NSFlow: An End-to-End FPGA Framework with Scalable Dataflow Architecture for Neuro-Symbolic AI

Hanchen Yang[1*], Zishen Wan[1*], Ritik Raj[1], Joongun Park[1], Ziwei Li[1], Ananda Samajdar[2], Arijit Raychowdhury[1], Tushar Krishna[1]
(*Equal Contributions)

[1]Georgia Tech, Atlanta, GA, USA
[2]IBM Research, Yorktown Heights, NY, USA

62 THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY CEDA sigda

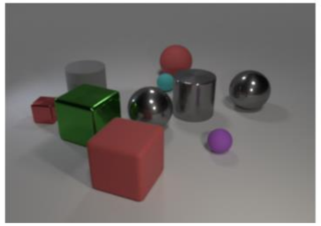
Background

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY CEDA sigda

Background

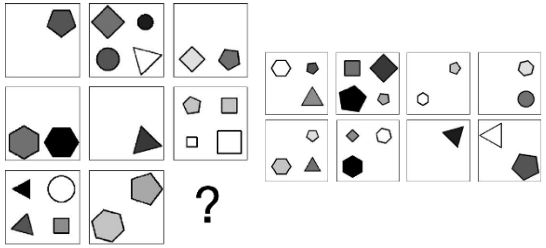
Traditional NNs are not good at reasoning



(i) *Remove all gray spheres. How many spheres are there?* (3), (ii) *Take away 3 cubes. How many objects are there?* (7), (iii) *How many blocks must be removed to get 1 block?* (2)

Complex Question Answering

NN accuracy: 50%

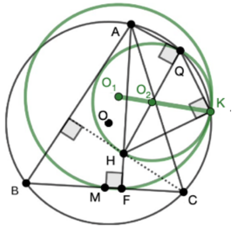


Abstract Reasoning

NN accuracy: 53%

IMO 2015 P3

"Let ABC be an acute triangle. Let (O) be its circumcircle, H its orthocenter, and F the foot of the altitude from A. Let M be the midpoint of BC. Let Q be the point on (O) such that QH ⊥ QA and let K be the point on (O) such that KH ⊥ KQ. Prove that the circumcircles (O_1) and (O_2) of triangles FKM and KQH are tangent to each other."



Automated Theorem Proving

NN accuracy: 0%

Interactive Learning

NN accuracy: 71%

Scenario

Imagine that a stranger will give Hank one thousand dollars to break all the windows in his neighbor's house without his neighbor's permission. Hank carries out the stranger's request.

Imagine that there are five people who are waiting in line to use a single-occupancy bathroom at a concert venue. Someone at the back of the line needs to throw up immediately. That person skips to the front of the line instead of waiting in the back.

At a summer camp, there is a pool. Right next to the pool is a tent where the kids at the camp have art class. The camp made a rule that there would be no cannonballing in the pool so that the art wouldn't get ruined by the splashing water. Today, there is a bee attacking this kid, and she needs to jump into the water quickly. This kid cannonballs into the pool.

Ethical Decision Making

NN accuracy: 65%

Farmer John has N cows ($2 \le N \le 10^5$). Each cow has a breed that is either Guernsey or Holstein. As is often the case, the cows are standing in a line, numbered $1 \cdots N$ in this order.

Over the course of the day, each cow writes down a list of cows. Specifically, cow i's list contains the range of cows starting with herself (cow i) up to and including cow E_i ($i \le E_i \le N$).

FJ has recently discovered that each breed of cow has exactly one distinct leader. FJ does not know who the leaders are, but he knows that each leader must have a list that includes all the cows of their breed, or the other breed's leader (or both).

Help FJ count the number of pairs of cows that could be leaders. It is guaranteed that there is at least one possible pair.

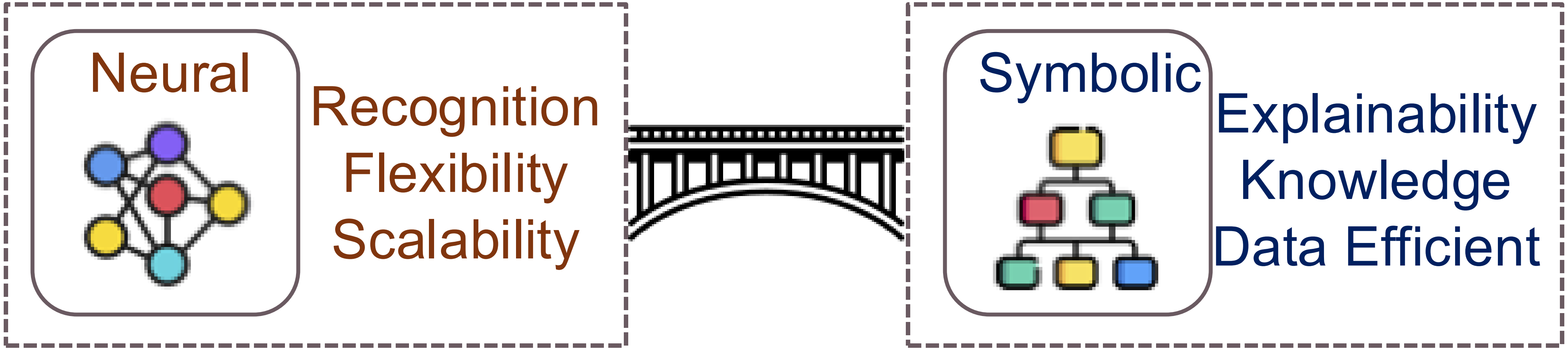
Competitive Programming

NN accuracy: 8.7%

Background

Neural-Symbolic AI (NSAI)

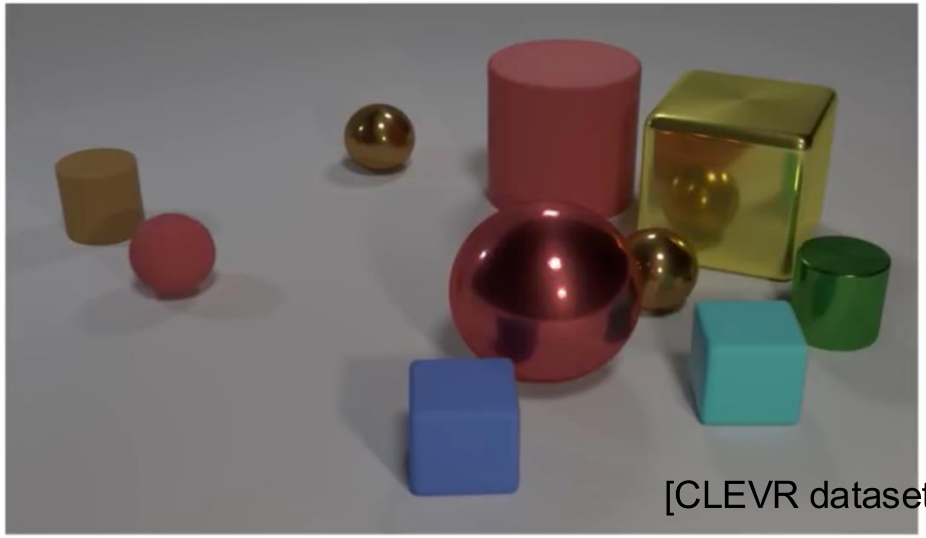
- A compositional system to enhance cognitive capability for reasoning tasks.



Background - NSAI

Neural-Symbolic AI example

- **Visual Reasoning**



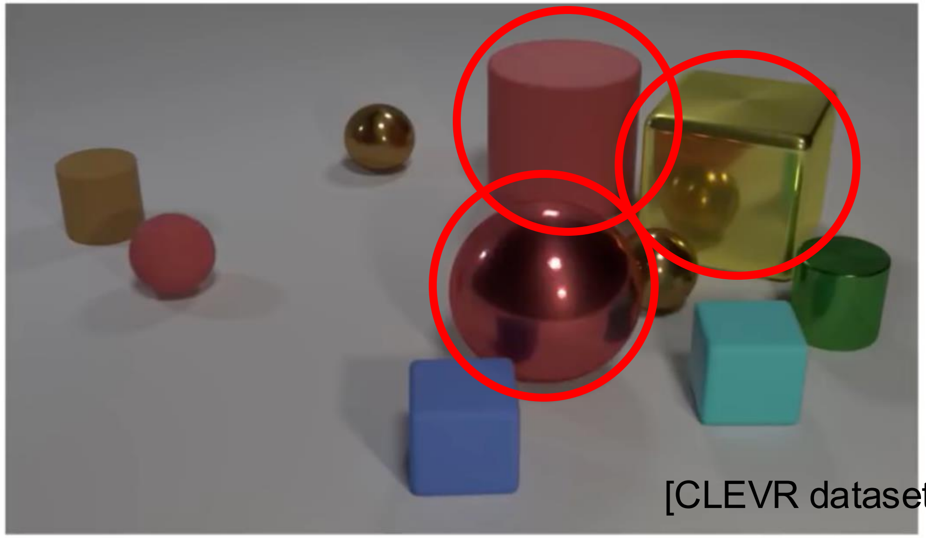
[CLEVR dataset]

Question: *Are there an **equal number** of large things and metal spheres?*

Background - NSAI

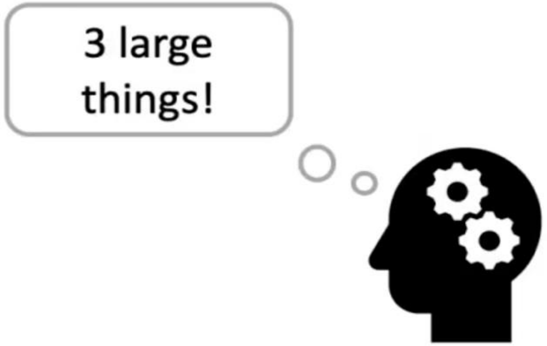
Neural-Symbolic AI example

- **Visual Reasoning**



[CLEVR dataset]

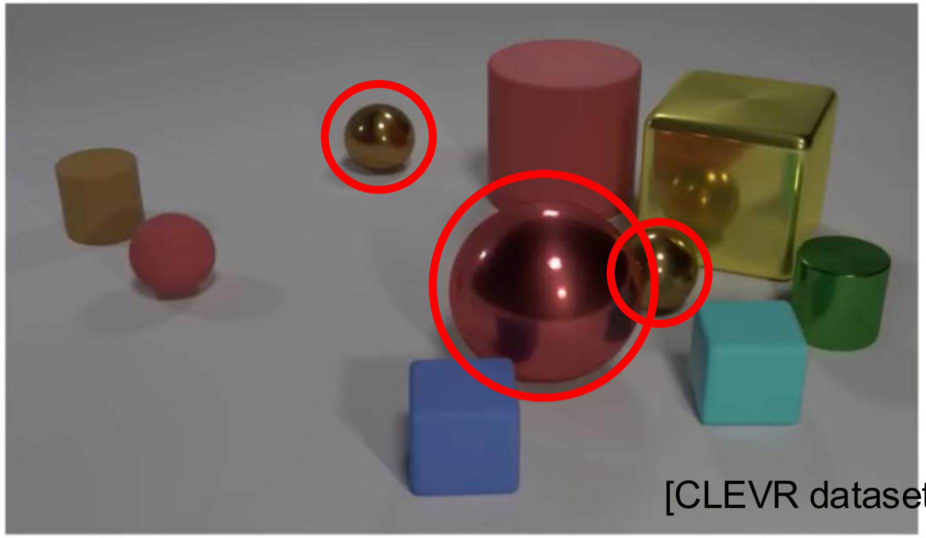
Question: *Are there an* ***equal number*** *of large things and metal spheres?*



Background - NSAI

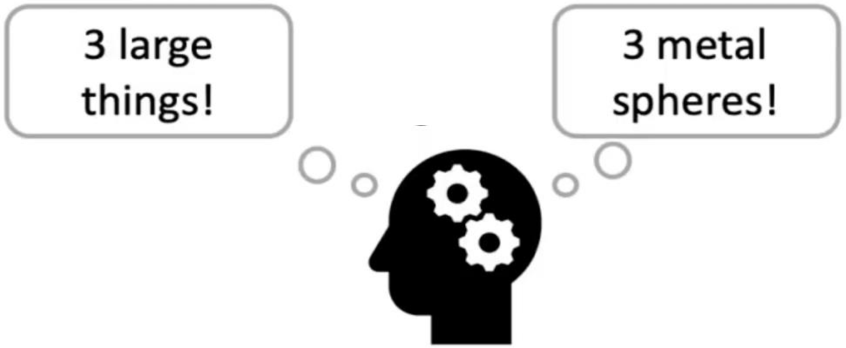
Neural-Symbolic AI example

- **Visual Reasoning**



[CLEVR dataset]

Question: *Are there an* ***equal number*** *of large things and metal spheres?*

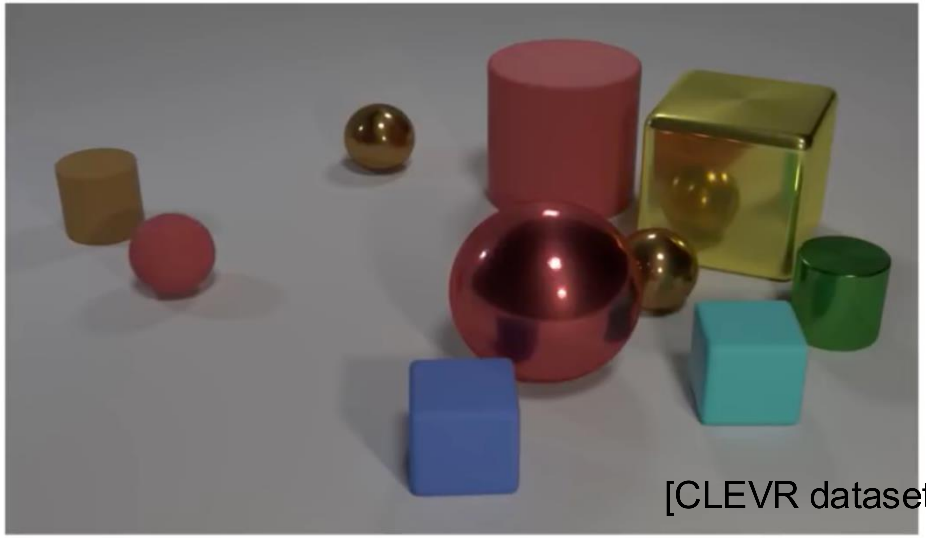


Slide Adapted from MIT 6.S191: Neurosymbolic AI

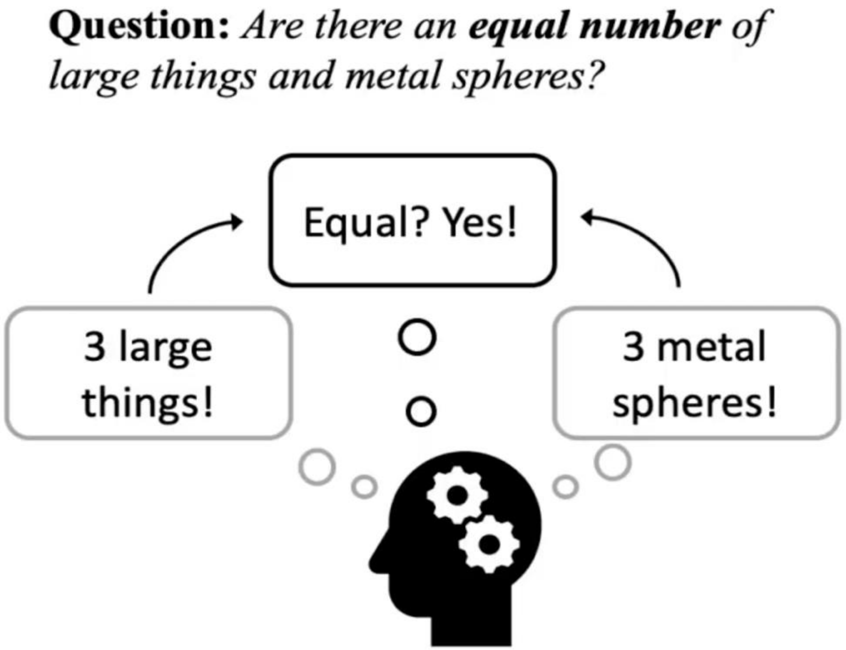
Background - NSAI

Neural-Symbolic AI example

- **Visual Reasoning**



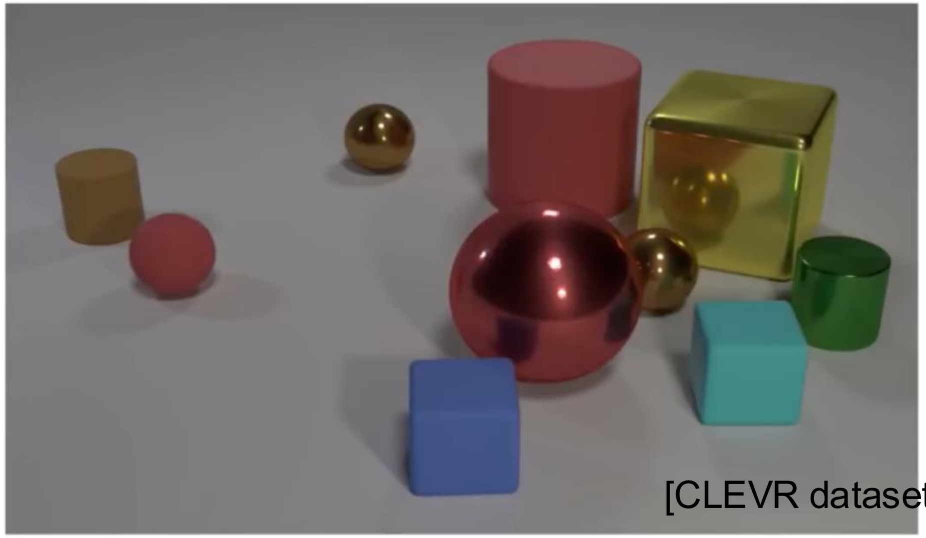
[CLEVR dataset]



