THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY: CEDA IEEE Council on Electronic Design Automation, SIGDA ACM

Adana: Accelerating Large Language Models via Adaptive Nonuniform Asymmetric Quantization

Xinkuang Geng[1], Siting Liu[2], Hui Wang[1], Leibo Liu[3], Jie Han[4], Honglan Jiang[1]

[1] School of Integrated Circuits, Shanghai Jiao Tong University, Shanghai, China

[2] School of Information Science and Technology, ShanghaiTech University, Shanghai, China

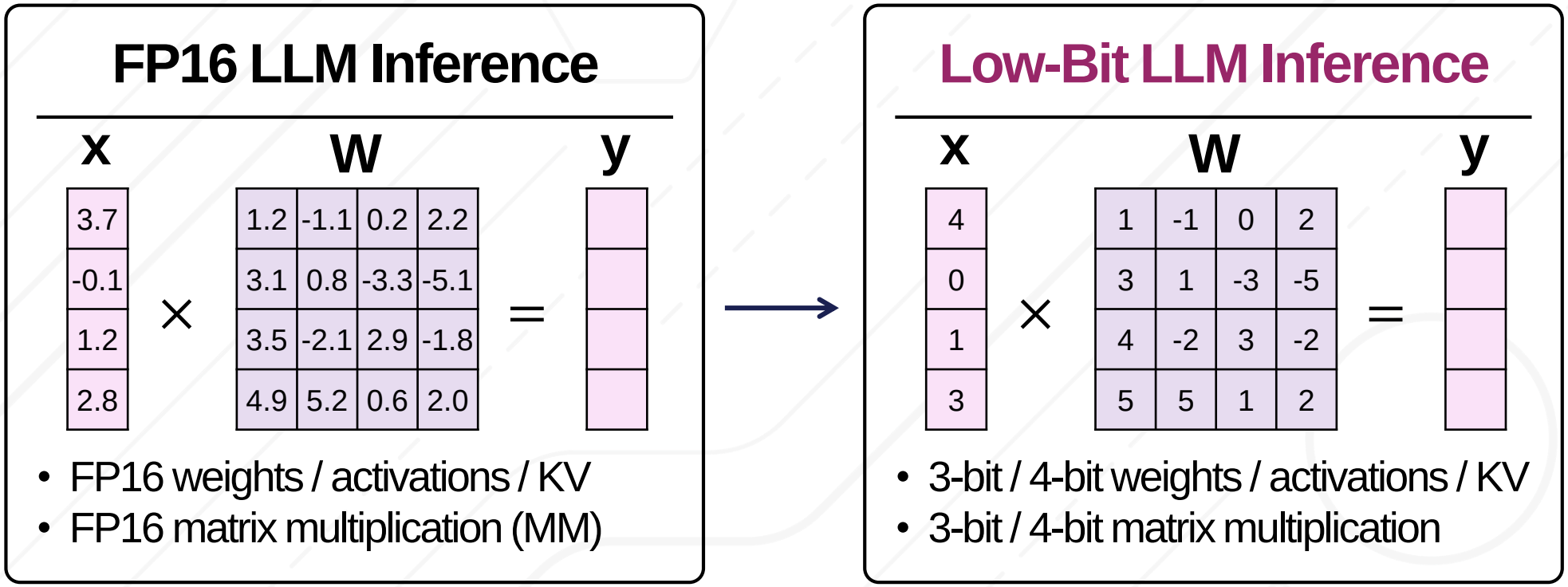
[3] School of Integrated Circuits, Tsinghua University, Beijing, China

[4] Department of Electrical and Computer Engineering, University of Alberta, Alberta, Canada

xinkuang@sjtu.edu.cn, liust@shanghaitech.edu.cn, ihuiwang@sjtu.edu.cn, liulb@tsinghua.edu.cn, jhan8@ualberta.ca, honglan@sjtu.edu.cn

