ETH zürich

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

PACE: An Optimal Piecewise Polynomial Approximation Unit for Flexible and Efficient Transformer Non-linearity Acceleration

Arpan Suravi Prasad*, Gamze İslamoğlu*, Luca Bertaccini*, Davide Rossi*, Francesco Conti*, Luca Benini*‡

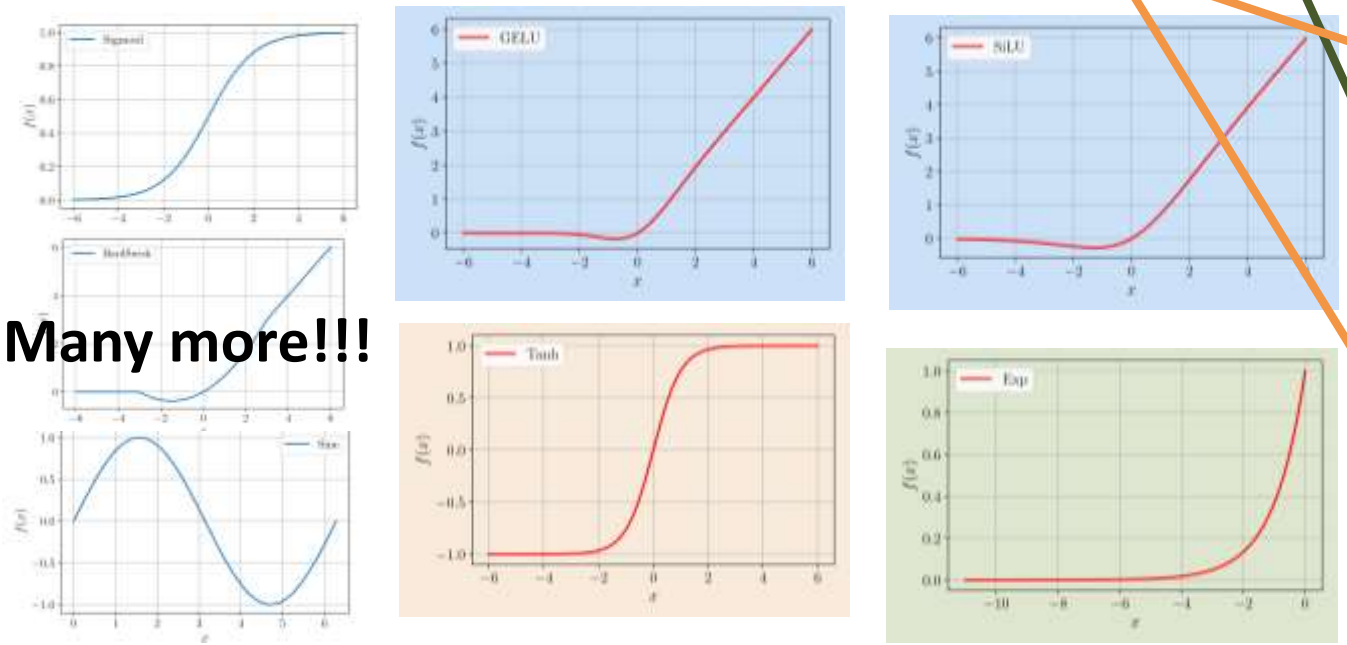
**Integrated Systems Laboratory, ETH Zurich;*
‡*DEI, University of Bologna*

pulp-platform.org
@pulp_platform
company/pulp-platform
youtube.com/pulp_platform

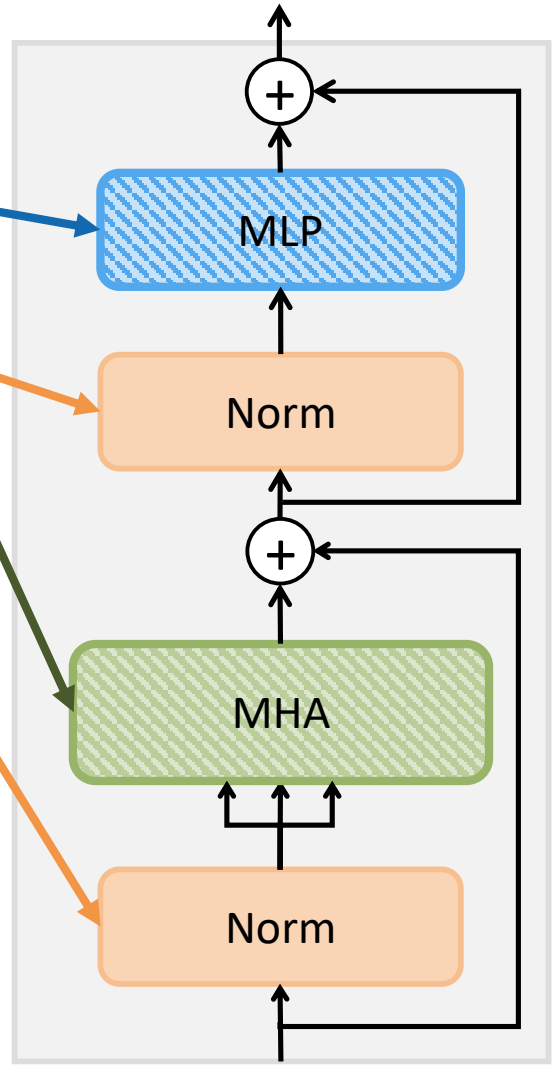
PULP Platform
Open Source Hardware, the way it should be!

Transformers Are Dominating AI

- Matrix multiplication remains the core operation
- Various non-linearities are being used
- New non-linearities are being evaluated*



Many more!!!

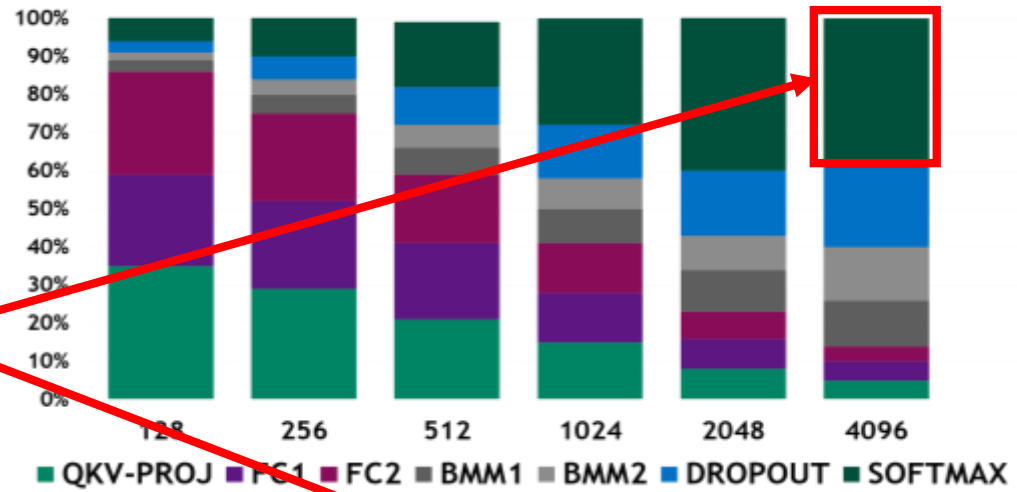


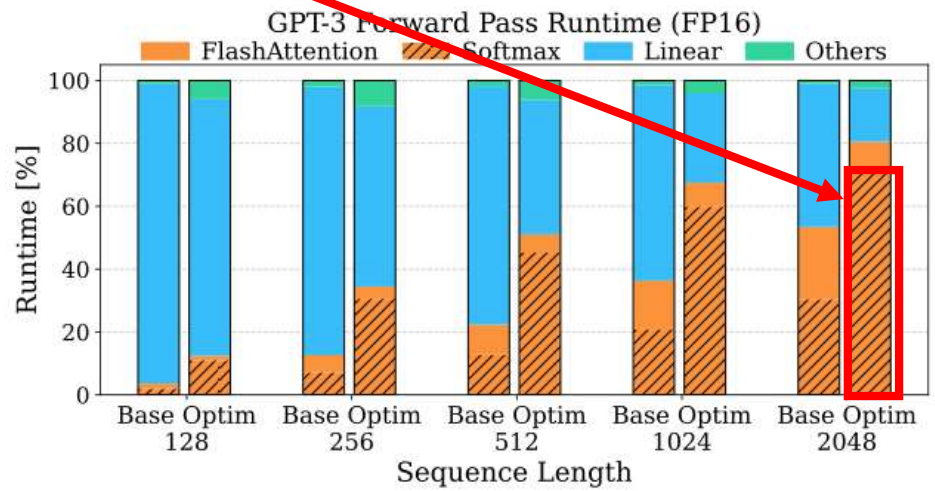
* Zhu, Jiachen, et al. "Transformers without normalization." *Proceedings of the CVPR* 2025

Non-linearities Are Emerging as a New Bottleneck

- Linear operation is heavily optimised
- Non-linear operation execution is slow
- Multiple and different nonlinearities
- Reliance on complex functions

PACE efficiently accelerates a wide range of non-linearities





Stevens, Jacob R., et al. "Softermax: Hardware/software co-design of an efficient softmax for transformers", DAC-2021

Wang, Run, et al. "VEXP: A Low-Cost RISC-V ISA Extension for Accelerated Softmax Computation in Transformers." *arXiv preprint arXiv:2504.11227* (2025).

Existing Methods for Non-linearity Acceleration

Specialised hardware for a single function
High performance & Energy Efficiency
Not adaptable to other functions

Pure polynomial approximation
Very high Accuracy and dynamic range
Very High computational cost

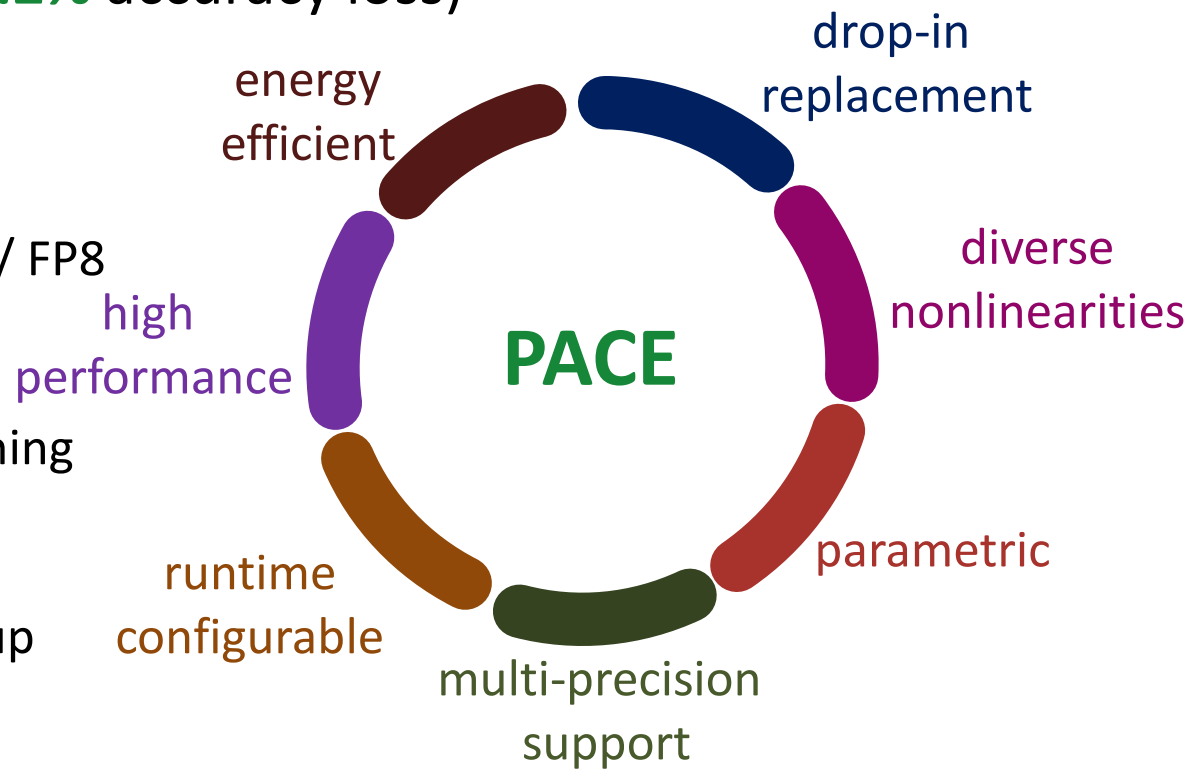
Lookup table method
No computation required at runtime
High area cost to support fine granularity
or a wide dynamic range

Piecewise linear approximation(PwLA)
High Accuracy and good dynamic range
Higher partition count → larger area
requirement to maintain accuracy

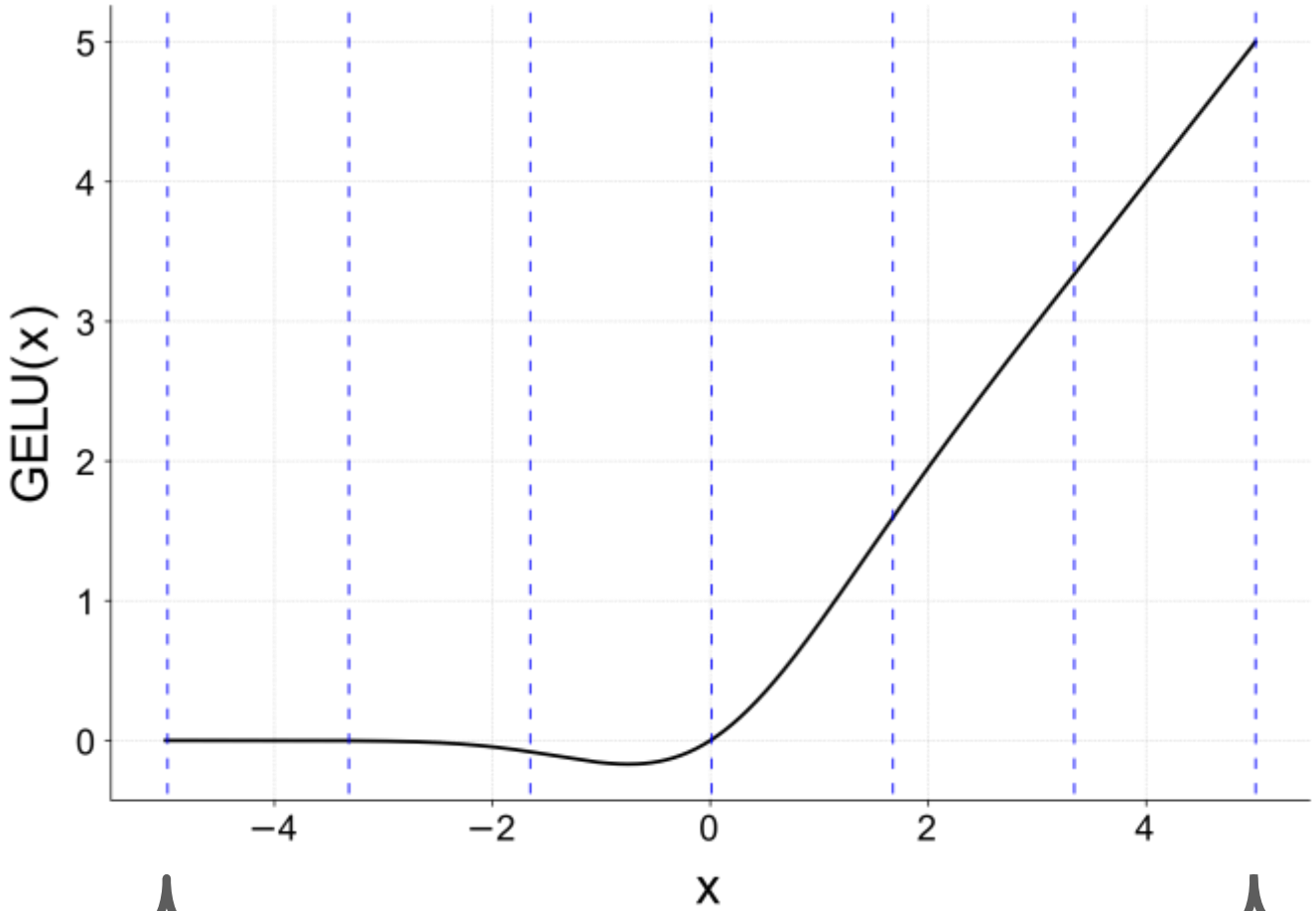
This Work – Piecewise Polynomial Approximation(PwPA)
High Accuracy and wide dynamic range
significantly fewer partitions compared to PwLA