Background - NSAI

Neural-Symbolic AI example

- **Visual Reasoning**

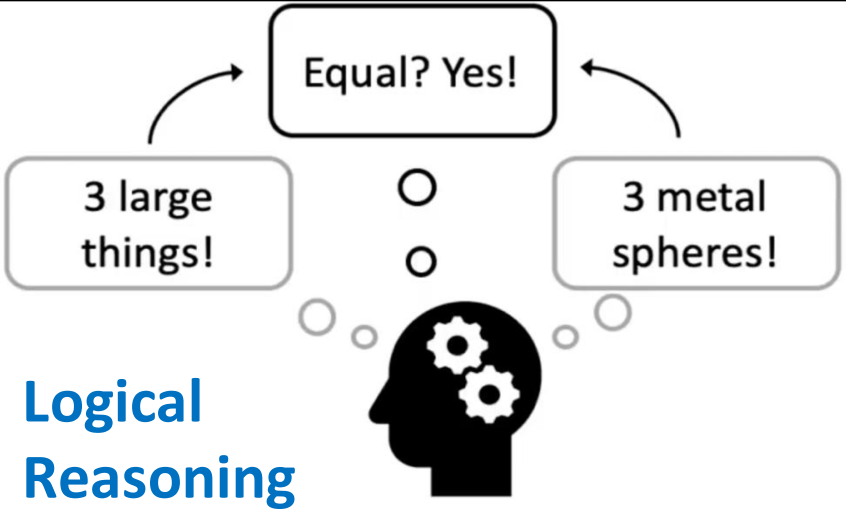


[CLEVR dataset]

Visual Perception

Question Understanding

Question: *Are there an **equal number** of large things and metal spheres?*



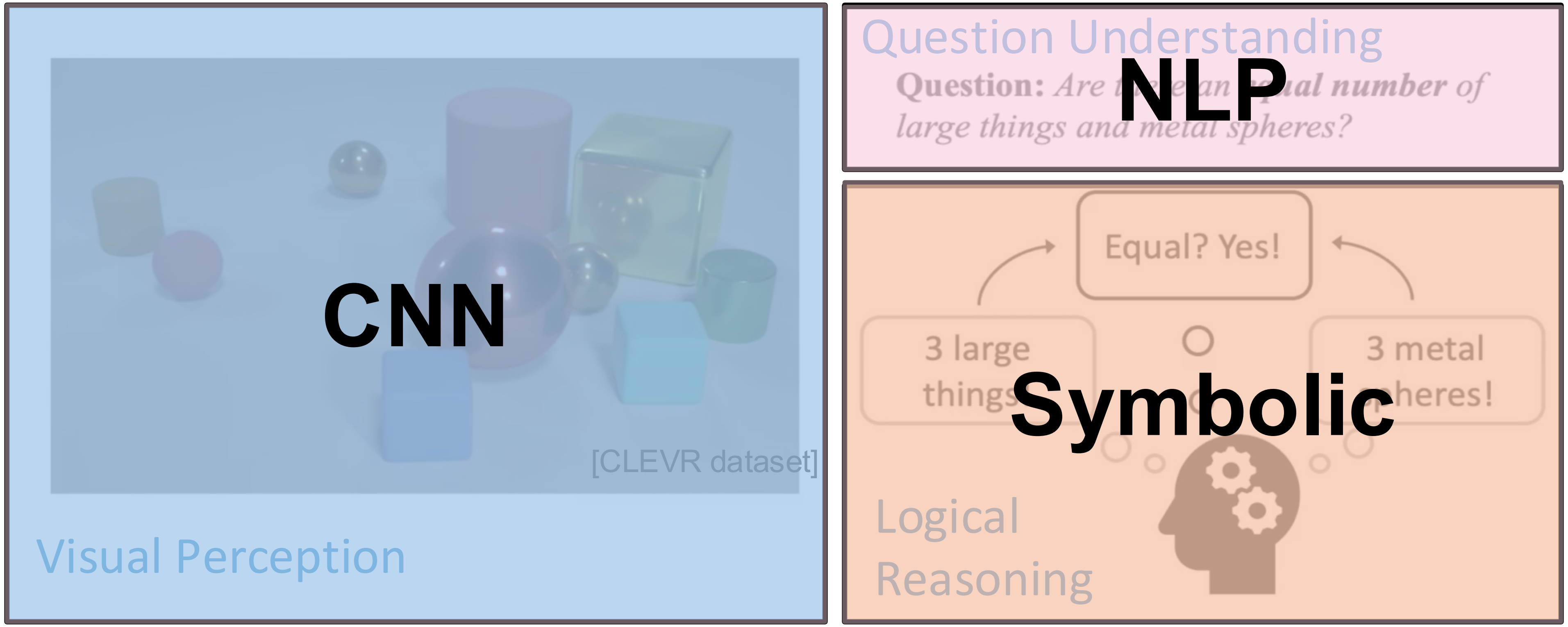
Logical Reasoning

Slide Adapted from MIT 6.S191: Neurosymbolic AI

Background - NSAI

Neural-Symbolic AI example

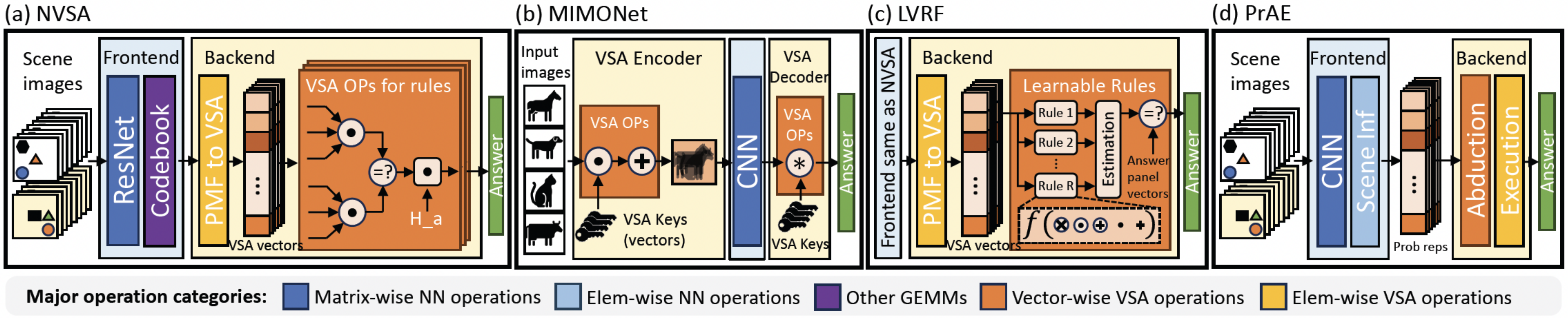
- **Visual Reasoning**



Background - NSAI

NSAI algorithm structure

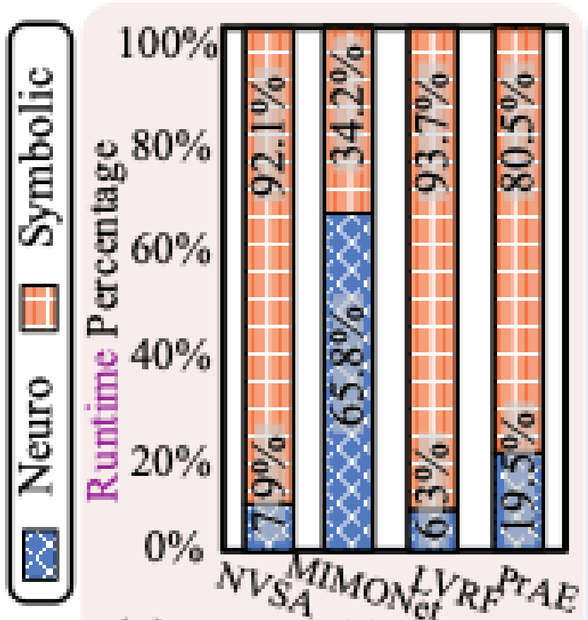
Representative Neuro-Symbolic AI Workloads		Neuro-Vector-Symbolic Architecture (NVSA) [13]	Multiple-Input-Multiple-Output Neural Networks (MIMONet) [24]	Probabilistic Abduction via Learning Rules in Vector-symbolic Architecture (LVRF) [12]	Probabilistic Abduction and Execution Learner (PrAE) [40]
Compute Pattern	Neuro	CNN	CNN/Transformer	CNN	CNN
	Symbolic	VSA binding/unbinding (Circular Conv)	VSA binding (Circular Conv)	VSA binding/unbinding (Circular Conv)	Probabilistic abduction
Application Scenario	Use Case	Spatial-temporal and abstract reasoning	Multi-input simultaneously processing	Probabilistic reasoning, OOD data processing	Spatial-temporal and abstract reasoning
	Advantage vs. Neural	Higher joint representation efficiency, Better reasoning capability, Transparency	Higher throughput, Lower latency, Compositional compute, Transparency	Stronger OOD handling capability, One-pass learning, Higher flexibility, Transparency	Higher generalization, Transparency, Interpretability, Robustness



Motivation

Motivation

NSAI workloads system characterization

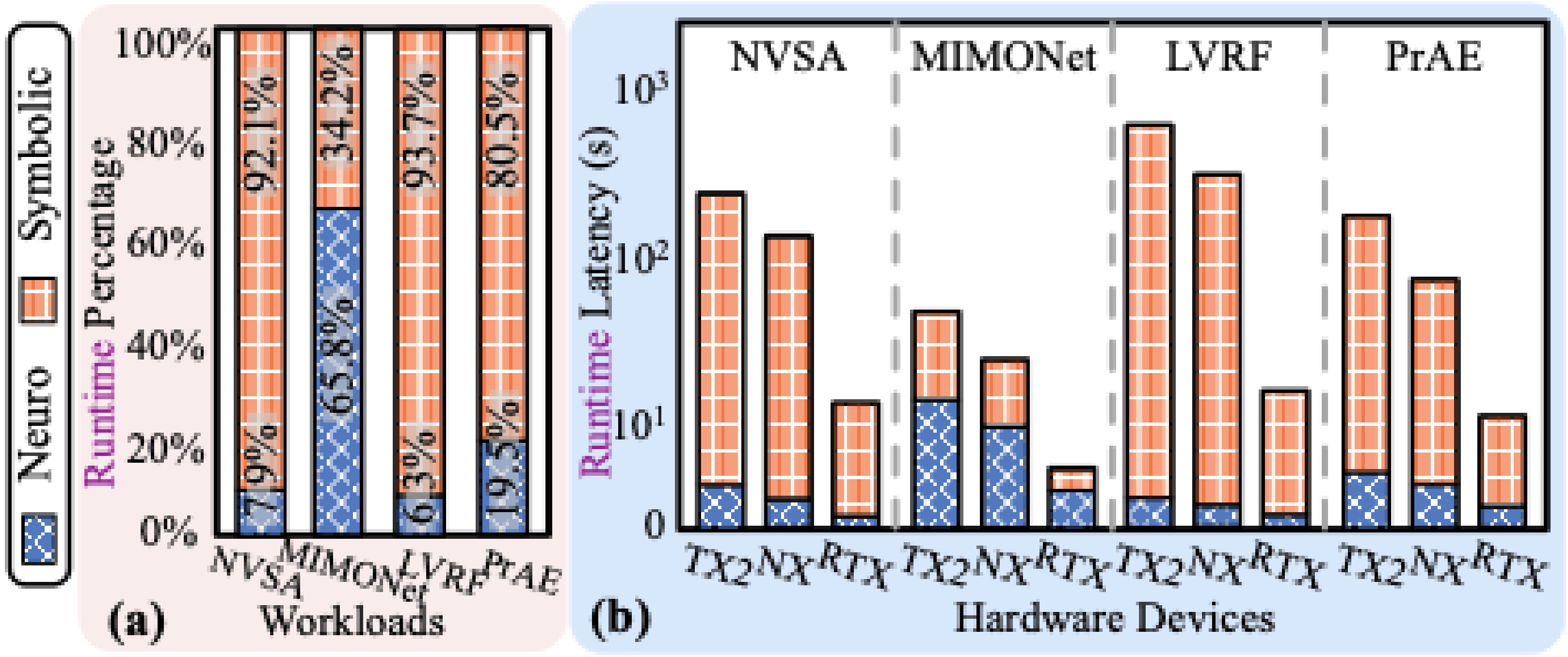


(a)

Z. Wan *et al.*, "Towards Cognitive AI Systems: Workload and Characterization of Neuro-Symbolic AI," *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*

Motivation

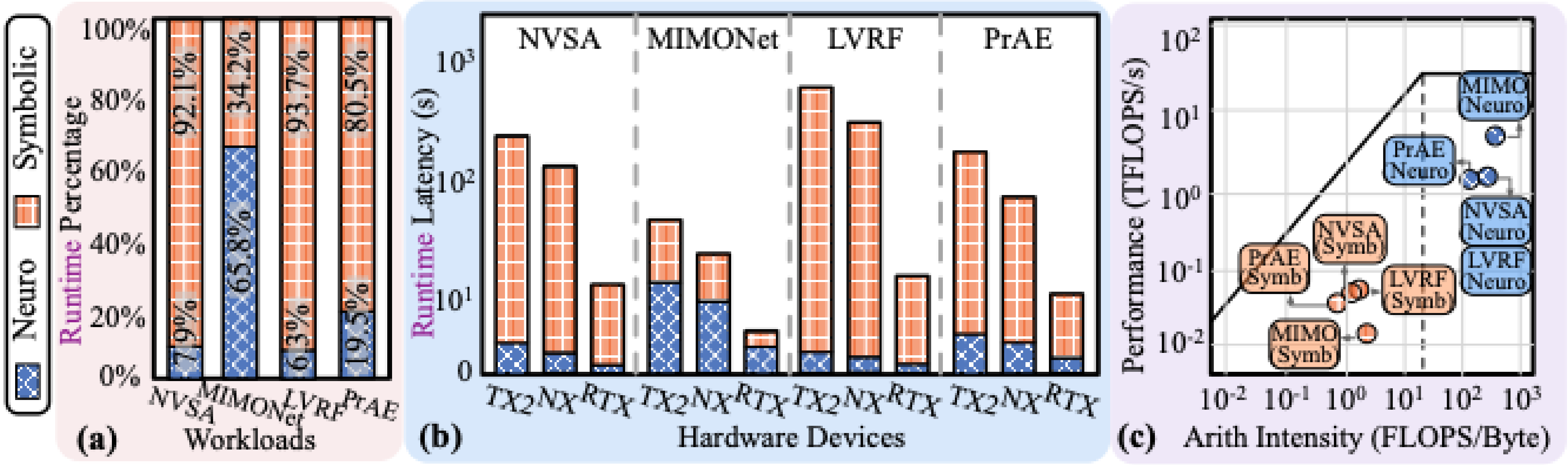
NSAI workloads system characterization



Z. Wan *et al.*, "Towards Cognitive AI Systems: Workload and Characterization of Neuro-Symbolic AI," *2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*

