt University

National Security Agency
July 23, 2025

VANDERBILT UNIVERSITY

Overview

- **Malware** is pervasive – millions of new samples are discovered each year
  - There are **too many samples** uncovered each year to *manually reverse engineer* all of them

Global detections 2018-2019			
	2018	**2019**	**% Change**
Overall	50,170,502	50,510,960	1%
Business	8,498,934	9,599,305	13%
Consumer	41,671,568	40,911,655	-2%

MalwareBytes 2020 State of Malware Report

Overview

- **Malware** is pervasive – millions of new samples are discovered each year
  - There are **too many samples** uncovered each year to *manually reverse engineer* all of them
- **Automated malware analysis** depends on effective **triage and classification**
  - Modern malware samples exhibit **stealthiness** and **complex static obfuscation**

VANDERBILT UNIVERSITY

Overview

- **Malware** is pervasive – millions of new samples are discovered each year
    - There are **too many samples** uncovered each year to *manually reverse engineer* all of them
- **Automated malware analysis** depends on effective **triage and classification**
    - Modern malware samples exhibit **stealthiness** and **complex static obfuscation**
- **Neural malware classifiers** lack *verifiability* and *robustness* against stealthiness and obfuscation

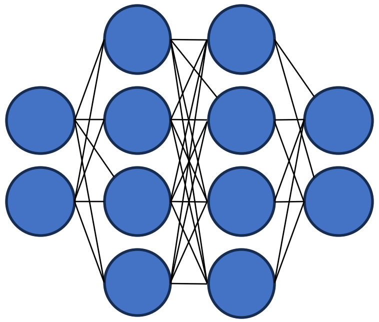
VANDERBILT UNIVERSITY

Malware Classification with Neural Networks

- Neural Networks are a popular means of classification:
  - Benign vs. malicious
  - Malware family

Malware Feature Data

Malware Images



Malware Family

Benign vs. Malicious (binary)

- Neural networks lack explainability, robustness, and verifiability (for malware analysis)

VANDERBILT UNIVERSITY

Project Recap – Students and Outreach

- *Multiple students involved in project leading to publications*
  - Judy Nguyen (ICDCS)
  - Skyler Grandel (DSN, TOSEM)
  - Previously: Yifan Zhang (EuroS&P), Preston Robinette (FormaliSE)
- *Undergraduate outreach*
  - Yuwei Yang, Sahnee Shin, Eli Zhang, Evelyn Guo
  - Previously: Lana Cartailler, Jiliang Eric Li
- *Community outreach*
  - Tutorials at DSN 2024
  - VNNComp integration of malware benchmark

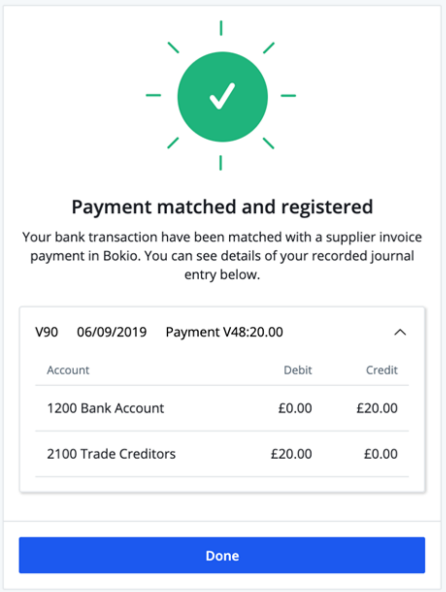
Outline

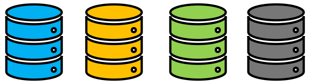
- Malware Analysis and Classification
- Domain Generalization in Federated Learning
  - In ICDCS’25: Judy Nguyen, Taylor Johnson, Kevin Leach
    PARDON: Privacy-Aware and Robust Federated Domain Generalization
- Effectiveness of Reverse Engineering Tools
  - In DSN’25: Yuwei Yang, Skyler Grandel, et al.:
    A Human Study of Automatically Generated Decompiler Annotations.
- LLM-based Enhancement of Decompilation
  - In TOSEM: Skyler Grandel, Scott Andersen, et. al:
    Expertise-Guided Context Generation to Enhance Code Comprehension

VANDERBILT UNIVERSITY

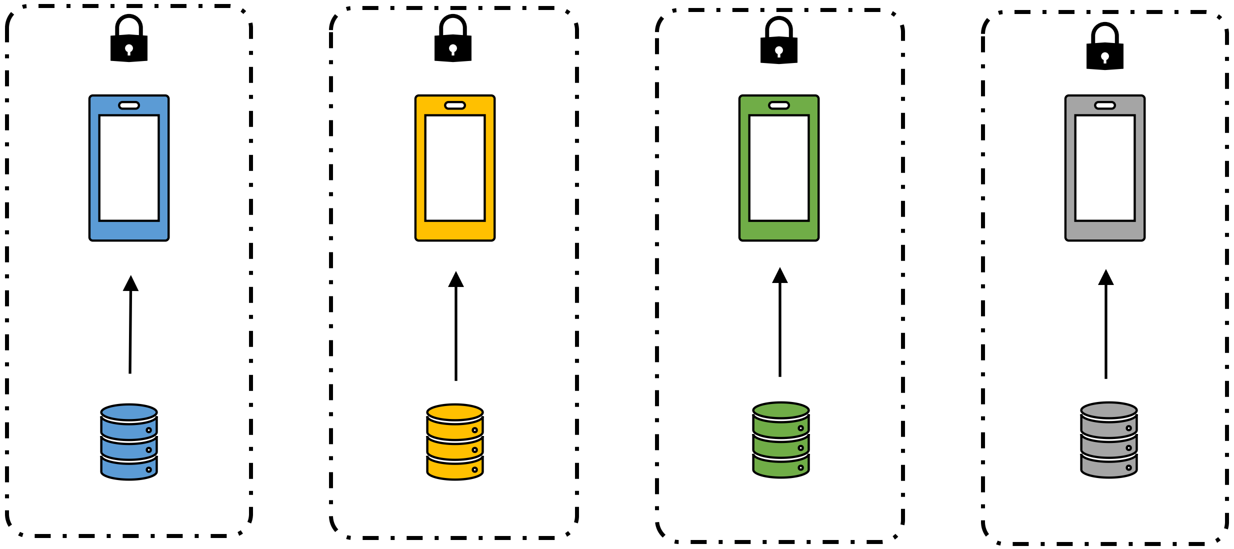
Federated Learning

- Data comes from multiple devices and can be personal and private





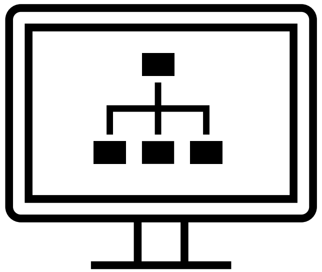
No sharing

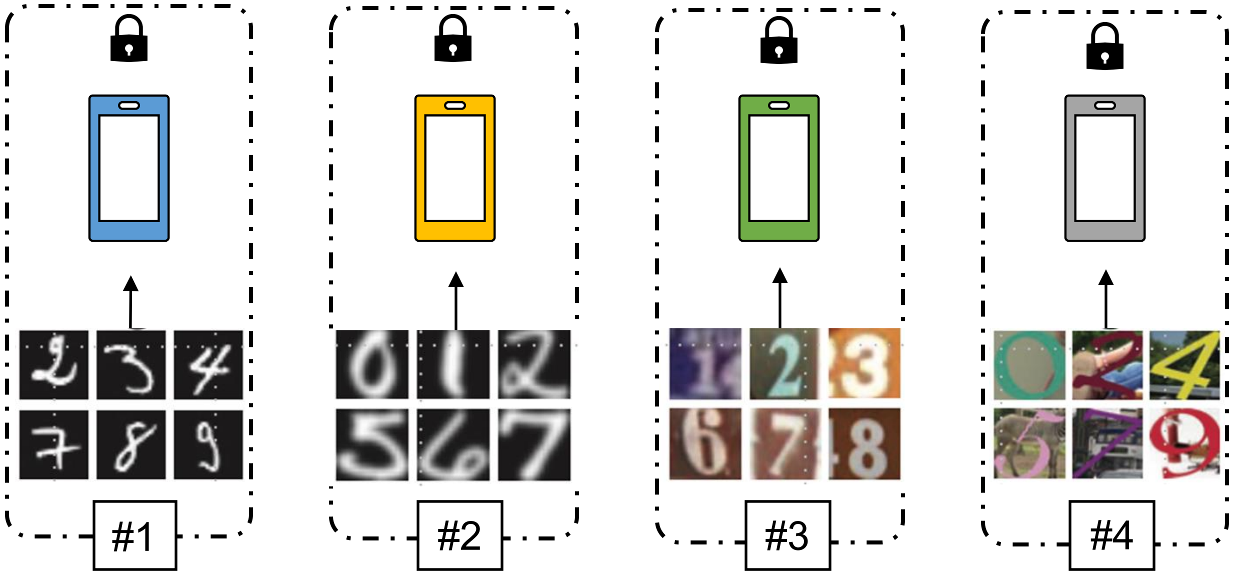


Federated Domain Shift

- In practical FL systems, data across clients may come from different domains

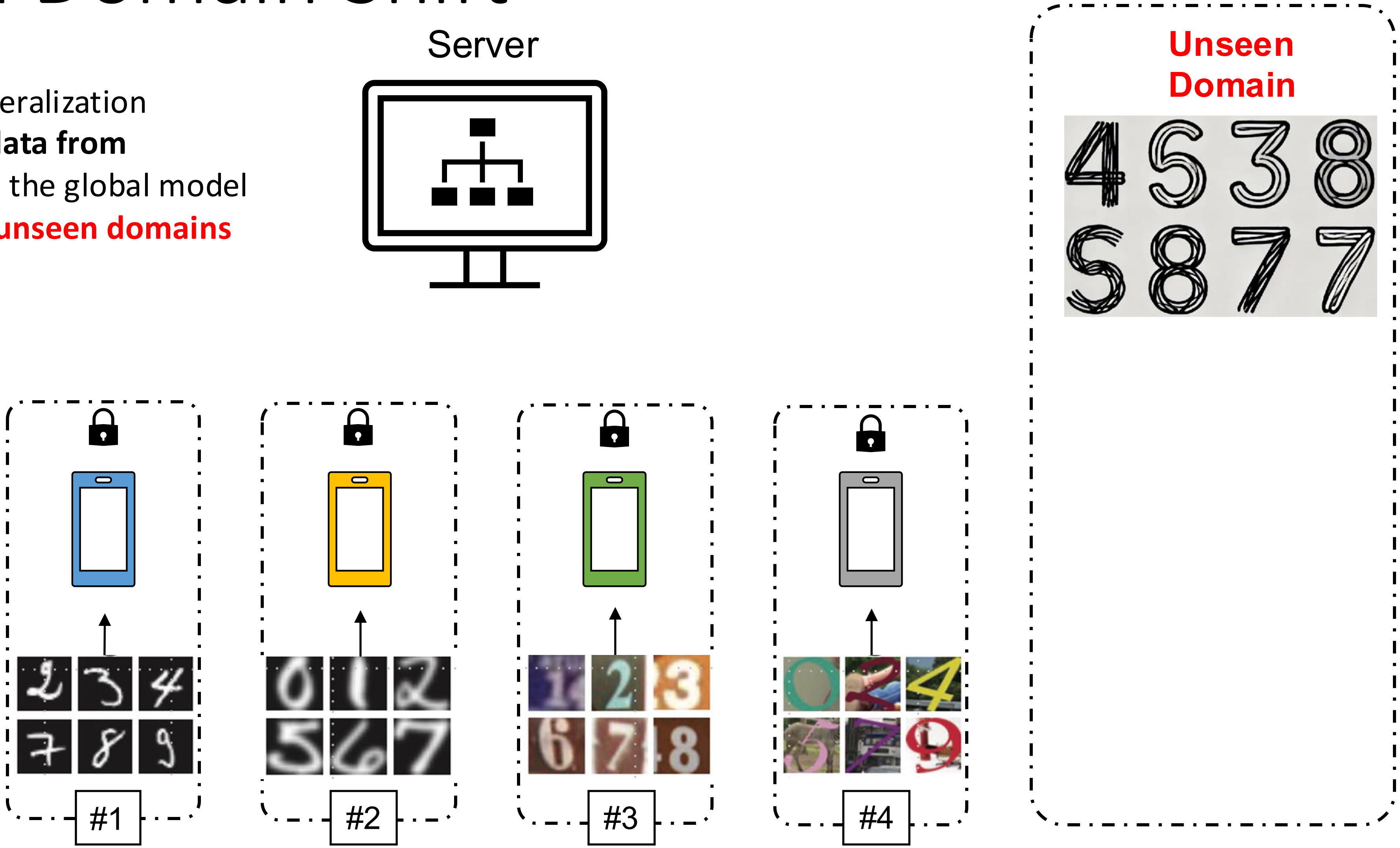
Domains: shape, color, brightness, artistic factors





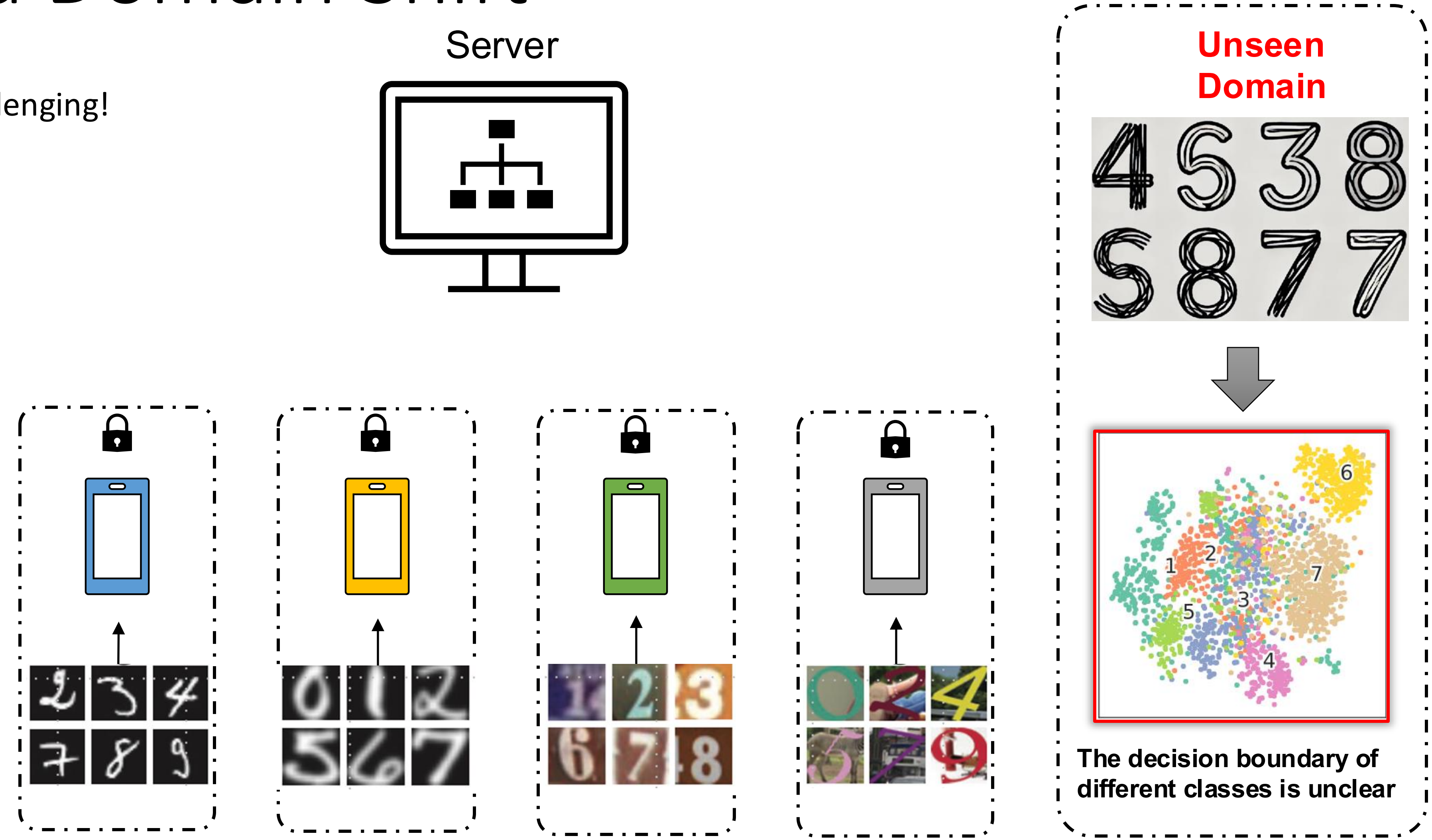
Federated Domain Shift

- Federated Domain Generalization (FedDG): **clients have data from different domains,** and the global model should predict well on **unseen domains**



Federated Domain Shift

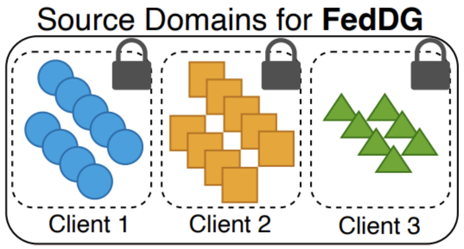
✗ However, FedDG is challenging!