Contributions: Our Approach to Non-linearities

- **PACE –** Piecewise polynomial Approximation Compute Engine
- Evaluations on various SoA AI models (**< 0.2%** accuracy loss)
- A parametric datapath
  - Arbitrary partitions, degrees
  - Multi-precision support → FP32 / FP16 / BFP16 / FP8
- Runtime configurable
  - arbitrary degrees and partitions → accuracy tuning
- System-level integration
  - ISA extension to RISC-V cores in a multicore setup
  - speedups of **65x** and **750x** over baselines
  - Energy efficiency boosts **19x** and **150x**



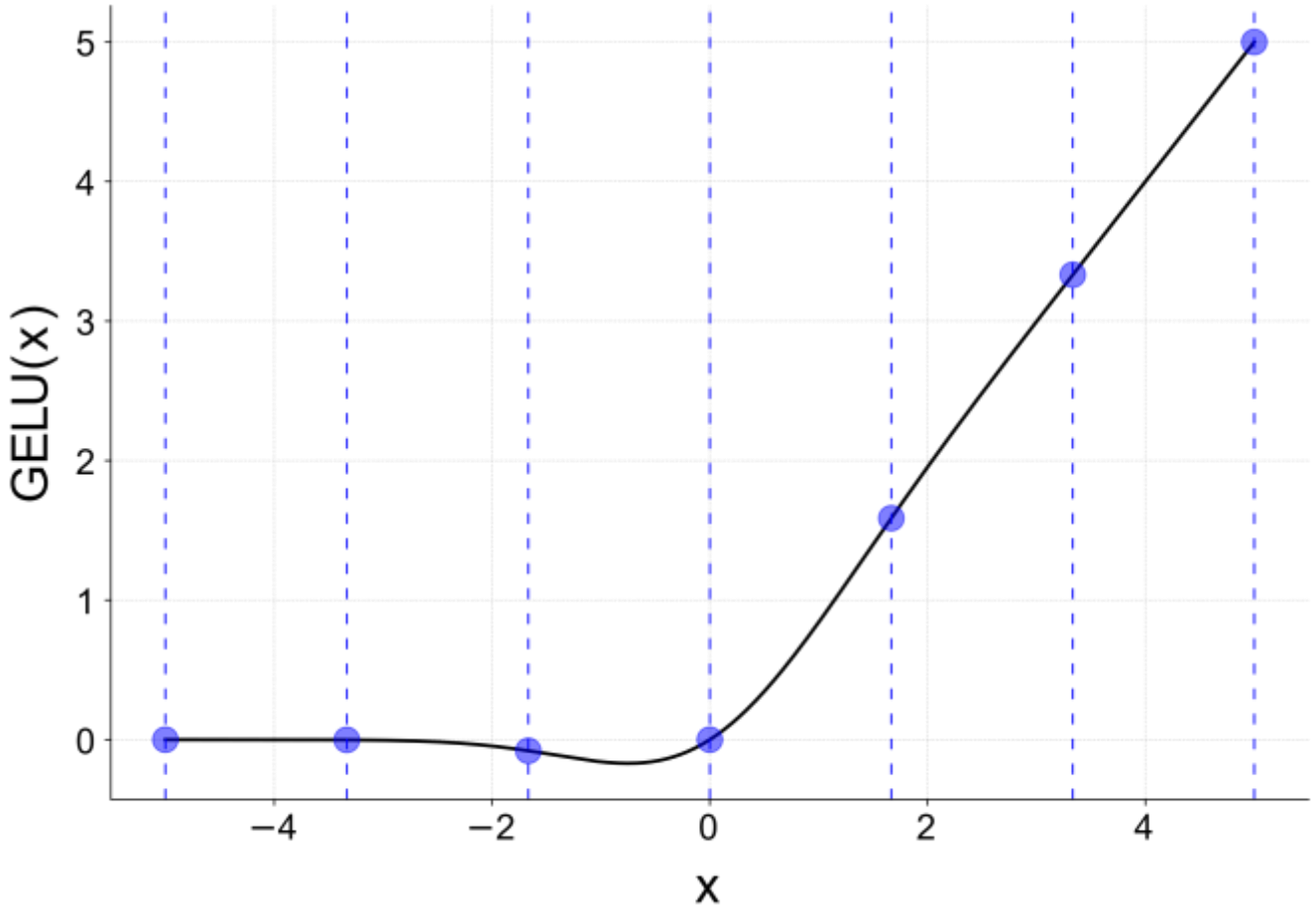
PwPA Method: Uniform Partitioning



Degree = 2, Partitions = 8

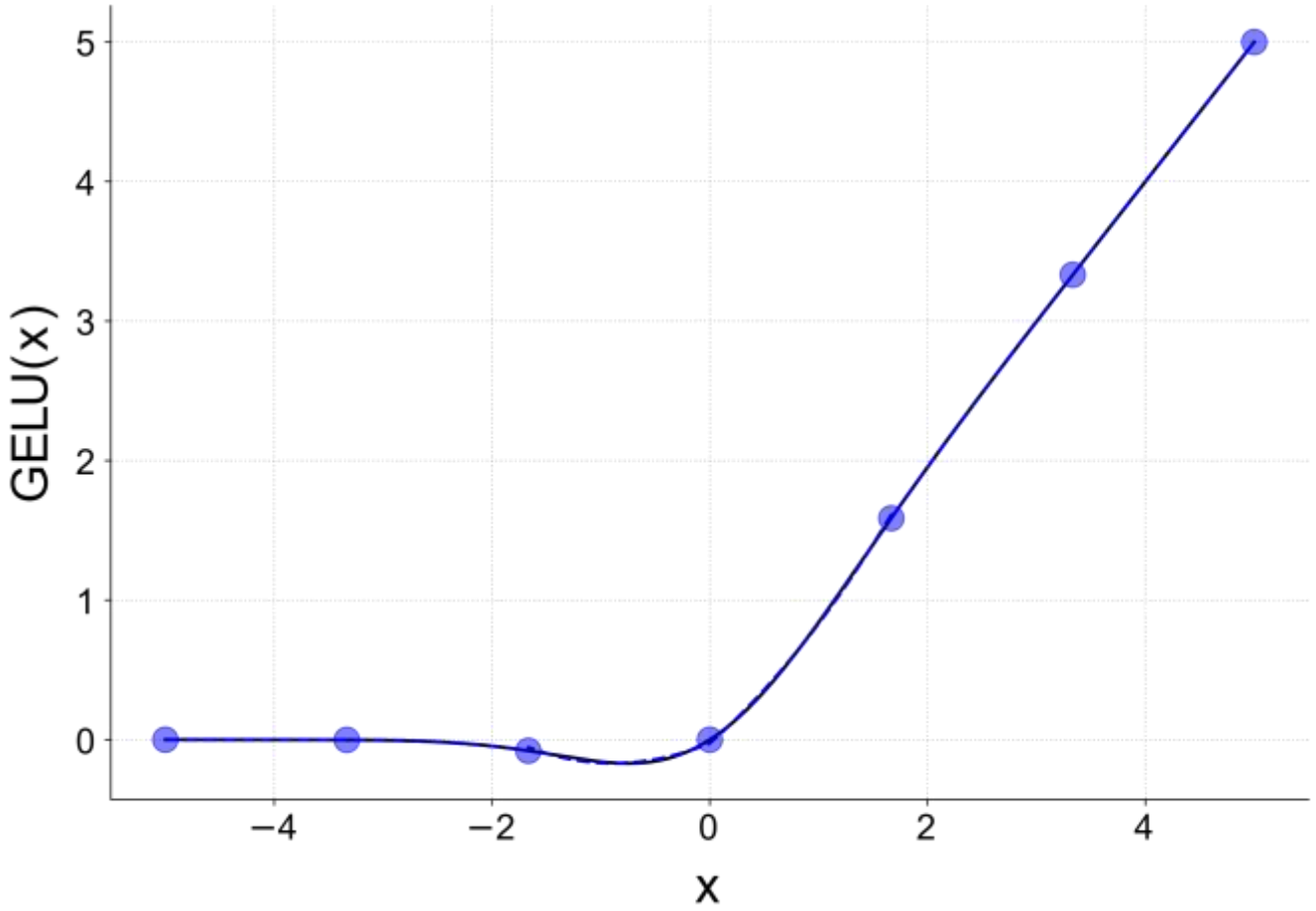
[-∞, 5] | 6 partitions | [5, ∞]

PwPA Method: Uniform Breakpoint Initialisation



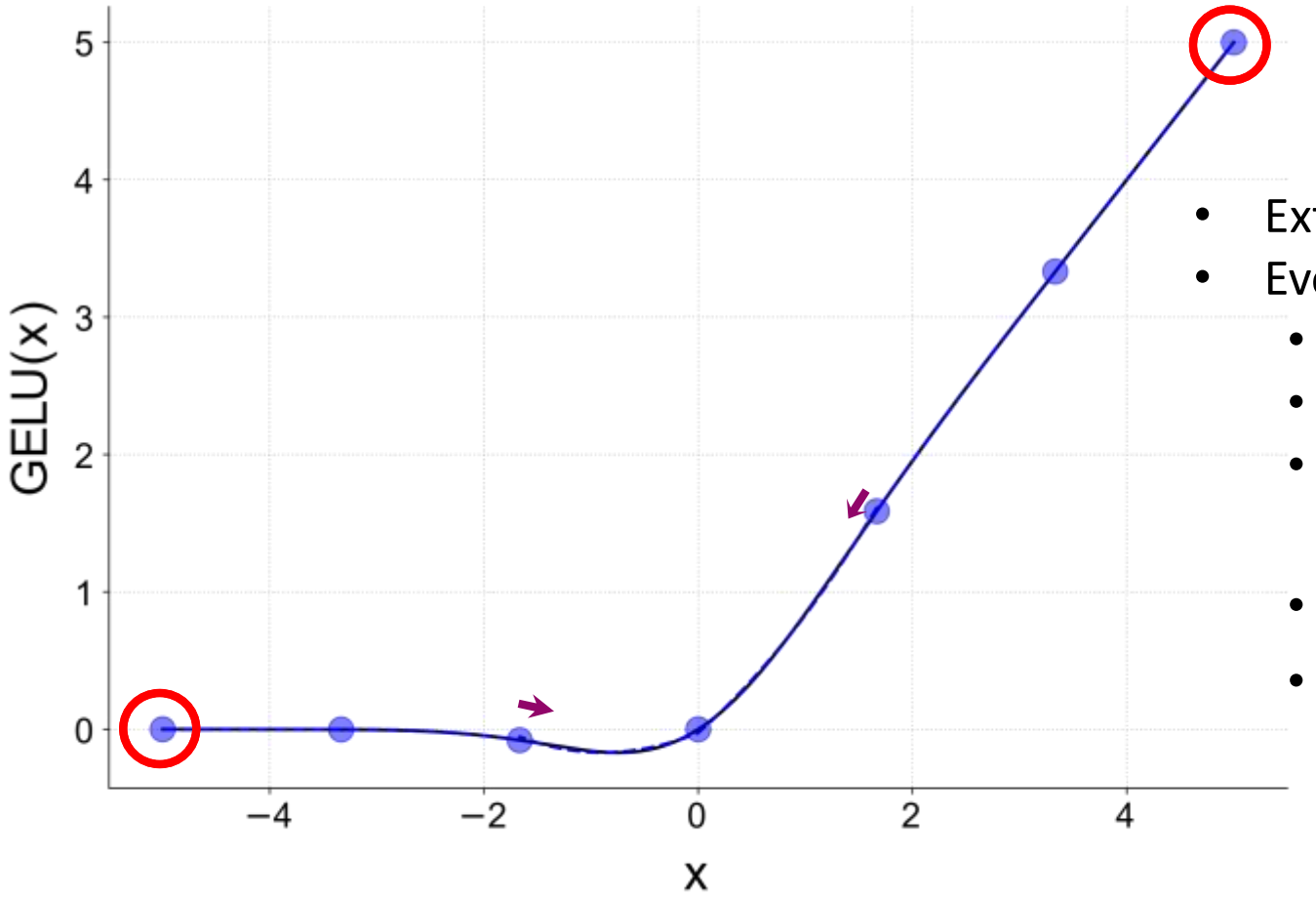
Breakpoint(***bp***) initialisation

PwPA Method: Polynomial Fit with Uniform Breakpoints



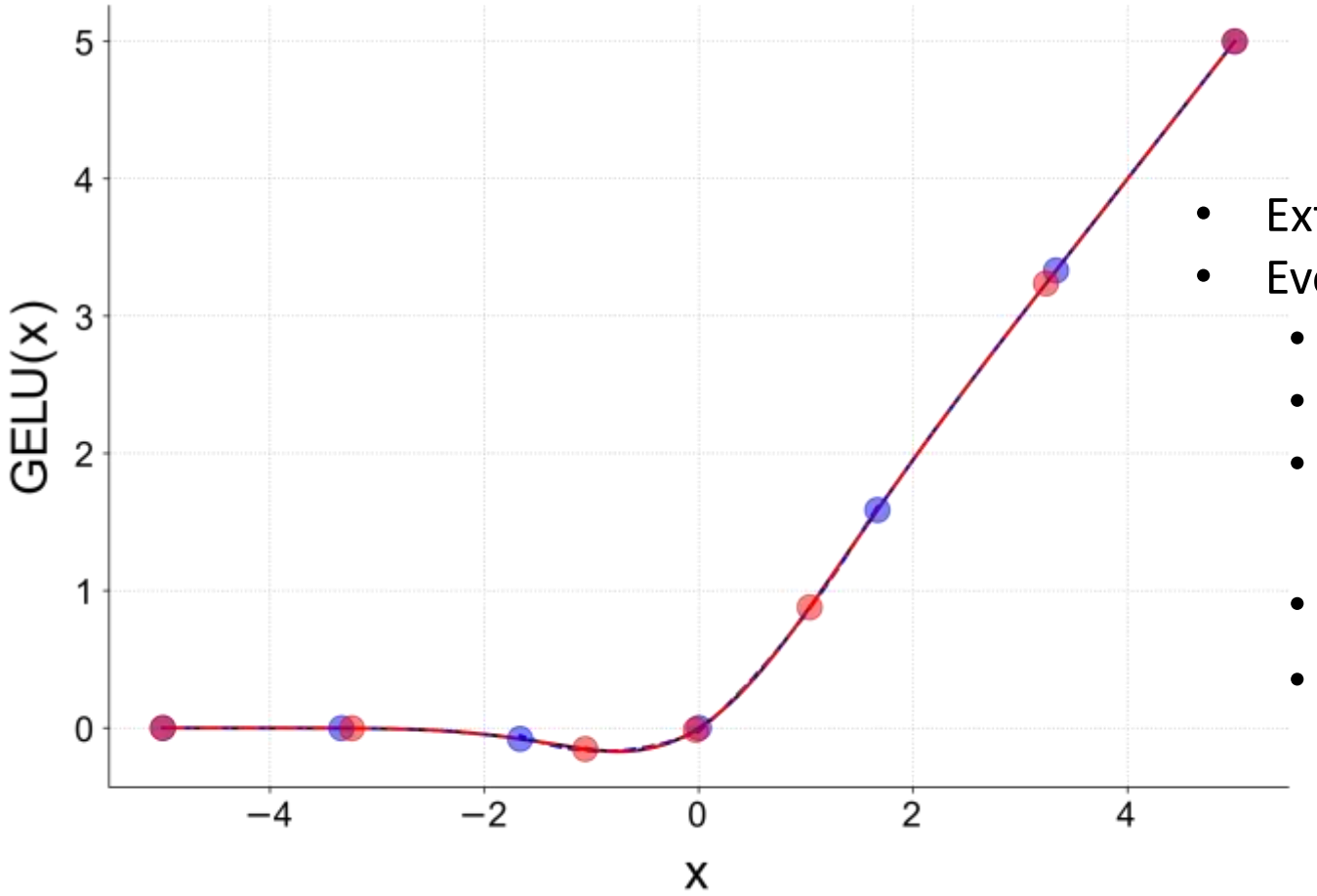
Chebyshev fit per partition using uniform ***bps***