Motivation

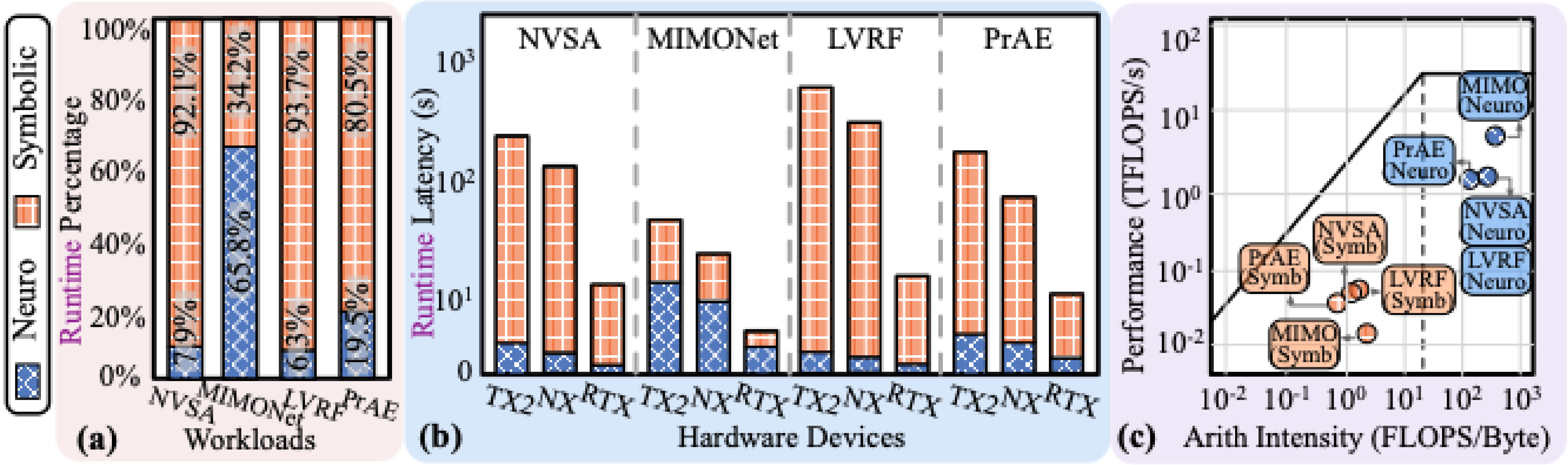
NSAI workloads system characterization



Z. Wan *et al.*, "Towards Cognitive AI Systems: Workload and Characterization of Neuro-Symbolic AI,"
2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)

Motivation

NSAI workloads system characterization



Neuro-symbolic workload exhibits high latency compared to neural models;
Symbolic component *is processed inefficiently on off-the-shelf CPU/GPUs*

Motivation

NSAI algorithms deployment challenges:

- **Memory & compute inefficiency**

Motivation

NSAI algorithms deployment challenges:

- **Memory & compute inefficiency**
- **Heterogeneous compute kernel**

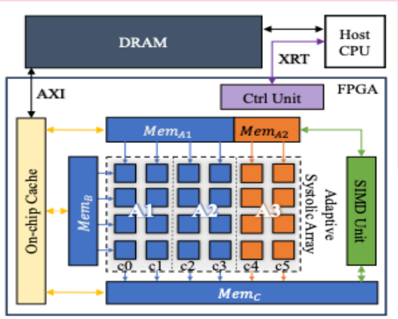
Motivation

NSAI algorithms deployment challenges:

- **Memory & compute inefficiency**
- **Heterogeneous compute kernel**

- **Algorithm diversity**

Motivation



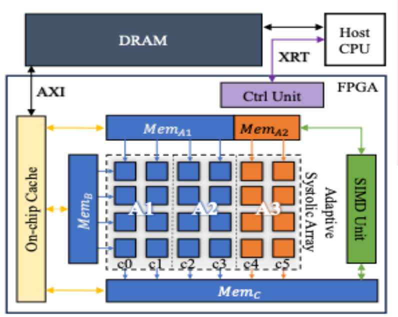
NSAI algorithms deployment challenges:

- **Memory & compute inefficiency**
- **Heterogeneous compute kernel**

Efficient and flexible architecture

- **Algorithm diversity**

Motivation



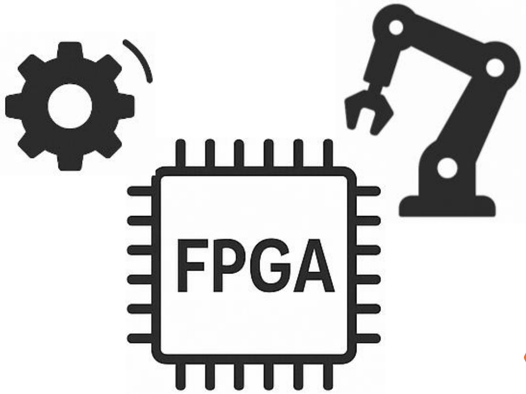
NSAI algorithms deployment challenges:

- **Memory & compute inefficiency**
- **Heterogeneous compute kernel**

Efficient and flexible architecture

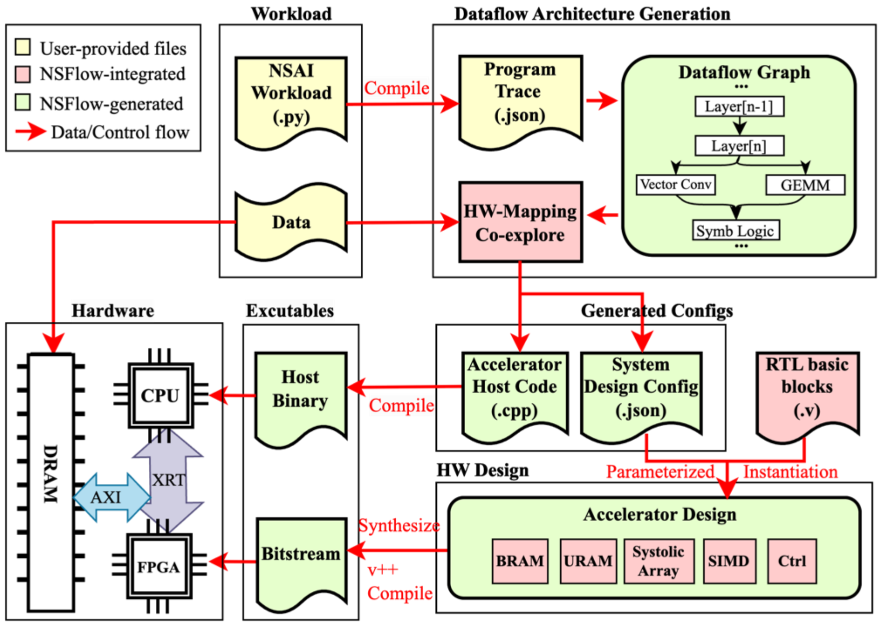
- **Algorithm diversity**

Automated and customized FPGA deployment



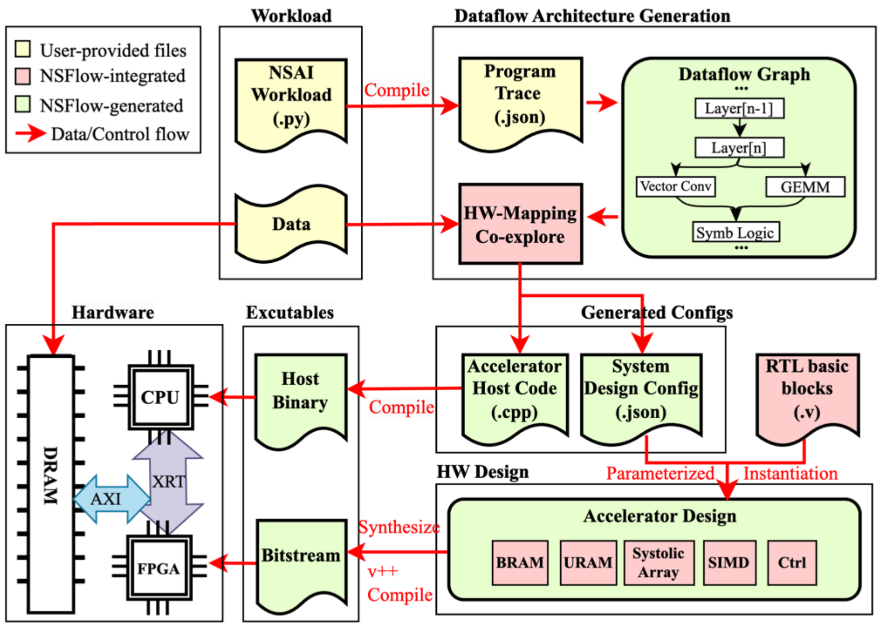
Motivation

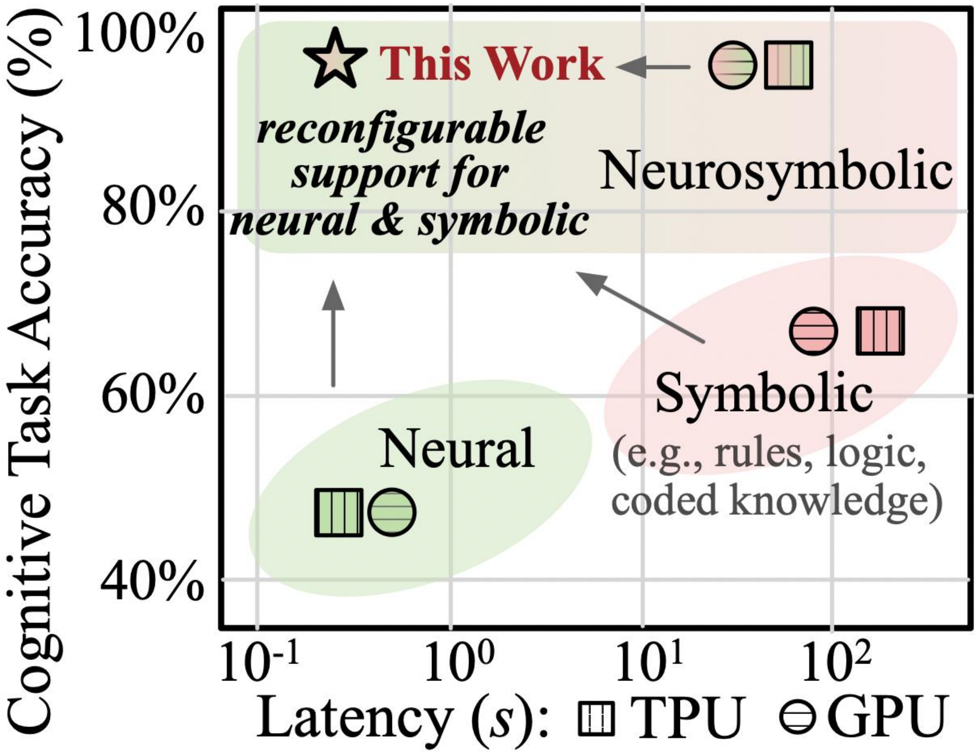
NSFlow



Motivation

NSFlow





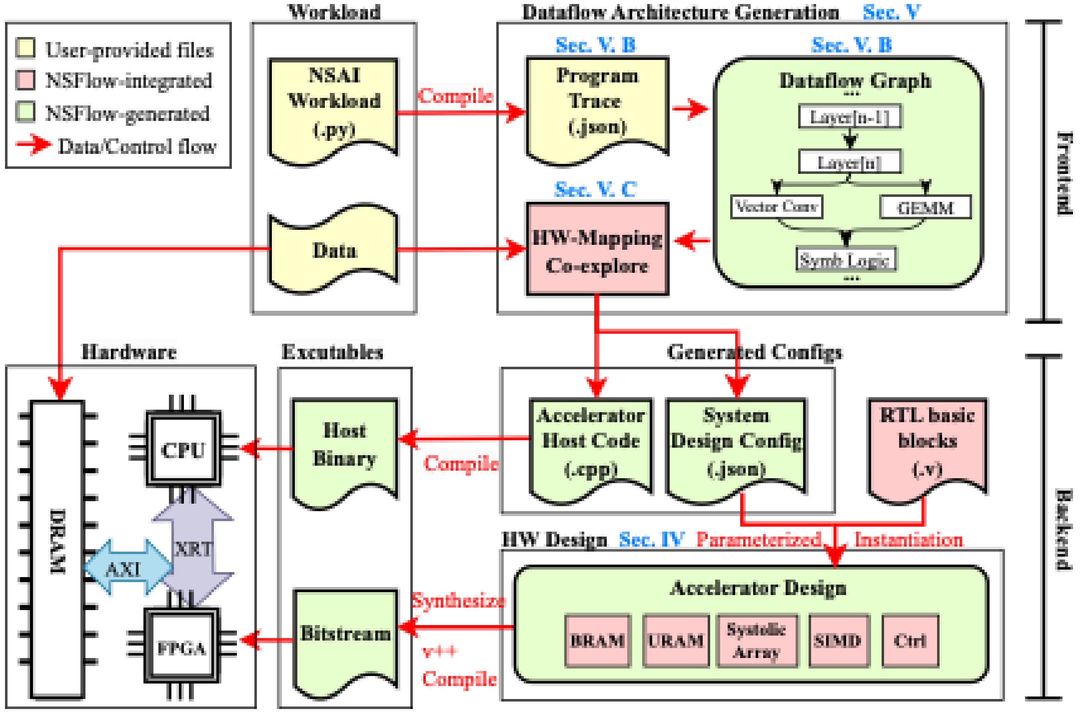
NSFlow Framework

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

NSFlow Framework

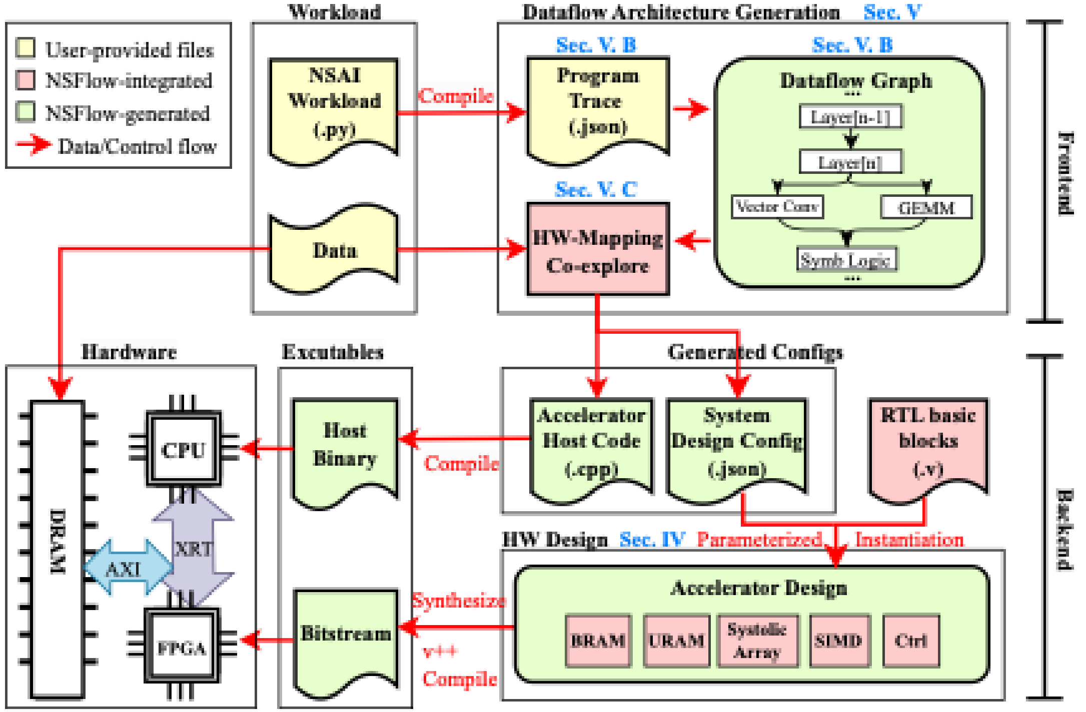
An end-to-end automated FPGA framework for accelerating and deploying generic NSAI workloads.



NSFlow Framework

An end-to-end automated FPGA framework for accelerating and deploying generic NSAI workloads.