From FP16 Model to Low-Bit Quantization

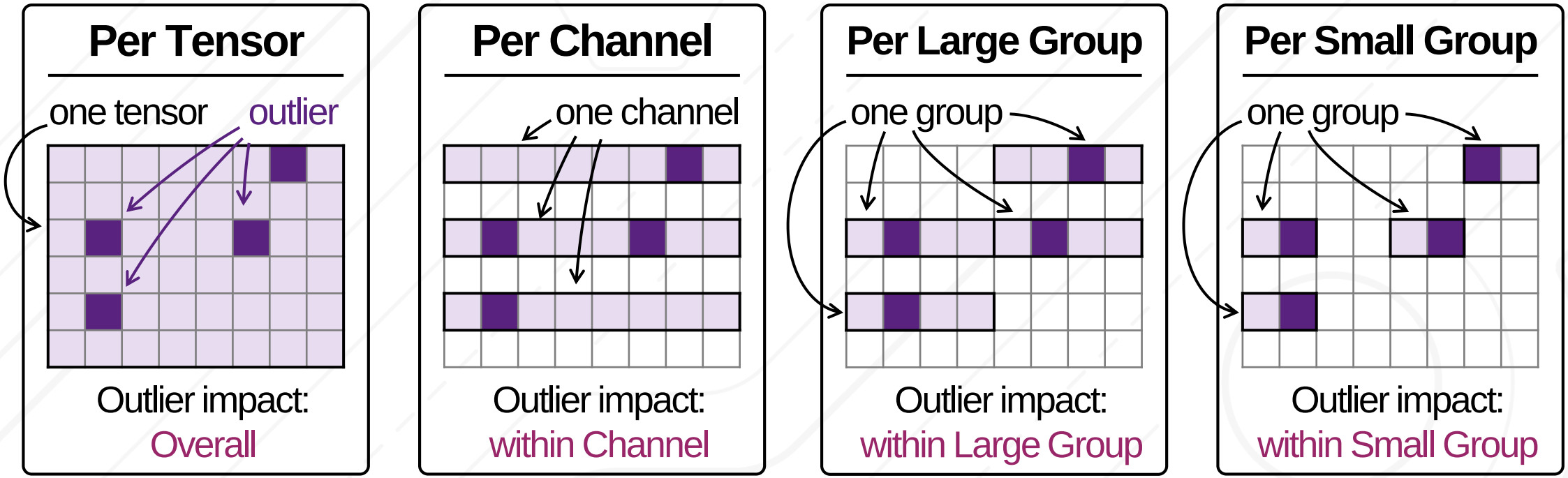
- **LLM quantization reduces memory footprint and inference cost [1].**



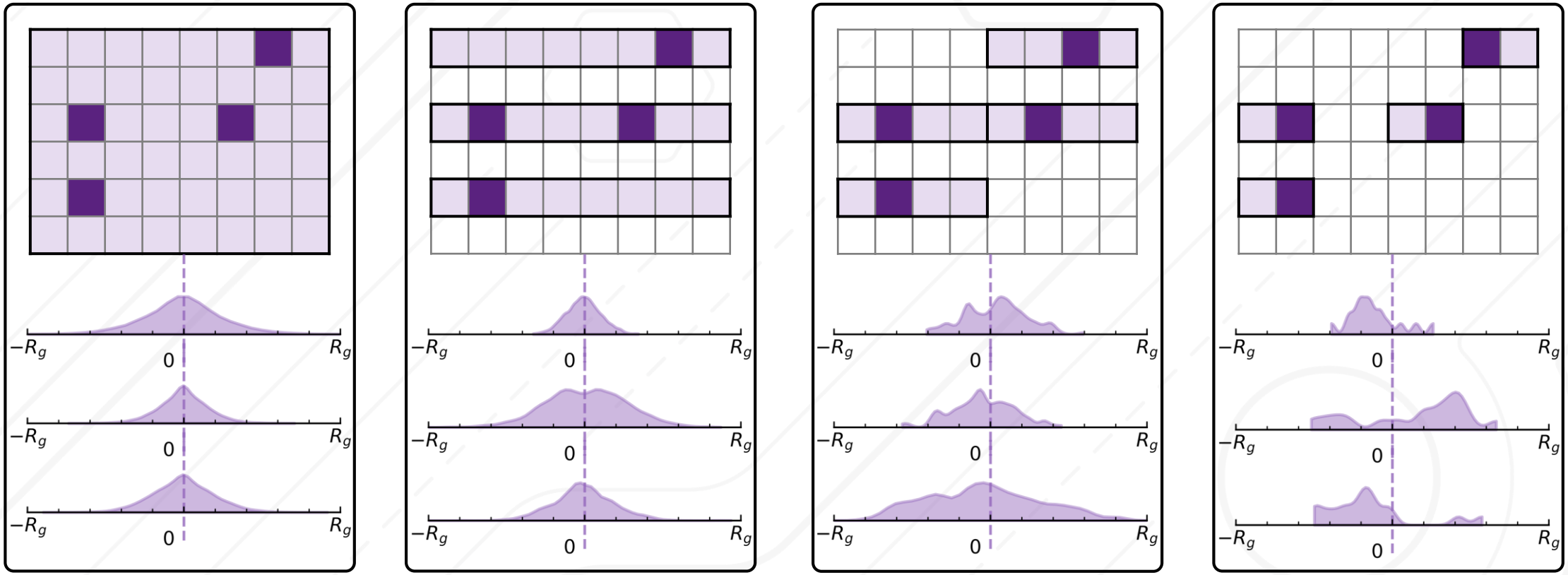
[1] Guo, Cong, et al. "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization." ISCA. 2023.