PwPA Method: Algorithm to Obtain Optimal Breakpoints



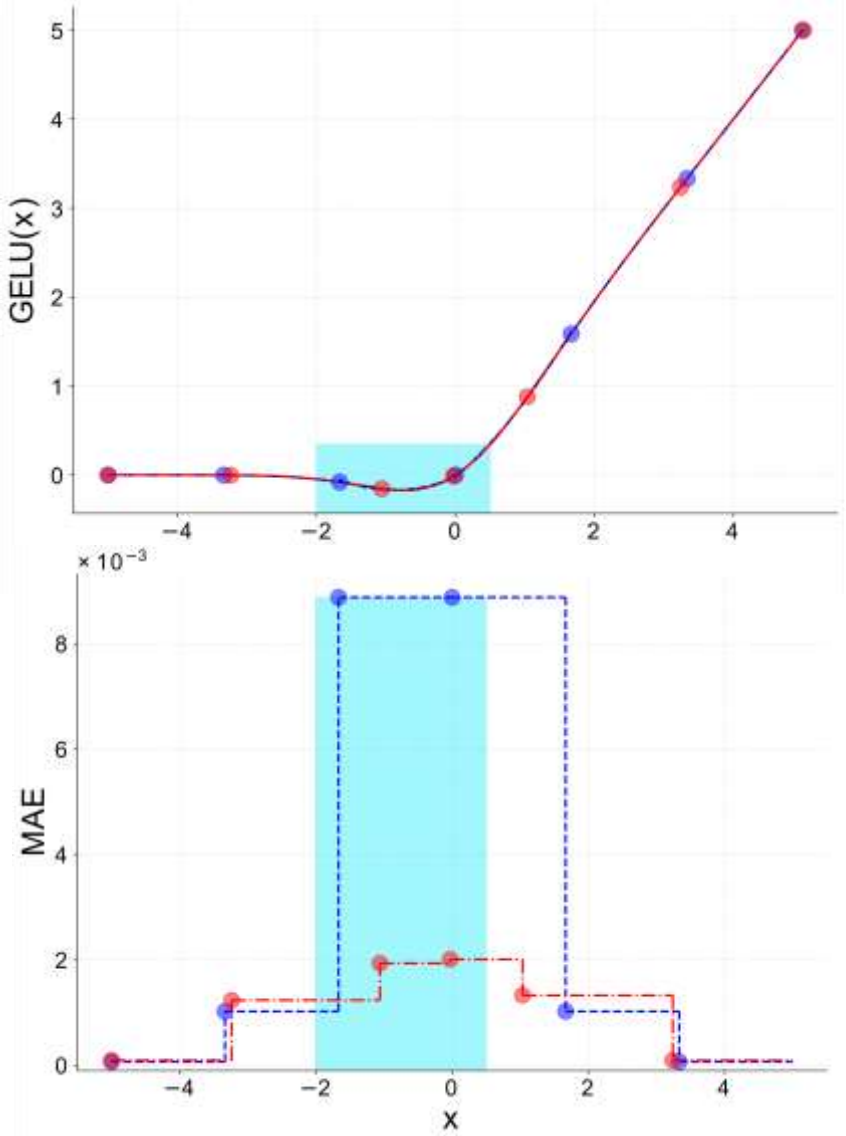
- Extreme ***bps*** are fixed
- Every iteration
  - each ***bp*** perturbed by ***h***
  - Chebyshev fit with updated ***bp***
  - Loss(L) → Mean Square Error golden vs Chebyshev fit
  - Gradient = $\frac{L_{updated} - Li_{nitial}}{h}$
  - ***bps*** updated using gradient descent

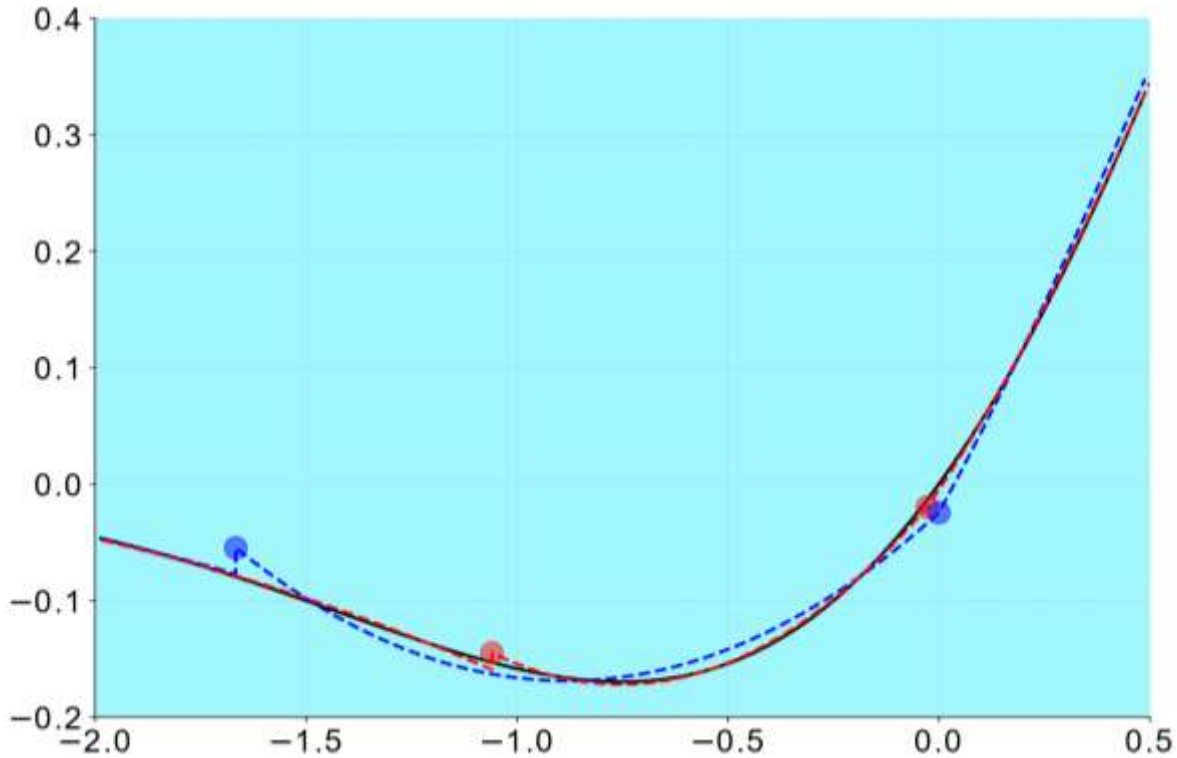
PwPA Method: Polynomial Fit Using Optimal Breakpoints



- Extreme ***bps*** are fixed
- Every iteration
  - each ***bp*** perturbed by ***h***
  - Chebyshev fit with updated ***bp***
  - Loss(L) → Mean Square Error golden vs Chebyshev fit
  - Gradient = $\frac{L_{updated} - Li_{nitial}}{h}$
  - ***bps*** updated using gradient descent

PwPA Method: Initial vs. Optimal Fit



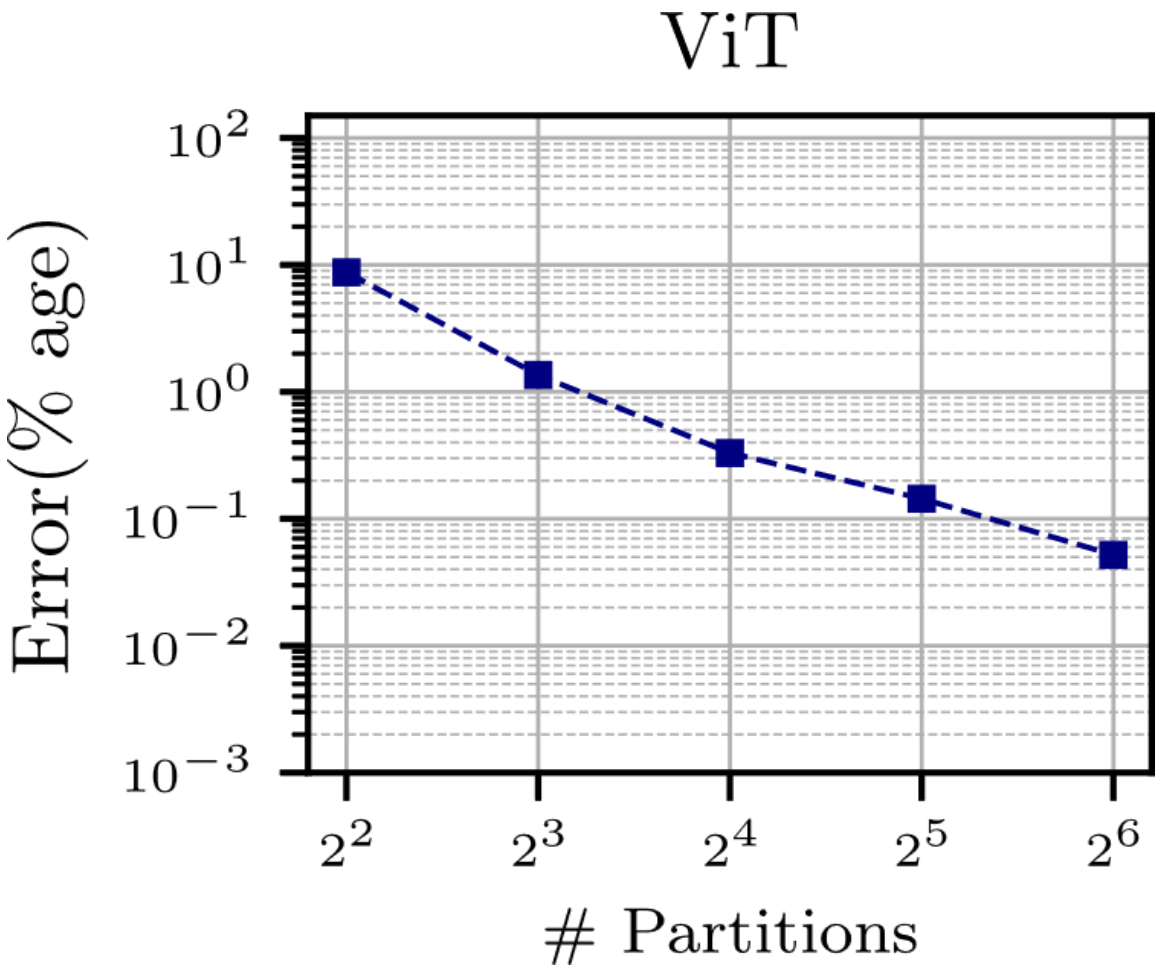


PwPA with optimal bps achieves a superior fit

Results: PwPA Evaluation Setup on AI Workloads

- Hardware: NVIDIA RTX 3080 GPU
- Drop-in replacement on pretrained models (inference)
- Models Evaluated:
  - *Image:* MobileNetV3, EfficientNetV2-S, MobileViT, ViT
  - *LLM:* OpenLLaMA 3B v2
- Nonlinearities Targeted:
  - SiLU, GELU, Exp, Hardswish, Sigmoid, Sine, Cosine
- Tasks:
  - *Image Classification:* ImageNet
  - *NLP:* PIQA, OpenBookQA, Winogrande, ARC-Easy, ARC-Challenge
- PwPA Configurations Swept:
  - Degrees: [1, 2, 3] Partitions: [4, 8, 16, 32, 64]