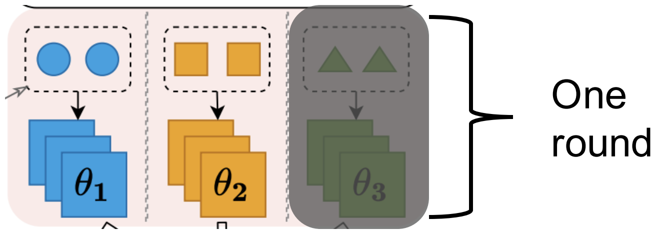


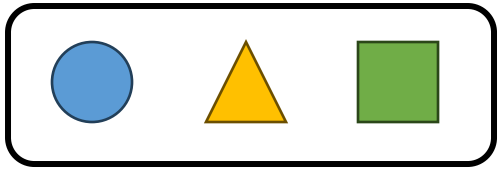
VANDERBILT UNIVERSITY

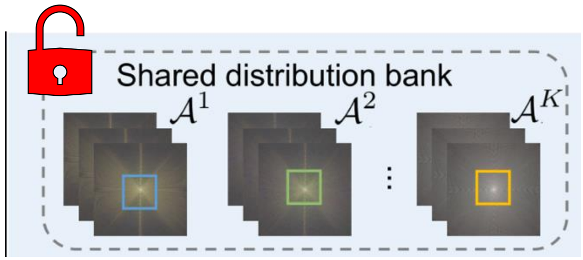
Previous FedDG Methods: Weaknesses

- Designated for domain-isolated settings
  - lowering variance of local losses, regularization, etc.
  - **each client** only contains data from **one domain**: #clients = #domains
  - limited performance under **client sampling**
- Evaluations are confined to testing on datasets with **limited domain diversity**
- Cross-sharing information can lead to **privacy breaches**
  - Augmentation using per-sample information



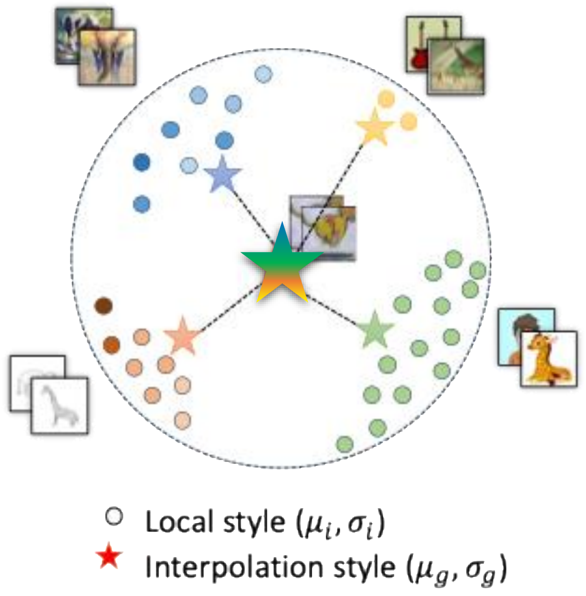


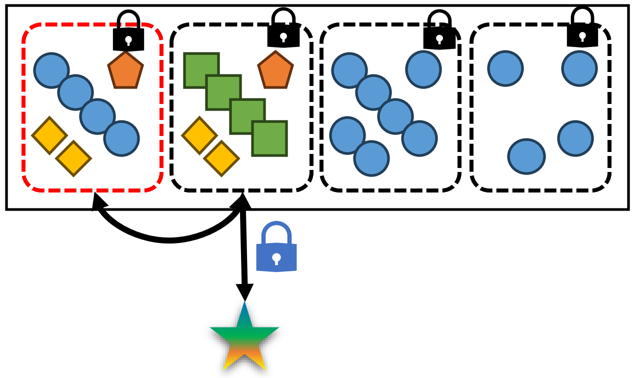




PARDON: Contribution

- Handling domain-shift more **EFFECTIVELY**
  - Better utility on unseen domains
- … while keeping **PRIVACY** of client data
- … while demonstrating **GENERALIZABILITY** through improved utility with:
  - Decreased proportion of client sampling
  - Diverse distribution of domains across clients
  - Large number of domains

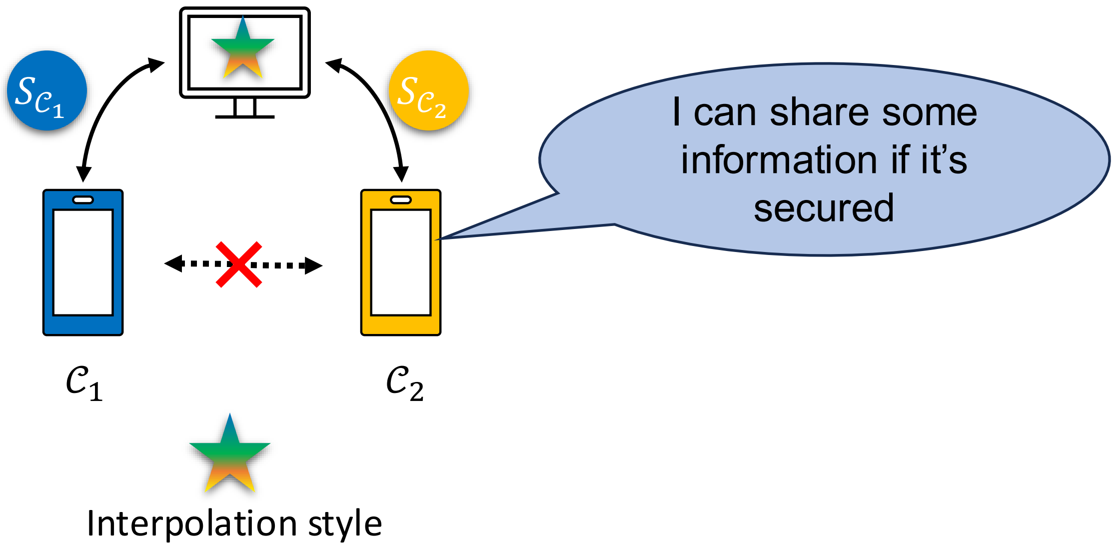


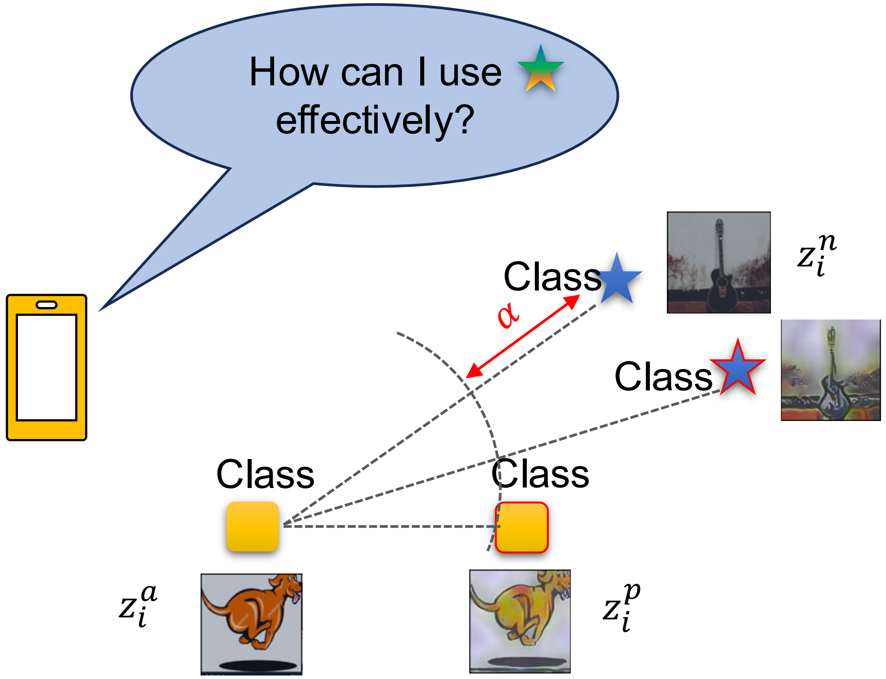


VANDERBILT UNIVERSITY

Key Insights for PARDON

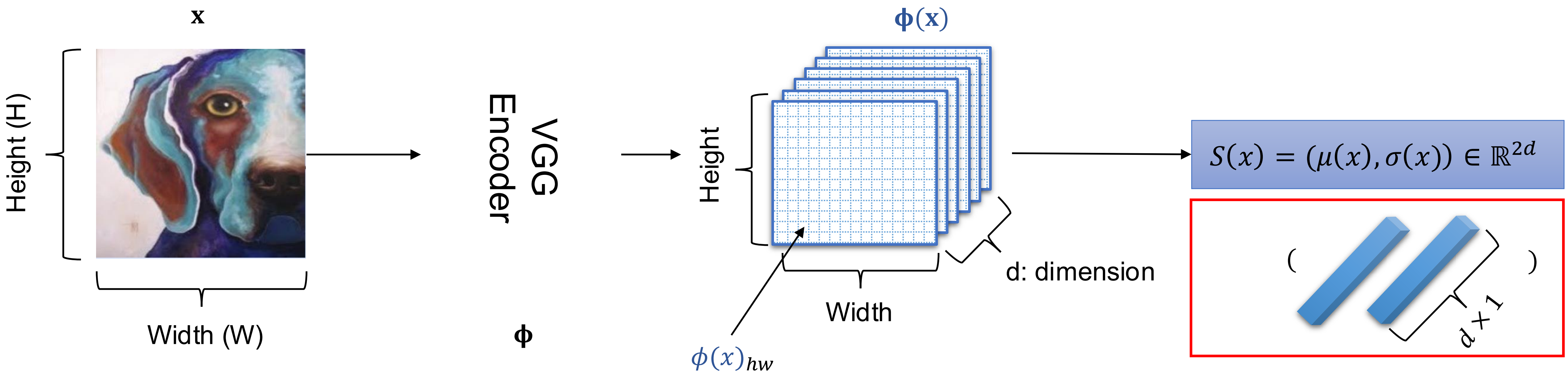
- Securely extract interpolation information
  - Only share as much information as needed (i.e., no specifics of samples)
- *Contrastive learning* on style transferred images
  - Forces model to learn domain-agnostic features





VANDERBILT UNIVERSITY

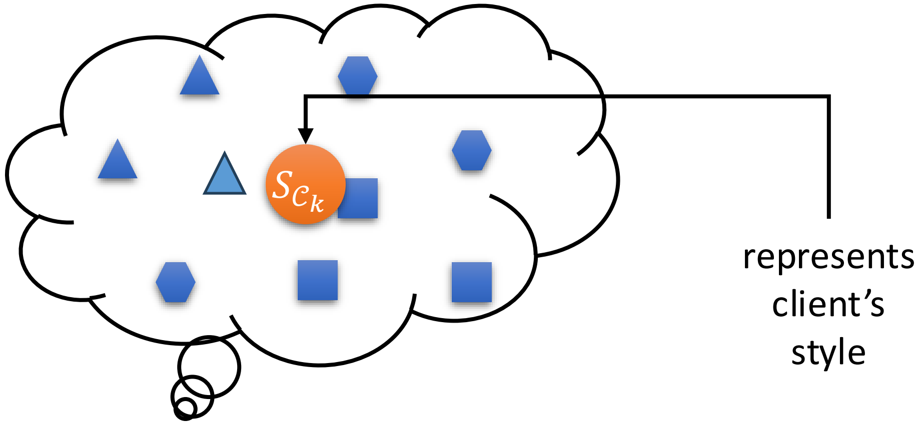
Interpolation Style



- Each style in each client is abstracted as a vector of (*mean, variance*) pairs for channels of pixels
  - Removes critical details of individual samples
  - **Interpolation Style can help clients transfer styles without sharing data**

Interpolation Style

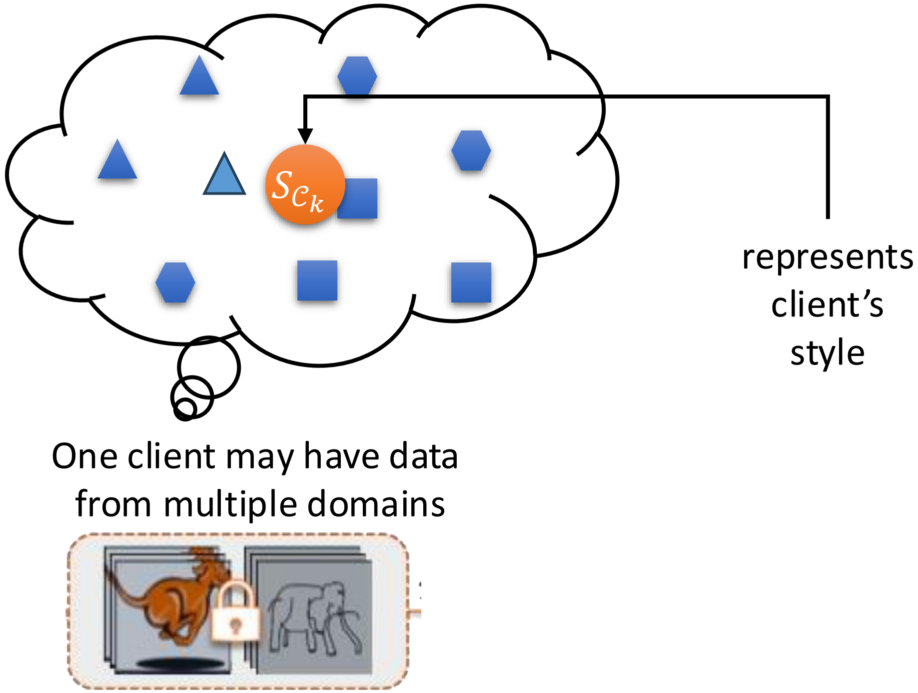
- Hierarchical unsupervised style clustering:
  - **Intra-client level**
  - Inter-client level