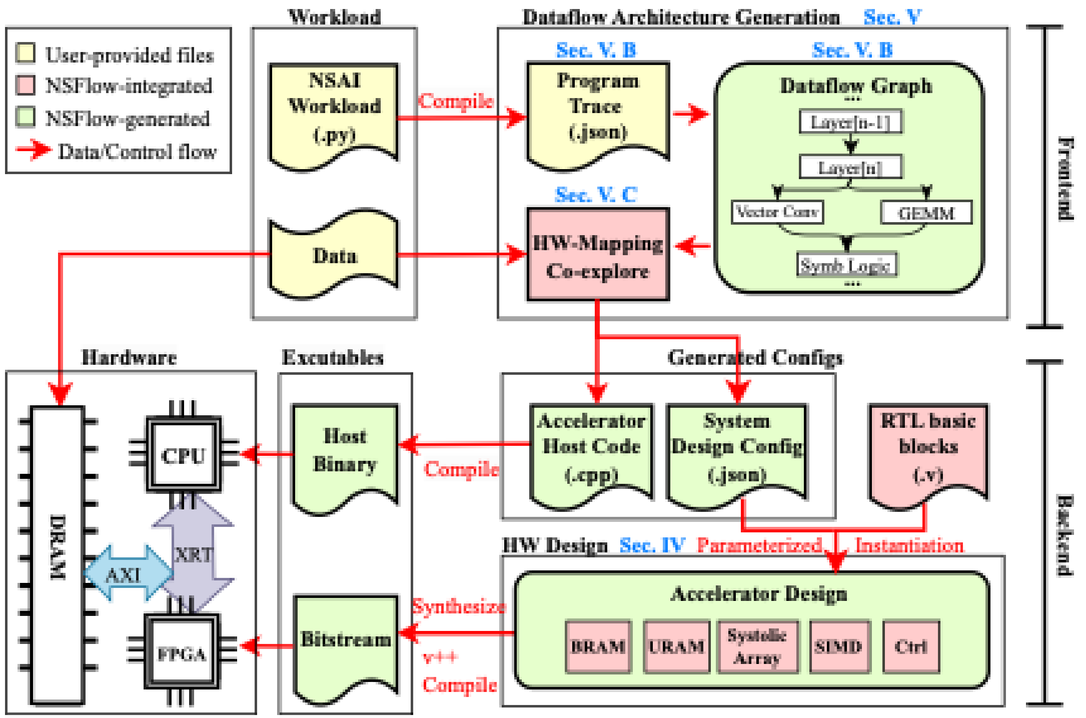
- **Identifies data dependency** for the workload



NSFlow Framework

An end-to-end automated FPGA framework for accelerating and deploying generic NSAI workloads.

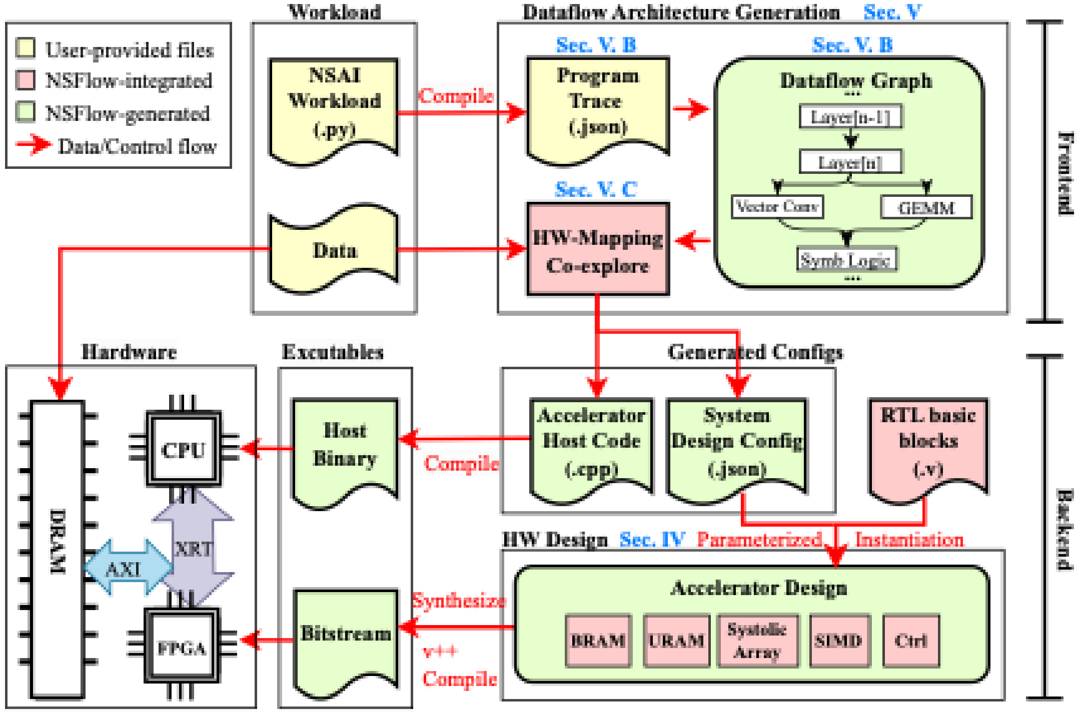
- **Identifies data dependency** for the workload
- **Explores design space** with parameterizable HW blocks.



NSFlow Framework

An end-to-end automated FPGA framework for accelerating and deploying generic NSAI workloads.

- ➢ **Identifies data dependency** for the workload
- ➢ **Explores design space** with parameterizable HW blocks.
- ➢ Generates and deploys **optimal dataflow architecture** on FPGA

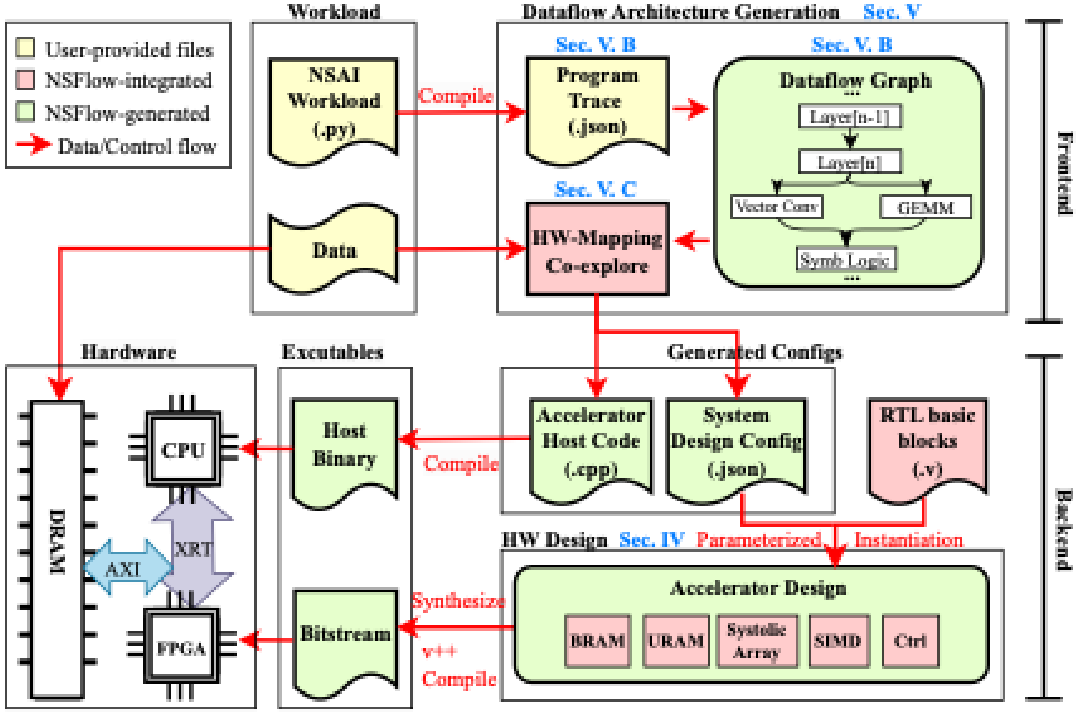


NSFlow Framework

An end-to-end automated FPGA framework for accelerating and deploying generic NSAI workloads.

- ➢ **Identifies data dependency** for the workload
- ➢ **Explores design space** with parameterizable HW blocks.
- ➢ Generates and deploys **optimal dataflow architecture** on FPGA

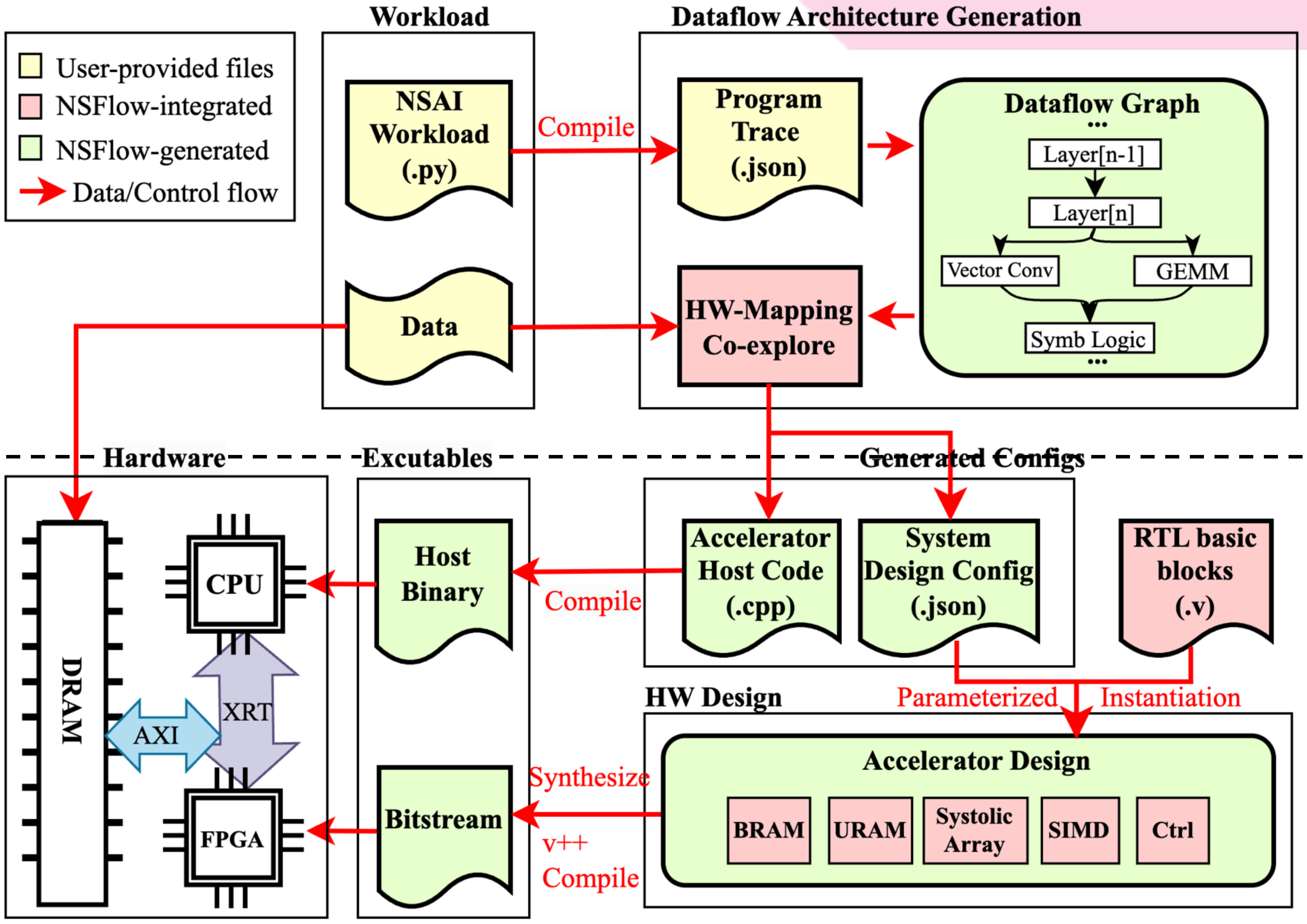
✓ *Enables automated, efficient and scalable dataflow and architecture solutions for NSAI*