Results: PwPA Accuracy on ViT

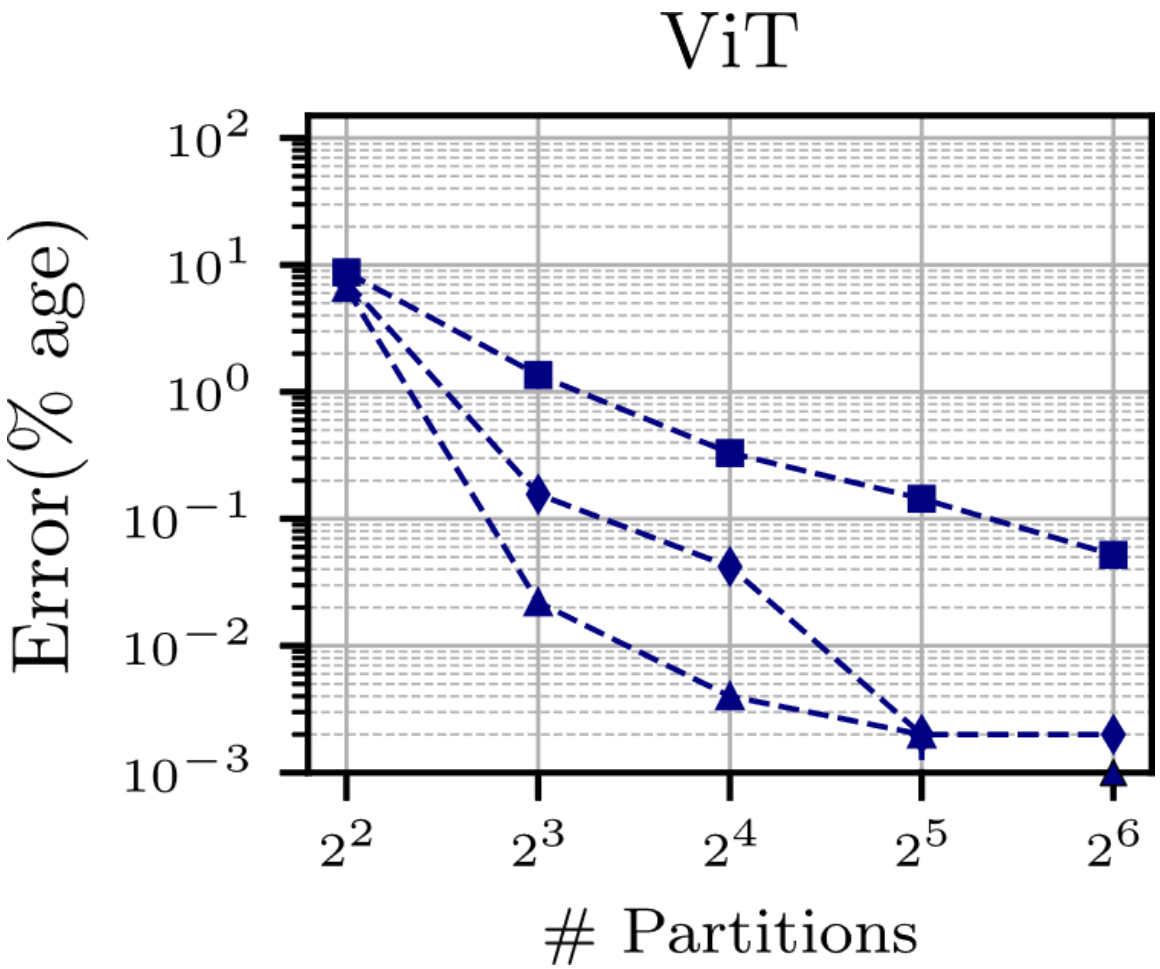


Degree 1 Degree 2 Degree 3

gelu

- ↑ Partitions → Error↓

Results: PwPA Accuracy on ViT

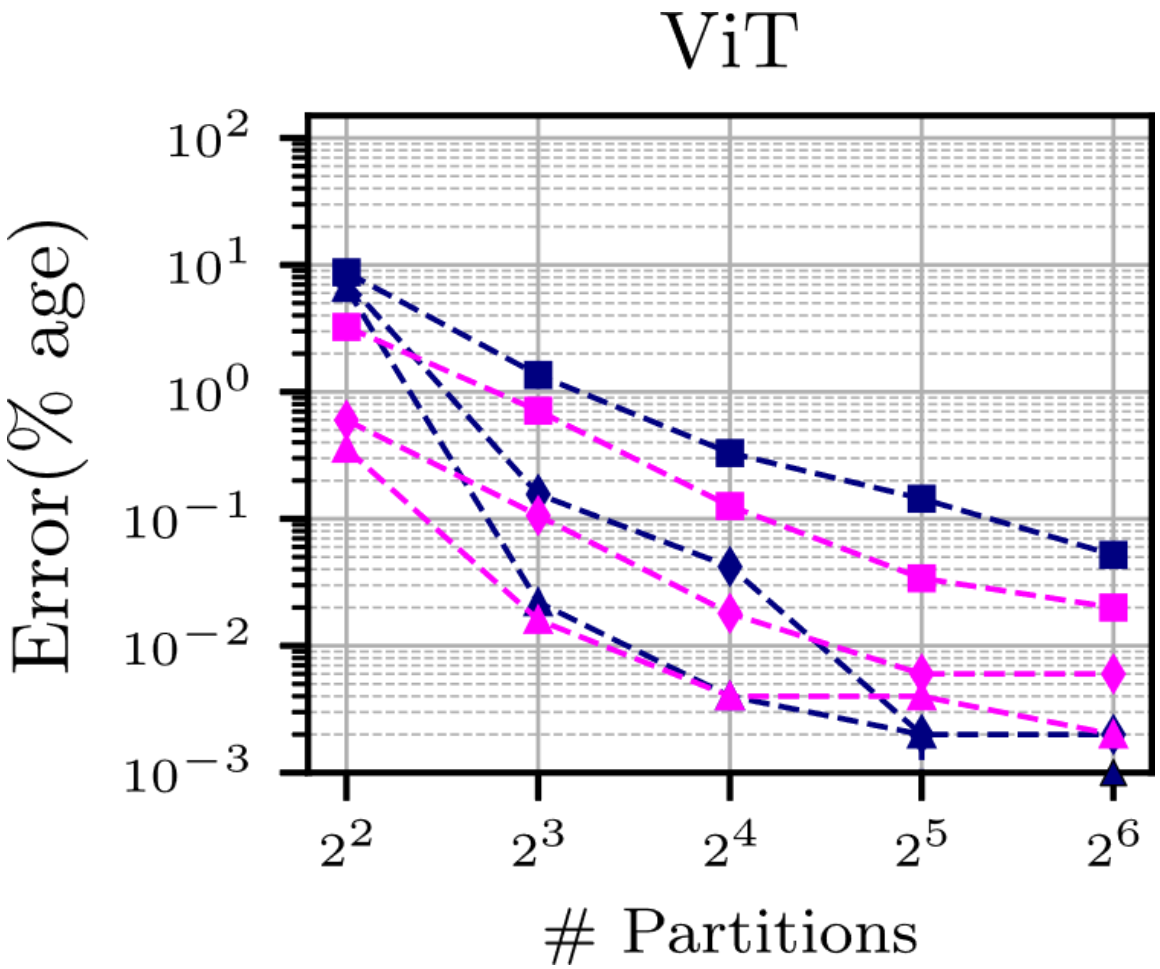


Degree 1 Degree 2 Degree 3

gelu

- ↑ Partitions → Error ↓
- ↑ Degree → Error ↓

Results: PwPA Accuracy on ViT

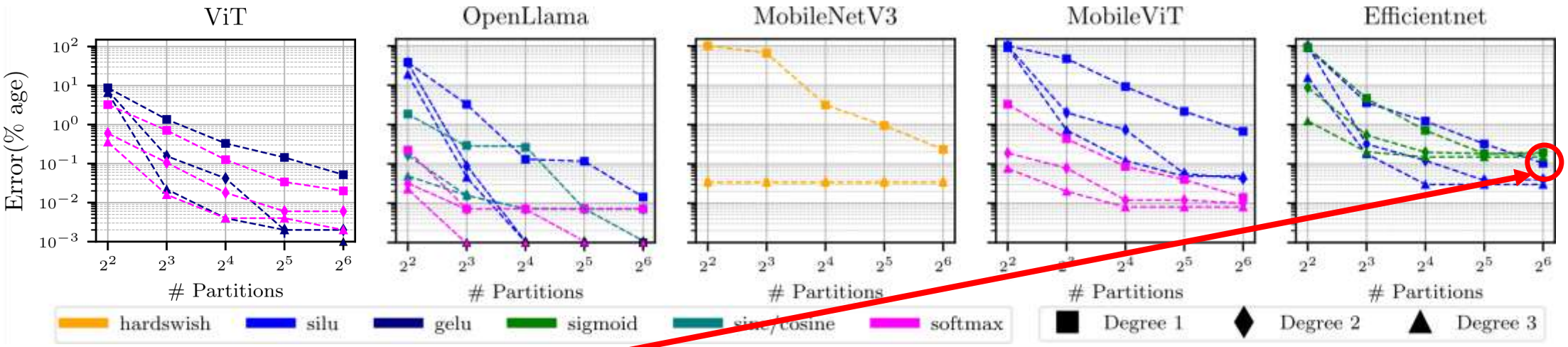


Degree 1 Degree 2 Degree 3

gelu softmax

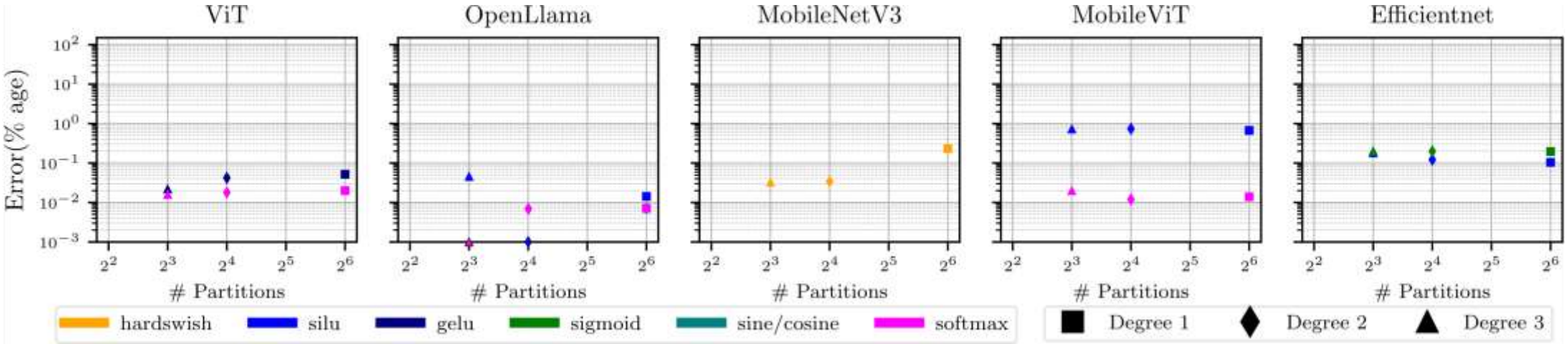
- ↑ Partitions → Error↓
- ↑ Degree → Error↓
- Similar trend for exp in softmax

Results: PwPA Accuracy on Diverse AI Workloads



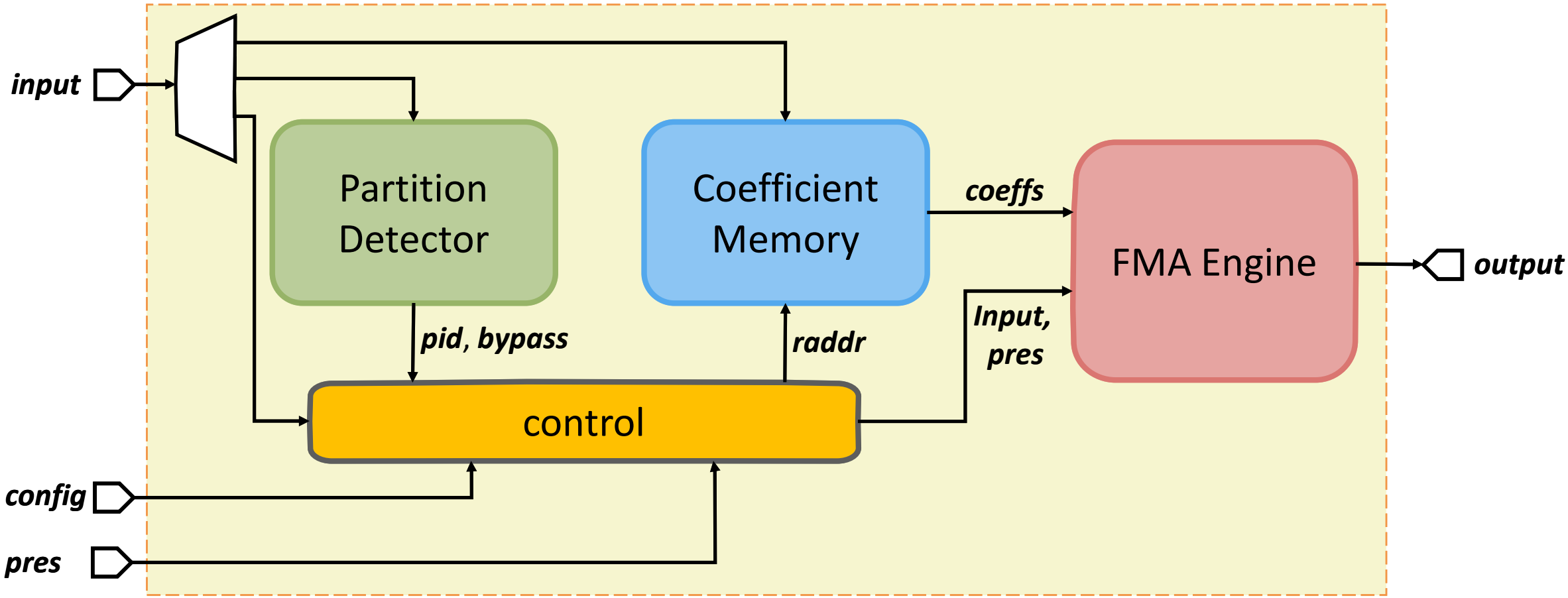
- Less than **0.2%** error across all evaluated workloads

Results: PwPA Accuracy on Diverse AI Workloads

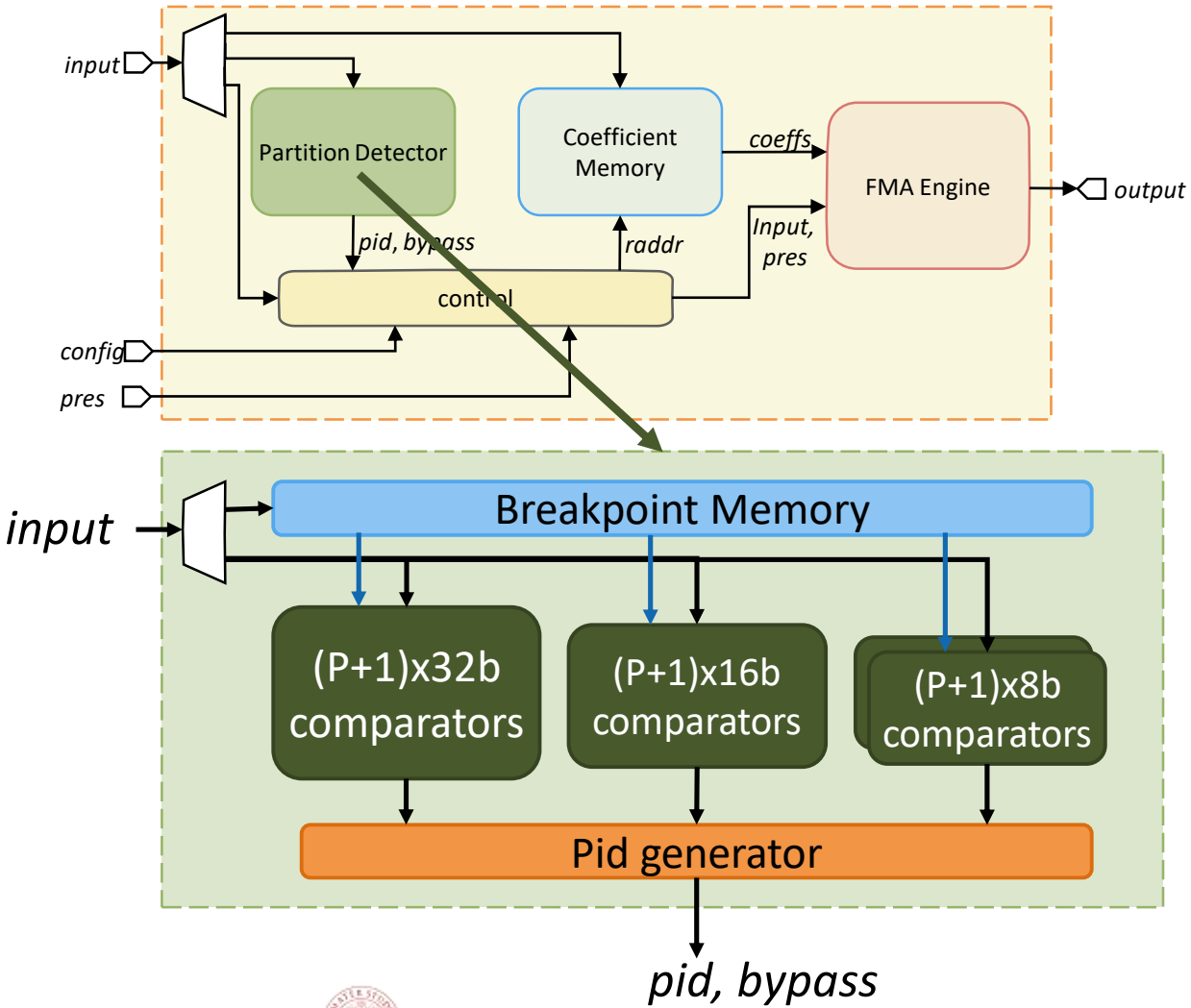


- Less than **0.2%** error across all evaluated workloads
- **Comparable accuracy** observed for (Degree, Partitions): (1, 64), (2, 16), and (3, 8)
- Hardswish function benefits with higher degree polynomial