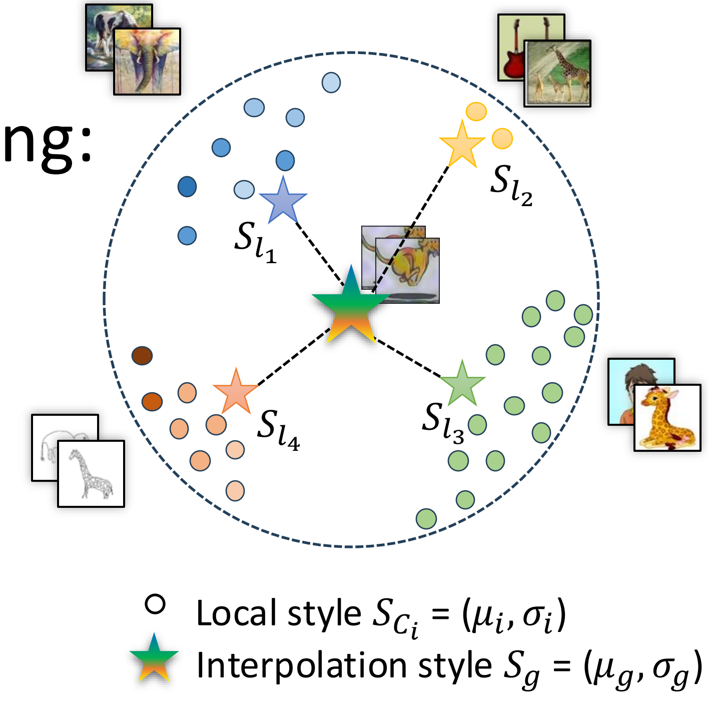
One client may have data from multiple domains

Interpolation Style

- Hierarchical unsupervised style clustering:
  - Intra-client level
  - **Inter-client level**



One client may have data from multiple domains



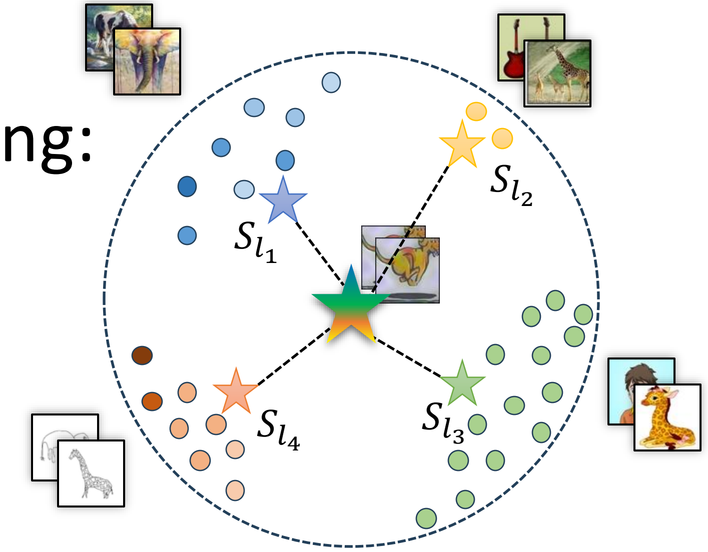
- ➢ **client-level** clustering
- ➢ There can be clients having similar styles

Interpolation Style

- Hierarchical unsupervised style clustering:
  - **Intra-client level**
  - **Inter-client level**

Interpolation style

✓ Domains with low cardinality
✓ Fair and comprehensive knowledge across all domains



○ Local style S_{C_i} = (μ_i, σ_i)
Interpolation style S_g = (μ_g, σ_g)

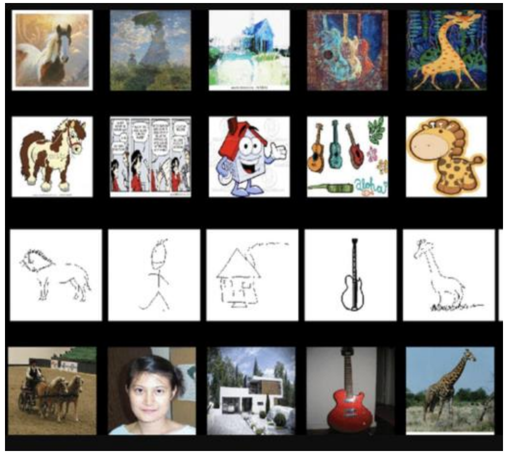
- **client-level** clustering
- there can be clients having similar styles

VANDERBILT UNIVERSITY

Experimental Setup

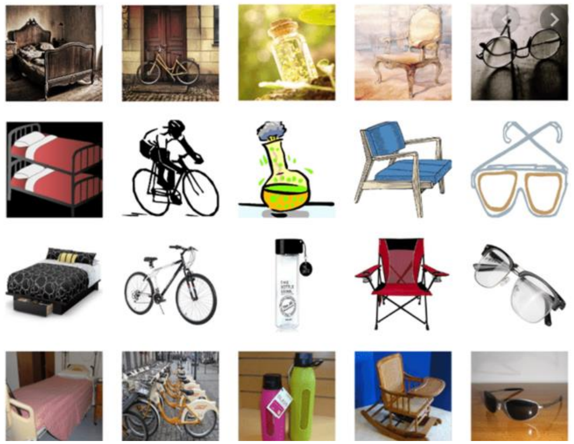
- Datasets: PACS, Office-Home, and IWildCam
  - Small number of domains and large number domains

PACS[1]



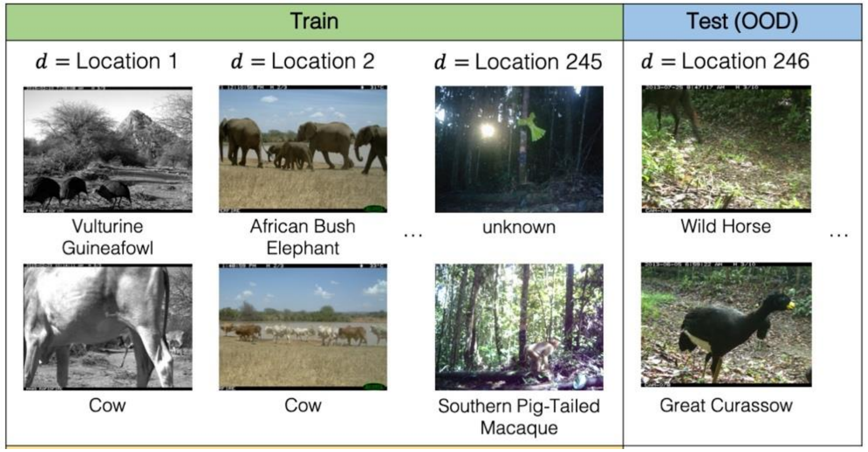
4 Domains – 7 Classes

Office-Home[1]



4 Domains – 65 Classes

IWildCam[2]



323 domains - 182 classes

[1] Li, D.; Yang, Y.; Song, Y.-Z.; and Hospedales, T. M. 2017. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, 5542–5550.

[2] Koh, P. W.; Sagawa, S.; Marklund, H.; Xie, S. M.; Zhang, M.; Balsubramani, A.; Hu, W.; Yasunaga, M.; Phillips, R. L.; Gao, I.; et al. 2021. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*, 5637–5664. PMLR.

VANDERBILT UNIVERSITY

Experimental Results

1. **RQ1:** Does PARDON perform well compared to SOTA?
    a. "Leave One Domain Out" (LODO) Split
    b. "Leave Two Domains Out" (LOTO) Split
    c. Large-domain Dataset: I-WildCam
2. **RQ2:** Can PARDON perform well across many settings?
    a. Different client sampling
    b. Different domain heterogeneity
3. **RQ3:** How well does PARDON improve client data privacy?
4. **RQ4:** What is the computational overhead of using PARDON?

VANDERBILT
UNIVERSITY

Experimental Results

1. **RQ1:** Does PARDON perform well compared to SOTA?
   a. "Leave One Domain Out" (LODO) Split
   b. "Leave Two Domains Out" (LTDO) Split
   c. Large-domain Dataset: I-WildCam
2. **RQ2:** Can PARDON perform well across many settings?
   a. Different client sampling
   b. Different domain heterogeneity
3. **RQ3:** How well does PARDON improve client data privacy?
4. **RQ4:** What is the computational overhead of using PARDON?

VANDERBILT UNIVERSITY

RQ1.b. Comparison with SOTA: LTDO

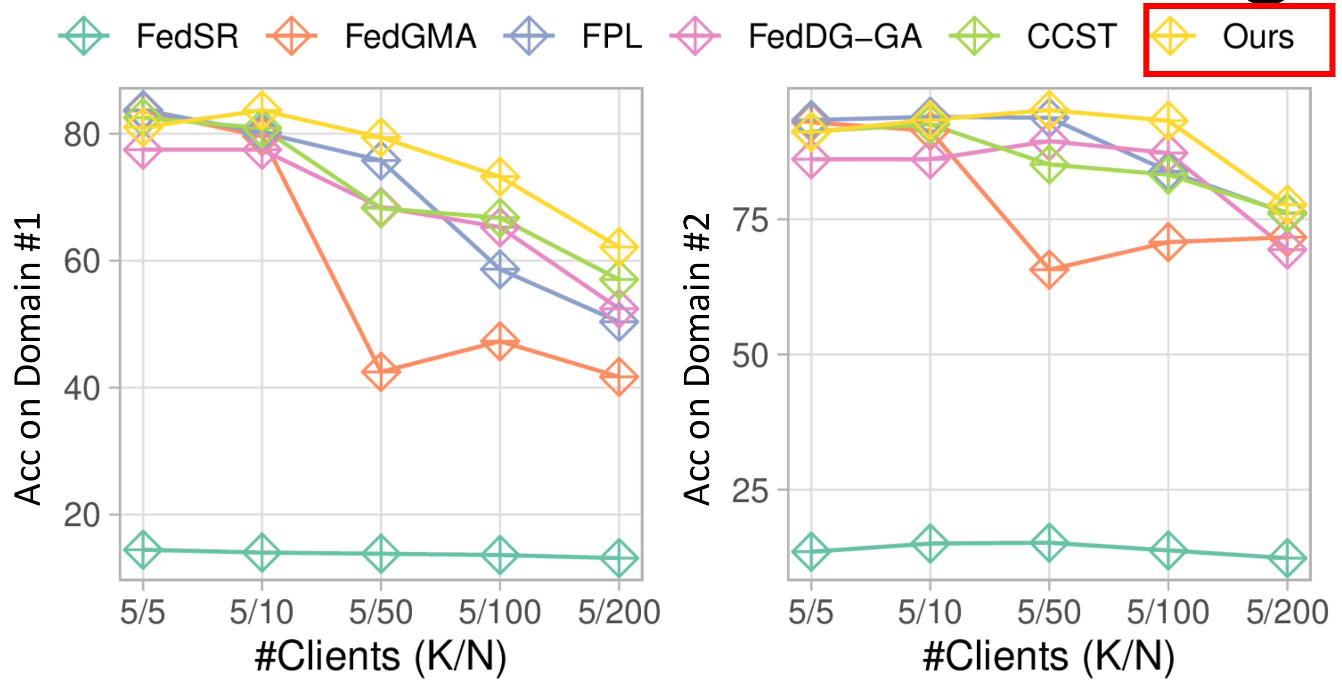
Dataset	Methods	Validation Accuracy					Test Accuracy				
		A	P	C	S	AVG	P	S	A	C	AVG
PACS	FedSR	14.80%	14.67%	13.39%	13.36%	14.06%	13.80%	13.97%	14.55%	12.87%	13.80%
	FedGMA	39.31%	94.13%	63.95%	36.22%	58.40%	73.83%	64.85%	73.10%	52.73%	66.13%
	FPL	77.93%	94.49%	64.97%	31.61%	67.25%	93.53%	55.97%	62.01%	51.83%	65.84%
	FedDG-GA	64.99%	92.46%	63.18%	32.73%	63.34%	84.19%	63.55%	61.87%	48.08%	64.42%
	CCST	68.51%	96.41%	59.26%	35.68%	64.97%	86.89%	59.91%	71.78%	50.94%	67.38%
	Ours	73.63%	95.57%	69.41%	35.91%	**68.63%**	93.05%	66.20%	71.73%	53.11%	**71.02%**
		C	A	R	P	AVG	A	P	C	R	AVG
OfficeHome	FedSR	1.40%	1.24%	1.36%	1.31%	1.33%	1.15%	1.14%	1.34%	1.33%	1.24%
	FedGMA	43.18%	54.92%	66.81%	54.29%	54.80%	55.71%	66.43%	39.91%	56.83%	54.72%
	FPL	45.72%	56.82%	69.45%	46.18%	54.54%	59.95%	65.22%	43.99%	52.54%	55.43%
	FedDG-GA	38.99%	51.38%	63.85%	48.07%	50.57%	51.63%	62.38%	36.68%	54.79%	51.37%
	CCST	44.81%	52.48%	62.29%	49.85%	52.36%	52.20%	62.79%	38.37%	54.88%	52.06%
	Ours	46.74%	58.84%	71.13%	55.31%	**58.01%**	60.09%	67.54%	45.41%	61.62%	**58.67%**

PACS: *A: Art-Painting, P: Photo, C: Cartoon, S: Sketch*
OfficeHome: *C: Clipart, A: Art, R: Real World, P: Product*

With smaller number of training domains,
PARDON outperforms other baselines by a larger margin

VANDERBILT UNIVERSITY

RQ2.a. Different FL Settings: Client Sampling



The higher the ratio K:N is, the larger the amount of data that participates in each training round.

✗ **Baseline:** strong performance with no client sampling (5/5) but diminished performance with increasingly sparse sampling

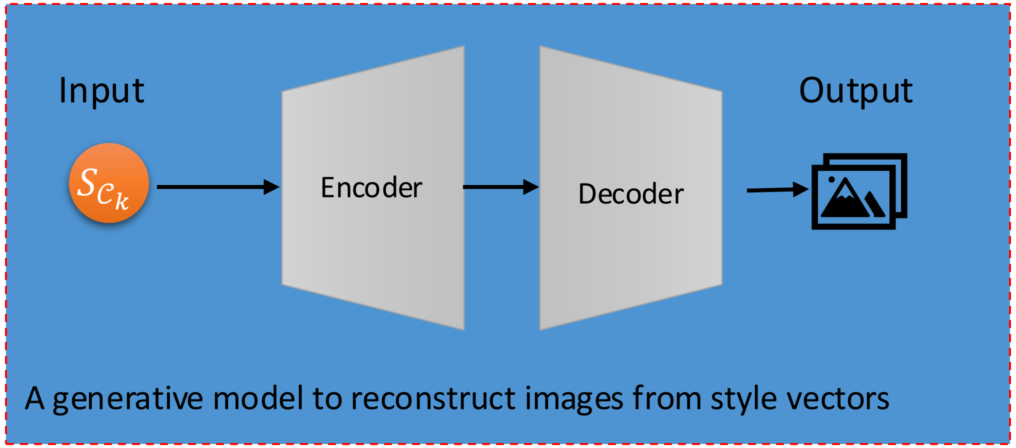
✓ **PARDON:** outperforms in terms of stability and efficiency