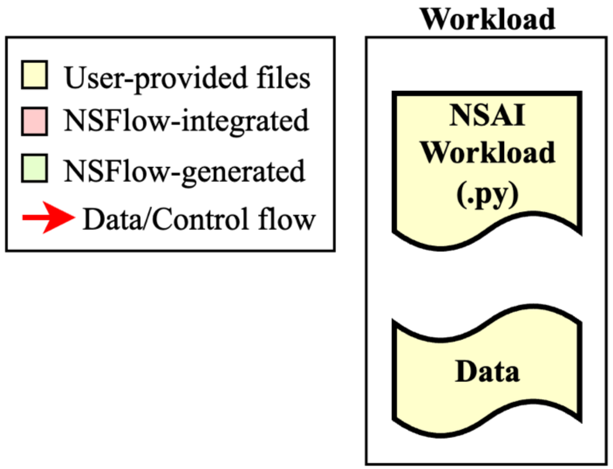
NSFlow Framework

Frontend

Backend



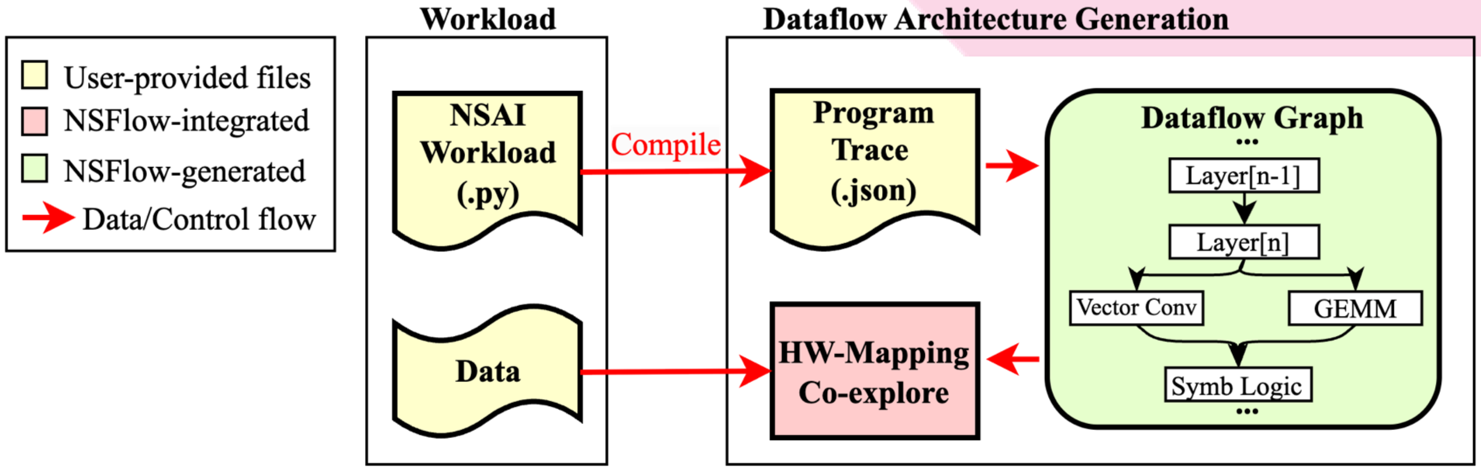
NSFlow Framework



Frontend

Backend

NSFlow Framework



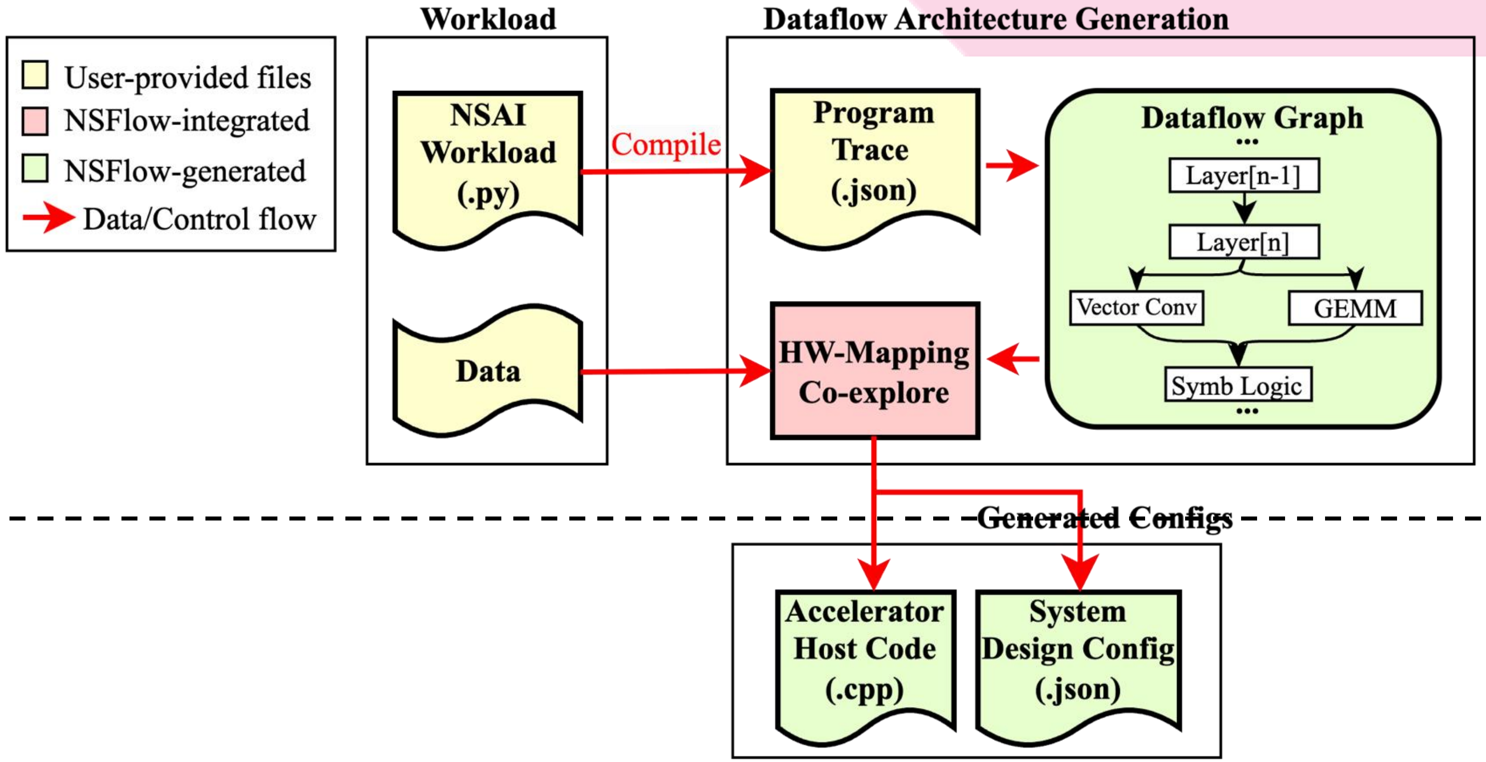
Frontend

Backend

NSFlow Framework

Frontend

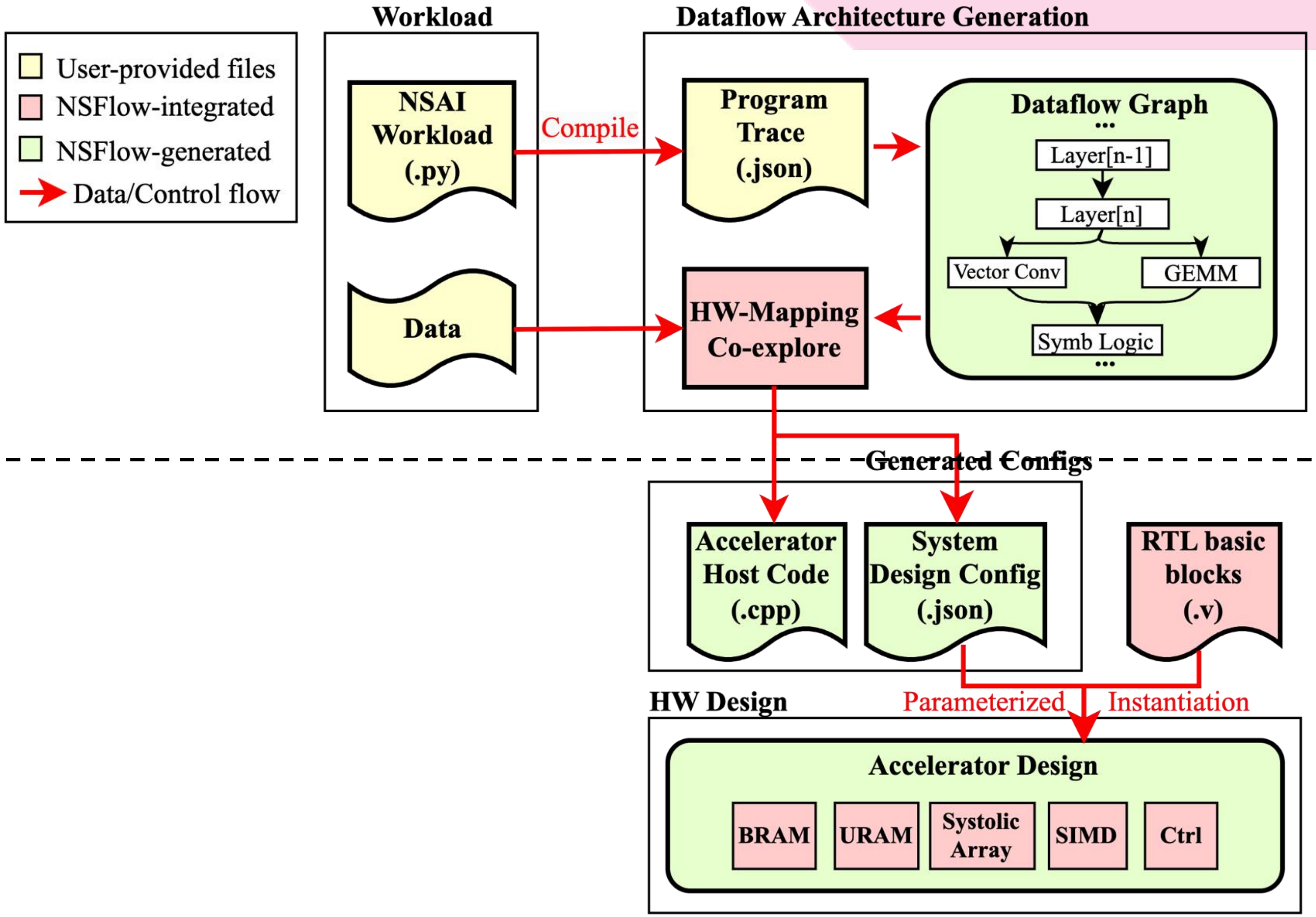
Backend



NSFlow Framework

Frontend

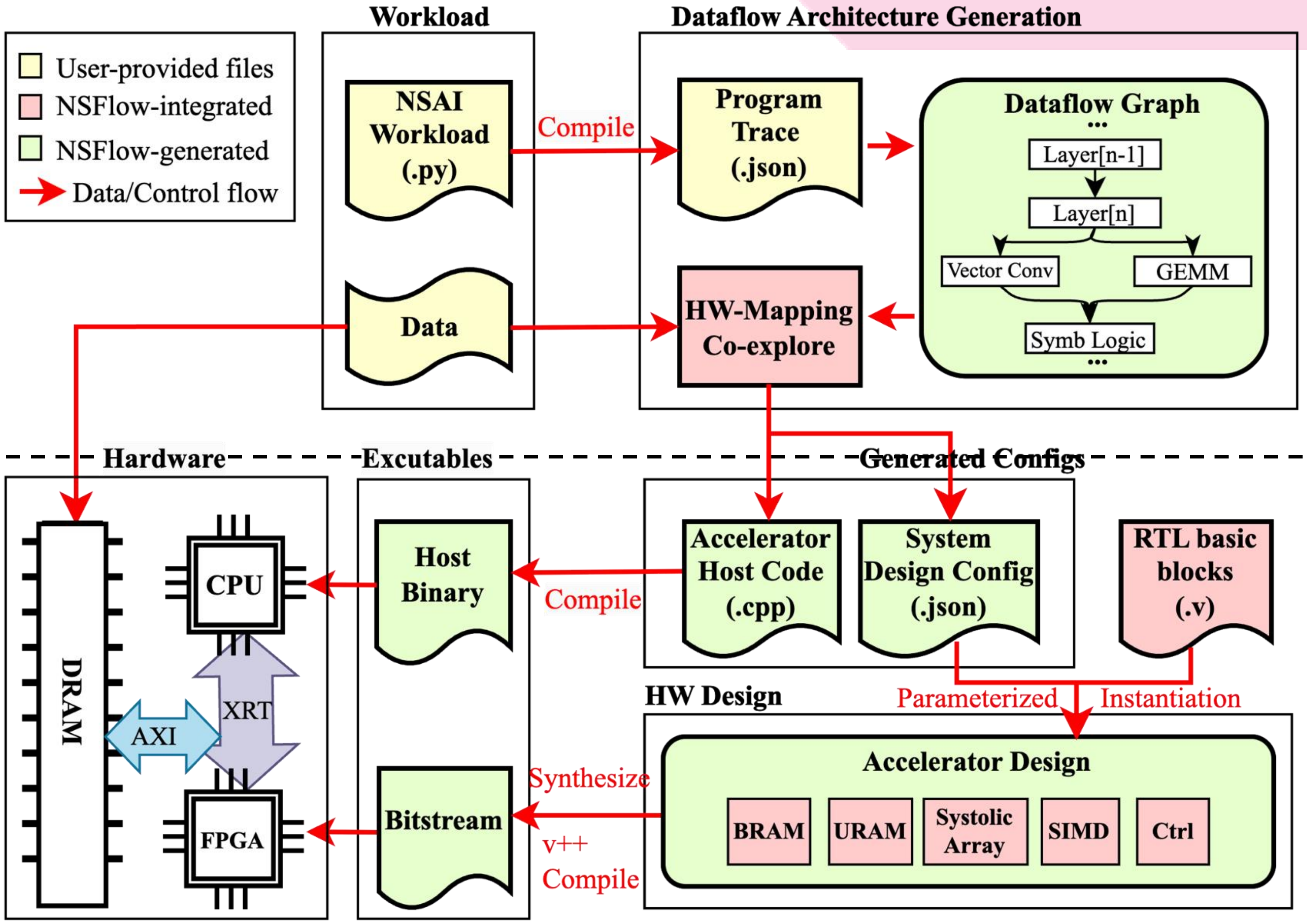
Backend



NSFlow Framework

Frontend

Backend



NSFlow Backend

Flexible Hardware Architecture

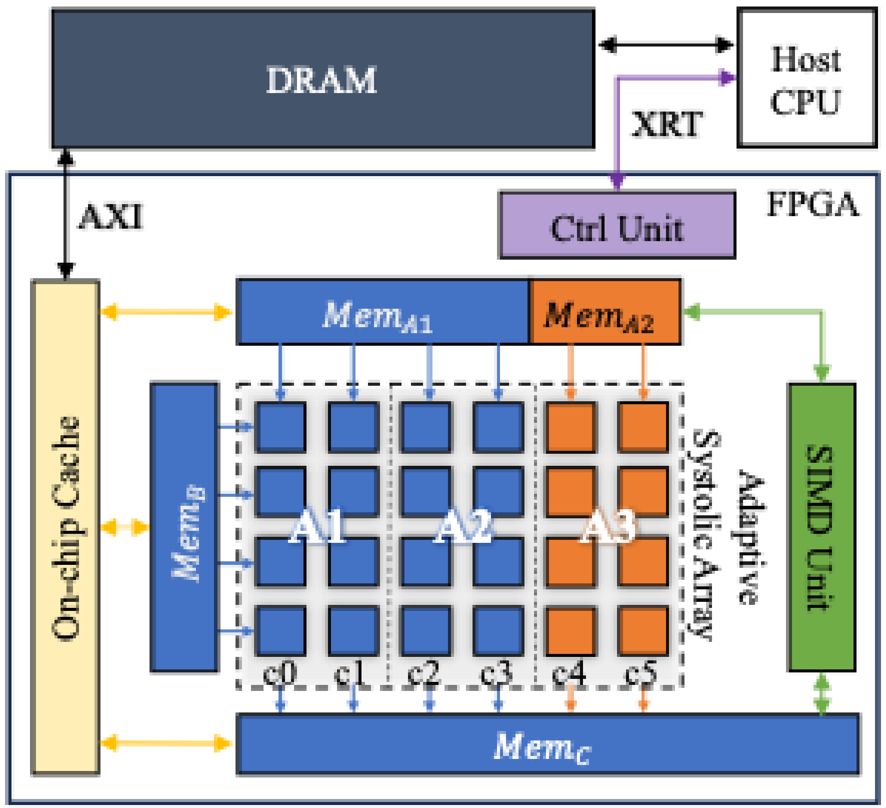
THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

NSFlow Backend

An **accelerator template**

w/ RTL blocks **parameterizable** by config files



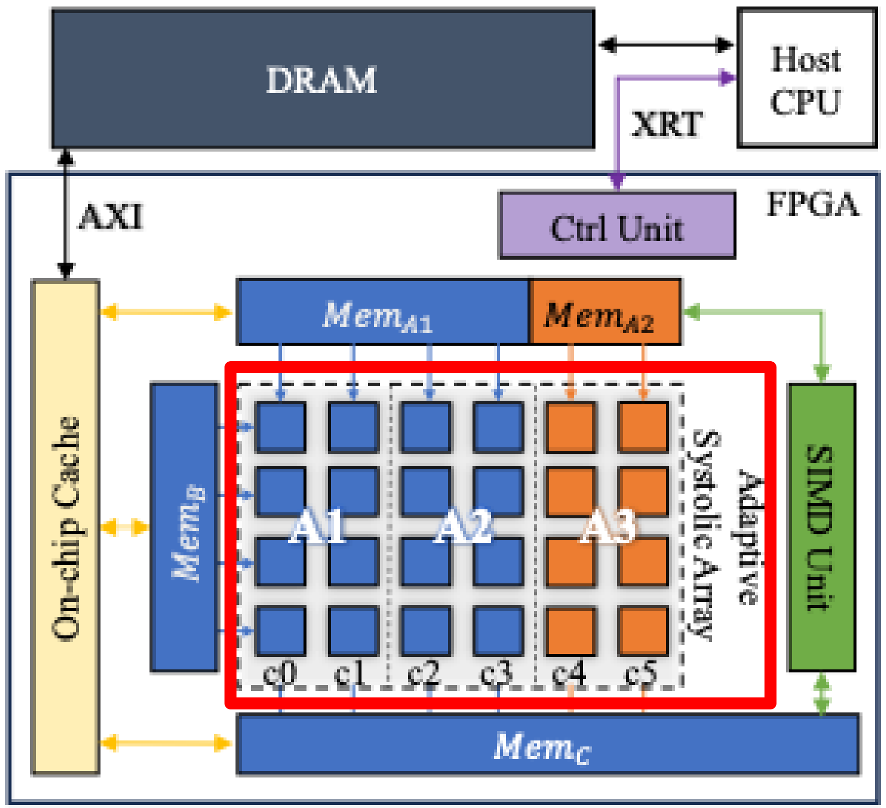
NSFlow Backend

An **accelerator template**

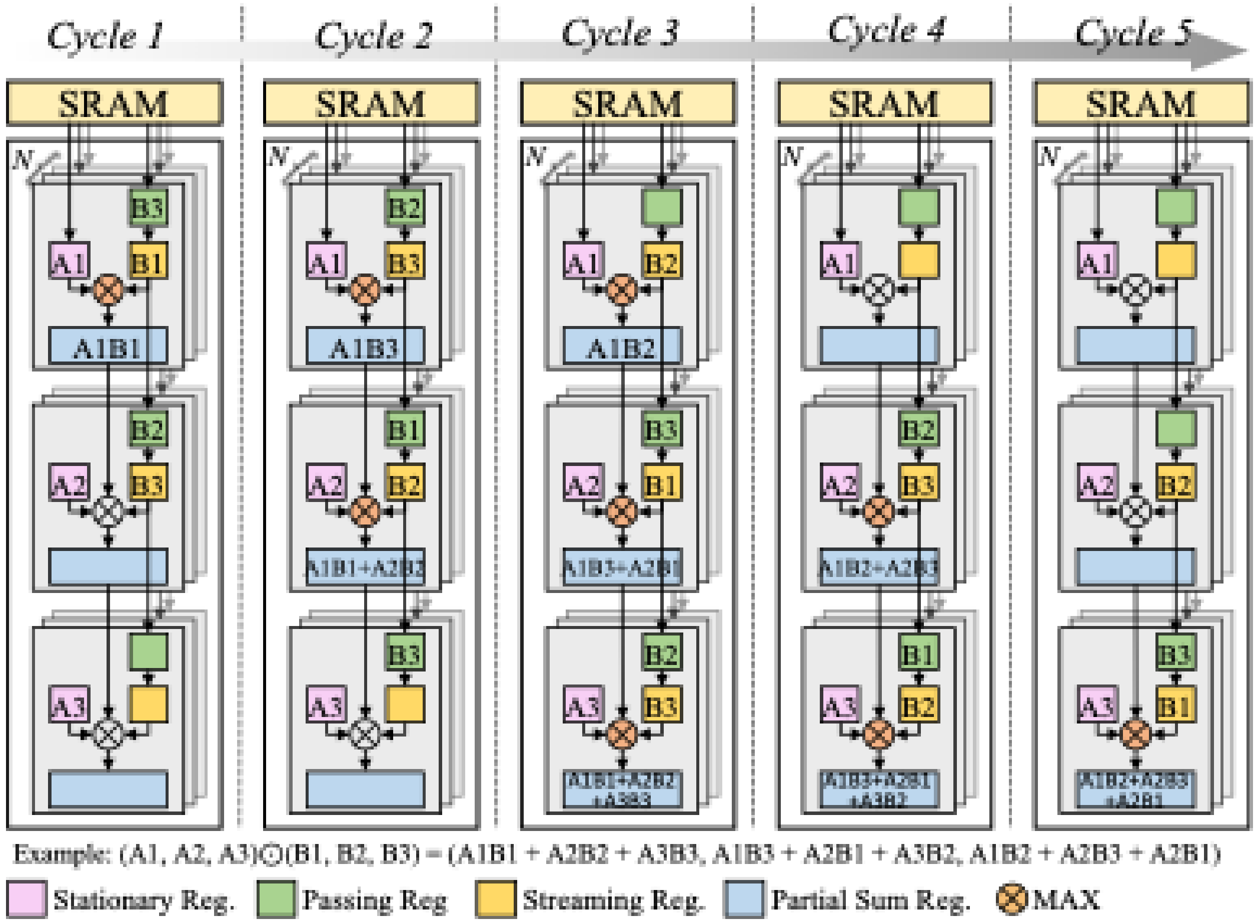
w/ RTL blocks **parameterizable** by config files