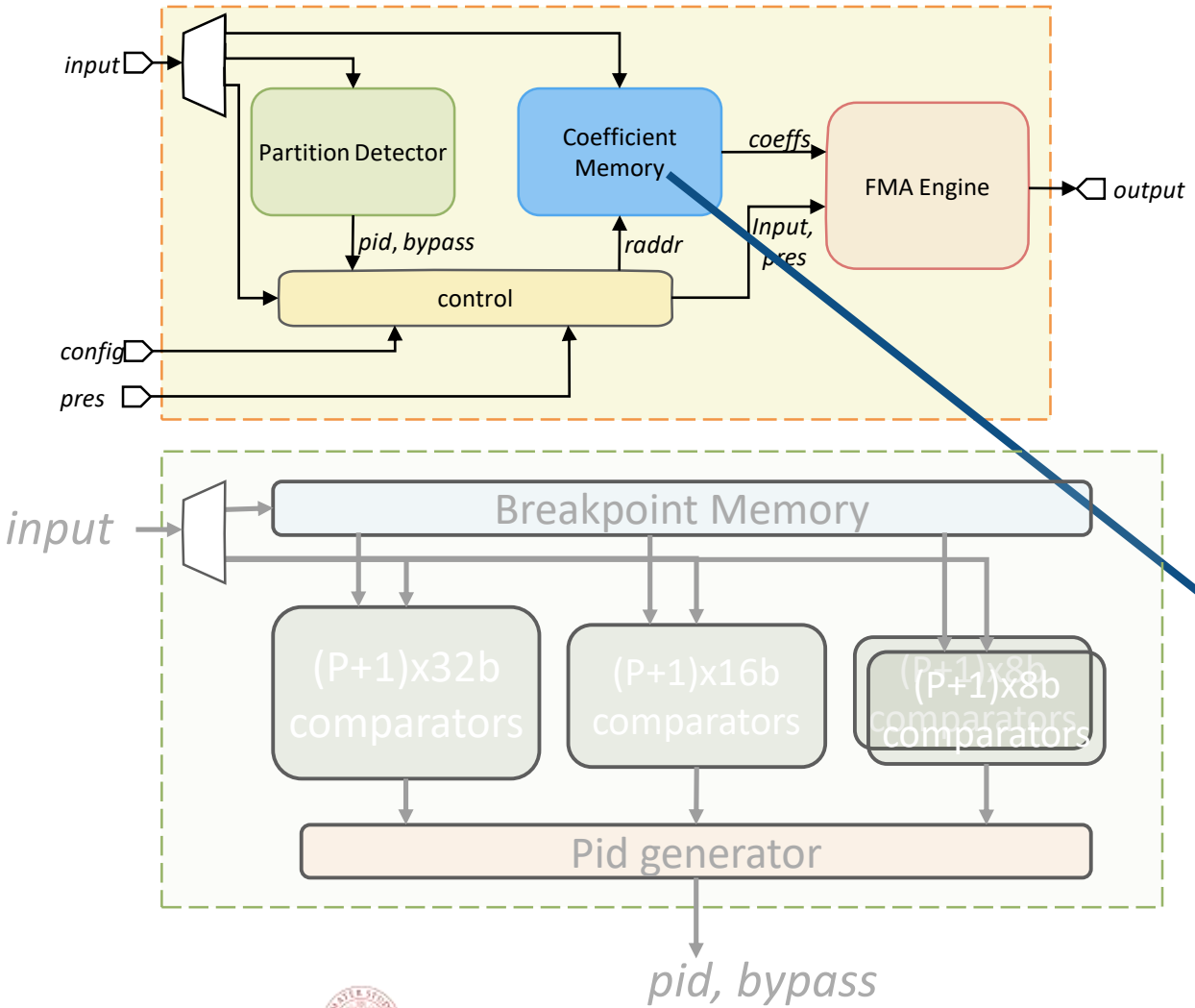
PACE Datapath: Overview

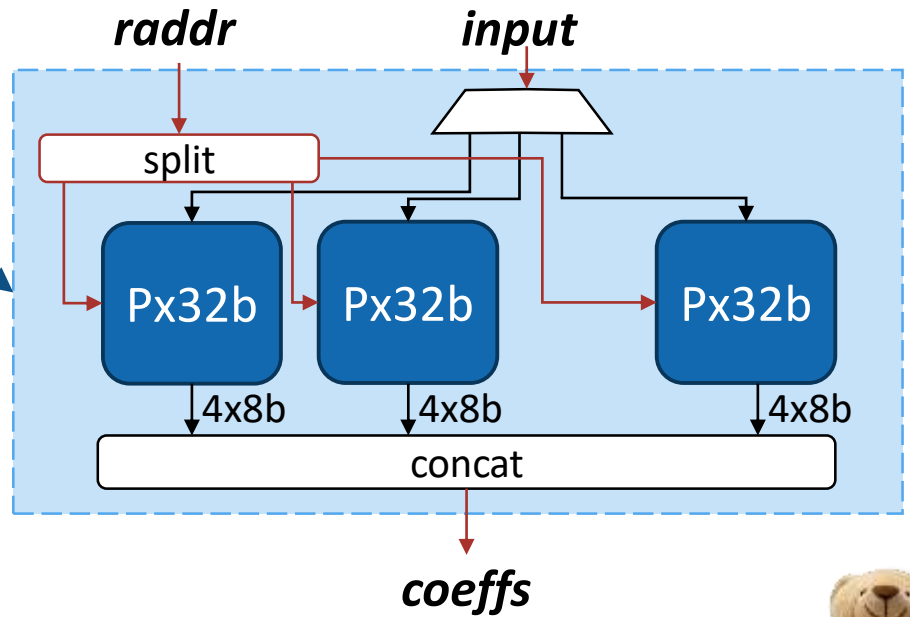


PACE datapath: Partition Detector

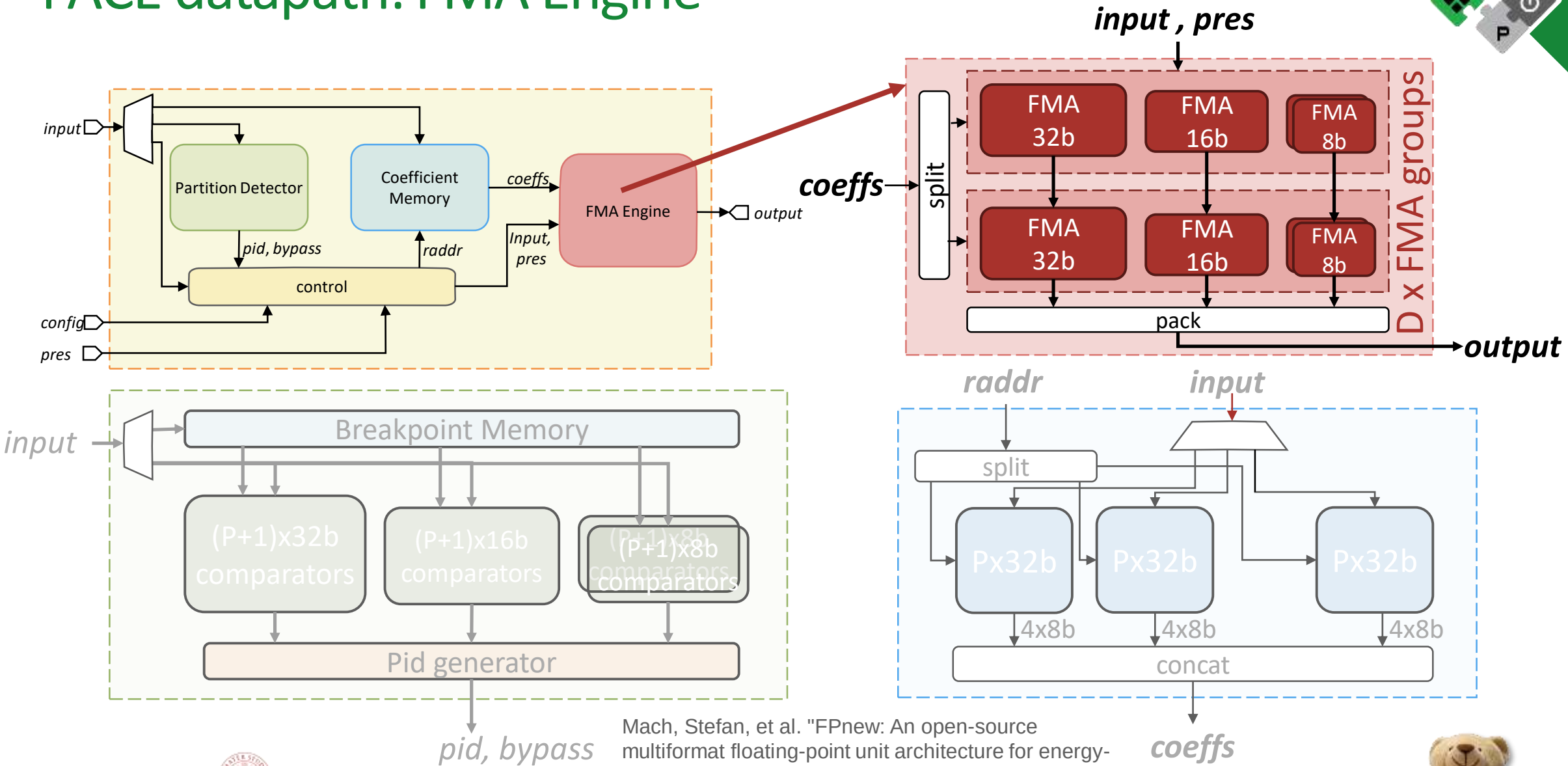


PACE datapath: Coefficient Memory



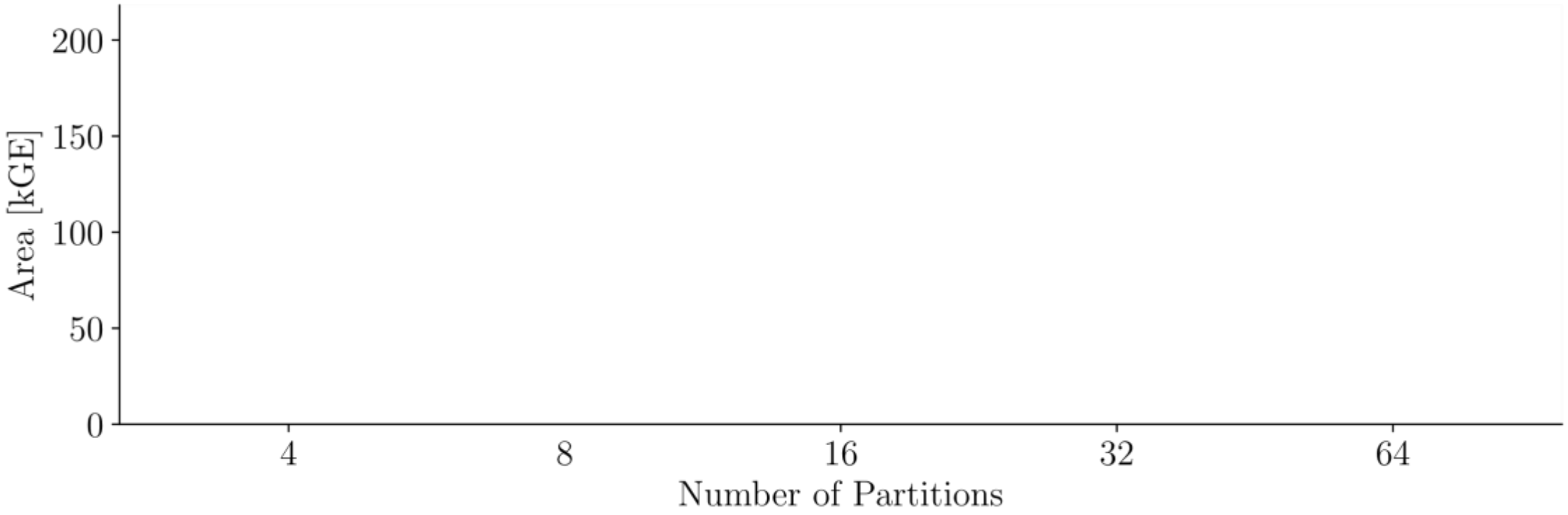


PACE datapath: FMA Engine



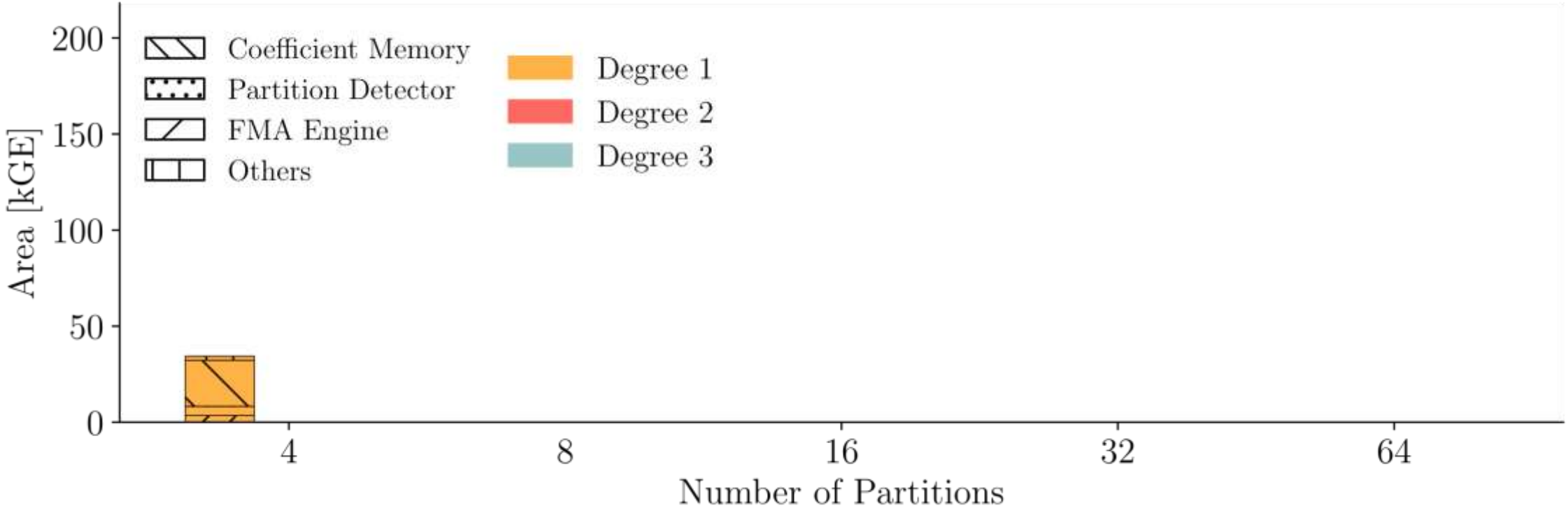
Mach, Stefan, et al. "FPnew: An open-source multiformat floating-point unit architecture for energy-proportional transprecision computing." *TVLSI 2020*.