RQ3: Security Analysis

Adversary

HAS: style vectors S_{c_k}

WANTS: private training images



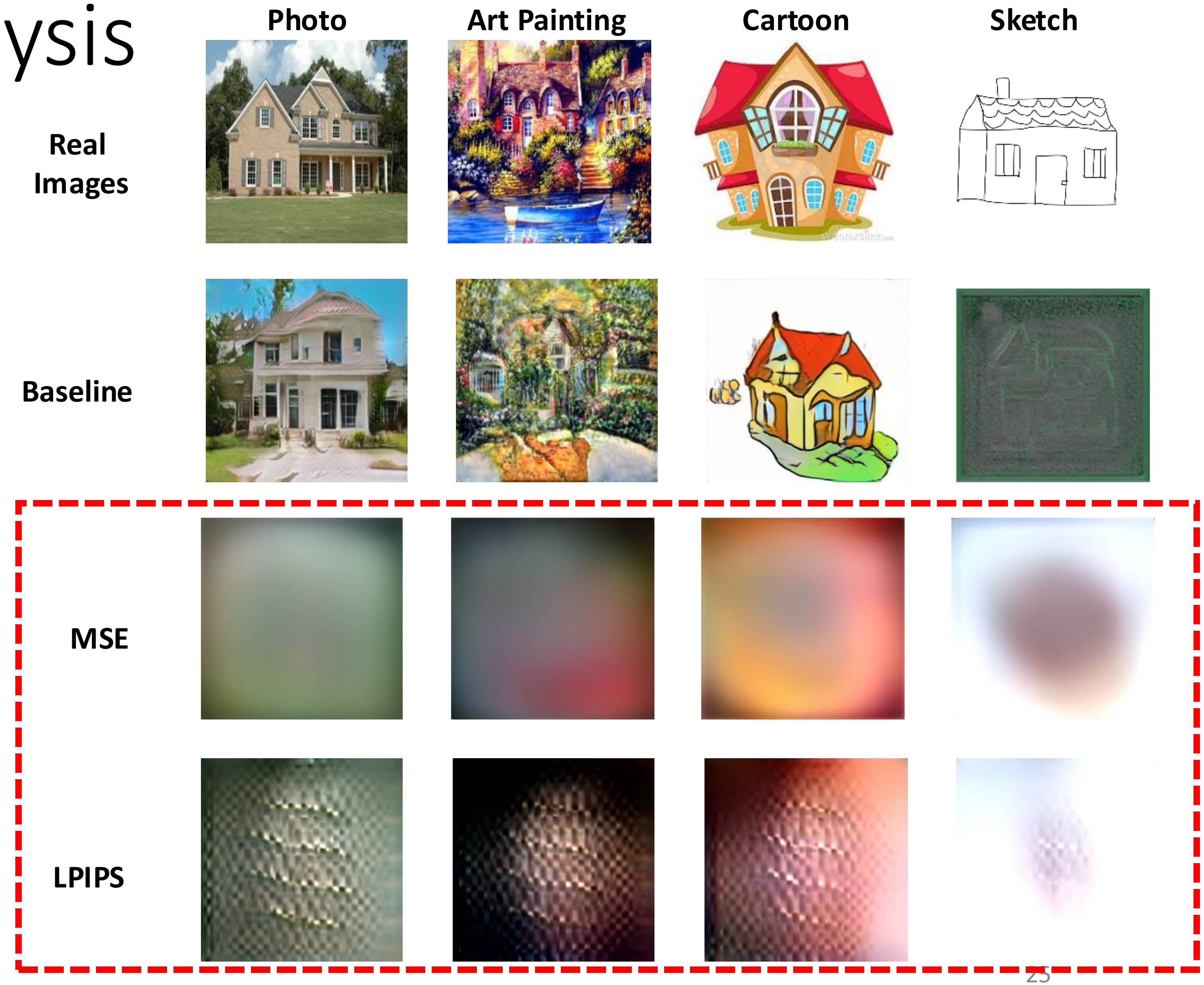
Baseline Case:
GAN model is trained on REAL images from clients (to assume a strong adversary)

VANDERBILT UNIVERSITY

RQ3: Security Analysis

- Reconstructed images are **far different** from the real images
- It is **non-trivial** to reconstruct a client's data using only style vectors as in our approach

Reconstructed images by using style vectors and public images



VANDERBILT UNIVERSITY

Summarizing PARDON

1. PARDON outperforms SOTA on both LODO and LTDO and when applied to a large number of domains
2. PARDON maintains improved performance under client sampling and with increased domain heterogeneity
3. PARDON’s style vectors create challenges for violating data privacy across clients

PARDON can be applied to malware classification settings, enabling style transfer across datasets to unify data and develop novel plausible samples

VANDERBILT UNIVERSITY

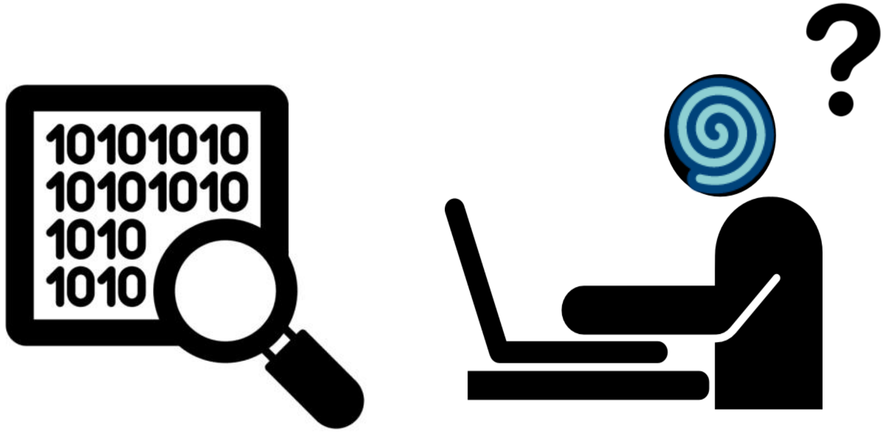
Outline

- Malware Analysis and Classification
- Domain Generalization in Federated Learning
  - In ICDCS’25: Judy Nguyen, Taylor Johnson, Kevin Leach
    PARDON: Privacy-Aware and Robust Federated Domain Generalization
- Effectiveness of Reverse Engineering Tools
  - In DSN’25: Yuwei Yang, Skyler Grandel, et al.:
    A Human Study of Automatically Generated Decompiler Annotations.
- LLM-based Enhancement of Decompilation
  - In TOSEM: Skyler Grandel, Scott Andersen, et. al:
    Expertise-Guided Context Generation to Enhance Code Comprehension

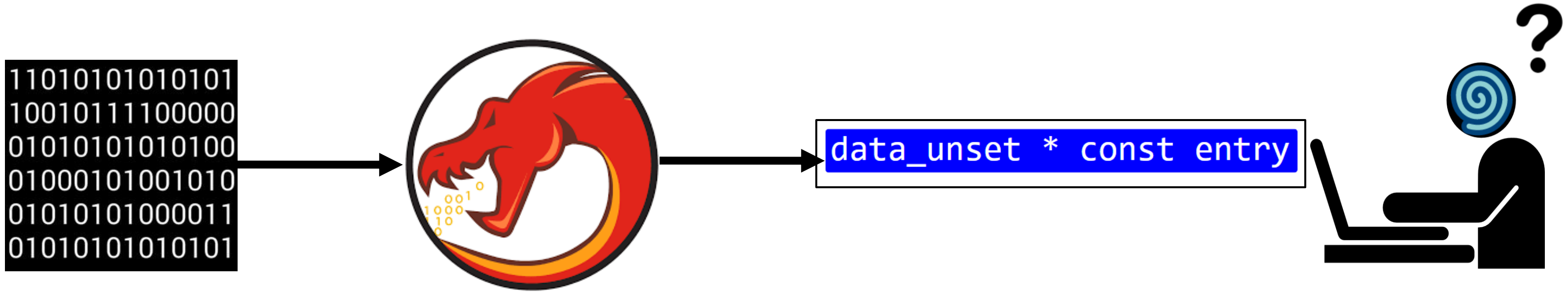
VANDERBILT UNIVERSITY

Manual Analysis

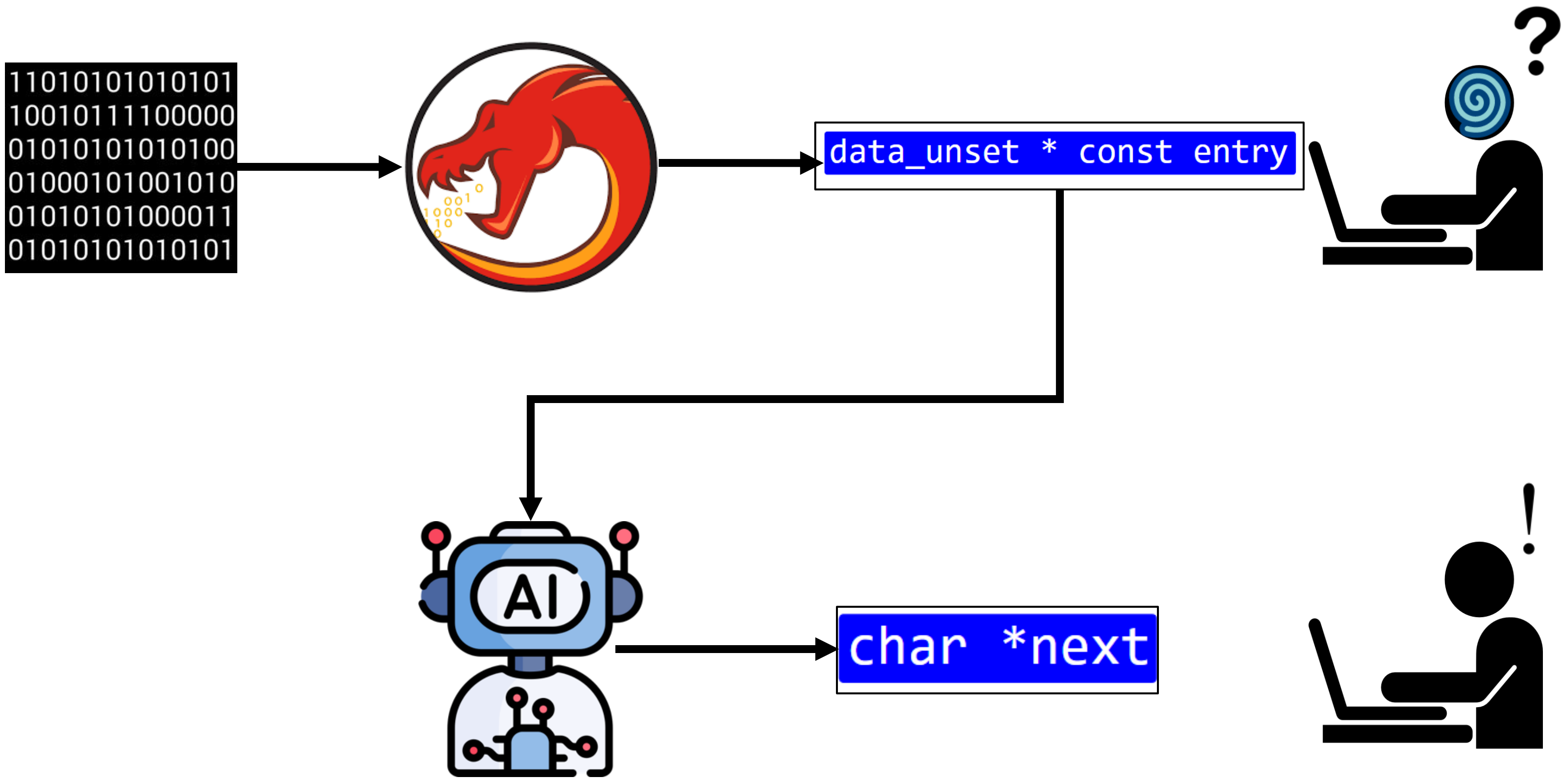
- **Automated malware analysis** isn't always enough.
  - Further **manual analysis** may be required after classification.
- How can we make this process **as easy as possible**?