Toward Finer Quantization Granularity

- **Smaller quantization groups preserve more normal values.**



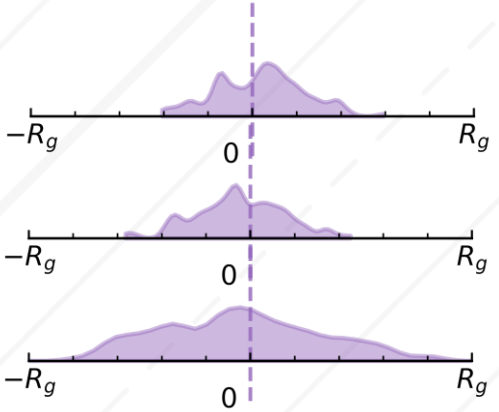
Smaller Groups, Greater Distribution Variability



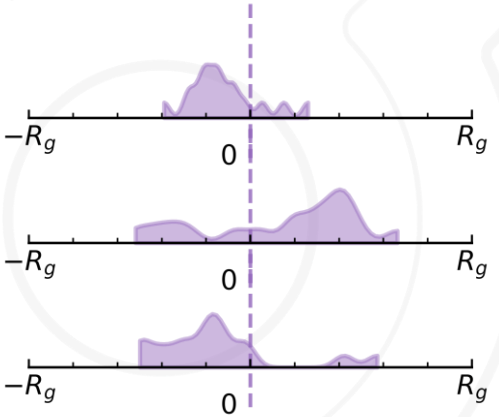
Finer Quantization Granularity

Gaps in Existing Quantization Methods

- **Insufficient exploitation of fine-grained group-wise quantization**
  - Smaller groups reduce quantization error, but existing numeric types [2-4] do not fully adapt to group sizes as small as 16.
- **Trade-off between accuracy and efficiency**
  - MSE-based search [2,4] improves accuracy but adds high hardware cost; simpler mapping [3] is efficient but less accurate.



Large Groups



Small Groups

[2] Guo, Cong, et al. “Ant: Exploiting adaptive numerical data type for low-bit deep neural network quantization.“ MICRO. IEEE, 2022
[3] Hu, Weiming, et al. "M-ANT: Efficient low-bit group quantization for LLMs via mathematically adaptive numerical type.“ HPCA. IEEE, 2025.
[4] Chen, Yuzong, et al. "Bitmod: Bit-serial mixture-of-datatype llm acceleration.“ HPCA. IEEE, 2025.

Algorithm & Architecture

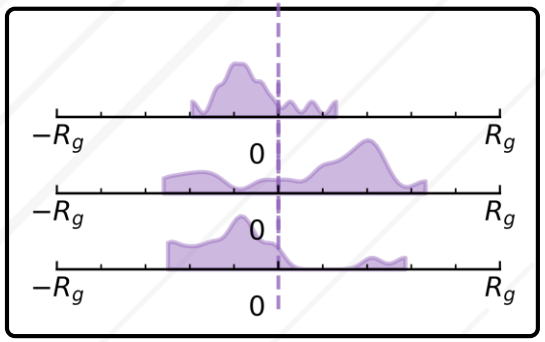
Our Approach

Algorithm side

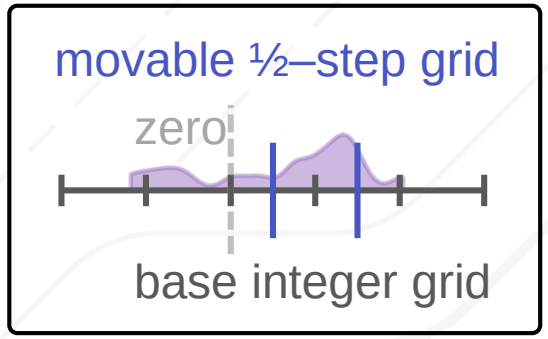
- Adaptive nonuniform asymmetric (Adana) numeric type
- Efficient online activation / KV quantization

Architecture side

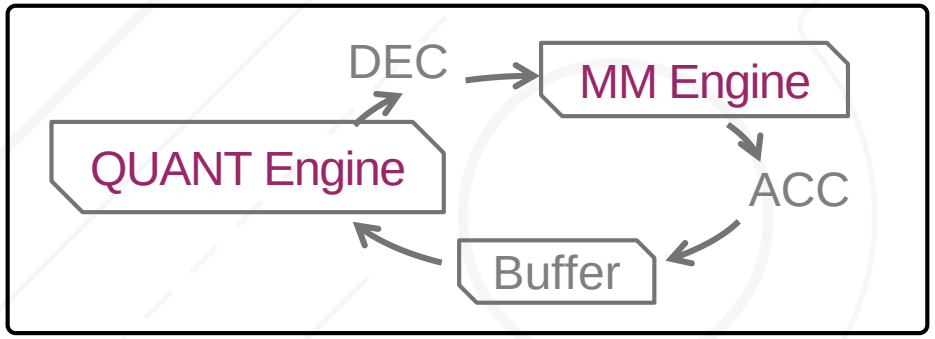
- Low-bit group-wise MM microarchitecture and dataflow
- Hardware-efficient QUANT engine