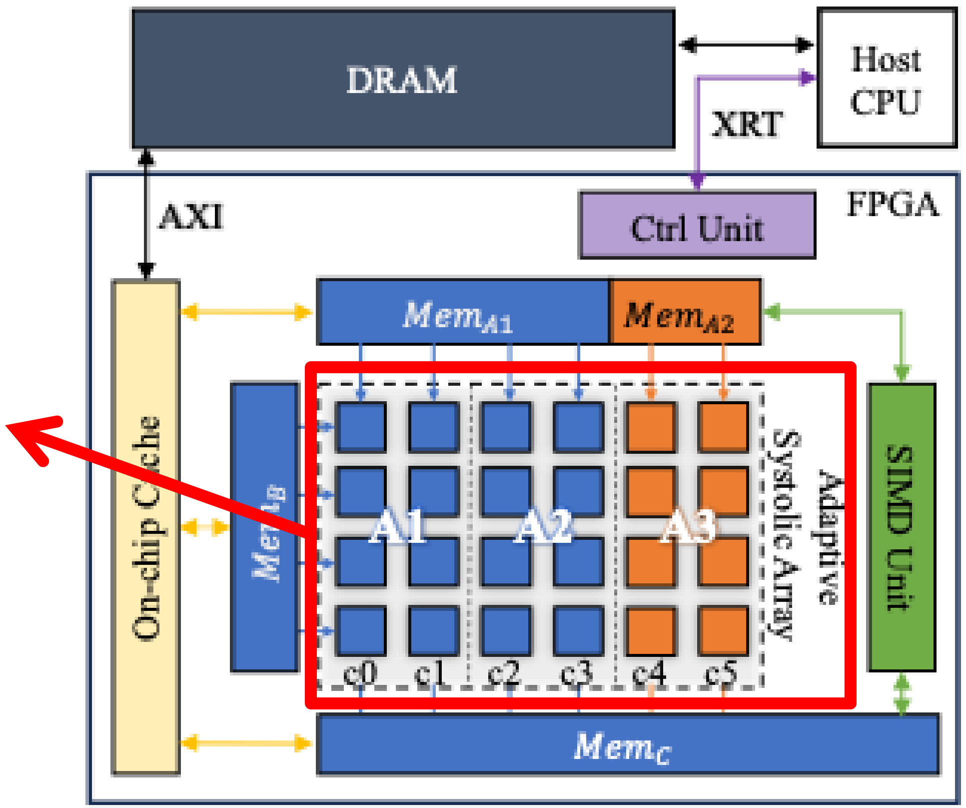
- ❑ **Adaptive Systolic Array:**

  for efficient Neuro & Symbolic processing



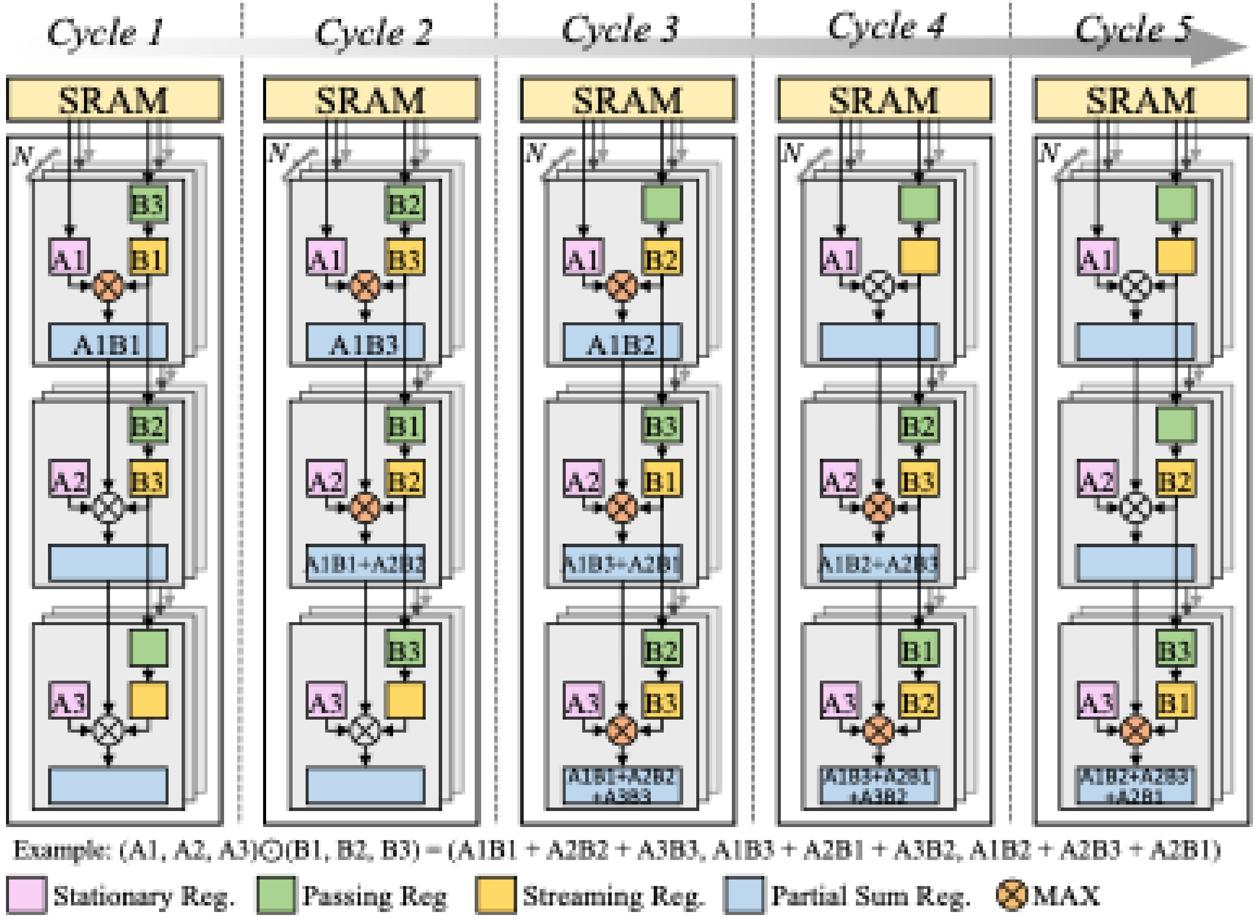
NSFlow Backend

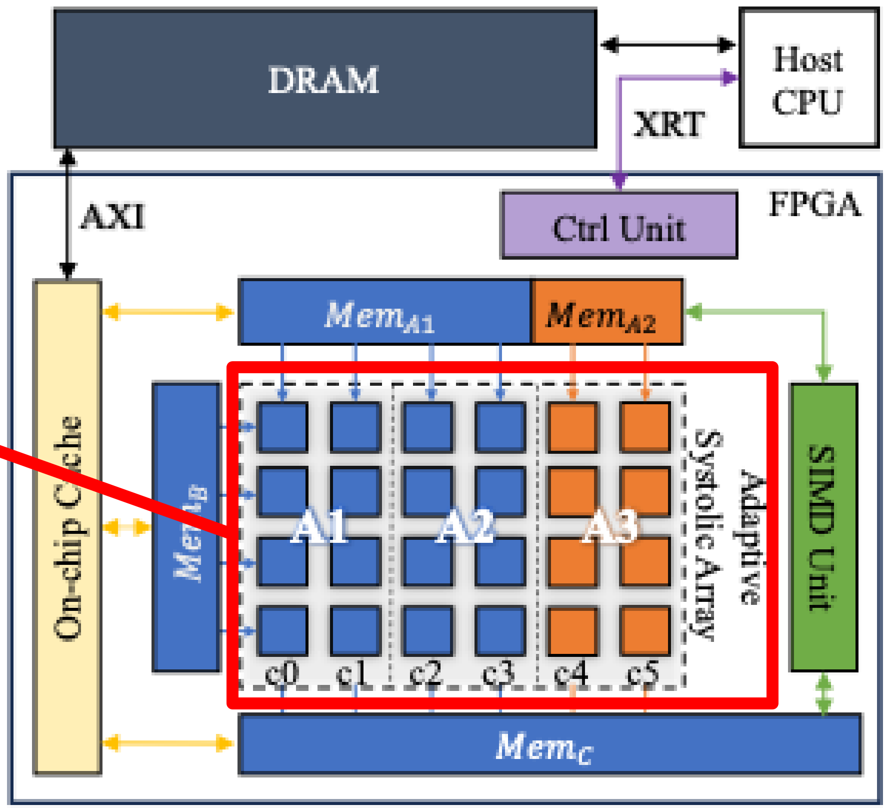




Z. Wan, H. Yang, R. Raj et al., "CogSys: Efficient and Scalable Neurosymbolic Cognition System via Algorithm-Hardware Co-Design," 2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)

NSFlow Backend





Enables efficient Vector Symbolic operation processing on systolic array

NSFlow Backend

An **accelerator template**

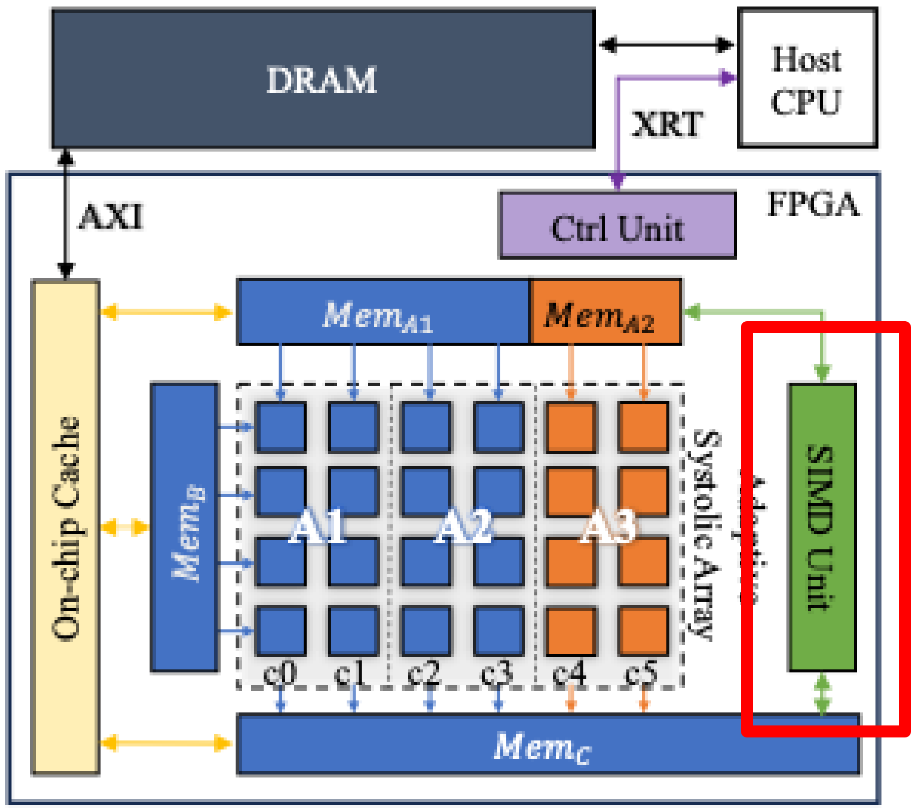
w/ RTL blocks **parameterizable** by config files

- ❑ **Adaptive Systolic Array:**

  for efficient Neuro & Symbolic processing

- ❑ **SIMD Unit:**

  for element-wise ops



NSFlow Backend

An **accelerator template**

w/ RTL blocks **parameterizable** by config files

- ❑ **Adaptive Systolic Array:**

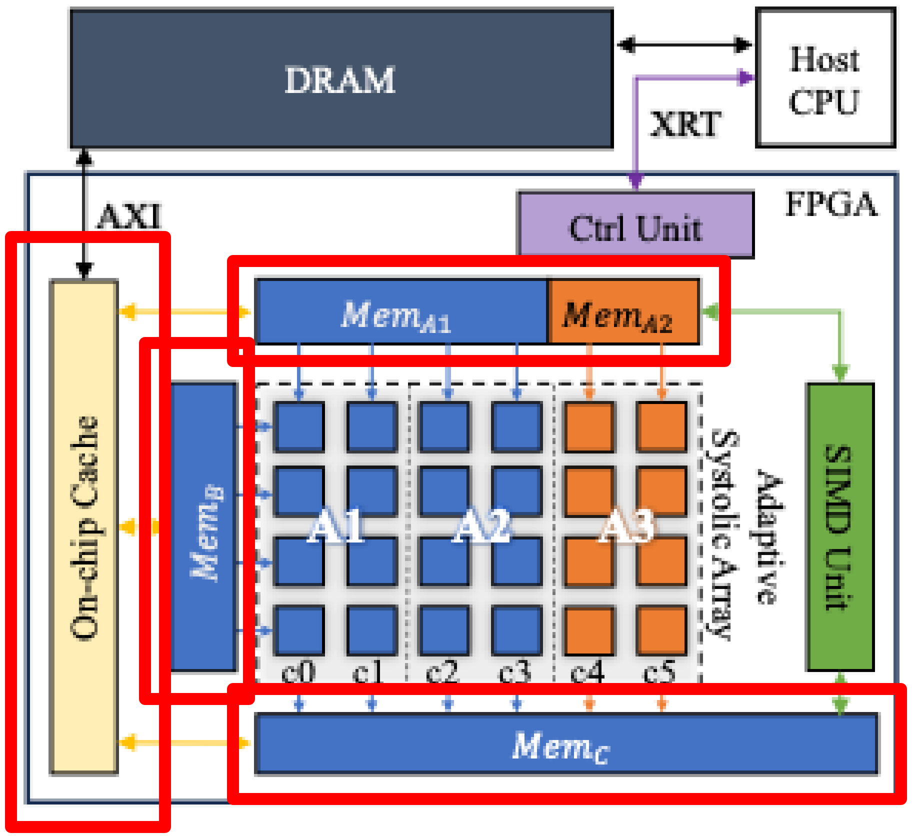
  for efficient Neuro & Symbolic processing