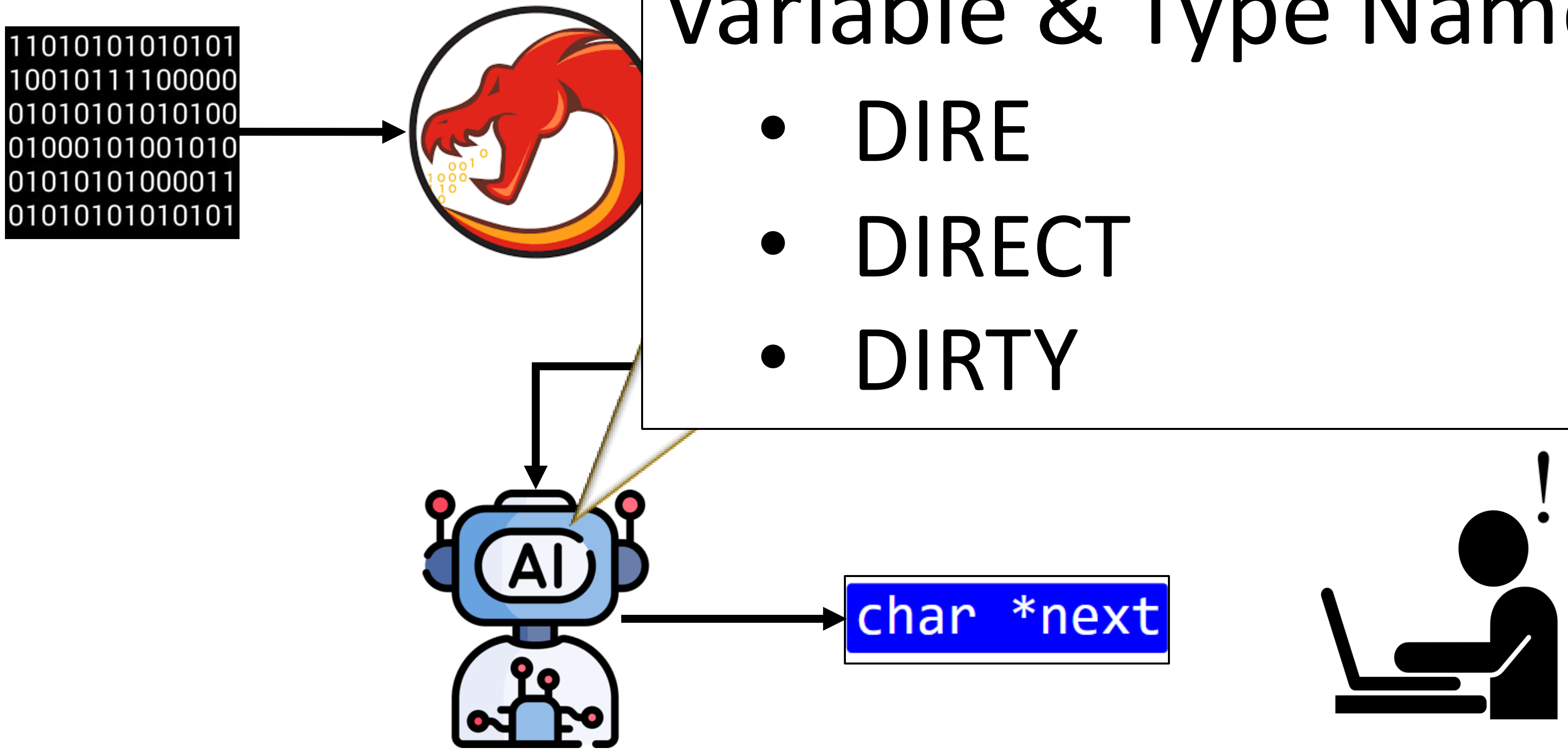
Manual Analysis



ML Assisted Manual Analysis



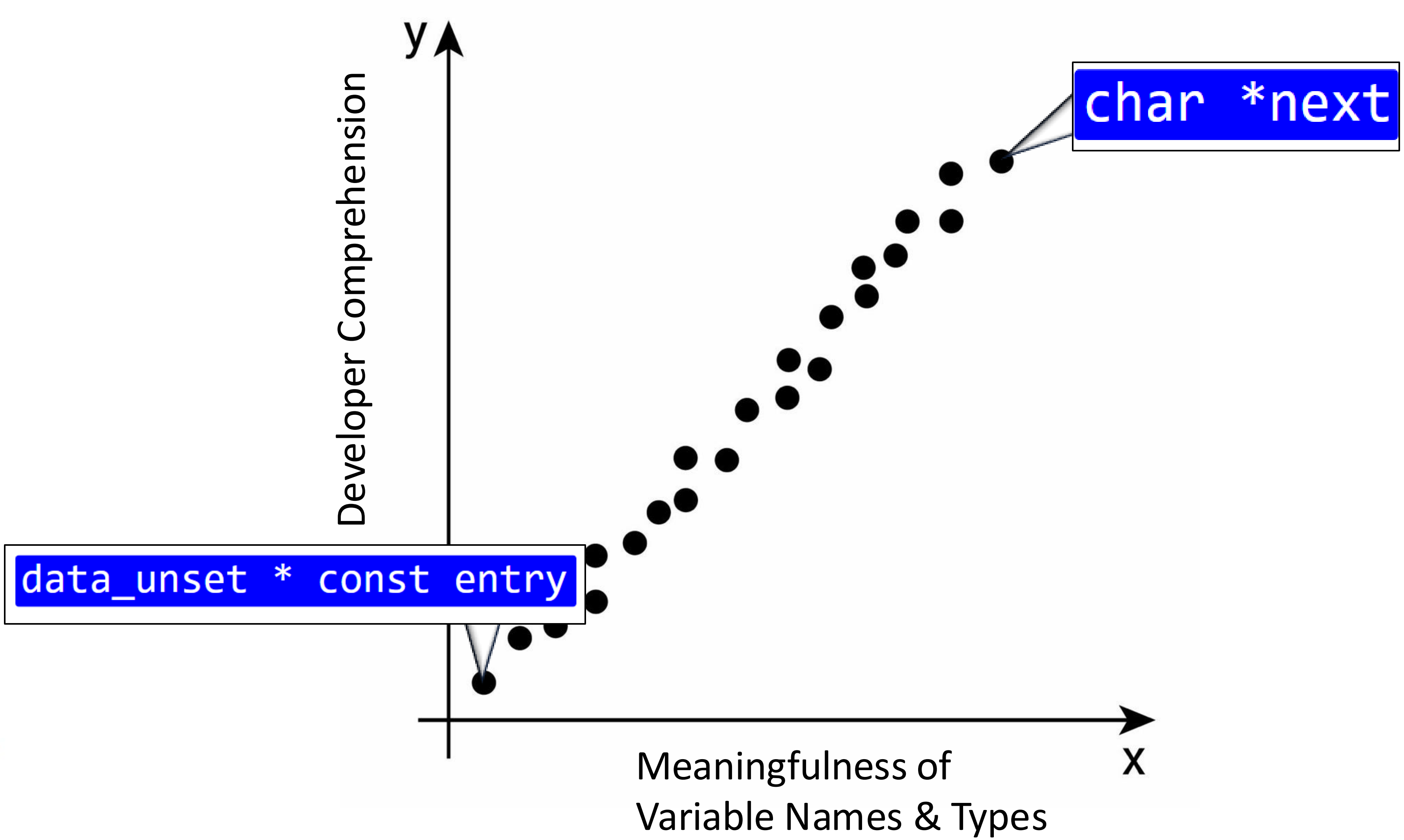
ML Assisted Manual Analysis



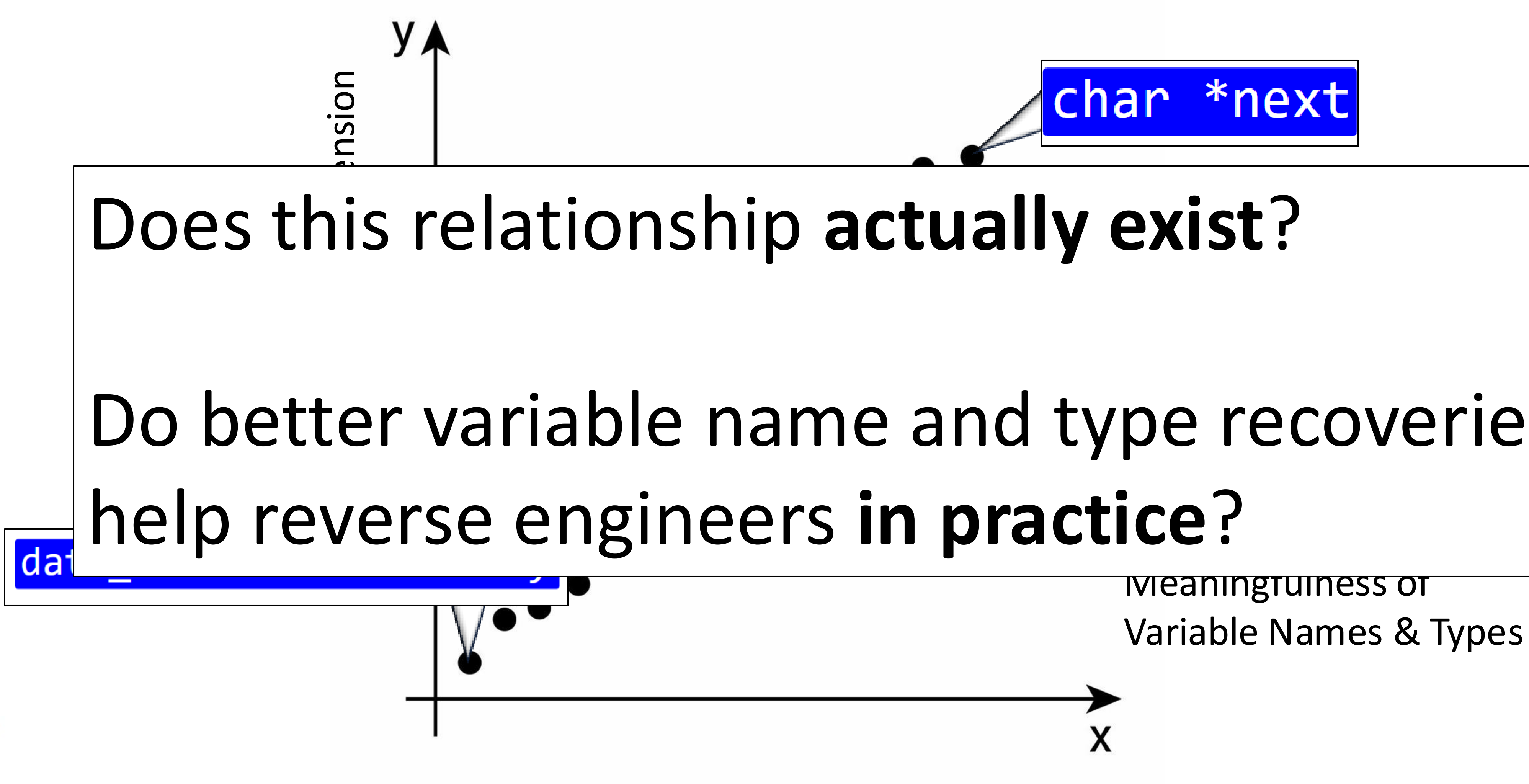
Variable & Type Name Recovery

- DIRE
- DIRECT
- DIRTY

Assumed Relationship



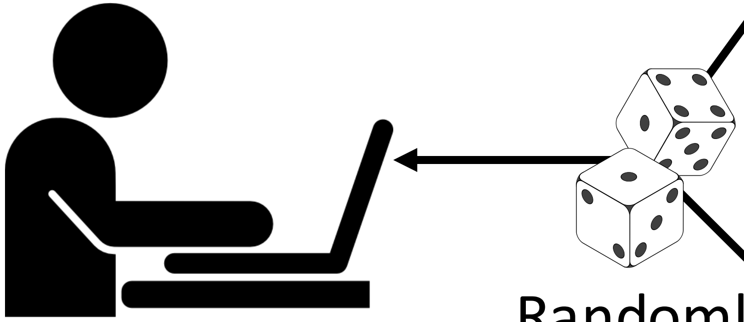
Assumed Relationship



Does this relationship **actually exist**?

Do better variable name and type recoveries help reverse engineers **in practice**?

Study Design



Randomly Assigned

```
__int64 __fastcall array_extract_element_klen(__int64 a1,
        __int64 a2, unsigned int a3) {
    //...
    //...
    int index;
    __int64 v7;
    //...
    if ( index < 0 )
        return 0LL;
    v7 = *(_QWORD *)(8LL * index + *(_QWORD *)(a1 + 8));
    //...
    return v7;
}
```

(a) Hex-Rays Output

```
char *__fastcall array_extract_element_klen(array_t_0 *
        array, void *key, int index) {
    //...
    int indexa;
    int ret;
    char *next;
    //...
    if ( indexa < 0 )
        return 0LL;
    next=*(char**)(8LL*indexa + *(_QWORD*)&array->size);
     //...
    return next;
}
```

(b) DIRTY Output

Study Design

```
1 __int64 __fastcall array_extract_element_klen(__int64 a1,
        __int64 a2, unsigned int a3) {
2     //...
3     //...
4     int index;
```

```
(_QWORD *)(a1 + 8));
```

```
nt_klen(array_t_0 *
```

```
WORD*)&array->size);
```

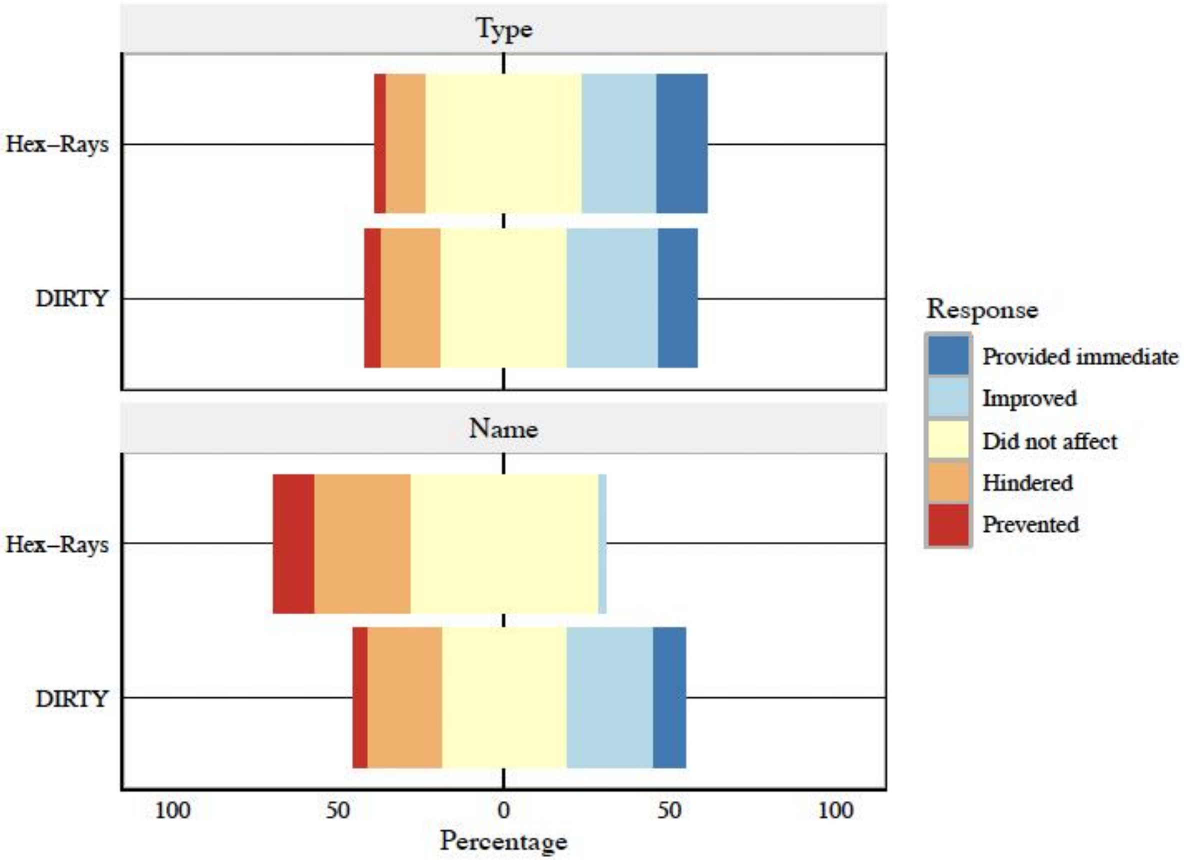
If `a1 + 8` points to an array and the `array_get_index` call on line 8 returns an index, what is the purpose of the `if` and `memmove` on lines 13-17?

Please write your answer here:

Developer correctness and time taken to complete each task are used to measure comprehension.

Results: User Preference

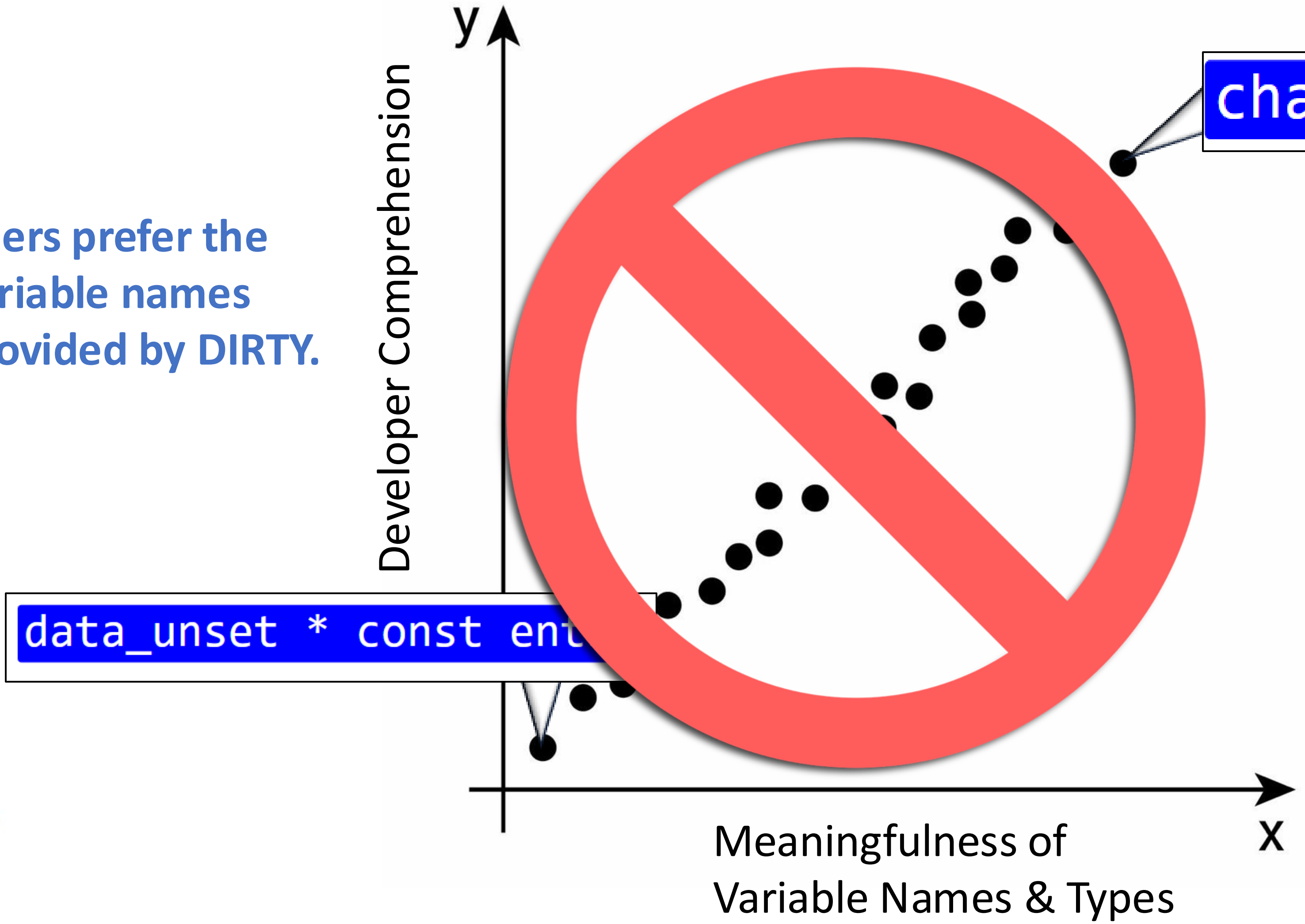
Users prefer the variable names provided by DIRTY.



Spearman correlation:
p-value = 0.02459 and ρ = 0.1035.