Results: Area Breakdown – PACE Datapath



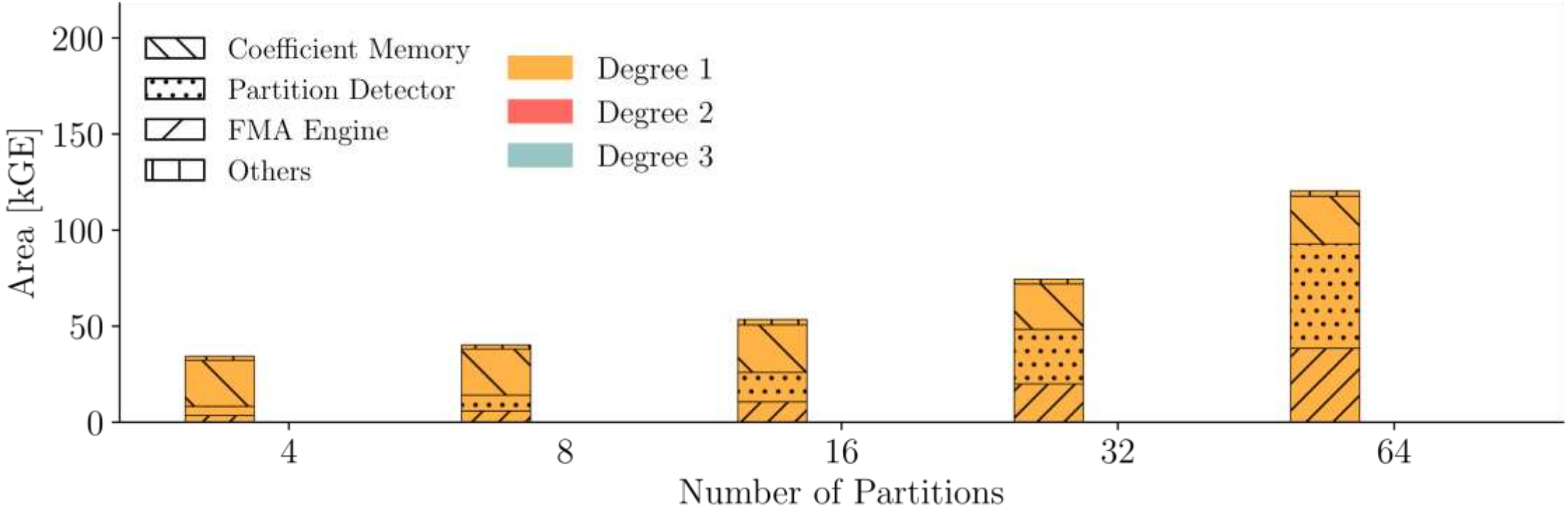
Synthesis results – GF12, 0.95GHz, TT/0.8V/25C

Results: Area Breakdown – PACE Datapath



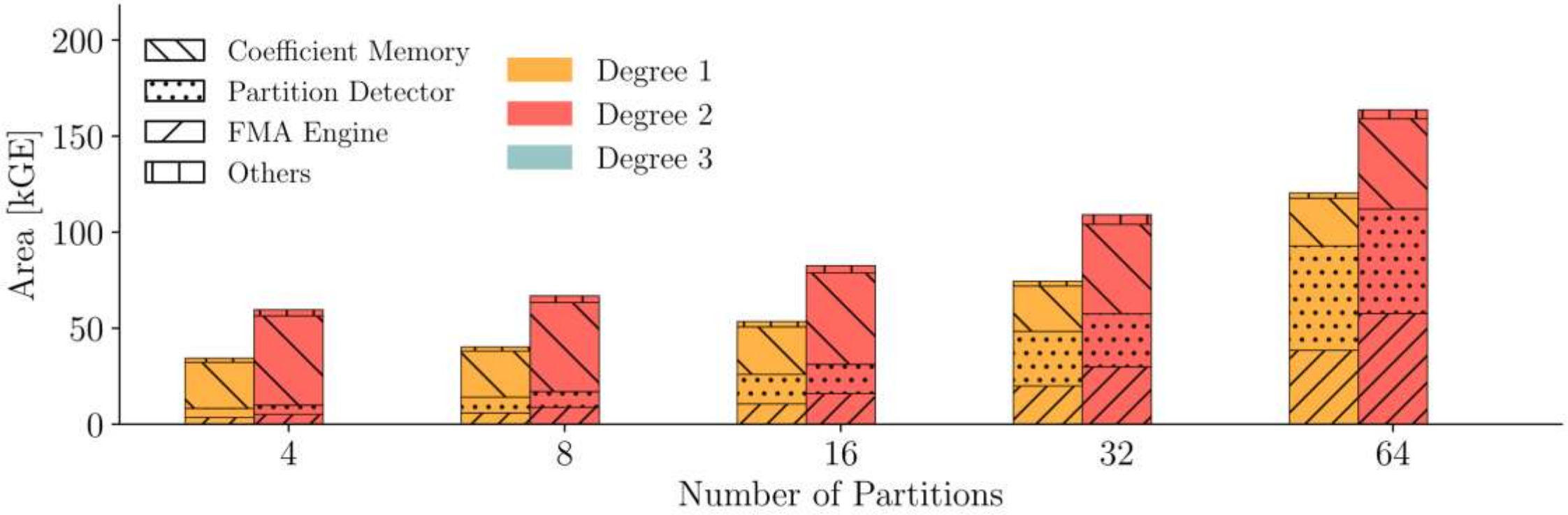
- **FMA engine area** dominates when **partition count** is **low**

Results: Area Breakdown – PACE Datapath



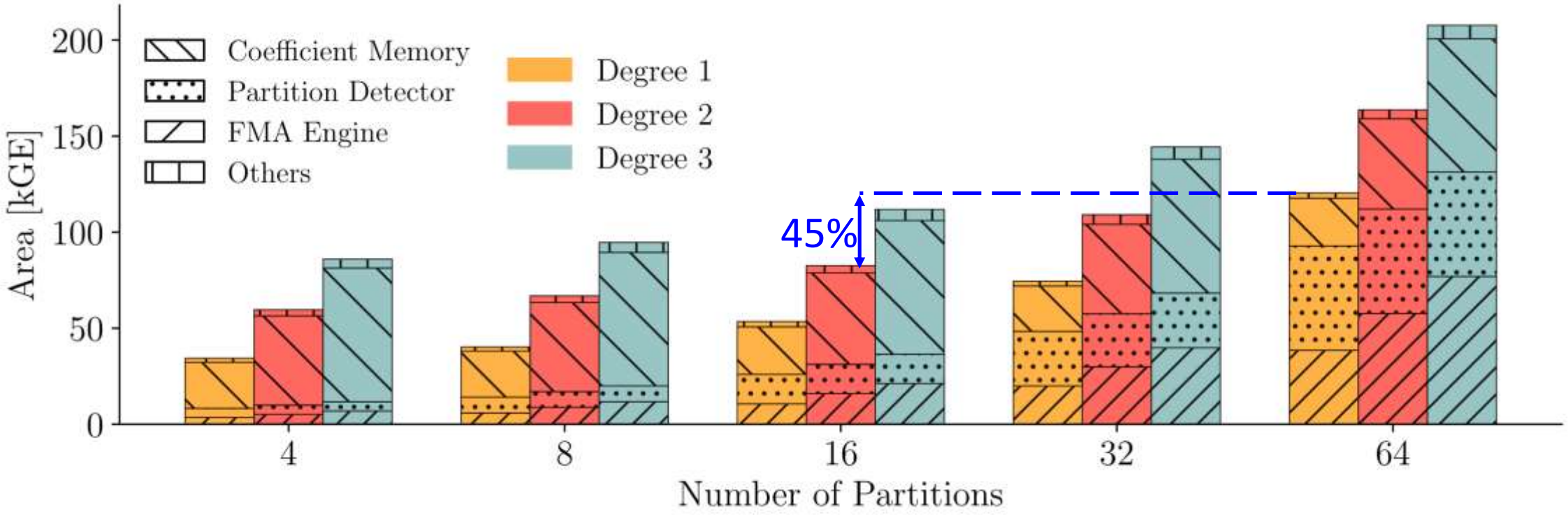
- **FMA engine** area remains **constant** with increasing **partitions**
- **Partition detector** and **coefficient memory** area **scale** with the number of **partitions**

Results: Area Breakdown – PACE Datapath



- **FMA Engine** area **scales** with polynomial **degree**
- **Coefficient memory** area **increases** with degree
- **Partition detector area** remains **constant** with changing degrees

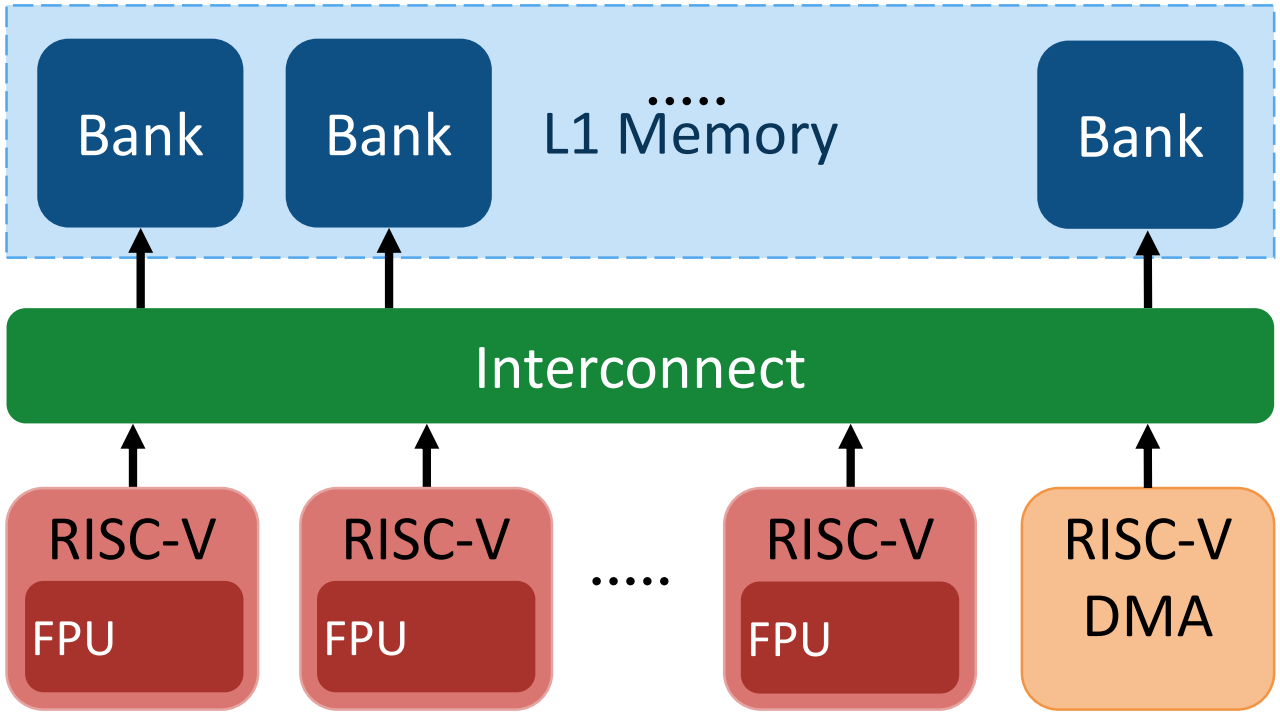
Results: Area Breakdown – PACE Datapath



At Iso-Accuracy: Area Comparison

- **(1, 64)** requires approximately **45% more area** than **(2, 16)**
- **(3, 8)** requires approximately **14% more area** than **(2, 16)**

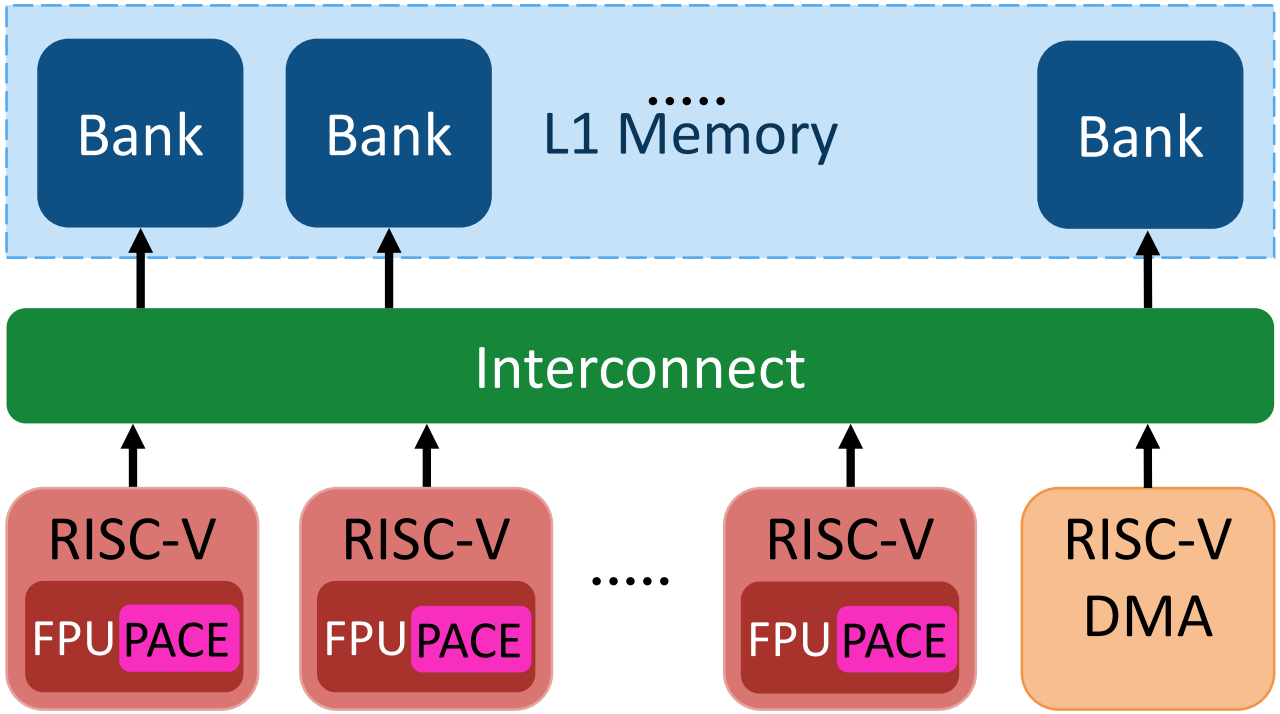
System-Level Integration in a RISC-V Multi-Core Platform



- 8x RISC-V streaming enhanced cores
  - Dedicated SIMD 32b **FPU** per core
- 1x RISC-V control core
- Multi-bank interleaved L1 memory
  - Shared through an interconnect

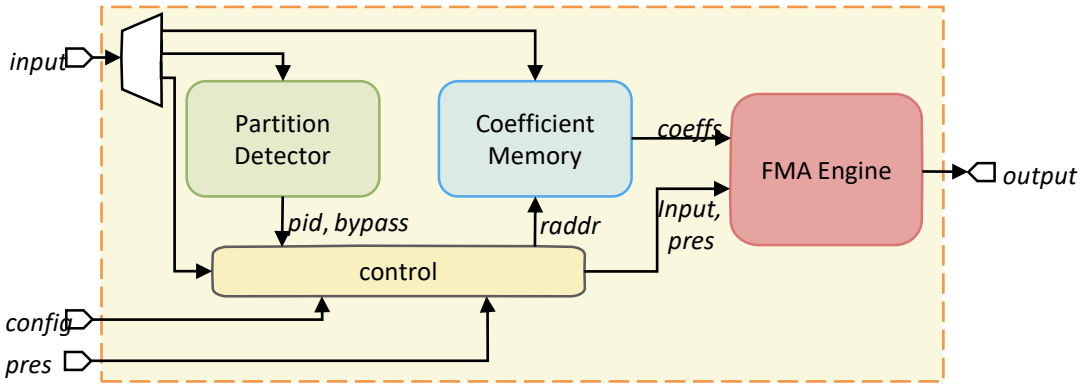
Schuiki, Fabian, et al. "Stream semantic registers: A lightweight risc-v isa extension achieving full compute utilization in single-issue cores." *IEEE Transactions on Computers* 70.2 (2020): 212-227