- ❑ **SIMD Unit:**

  for element-wise ops

- ❑ **BRAM Blocks:**

  for flexible on-chip memory



NSFlow Backend

An **accelerator template**

w/ RTL blocks **parameterizable** by config files

- ❑ **Adaptive Systolic Array:**

  for efficient Neuro & Symbolic processing

- ❑ **SIMD Unit:**

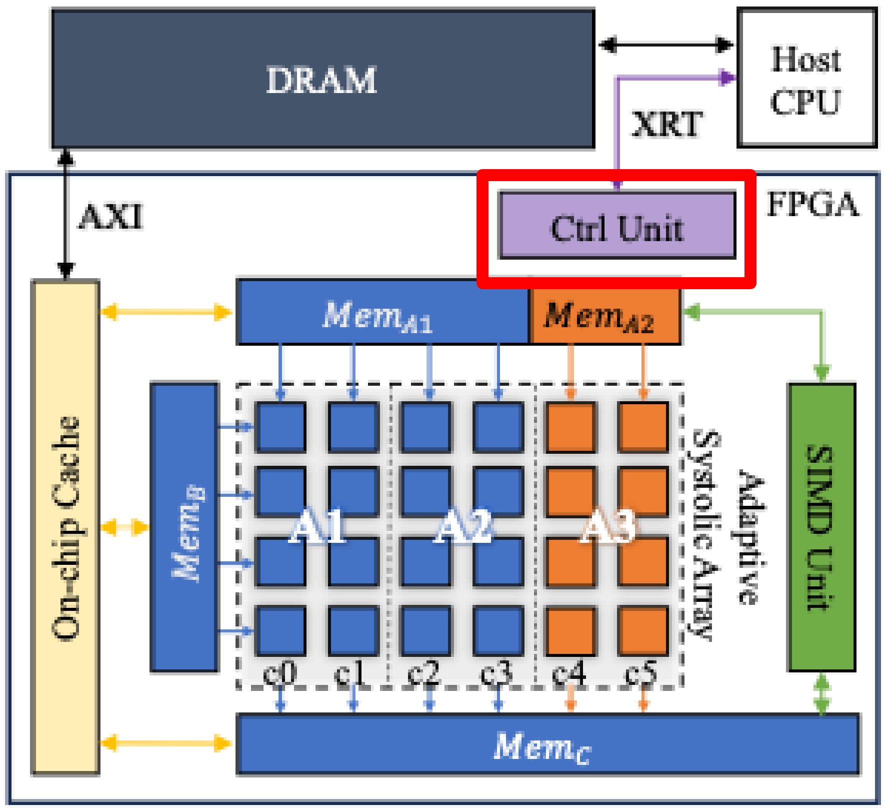
  for element-wise ops

- ❑ **BRAM Blocks:**

  for flexible on-chip memory

- ❑ **Control Logic:**

  for HW-level task scheduling



NSFlow Frontend

Dataflow Architecture Generation (DAG)

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

NSFlow Frontend

- Analyze workload characteristics and data dependencies
- Explore optimal HW configurations and dataflow for backend

NSFlow Frontend

1. Extract Execution Trace from the workload:

wokload.py

NSFlow Frontend

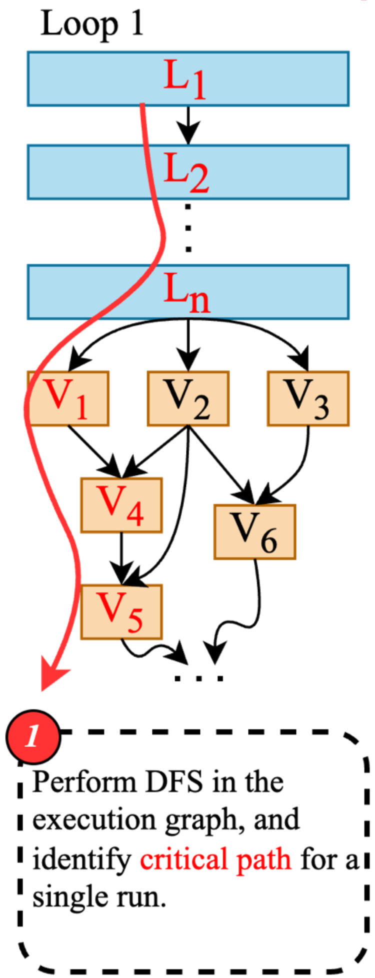
1. Extract Execution Trace from the workload:

wokload.py => et.json

```
graph():
    ...
    // Neuro Operation - CNN (Resnet18)
    %relu_1[16,64,160,160] : call_module[relu](args = (%bn1
         [16,64,160,160]))
    %maxpool_1[16,64,160,160] : call_module[maxpool](args =
         (%relu_1[16,64,160,160]))
    %conv2d_1[16,64,160,160] : call_module[conv2d](args =
         (%maxpool_1[16,64,160,160]))
    ...
    // Symbolic Operations
    // Inverse binding of two block codes vectors by
         blockwise cicular correlation
    %inv_binding_circular_1[1,4,256] : call_function[nvsa.
         inv_binding_circular](args = (%vec_0[1,4,256], %
         vec_1[1,4,256]))
    %inv_binding_circular_2[1,4,256] : call_function[nvsa.
         inv_binding_circular](args = (%vec_3[1,4,256], %
         vec_4[1,4,256]))
    // Compute similarity between two block codes vectors
    %match_prob_1[1] : call_function[nvsa.match_prob](args
         = (%inv_binding_circular_1[1,4,256], %vec_2
         [1,4,256]))
    // Compute similarity between a dictionary and a batch
         of query vectors
    %match_prob_multi_batched_1[1]: call_function[nvsa.
         match_prob_multi_batched](args = (%
         inv_binding_circular_2[1,4,256], %vec_5[7,4,256]))
    %sum_1[1] : call_function[torch.sum](args = (%
         match_prob_multi_batched_1[1]))
    %clamp_1[1] : call_function[torch.clamp](args = (%sum_1
         [1]))
    %mul_1[1] : call_function[operator.mul](args = (%
         match_prob_1[1], %clamp_1[1]))
    ...
```

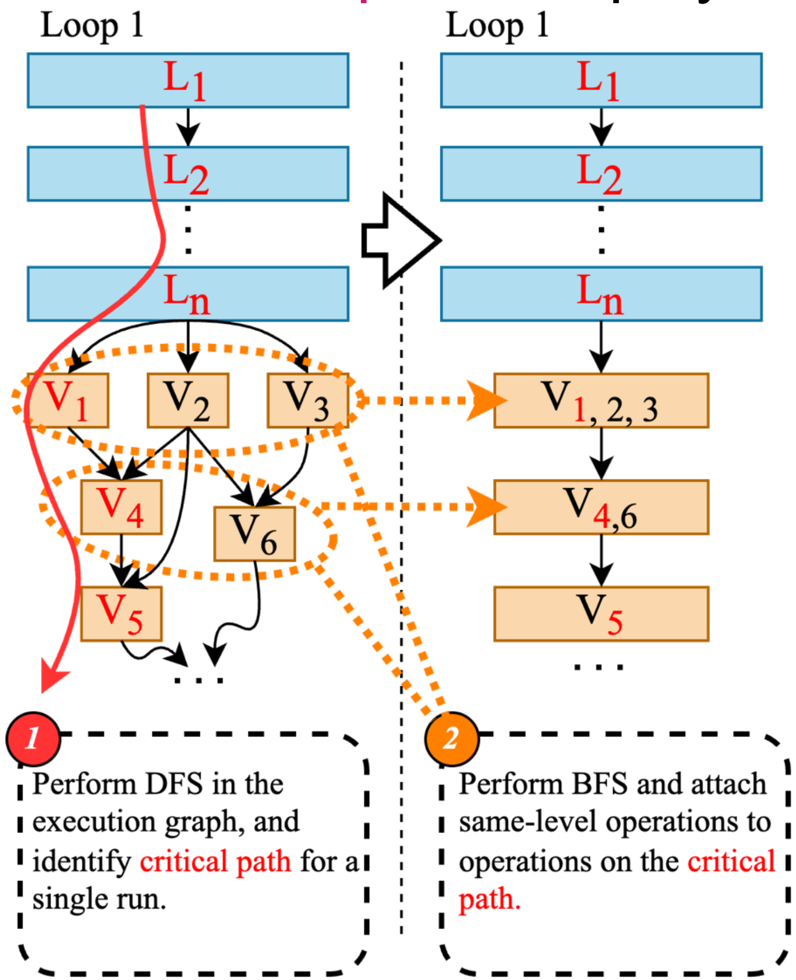
NSFlow Frontend

2. Generate Dataflow Graph for deployment:



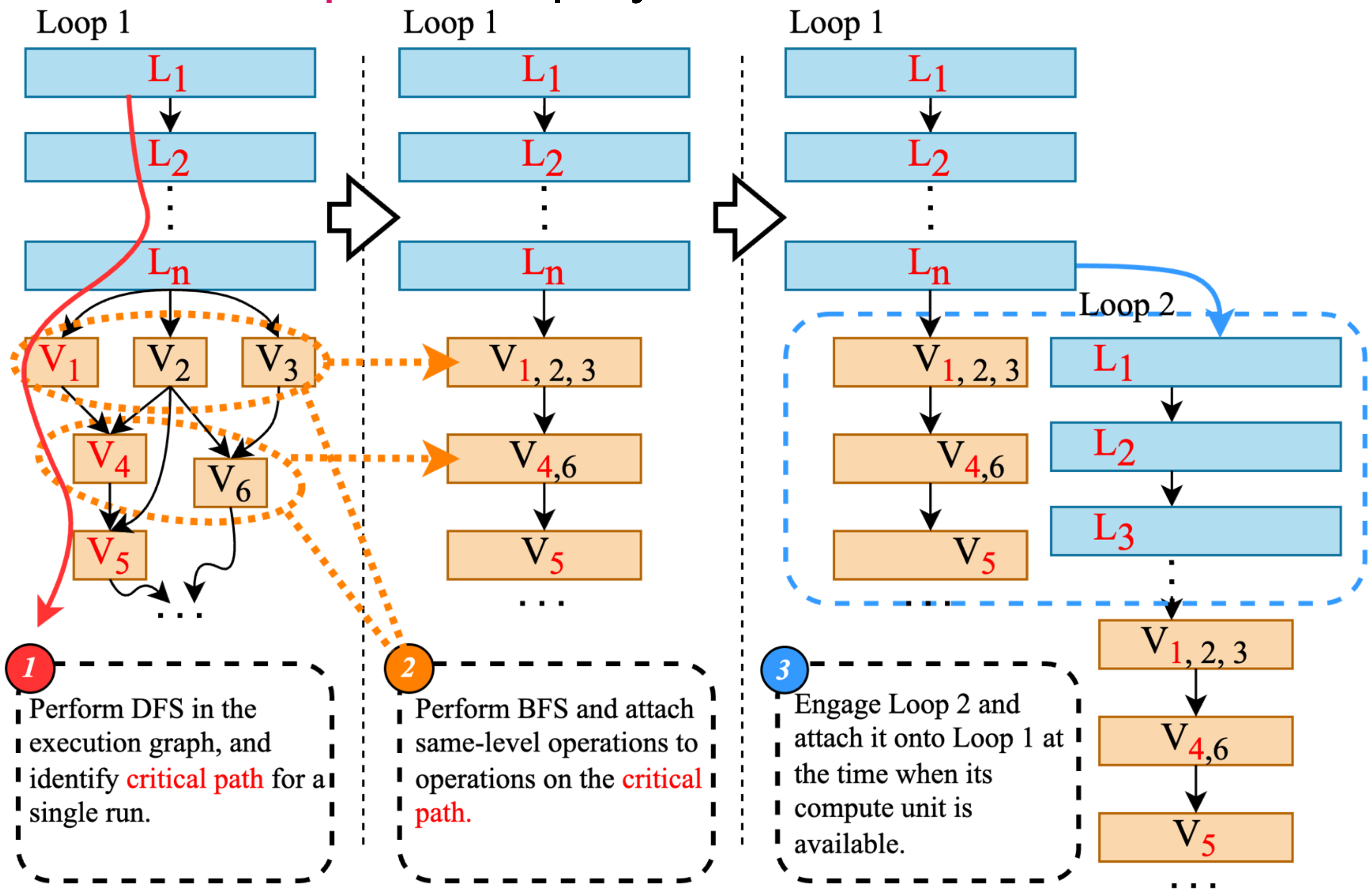
NSFlow Frontend

2. Generate Dataflow Graph for deployment:



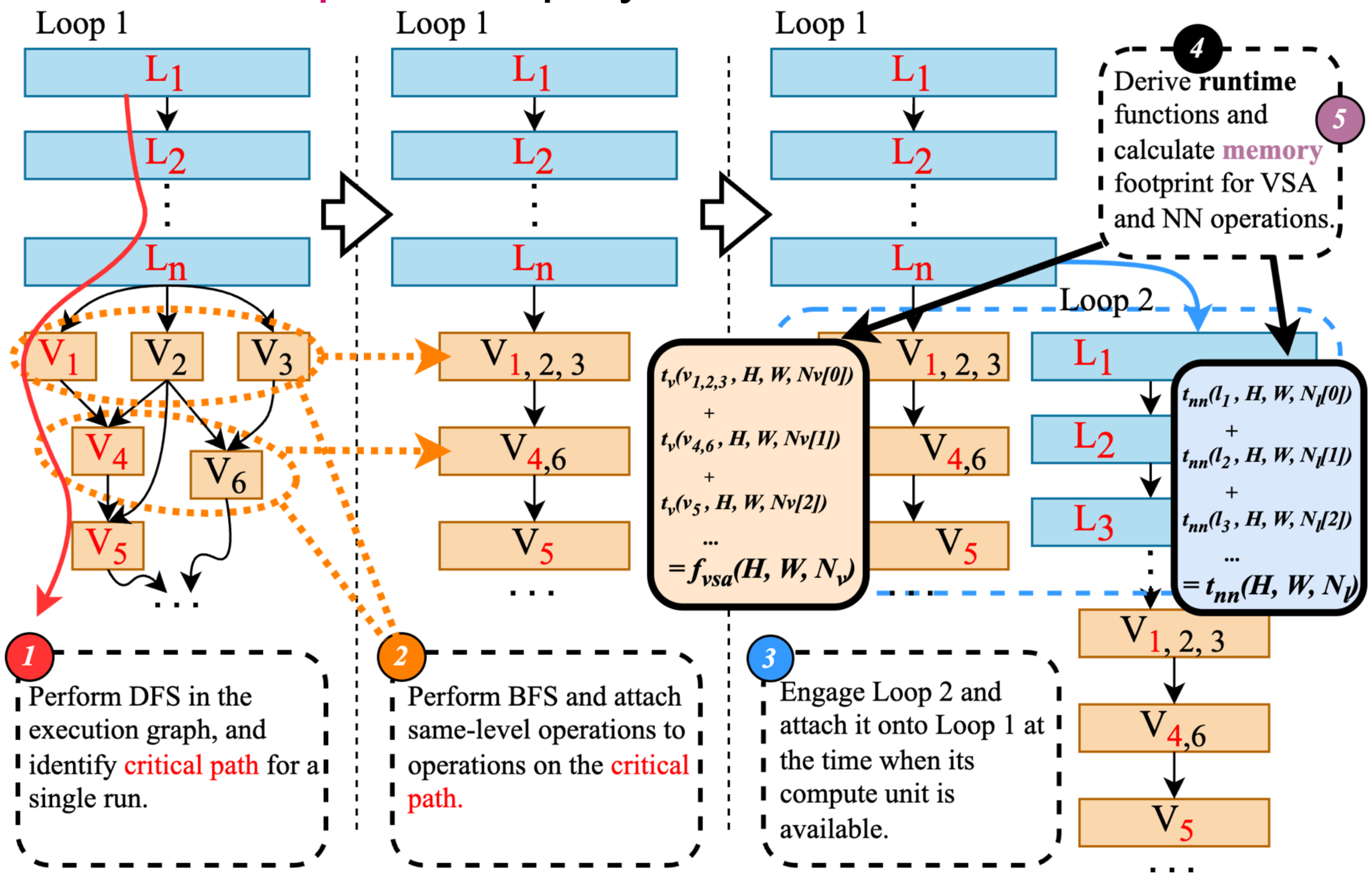
NSFlow Frontend

2. Generate Dataflow Graph for deployment:



NSFlow Frontend

2. Generate Dataflow Graph for deployment:



NSFlow Frontend

3. Explore optimal HW config and array partition strategy

NSFlow Frontend

3. Explore optimal HW config and array partition strategy

Algorithm 1: NSFlow Two-Phase DSE Algorithm

Data: R_l, R_v, $Range_H$ (H search range), $Range_W$ (W search range), M (max #PEs), $Iter_{max}$ (Phase II max iterations)
Result: H, W, N (total #sub-arrays), N_l, N_v

$/* \ Phase\ I \ */$
for H *in* $Range_H$, W *in* $Range_W$ **do**
    $N = \lfloor M/(H \times W) \rfloor$ // get total #sub-arrays
    for $\bar{N}_l$ *in* $[1,\ N)$ **do**
        // get optimal HW config for parallel mapping
        Set all elements in N_l to $\bar{N}_l$
        Set all elements in N_v to $N - \bar{N}_l$
        $t_{para} = max(t_{nn}(H, W, N_l), t_{vsa}(H, W, N_v))$
        Save the $H, W, \bar{N}_l$ *(and* $\bar{N}_v$*) with minimal* t_{para}.
    end
    // get sequential runtime
    $t_{seq} = \Sigma_i^G f_{l_i}(H, W, N) + min(\Sigma_j^G f_{v_j,temp}(H, W, N),\ \Sigma_j^G f_{v_j,spatial}(H, W, N))$
    // Set to sequential mode in case it has better performance
    Return and set sequential mode **if** $t_{seq} < t_{para}$ **else** Continue
end
$/* \ Phase\ II \ */$
for it *in* $Iter_{max}$ **do**
    for *layer* i *in* R_l **do**
        Locate VSA node j' and j'' where layer i starts and ends
        if $t_{seq} < t_{para}$ **do** $N_l[i] - -$; $N_v[j' : j''] + +$;
        else do $N_l[i] + +$; $N_v[j' : j''] - -$;
        $t_{para} = max(t_{nn}(H, W, N_l),\ t_{vsa}(H, W, N_v))$
        Save the H, W, N_l, N_v *with minimal* t_{para}.
    end
end
Return H, W, N, N_l, N_v.