Results: User Performance

Users prefer the variable names provided by DIRTY.



But there is ❌ **No statistically significant** difference in accuracy and correctness or time.

Results: Similarity Metrics & Code Comprehension

Correlation Between Similarity Metrics and Participant Time Taken on DIRTY Annotated Code Snippets

Similarity Metric	Correlation	ρ	p-value
BLEU	↗	0.2568	**0.0010**
codeBLEU	↗	0.2568	**0.0010**
Jaccard Similarity	↗	0.5193	<**0.0001**
BERTScore F1	↗	0.006	0.94
VarCLR	↗	0.2568	**0.0010**
Human Evaluation (Variables)	↗	0.2611	<**0.0001**
Human Evaluation (Types)	↗	0.1065	**0.0004542**

Correlation Between Similarity Metrics and Participant Correctness on DIRTY Annotated Code Snippets

Similarity Metric	Correlation	ρ	p-value
BLEU	↗	0.0792	0.3437
codeBLEU	↗	0.0792	0.3437
Jaccard Similarity	↘	-0.2173	**0.0086**
BERTScore F1	↗	0.2302	**0.0053**
VarCLR	↗	0.0792	0.3437
Human Evaluation (Variables)	↘	-0.1241	<**0.0001**
Human Evaluation (Types)	↗	0.0517	0.1072

Results: Similarity Metrics & Code Comprehension

Correlation Between Similarity Metrics and Participant Time Taken on DIRTY Annotated Code Snippets

Similarity Metric	**Correlation**	ρ	**p-value**
BLEU	↗	0.2568	**0.0010**
codeBLEU	↗	0.2568	**0.0010**
Jaccard Similarity	↗	0.5193	<**0.0001**
BERTScore F1	↗	0.006	0.94
VarCLR	↗	0.2568	**0.0010**
Human Evaluation (Variables)	↗	0.2611	<**0.0001**
Human Evaluation (Types)	↗	0.1065	**0.0004542**

Correlation Between Similarity Metrics and Participant Correctness on DIRTY Annotated Code Snippets

Similarity Metric	**Correlation**	ρ	**p-value**
BLEU	↗	0.0792	0.3437
codeBLEU	↗	0.0792	0.3437
Jaccard Similarity	↘	-0.2173	**0.0086**
BERTScore F1	↗	0.2302	**0.0053**
VarCLR	↗	0.0792	0.3437
Human Evaluation (Variables)	↘	-0.1241	<**0.0001**
Human Evaluation (Types)	↗	0.0517	0.1072

Results: Similarity Metrics & Code Comprehension

Correlation Between Similarity Metrics and Participant Time Taken on DIRTY Annotated Code Snippets

Similarity Metric	**Correlation**	ρ	**p-value**
BLEU	↗	0.2568	**0.0010**
codeBLEU	↗	0.2568	**0.0010**
Jaccard Similarity	↗	0.5193	<**0.0001**
BERTScore F1	↗	0.006	0.94
VarCLR	↗	0.2568	**0.0010**
Human Evaluation (Variables)	↗	0.2611	<**0.0001**
Human Evaluation (Types)	↗	0.1065	**0.0004542**

Correlation Between Similarity Metrics and Participant Correctness on DIRTY Annotated Code Snippets

Similarity Metric	**Correlation**	ρ	**p-value**
BLEU	↗	0.0792	0.3437
codeBLEU	↗	0.0792	0.3437
Jaccard Similarity	↘	-0.2173	**0.0086**
BERTScore F1	↗	0.2302	**0.0053**
VarCLR	↗	0.0792	0.3437
Human Evaluation (Variables)	↘	-0.1241	<**0.0001**
Human Evaluation (Types)	↗	0.0517	0.1072

Results: Similarity Metrics & Code Comprehension

Correlation Between Similarity Metrics and Participant Time Taken on DIRTY Annotated Code Snippets

Similarity Metric	Correlation	ρ	p-value
BLEU	↗	0.2568	**0.0010**
codeBLEU	↗	0.2568	**0.0010**
Jaccard Similarity	↗	0.5193	<**0.0001**
BERTScore F1	↗	0.006	0.94
VarCLR	↗	0.2568	**0.0010**
Human Evaluation (Variables)	↗	0.2611	<**0.0001**
Human Evaluation (Types)	↗	0.1065	**0.0004542**

Correlation Between Similarity Metrics and Participant Correctness on DIRTY Annotated Code Snippets

Similarity Metric	Correlation	ρ	p-value
BLEU	↗	0.0792	0.3437
codeBLEU	↗	0.0792	0.3437
Jaccard Similarity	↘	-0.2173	**0.0086**
BERTScore F1	↗	0.2302	**0.0053**
VarCLR	↗	0.0792	0.3437
Human Evaluation (Variables)	↘	-0.1241	<**0.0001**
Human Evaluation (Types)	↗	0.0517	0.1072

Insight: commonly used metrics may not effectively reflect human code comprehension.

Contributions

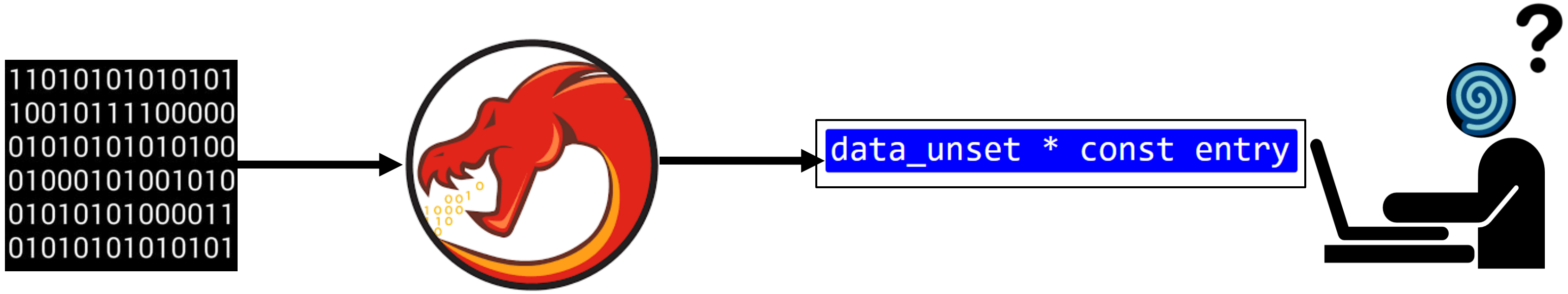
- **Empirical Evaluation of ML Performance Metrics**:
  We show that commonly used machine learning metrics for variable and type name recovery do not correlate with actual improvements in human code comprehension.
- **User Preference for ML-Augmented Decompiler Output**:
  Despite limited performance gains, users consistently preferred decompiled code enhanced with machine-generated names and types.
- **Developer Performance and Enriched Code Analysis**:
  Our study finds no significant improvement in task performance from enriched decompiler output, suggesting current augmentation techniques have limited practical effectiveness.

Outline

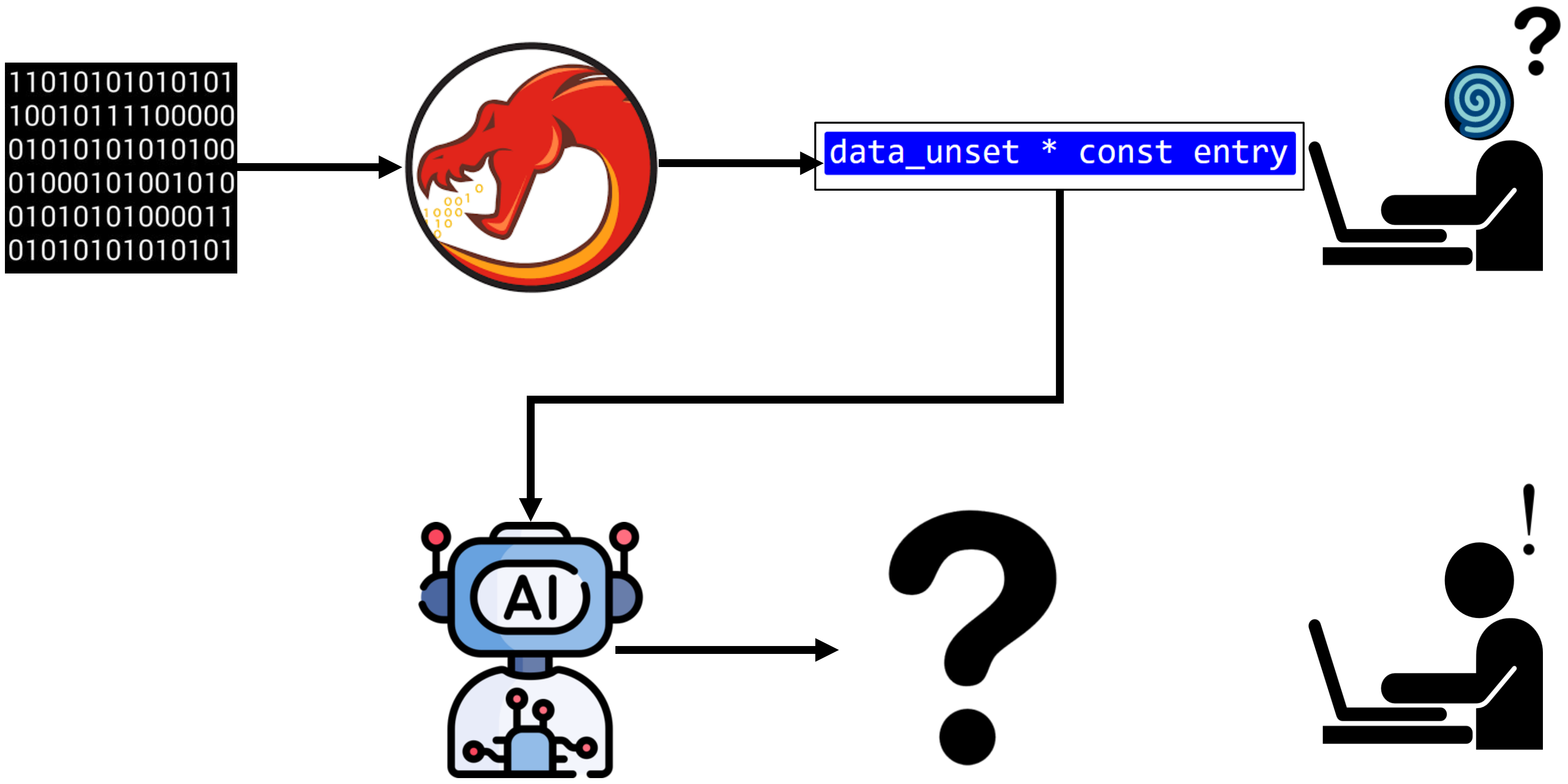
- Malware Analysis and Classification
- Domain Generalization in Federated Learning
  - In ICDCS’25: Judy Nguyen, Taylor Johnson, Kevin Leach
    PARDON: Privacy-Aware and Robust Federated Domain Generalization
- Effectiveness of Reverse Engineering Tools
  - In DSN’25: Yuwei Yang, Skyler Grandel, et al.:
    A Human Study of Automatically Generated Decompiler Annotations.
- LLM-based Enhancement of Decompilation
  - In TOSEM: Skyler Grandel, Scott Andersen, et. al:
    Expertise-Guided Context Generation to Enhance Code Comprehension