System-Level Integration in a RISC-V Multi-Core Platform



- 8x RISC-V streaming enhanced cores
  - Dedicated SIMD 32b **FPU** per core
- 1x RISC-V control core
- Multi-bank interleaved L1 memory
  - Shared through an interconnect
- **RISC-V cores enhanced with PACE**
  - **Integration into the FPU**

PACE as a RISC-V ISA Extension



config port is mapped to CSRs

`PwPA_init rs1`

The values from rs1 are used for Initialising the breakpoint and coefficient memory

`PwPA_exec rd, rs1, rs2`

The input x and pres are read from rs2, rs1 and the output is written to the rd register

```
//Initialise the configurations
write_csr(CSR1, degree)
write_csr(CSR2, partitions)
write_csr(CSR3, (type,mode,init))

//Initialise the bp and coeff memory using
//value stored in rs1
for (i=0; i<NumBps+NumCoeffs; i++):
  PwPA_init rs1;

//Tensor wise PwPA execution. pres, x read
//from rs1, rs2 to produce result to rd
for (i=0; i<TensorLen; i++):
  PwPA_exec rd, rs1, rs2;
```

Results: System-Level Performance Evaluation

Baseline-1

EXP-C (math library)[FP32]

```
for(i=0; i<TensorLen; i++)
  y[i] = expf(x[i])
```

Baseline-2

PwPA-SW[FP32]

```
for(i=0; i<TensorLen; i++)
{
  pid = compute_pid();
  res=coeff[pid][0];
  for(d=1; d<=degree; d++)
    fma(res, x[i], coeff[pid][d]);
  y[i] = res;
}
```

PACE-FP32

```
initialize(2, 16, FP32);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```

PACE-FP16

```
initialize(2, 16, FP16);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```


Results: System-Level Performance Evaluation

PwPA-SW[FP32]

```
for(i=0; i<TensorLen; i++)
{
  pid = compute_pid();
  res=coeffs[pid][0];
  for(d=1; d<=degree; d++)
      fma(res, x[i], coeff[pid][d]);
  y[i] = res;
}
```

EXP-C (math library)[FP32]

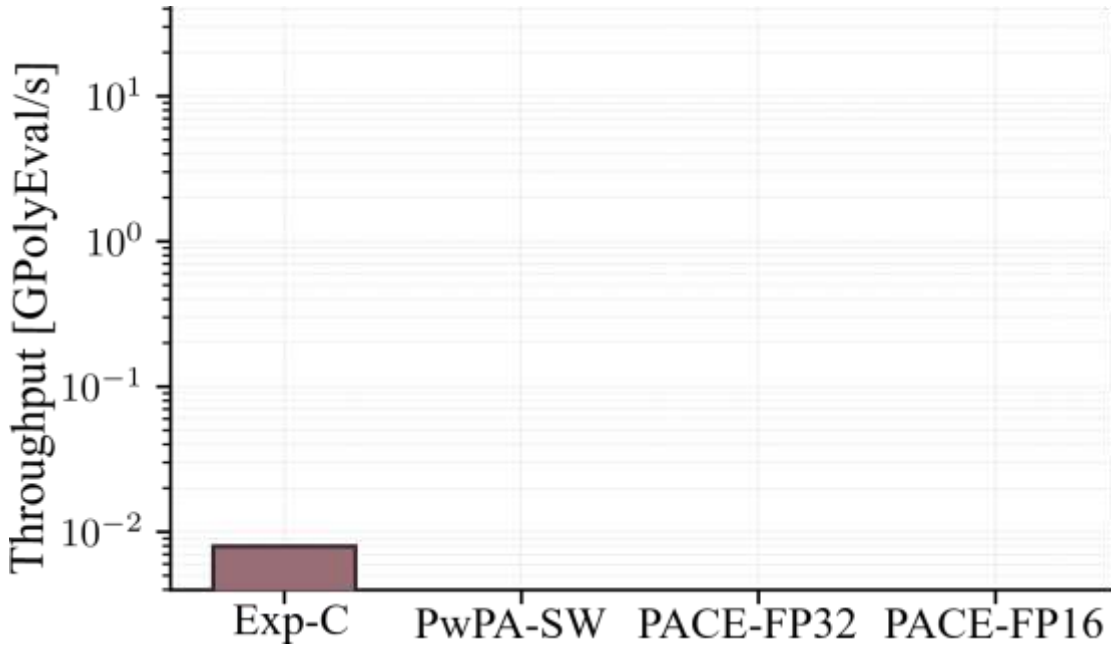
```
for(i=0; i<TensorLen; i++)
  y[i] = expf(x[i])
```

PACE-FP32

```
initialize(2, 16, FP32);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```

PACE-FP16

```
initialize(2, 16, FP16);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```



Results: System-Level Performance Evaluation

PwPA-SW[FP32]

```
for(i=0; i<TensorLen; i++)
{
  pid = compute_pid();
  res=coeffs[pid][0];
  for(d=1; d<=degree; d++)
       fma(res, x[i], coeff[pid][d]);
  y[i] = res;
}
```

EXP-C (math library)[FP32]

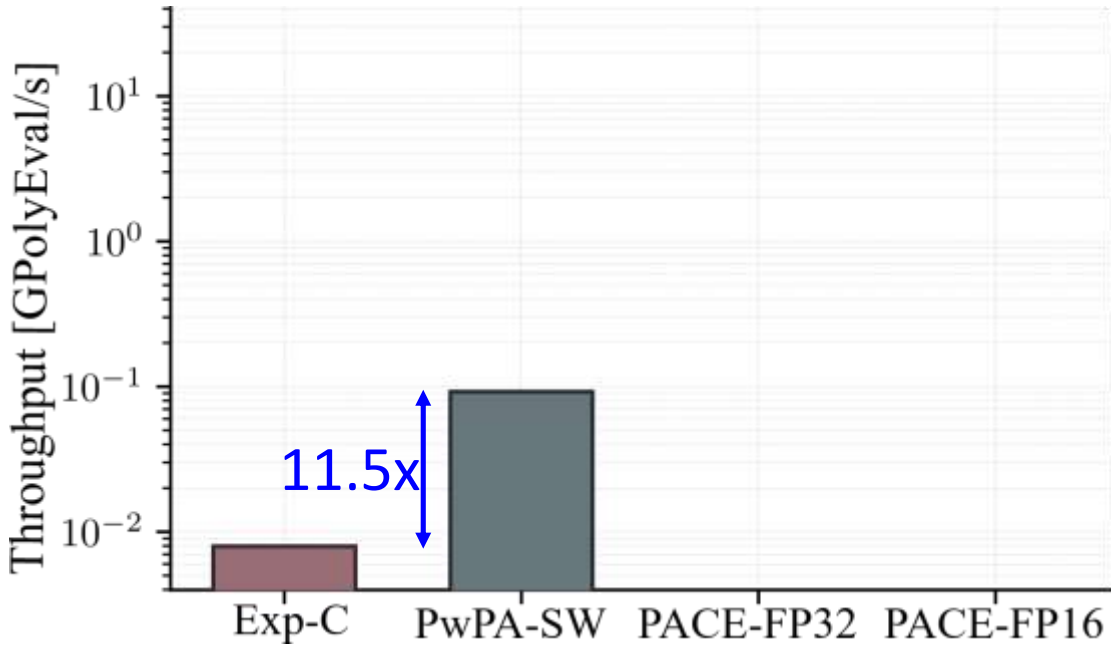
```
for(i=0; i<TensorLen; i++)
  y[i] = expf(x[i])
```

PACE-FP32

```
initialize(2, 16, FP32);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```

PACE-FP16

```
initialize(2, 16, FP16);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```



Results: System-Level Performance Evaluation

PwPA-SW[FP32]

```
for(i=0; i<TensorLen; i++)
{
  pid = compute_pid();
  res=coeffs[pid][0];
  for(d=1; d<=degree; d++)
       fma(res, x[i], coeff[pid][d]);
  y[i] = res;
}
```

EXP-C (math library)[FP32]

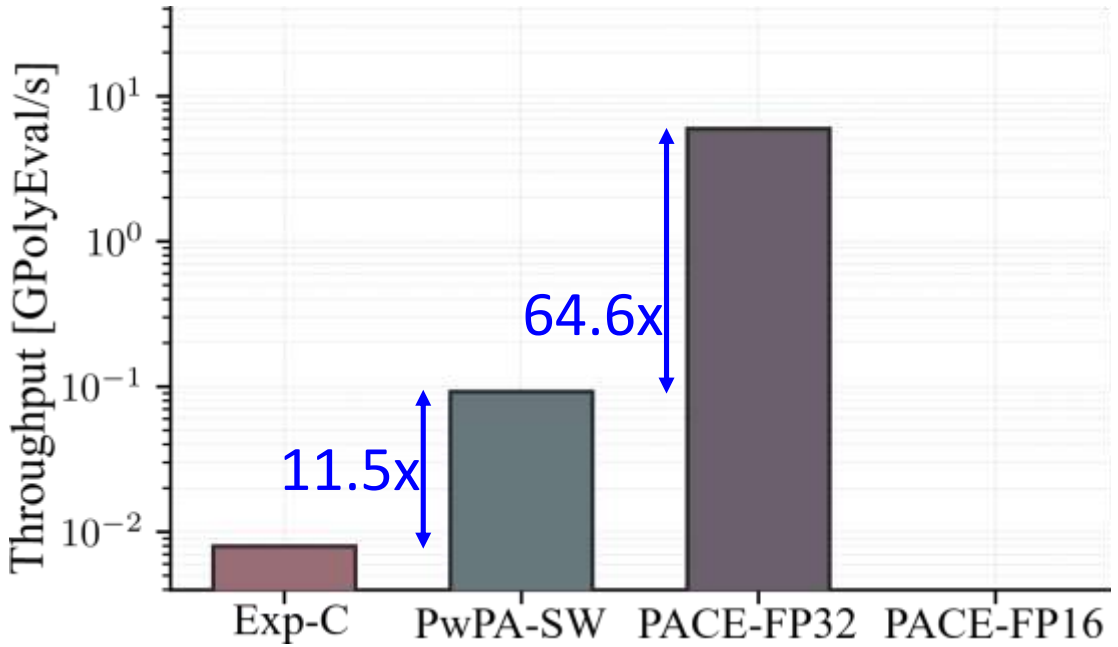
```
for(i=0; i<TensorLen; i++)
  y[i] = expf(x[i])
```

PACE-FP32

```
initialize(2, 16, FP32);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```

PACE-FP16

```
initialize(2, 16, FP16);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```



Results: System-Level Performance Evaluation

PwPA-SW[FP32]

```
for(i=0; i<TensorLen; i++)
{
  pid = compute_pid();
  res=coeffs[pid][0];
  for(d=1; d<=degree; d++)
       fma(res, x[i], coeff[pid][d]);
  y[i] = res;
}
```

EXP-C (math library)[FP32]

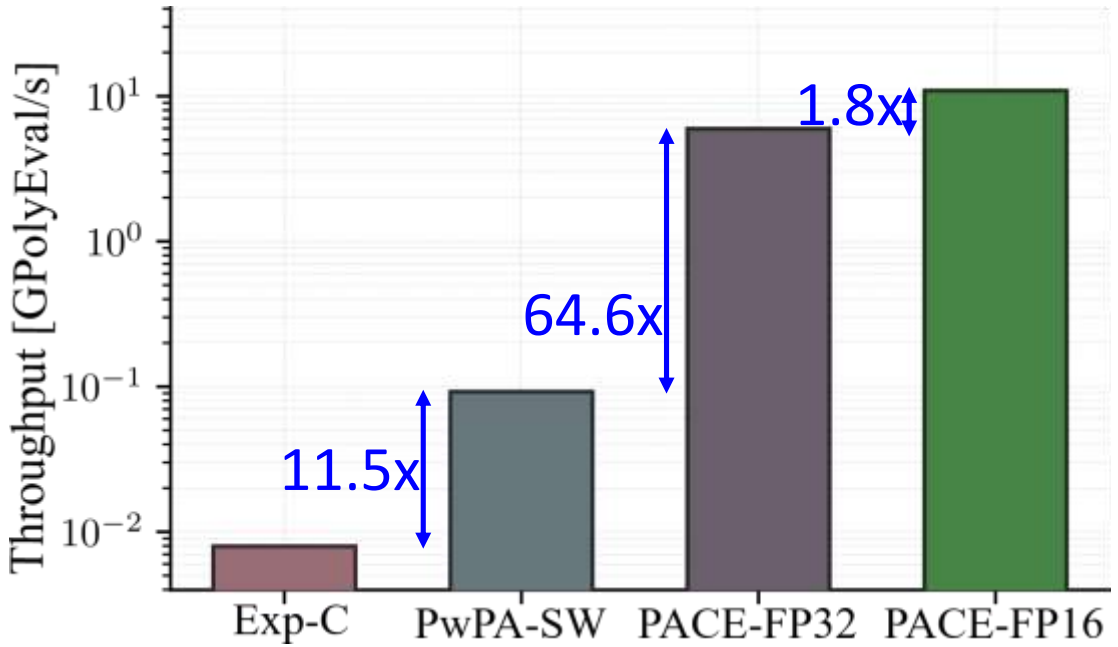
```
for(i=0; i<TensorLen; i++)
  y[i] = expf(x[i])
```

PACE-FP32

```
initialize(2, 16, FP32);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```

PACE-FP16

```
initialize(2, 16, FP16);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```



Results: System-Level Performance Evaluation

PwPA-SW[FP32]

```
for(i=0; i<TensorLen; i++)
{
  pid = compute_pid();
  res=coeffs[pid][0];
  for(d=1; d<=degree; d++)
       fma(res, x[i], coeff[pid][d]);
  y[i] = res;
}
```

EXP-C (math library)[FP32]

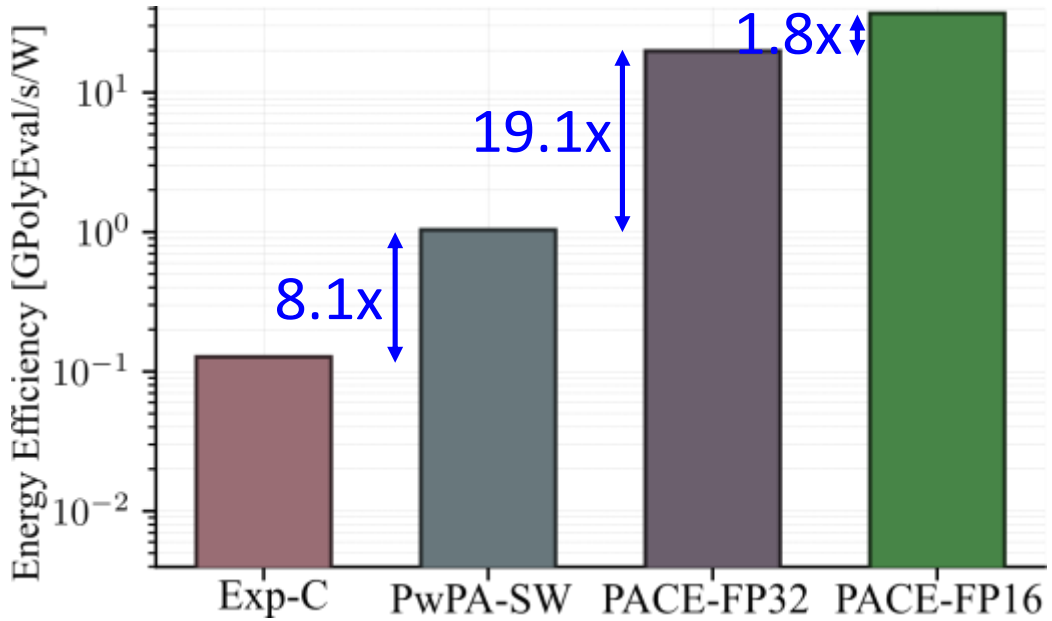
```
for(i=0; i<TensorLen; i++)
  y[i] = expf(x[i])
```