NSFlow Frontend

3. Explore optimal HW config and array partition strategy

- *Phase I:* Assuming **static partition**, find the optimal array size (H, W, N).

Algorithm 1: NSFlow Two-Phase DSE Algorithm

Data: R_l, R_v, $Range_H$ (H search range), $Range_W$ (W search range), M (max #PEs), $Iter_{max}$ (Phase II max iterations)
Result: H, W, N (total #sub-arrays), N_l, N_v

$/* \ Phase\ I\ */$
for H *in* $Range_H$, W *in* $Range_W$ **do**
  $N = \lfloor M/(H \times W) \rfloor$ // get total #sub-arrays
  for $\bar{N}_l$ *in* $[1,\ N)$ **do**
    // get optimal HW config for parallel mapping
    Set all elements in N_l to $\bar{N}_l$
    Set all elements in N_v to $N - \bar{N}_l$
    $t_{para} = max(t_{nn}(H, W, N_l), t_{vsa}(H, W, N_v))$
    Save the $H, W, \bar{N}_l$ *(and* $\bar{N}_v$*) with minimal* t_{para}.
  end
  // get sequential runtime
  $t_{seq} = \Sigma_i^G f_{l_i}(H, W, N) + min(\Sigma_j^G f_{v_j,temp}(H, W, N),\ \Sigma_j^G f_{v_j,spatial}(H, W, N))$
  // Set to sequential mode in case it has better performance
  Return and set sequential mode **if** $t_{seq} < t_{para}$ **else** Continue
end
$/* \ Phase\ II\ */$
for *it in* $Iter_{max}$ **do**
  for *layer* i *in* R_l **do**
    Locate VSA node j' and j'' where layer i starts and ends
    if $t_{seq} < t_{para}$ **do** $N_l[i] - -;\ N_v[j' : j''] + +;$
    else do $N_l[i] + +;\ N_v[j' : j''] - -;$
    $t_{para} = max(t_{nn}(H, W, N_l),\ t_{vsa}(H, W, N_v))$
    Save the H, W, N_l, N_v *with minimal* t_{para}.
  end
end
Return H, W, N, N_l, N_v.

NSFlow Frontend

3. Explore optimal HW config and array partition strategy

- *Phase I:* Assuming **static partition**, find the optimal array size (H, W, N).

- *Phase II:* Fine-tune for **dynamic partition** to better balance Neuro & Symb at runtime

Algorithm 1: NSFlow Two-Phase DSE Algorithm

Data: R_l, R_v, $Range_H$ (H search range), $Range_W$ (W search range), M (max #PEs), $Iter_{max}$ (Phase II max iterations)
Result: H, W, N (total #sub-arrays), N_l, N_v

$/ * \; Phase \; I * /$
for H *in* $Range_H$, W *in* $Range_W$ **do**
    $N = \lfloor M/(H \times W) \rfloor$ // get total #sub-arrays
    for $\bar{N_l}$ *in* $[1, \; N)$ **do**
        // get optimal HW config for parallel mapping
        Set all elements in N_l to $\bar{N_l}$
        Set all elements in N_v to $N - \bar{N_l}$
        $t_{para} = max(t_{nn}(H, W, N_l), t_{vsa}(H, W, N_v))$
        Save the $H, W, \bar{N_l}$ *(and* $\bar{N_v}$*) with minimal* t_{para}.
    end
    // get sequential runtime
    $t_{seq} = \Sigma_i^G f_{l_i}(H, W, N) + min(\Sigma_j^G f_{v_j,temp}(H, W, N), \; \Sigma_j^G f_{v_j,spatial}(H, W, N))$
    // Set to sequential mode in case it has better performance
    Return and set sequential mode **if** $t_{seq} < t_{para}$ **else** Continue
end
$/ * \; Phase \; II * /$
for *it in* $Iter_{max}$ **do**
    for *layer* i *in* R_l **do**
        Locate VSA node j' and j'' where layer i starts and ends
        if $t_{seq} < t_{para}$ **do** $N_l[i] - -$; $N_v[j' : j''] + +$;
        else do $N_l[i] + +$; $N_v[j' : j''] - -$;
        $t_{para} = max(t_{nn}(H, W, N_l), \; t_{vsa}(H, W, N_v))$
        Save the H, W, N_l, N_v *with minimal* t_{para}.
    end
end
Return H, W, N, N_l, N_v.

NSFlow Frontend

3. Explore optimal HW config and array partition strategy

- *Phase I:* Assuming **static partition**, find the optimal array size (H, W, N).
- *Phase II:* Fine-tune for **dynamic partition** to better balance Neuro & Symb at runtime

✓ Reduces search space x10^{100}

	HW config (H, W, N)	Array partition and mapping	Total design space, $m = 10$
Original	$m \times (m+1)/2$	$(N-1)^k$ for each N	10^{300}
DAG	$PhaseI$: $1/4 \leq H/W \leq 16$	$PhaseII$: Iter $\times$ #layers	10^3

Algorithm 1: NSFlow Two-Phase DSE Algorithm

Data: R_l, R_v, $Range_H$ (H search range), $Range_W$ (W search range), M (max #PEs), $Iter_{max}$ (Phase II max iterations)
Result: H, W, N (total #sub-arrays), N_l, N_v

$/ * $ *Phase I* $ * /$
for H *in* $Range_H$, W *in* $Range_W$ **do**
  $N = \lfloor M/(H \times W) \rfloor$ // get total #sub-arrays
  for $\bar{N_l}$ *in* $[1,\ N)$ **do**
    // get optimal HW config for parallel mapping
    Set all elements in N_l to $\bar{N_l}$
    Set all elements in N_v to $N - \bar{N_l}$
    $t_{para} = max(t_{nn}(H, W, N_l), t_{vsa}(H, W, N_v))$
    Save the $H, W, \bar{N_l}$ *(and* $\bar{N_v}$*) with minimal* t_{para}.
  end
  // get sequential runtime
  $t_{seq} = \Sigma_i^G f_{l_i}(H, W, N) + min(\Sigma_j^G f_{v_j,temp}(H, W, N),\ \Sigma_j^G f_{v_j,spatial}(H, W, N))$
  // Set to sequential mode in case it has better performance
  Return and set sequential mode **if** $t_{seq} < t_{para}$ **else** Continue
end
$/ * $ *Phase II* $ * /$
for *it in* $Iter_{max}$ **do**
  for *layer* i *in* R_l **do**
    Locate VSA node j' and j'' where layer i starts and ends
    if $t_{seq} < t_{para}$ **do** $N_l[i]--$; $N_v[j':j'']++$;
    else do $N_l[i]++$; $N_v[j':j'']--$;
    $t_{para} = max(t_{nn}(H, W, N_l),\ t_{vsa}(H, W, N_v))$
    Save the H, W, N_l, N_v *with minimal* t_{para}.
  end
end
Return H, W, N, N_l, N_v.

Evaluation

Evaluation

Experiments setup

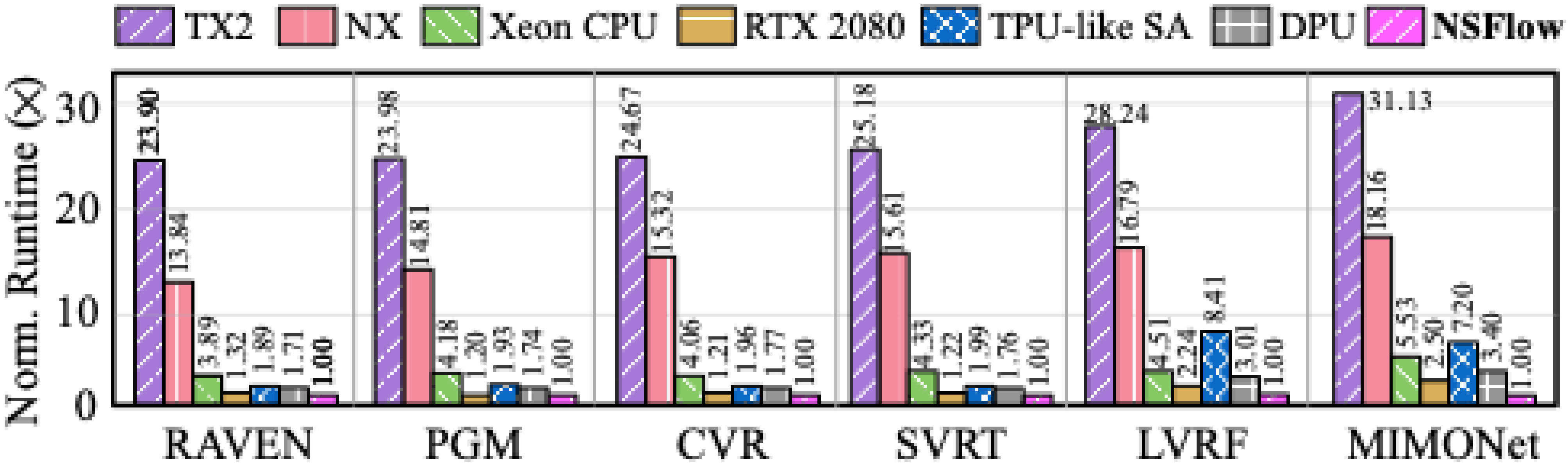
- Workloads:
  - Algorithms: *NVSA, MIMONet, LVRF*
  - Datasets: *RAVEN, I-RAVEN, PGM, CVR, and SVRT*
- Hardwares:
  - Baselines: *TX2, Xavier NX, Xeon CPU, RTX 3080, ML accelerators (TPU, Xilinx DPU)*
  - FPGA deployment: *AMD U250*

Workloads	Precision		AdArray Configuration		SIMD Size	On-chip SRAM Blocks (BRAM)			On-chip Cache (URAM)	AMD U250 Utilization						Frequency
	NN	Symb	Size (H, W, N)	Default Partition ($\bar{N}_l$: $\bar{N}_v$)		MemA1, MemA2	Mem B	Mem C		DSP	LUT	FF	BRAM	URAM	LUTRAM	
NVSA	INT8	INT4	32, 16, 16	14 : 2	64	2.7 MB, 1.1 MB	2.7 MB	1.6 MB	16.2 MB	89%	56%	60%	34%	8%	24%	272 MHz
MIMONet	INT8	INT8	32, 32, 8	6 : 2	64	3.4 MB, 1.2 MB	3.4 MB	2.1 MB	20.1 MB	89%	44%	52%	43%	10%	20%	272 MHz
LVRF	INT8	INT4	32, 16, 16	14 : 2	64	2.7 MB, 0.96 MB	2.7 MB	1.4 MB	15.5 MB	89%	56%	60%	31%	7%	24%	272 MHz

Evaluation

End-to-end runtime improvement

✓ ~2x speedup over GPU, 2~8x speedup over TPU, ~3x speedup over DPU



Evaluation

Mixed-precision optimization

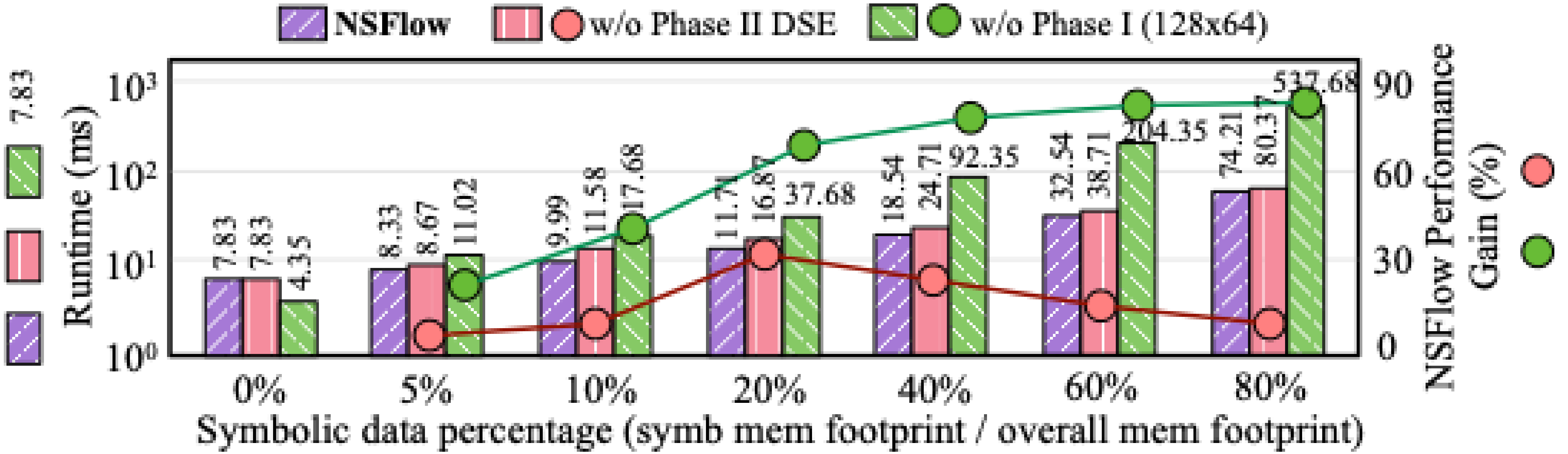
✓~6x Memory reduction

Reasoning Accuracy	FP32	FP16	INT8	MP (IN8 for NN, INT4 for Symb)	INT4
RAVEN [39]	98.9%	98.9%	98.7%	98.0%	92.5%
I-RAVEN [16]	99.0%	98.9%	98.8%	98.1%	91.3%
PGM [3]	68.7%	68.6%	68.4%	67.4%	59.9%
Memory	32MB	16MB	8MB	5.5MB	4MB

Evaluation

Scalability

✓ Only 4x runtime increase when symbolic workloads scale by 150x



Conclusion

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

Conclusion

NSFlow is the first end-to-end design automation framework dedicated to accelerate generic NSAI systems

Conclusion

NSFlow is the first end-to-end design automation framework dedicated to accelerate generic NSAI systems

- *Identifies the unique optimization opportunities for NSAI acceleration*

Conclusion

NSFlow is the first end-to-end design automation framework dedicated to accelerate generic NSAI systems

- *Identifies the unique optimization opportunities for NSAI acceleration*
- *Explores the dataflow and architecture design space with a novel algorithm*

Conclusion

NSFlow is the first end-to-end design automation framework dedicated to accelerate generic NSAI systems

- *Identifies the unique optimization opportunities for NSAI acceleration*
- *Explores the dataflow and architecture design space with a novel algorithm*
- *Generates a efficient scalable design for FPGA deployment*

Conclusion

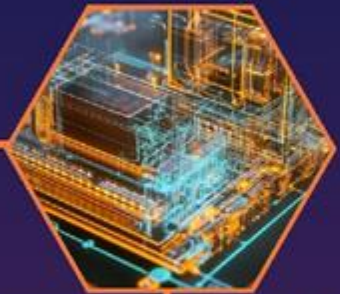
NSFlow paves the way
for advancing efficient cognitive reasoning systems and
unlocking new possibilities in NSAI.


AI


Security


Systems

Thank you!

EDA


Design


THE CHIPS TO SYSTEMS CONFERENCE

62

SPONSORED BY CEDA sigda