VANDERBILT UNIVERSITY

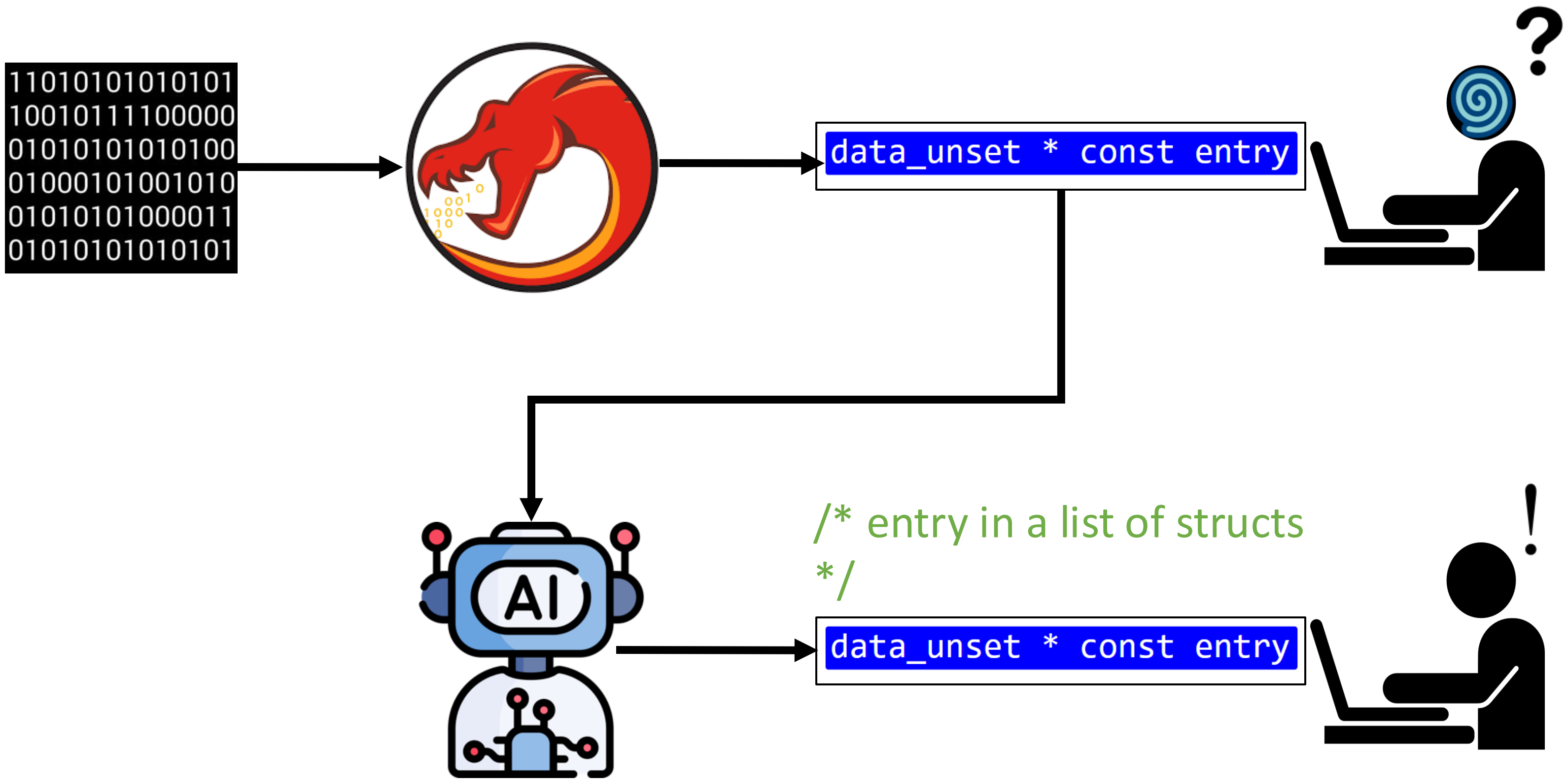
Manual Analysis



ML Assisted Manual Analysis



ML Assisted Manual Analysis

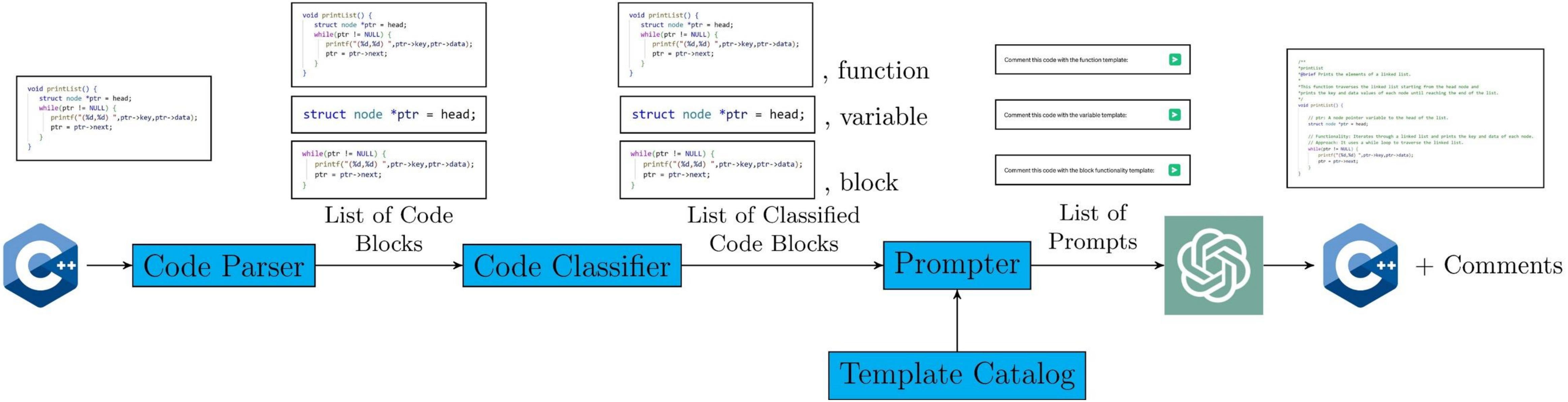


Expertise-Guided Context Generation

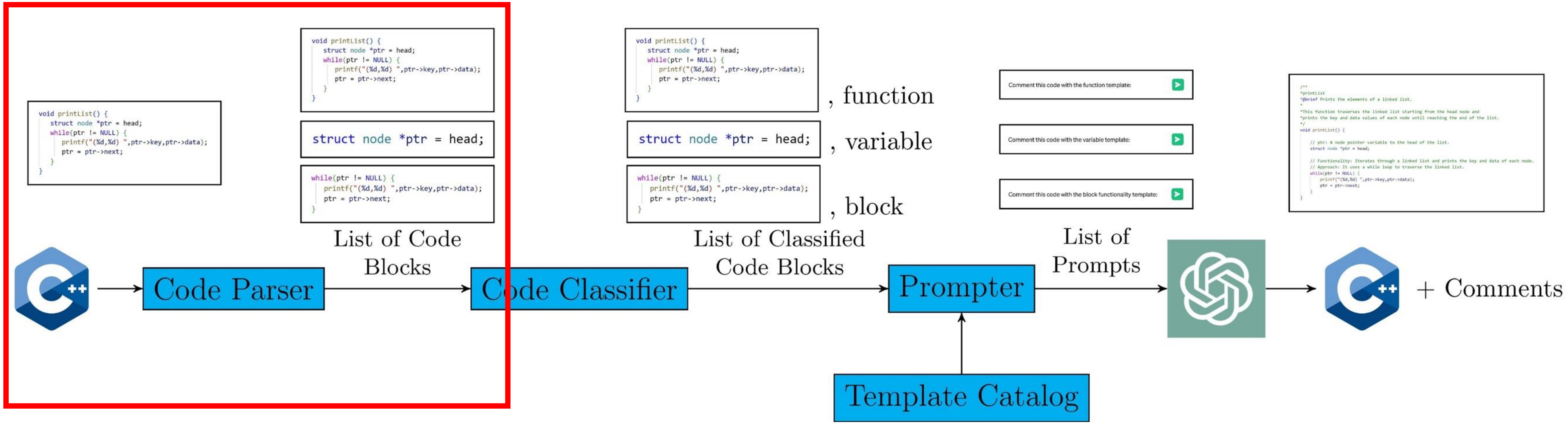
- **Core Idea:** Leverage LLMs augmented with developer insights to pick *where* and *what* to annotate in code.

- Use **experts** to identify
  - where useful comments are located in code
  - common structures of useful comments