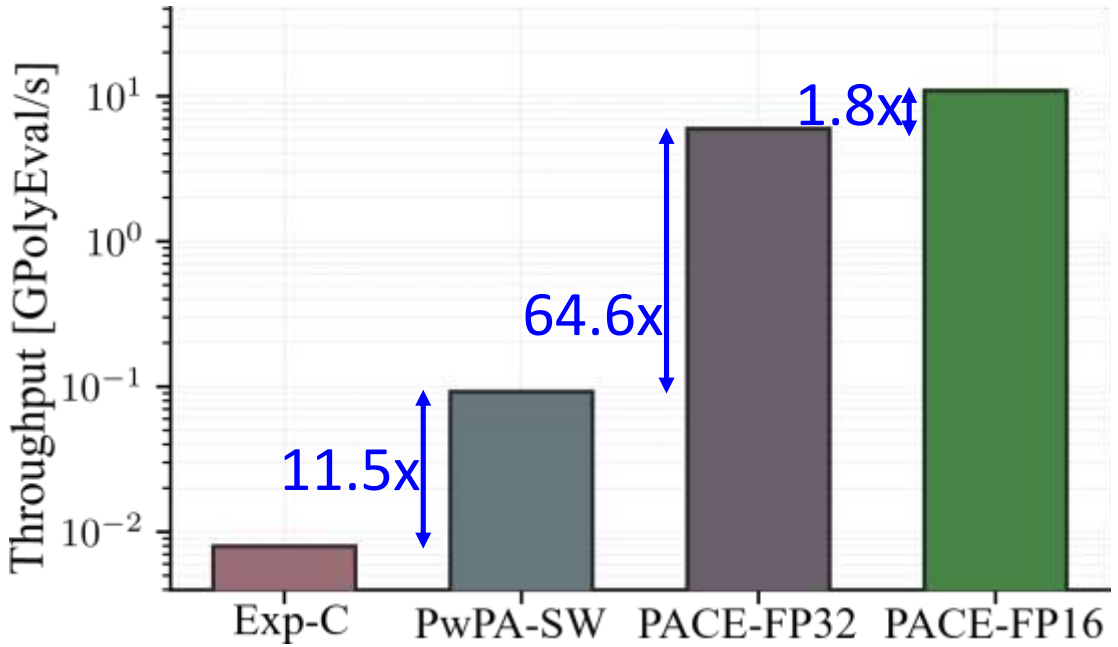
PACE-FP32

```
initialize(2, 16, FP32);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```

PACE-FP16

```
initialize(2, 16, FP16);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```





Results: System-Level Performance Evaluation

PwPA-SW[FP32]

```
for(i=0; i<TensorLen; i++)
{
  pid = compute_pid();
  res=coeffs[pid][0];
  for(d=1; d<=degree; d++)
       fma(res, x[i], coeff[pid][d]);
  y[i] = res;
}
```

EXP-C (math library)[FP32]

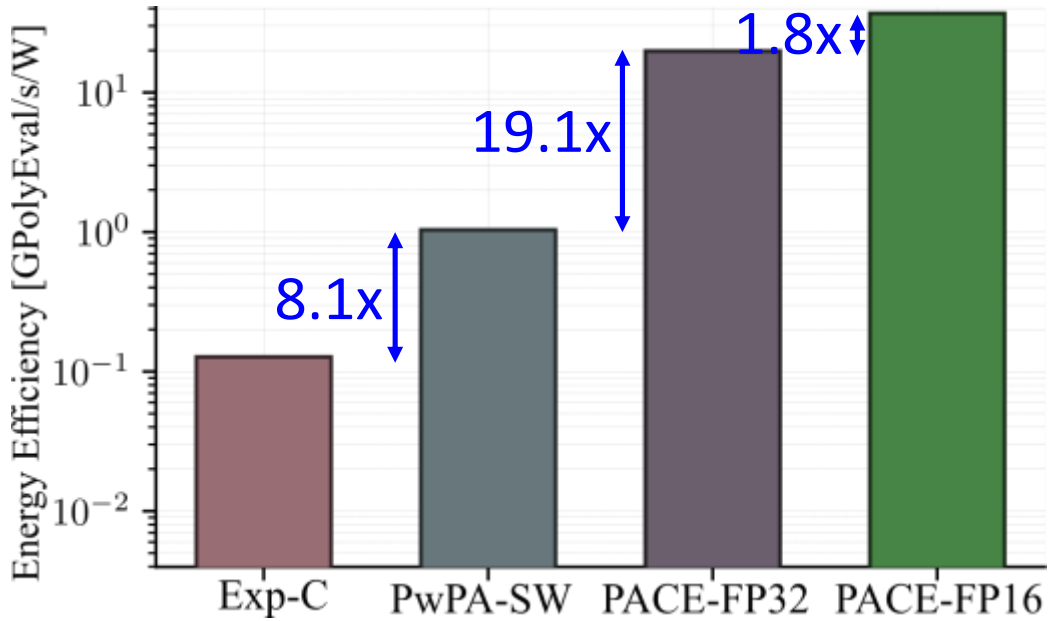
```
for(i=0; i<TensorLen; i++)
  y[i] = expf(x[i])
```

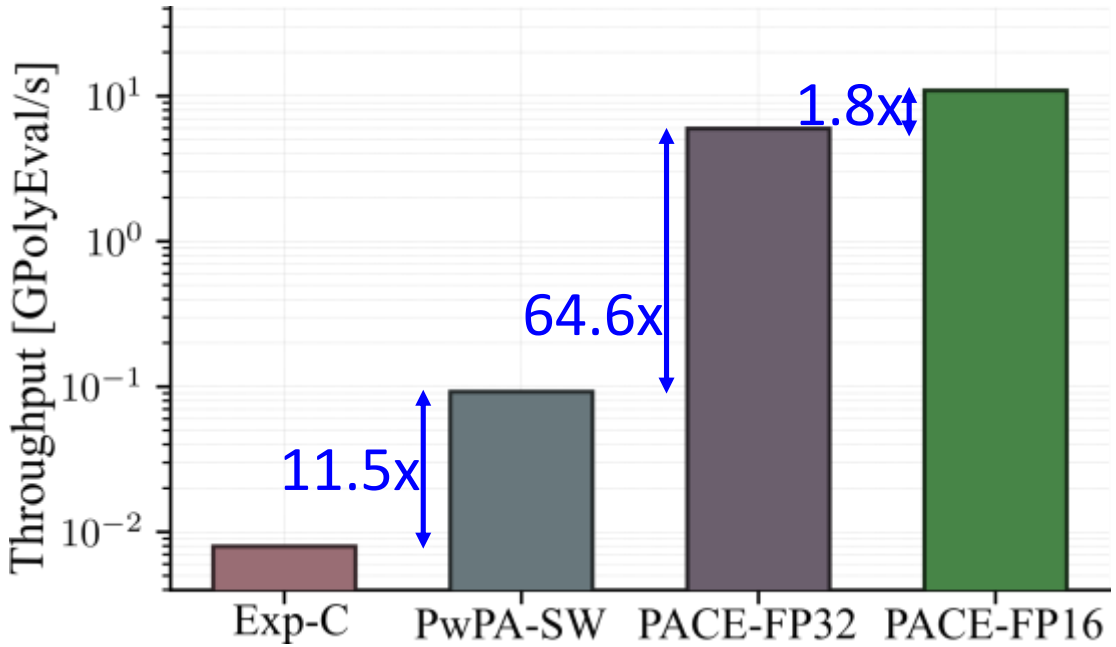
PACE-FP32

```
initialize(2, 16, FP32);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```

PACE-FP16

```
initialize(2, 16, FP16);
for(i=0; i<TensorLen; i++)
  y[i] = PwPA_exec(x[i]);
```





PACE(FP32) achieves **750x** & **65x** speedup with **150x** & **19x** efficiency gain compared to SW baselines

Comparative Analysis: PACE vs. Prior Work

Work	Diverse	Precision	Tech [nm]	Frequency [GHz]	Area [um^2]	Throughput [GPolyEval/s]	Efficiency [GPolyEval/s/W]
[1]Diaz-Conti et. al	✓	Fixed	NA	NA	NA	NA	NA
[2]Reggiani et. al	✓	I/FP32/16/8	28	0.6	14857	0.6/1.2/2.4	158
[3]Islamoglu et. al	✗	INT8	22	0.5	5709	NA	NA
[4]Zhu et.al	✗	NA	28	2.78	10081	22.24	NA
PACE (2, 16)	✓	FP32/16/8	12	0.95	10538	0.95/1.9/3.8	124/200/231

Comparative Analysis: PACE vs. Prior Work

Work	Diverse	Precision	Tech [nm]	Frequency [GHz]	Area [um^2]	Throughput [GPolyEval/s]	Efficiency [GPolyEval/s/W]
[1]Diaz-Conti et. al	✓	Fixed	NA	NA	NA	NA	NA
[2]Reggiani et. al	✓	I/FP32/16/8	28	0.6	14857	0.6/1.2/2.4	158
[3]Islamoglu et. al	✗	INT8	22	0.5	5709	NA	NA
[4]Zhu et.al	✗	NA	28	2.78	10081	22.24	NA
PACE (2, 16)	✓	FP32/16/8	12	0.95	10538	0.95/1.9/3.8	124/200/231

- Specialised architectures [3,4] do not cater to diverse non-linearities

Comparative Analysis: PACE vs. Prior Work

Work	Diverse	Precision	Tech [nm]	Frequency [GHz]	Area [um^2]	Throughput [GPolyEval/s]	Efficiency [GPolyEval/s/W]
[1]Diaz-Conti et. al	✓	Fixed	NA	NA	NA	NA	NA
[2]Reggiani et. al	✓	I/FP32/16/8	28	0.6	14857	0.6/1.2/2.4	158
[3]Islamoglu et. al	✗	INT8	22	0.5	5709	NA	NA
[4]Zhu et.al	✗	NA	28	2.78	10081	22.24	NA
PACE (2, 16)	✓	FP32/16/8	12	0.95	10538	0.95/1.9/3.8	124/200/231

- Specialised architectures [3,4] do not cater to diverse non-linearities
- [1], [2] and PACE can cater to diverse nonlinearity due to their reliance on PwLA/PA

Comparative Analysis: PACE vs. Prior Work

Work	Diverse	Precision	Tech [nm]	Frequency [GHz]	Area [um^2]	Throughput [GPolyEval/s]	Efficiency [GPolyEval/s/W]
[1]Diaz-Conti et. al	✓	Fixed	NA	NA	NA	NA	NA
[2]Reggiani et. al	✓	I/FP32/16/8	28	0.6	14857	0.6/1.2/2.4	158
[3]Islamoglu et. al	✗	INT8	22	0.5	5709	NA	NA
[4]Zhu et.al	✗	NA	28	2.78	10081	22.24	NA
PACE (2, 16)	✓	FP32/16/8	12	0.95	10538	0.95/1.9/3.8	124/200/231

- Specialised architectures [3,4] do not cater to diverse non-linearities
- [1], [2] and PACE can cater to diverse nonlinearity due to their reliance on PwLA/PA
- Evaluation on [1] is limited to only two CNNs
  - PACE is evaluated on a broad set with CNNs, ViTs and LLMs

Comparative Analysis: PACE vs. Prior Work

Work	Diverse	Precision	Tech [nm]	Frequency [GHz]	Area [um^2]	Throughput [GPolyEval/s]	Efficiency [GPolyEval/s/W]
[1]Diaz-Conti et. al	✓	Fixed	NA	NA	NA	NA	NA
[2]Reggiani et. al	✓	I/FP32/16/8	28	0.6	14857	0.6/1.2/2.4	158
[3]Islamoglu et. al	✗	INT8	22	0.5	5709	NA	NA
[4]Zhu et.al	✗	NA	28	2.78	10081	22.24	NA
PACE (2, 16)	✓	FP32/16/8	12	0.95	10538	0.95/1.9/3.8	124/200/231

- Specialised architectures [3,4] do not cater to diverse non-linearities
- [1], [2] and PACE can cater to diverse nonlinearity due to their reliance on PwLA/PA
- Evaluation on [1] is limited to only two CNNs
  - PACE is evaluated on a broad set with CNNs, ViTs and LLMs
- Main comparison point for PACE is [2]

Comparative Analysis: PACE vs. Prior Work

Work	General	Precision	Technology [nm]	Frequency [GHz]	Area [um^2]	Throughput [GPolyEval/s]	Efficiency [GPolyEval/s/W]
[2]	✓ PwLA	I/FP32/16/8	28	0.6	14857	0.6/1.2/2.4	158
PACE	✓ PwPA	FP32/16/8	12	0.95	10538	0.95/1.9/3.8	124/200/231

- We use (1,64) for PwLA to compare with PACE's (2,16) for iso-accuracy
- PACE offers ~1.6x more speedup compared to the [2]
- Energy efficiency calculation excludes VPU power in [2]
  - Excluding FMA power, PACE is 2.7x more energy efficient
- PACE offers runtime configurability with support for arbitrary degrees and partitions.
  - Higher degree support enables higher approximation accuracy.

Conclusion: Summarising PACE

PACE accelerates nonlinearities using
PwPA with arbitrary partition and degrees

Energy savings are **19x** and **150x** times compared to baselines

The most flexible and adaptable solution offering highest accuracy

Superior accuracy, No fine-tuning
< 0.2% error across diverse workloads

https://github.com/pulp-platform/cvfpu/tree/feature/pace

Systemlevel **speedup** by **65x** and **750x** compared to SW baselines

SoA References

1. González-Díaz_Conti, Griselda, et al. "Hardware-based activation function-core for neural network implementations." *Electronics* 11.1 (2021): 14.
2. Reggiani, Enrico, Renzo Andri, and Lukas Cavigelli. "Flex-sfu: Accelerating dnn activation functions by non-uniform piecewise approximation." *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023.
3. Islamoglu, Gamze, et al. "Ita: An energy-efficient attention and softmax accelerator for quantized transformers." *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2023.
4. Zhu, Danyang, et al. "Efficient precision-adjustable architecture for softmax function in deep learning." *IEEE Transactions on Circuits and Systems II: Express Briefs* 67.12 (2020): 3382-3386.

Backup Slides – PACE in Extended part mode

- $f(x) = a_2x^2 + a_1x + a0$
- First iteration
  - Initialize PACE with coefficients a_2, a_1, a_0 for partitions 0-P
  - $F(x) = (a_2x + a_1)x + a_0$ if x lies in partition 0-P
  - F(x) = x if bypass i.e.; x doesn't lie in the partition
- Second iteration
  - Initialize PACE with coefficients a_2, a_1, a_0 for partitions P+1-2P
  - $F(x) = (a_2x + a_1)x + a_0$ if x lies in partition P+1-2P
  - F(x) = pres if bypass i.e.; x doesn't lie in the partition

Backup Slides – PACE in Extended degree mode

- $f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a0 = (((a_4x + a_3)x + a_2)x+ a_1)x + a0$
- First iteration
  - Initialize PACE with coefficients a_4, a_3, a_2
  - pres = $(a_4x + a_3)x + a_2$
- Second iteration
  - Initialize PACE with coefficients a_1, a_0
  - pres = (pres * x + a_1)x + a_0