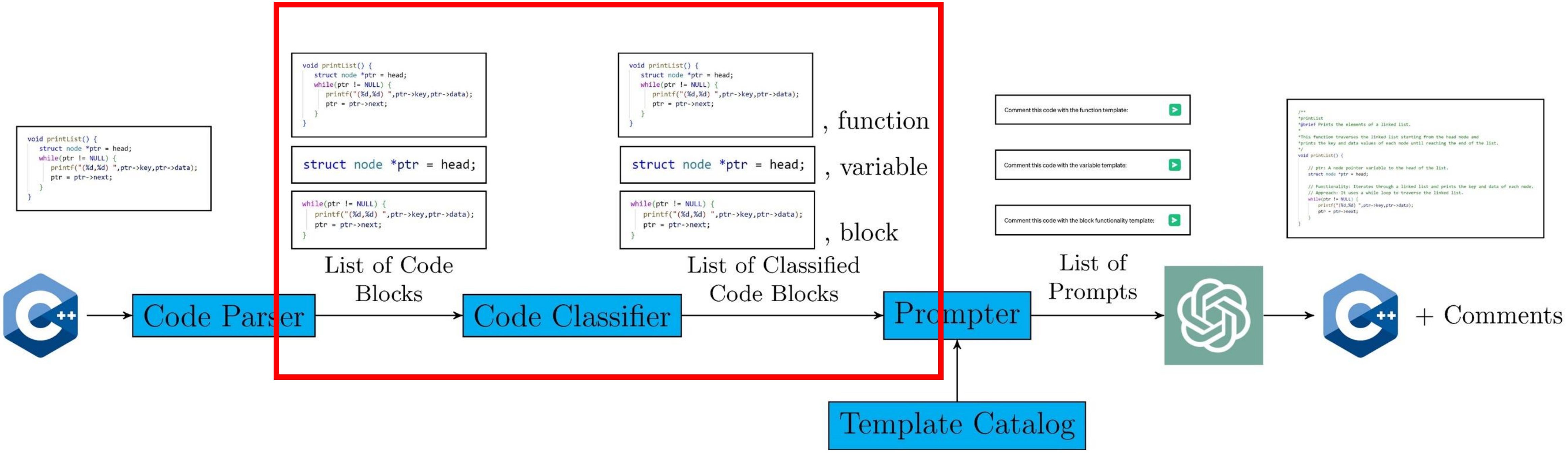
Expertise-Guided Context Generation



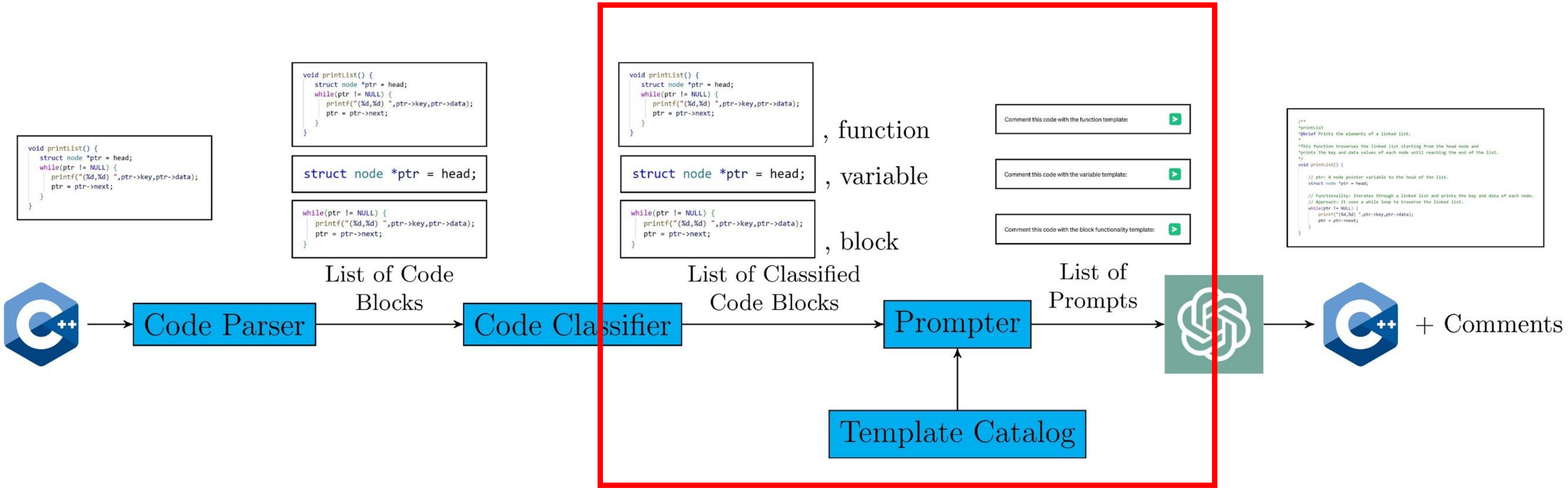
Expertise-Guided Context Generation



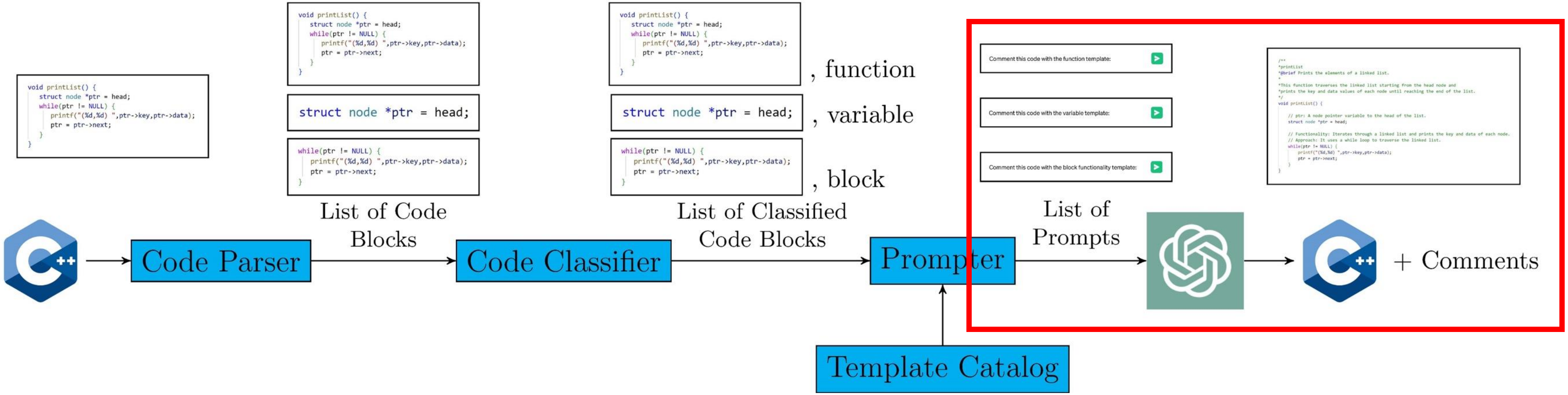
Expertise-Guided Context Generation



Expertise-Guided Context Generation

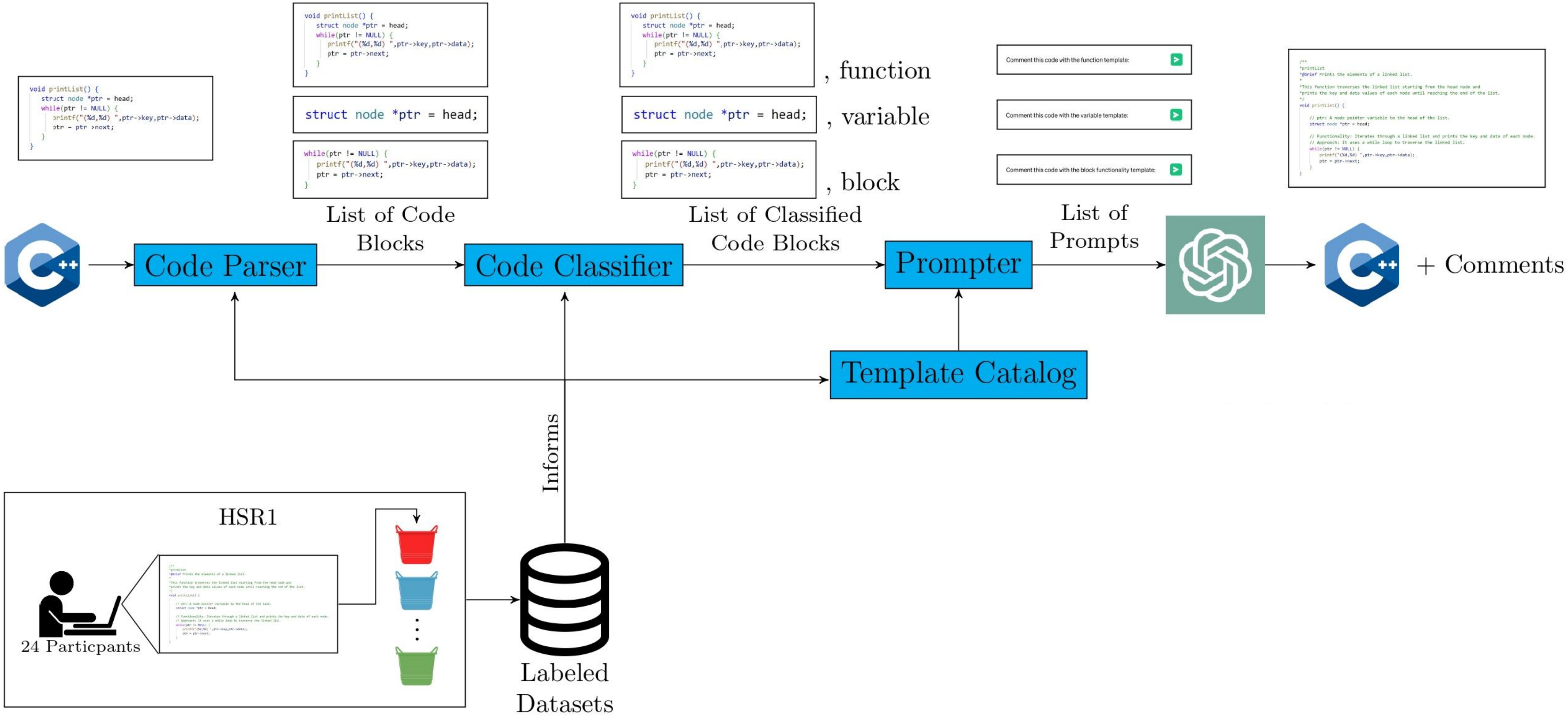


Expertise-Guided Context Generation



VANDERBILT UNIVERSITY

Expertise-Guided Context Generation



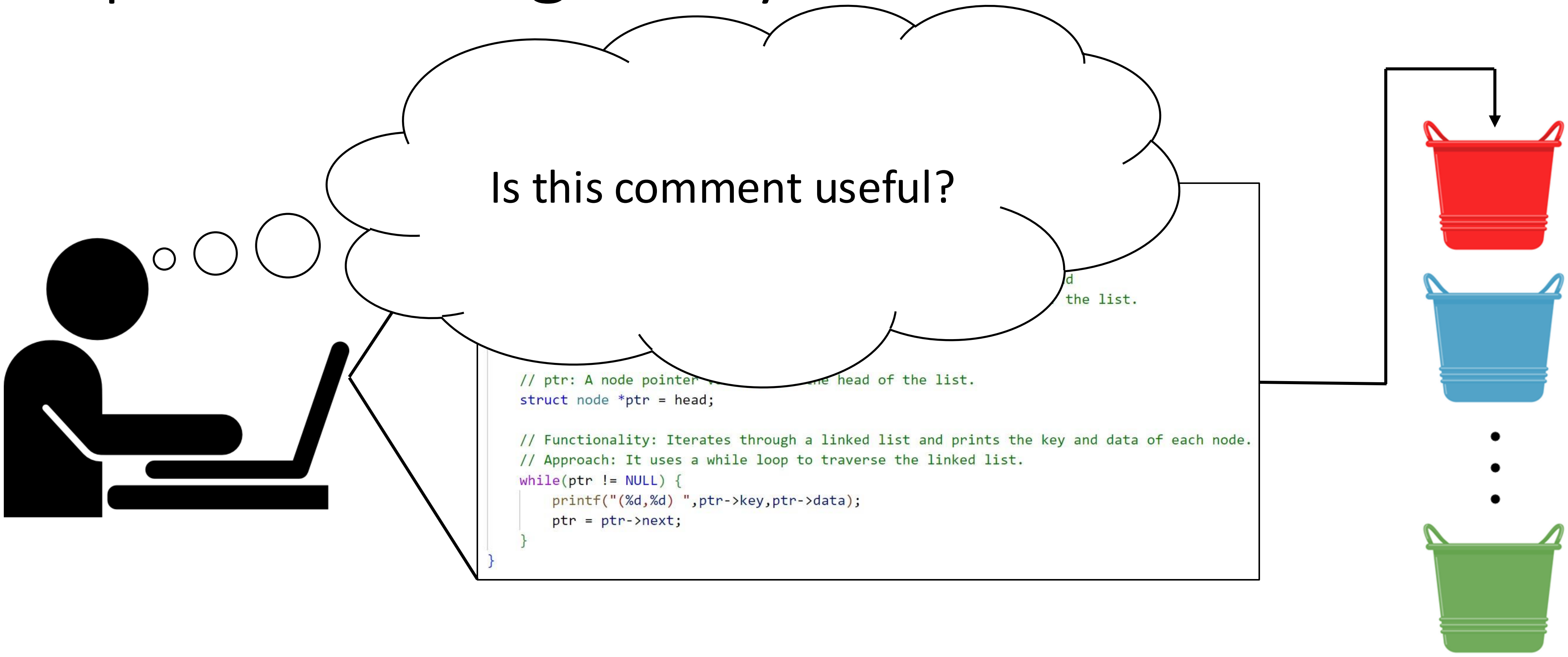
Expert Labeling Study

```
/**
*printList
*@brief Prints the elements of a linked list.
*
*This function traverses the linked list starting from the head node and
*prints the key and data values of each node until reaching the end of the list.
*/
void printList() {

    // ptr: A node pointer variable to the head of the list.
    struct node *ptr = head;

    // Functionality: Iterates through a linked list and prints the key and data of each node.
    // Approach: It uses a while loop to traverse the linked list.
    while(ptr != NULL) {
        printf("(%d,%d) ",ptr->key,ptr->data);
        ptr = ptr->next;
    }
}
```

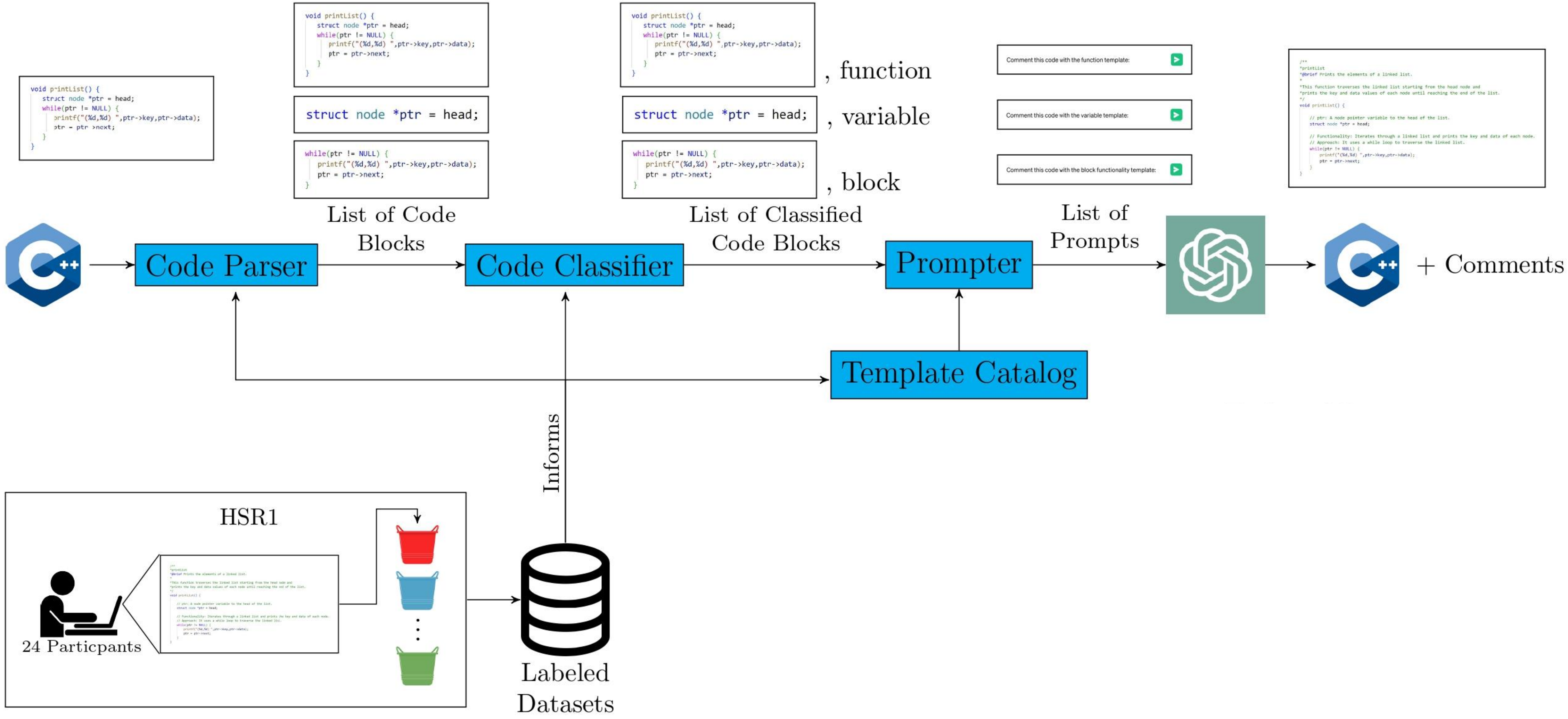
Expert Labeling Study



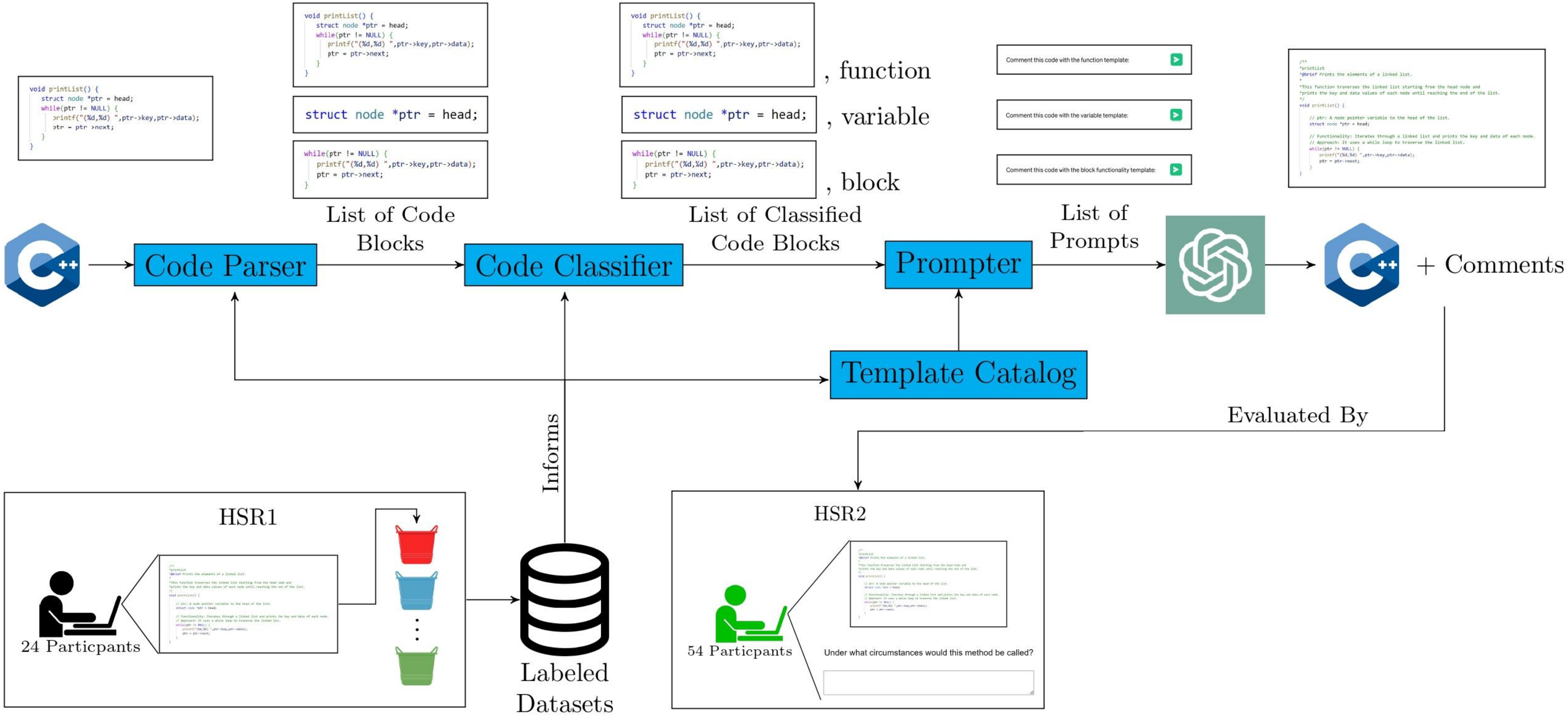
Full Comment Classification Schema		
Category	Description	% Useful
Function	Comments that describe an entire function, often in Javadoc or similar format. They tend to summarize the function and note parameters and return values.	92.18%
Variable	Comments that describe a variable, constant, or literal. They often note what a variable represents.	66.67%
Snippet Functionality	Comments that are inline and summarize or describe the functionality of code.	94.21%
Branch	Comments that describe possible branches of execution, often summarizing if-else or switch statements. This also includes preconditions for branches.	91.01%
Reasoning	Comments that describe the reasoning behind implementation decisions, but not functionality.	74.05%
Quirk	Comments that contain a random quirk of the code, author jokes, or some other unimportant information.	9.33%
Use Guidelines	Comments that guide readers on using or accessing functions, containers, or variables, or they detail compilation or execution instructions.	35.59%
Source	Comments that describe the source of the code. These might note that the code was copied from some documentation or StackOverflow link.	6.06%
Copyright	Comments that contain copyright, licensing, and author information, typically at the top of a file.	8.54%
Section	Comments that provide a section label for multiple functions, test cases, or global or class variables.	47.37%
Code	Commented out code.	10.00%
Task	Comments that note future work, e.g. a TODO or FIXME.	14.06%

Full Comment Classification Schema		
Category	Description	% Useful
Function	Comments that describe an entire function, often in Javadoc or similar format. They tend to summarize the function and note parameters and return values.	92.18%
Variable	Comments that describe a variable, constant, or literal. They often note what a variable represents.	66.67%
Snippet Functionality	Comments that are inline and summarize or describe the functionality of code.	94.21%
Branch	Comments that describe possible branches of execution, often summarizing if-else or switch statements. This also includes preconditions for branches.	91.01%
Reasoning	Comments that describe the reasoning behind implementation decisions, but not functionality.	74.05%
Quirk	Comments that contain a random quirk of the code, author jokes, or some other unimportant information.	9.33%
Use Guidelines	Comments that guide readers on using or accessing functions, containers, or variables, or they detail compilation or execution instructions.	35.59%
Source	Comments that describe the source of the code. These might note that the code was copied from some documentation or StackOverflow link.	6.06%
Copyright	Comments that contain copyright, licensing, and author information, typically at the top of a file.	8.54%
Section	Comments that provide a section label for multiple functions, test cases, or global or class variables.	47.37%
Code	Commented out code.	10.00%
Task	Comments that note future work, e.g. a TODO or FIXME.	14.06%

Expertise-Guided Context Generation



Expertise-Guided Context Generation



Empirical Evaluation in Practice

```
/**
*printList
*@brief Prints the elements of a linked list.
*
*This function traverses the linked list starting from the head node and
*prints the key and data values of each node until reaching the end of the list.
*/
void printList() {

    // ptr: A node pointer variable to the head of the list.
    struct node *ptr = head;

    // Functionality: Iterates through a linked list and prints the key and data of each node.
    // Approach: It uses a while loop to traverse the linked list.
    while(ptr != NULL) {
        printf("(%d,%d) ",ptr->key,ptr->data);
        ptr = ptr->next;
    }
}
```

Under what circumstances would this method be called?

Empirical Evaluation in Practice

- Evaluate programmer comprehension of code annotated by
  - ComCat
  - Humans
  - “Standard” ChatGPT

- Comprehension is measured through 3 tasks:
  - Short Answer
  - Code Writing
  - Debugging

Results: Developer Performance Using ComCat

	Compared to Human Generated		Compared to Standard ChatGPT	
Question Type	Change in Correctness	*p*	Change in Correctness	*p*
Short Answer	**+13.6%**	**<0.001**	**+14.3%**	**<0.001**
Code Writing	**+18.7%**	**<0.001**	**+30.9%**	**<0.001**
Debugging	**+7.0%**	**0.041**	**+11.4%**	**0.025**
Overall	**+13.3%**	**<0.001**	**+16.3%**	**<0.001**

Results: Developer Performance Using ComCat

	Compared to Human Generated		Compared to Standard ChatGPT	
Question Type	Change in Correctness	*p*	Change in Correctness	*p*
Short Answer	**+13.6%**	**<0.001**	**+14.3%**	**<0.001**
Code Writing	**+18.7%**	**<0.001**	**+30.9%**	**<0.001**
Debugging	**+7.0%**	**0.041**	**+11.4%**	**0.025**
Overall	**+13.3%**	**<0.001**	**+16.3%**	**<0.001**

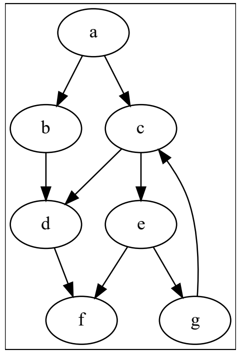
Key Takeaways

- ComCat Boosts Understandability
  - +13–16% improvement in **human** participants' comprehension accuracy.
  - Annotations are targeted and aligned with **developer mental models**.
- In Malware Analysis
  - Decompiled malware often has **zero semantic clues**—ComCat's inline annotations directly fill that gap.
  - Better comprehension → better identification of malicious routines.

Next Steps

- **Integration** with decompilers
- Future evaluation: User Study with **Malware Analysts**

- Extended capabilities
  - **Domain-Adapted** Templates/Prompts.
  - Combine with dynamic traces to annotate **control-flow graphs.**



VANDERBILT UNIVERSITY

VNN-Comp and MalBeWare Benchmark

- Previously-reported verification techniques for malware classifiers has been incorporated into VNN-Comp
  - MalBeWare benchmark available
  - Upcoming VNN-COMP’25 at CAV/SAIV 2025

VANDERBILT UNIVERSITY

Summary

- Malware samples are too voluminous for scalable analysis
- Automated analysis can be thwarted by perturbations and evasiveness

- Generating interpolation styles for diverse datasets can help improve robustness and generalizability of neural classifiers

- Techniques that attempt to improve decompilation do not necessarily improve reverse engineer comprehension, complicating analysis efforts

Skyler Grandel (skyler.h.grandel@vanderbilt.edu)

Kevin Leach (kevin.leach@vanderbilt.edu)   Taylor Johnson (taylor.johnson@vanderbilt.edu)

VANDERBILT
UNIVERSITY