Fit Data Distributions



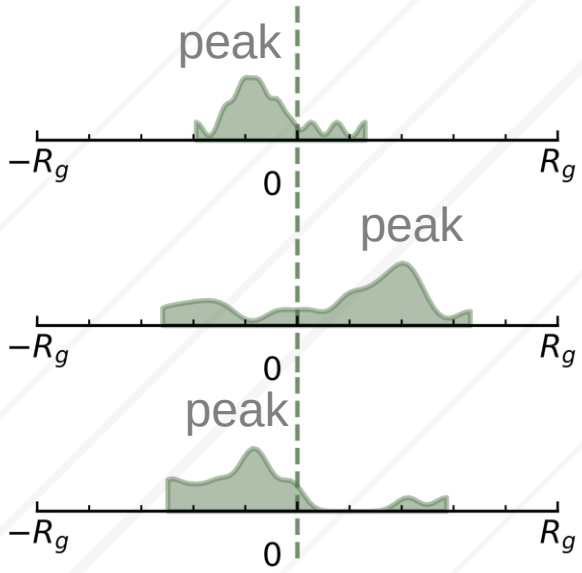
Efficient Quantization



Efficient Computation

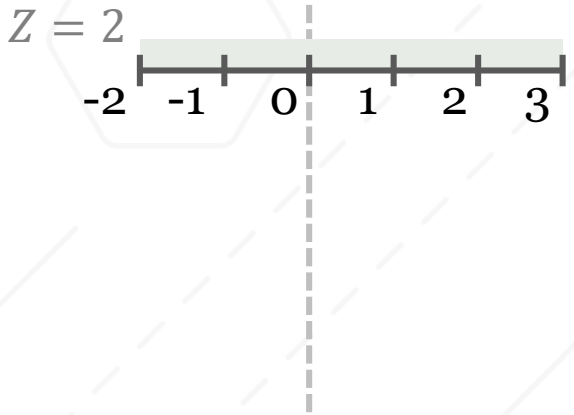
Numeric Type Design

diverse peak positions

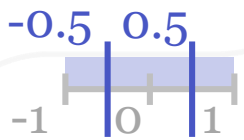


asymmetric data ranges

base integer grid

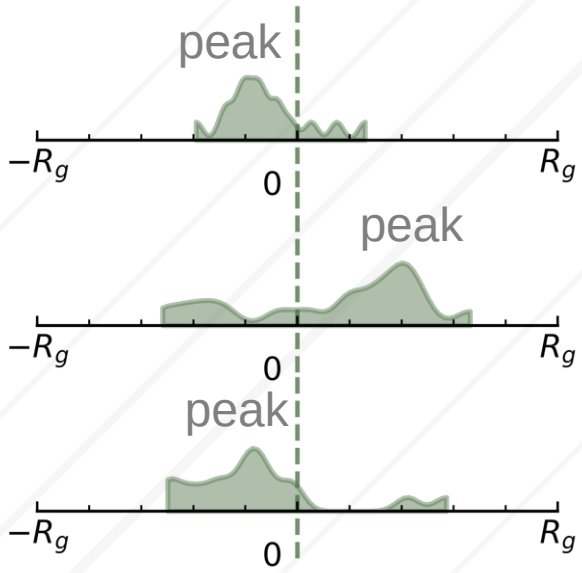


movable ½–step grid



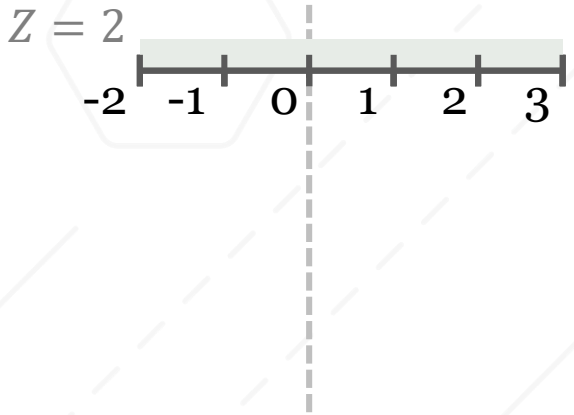
Numeric Type Design

diverse peak positions

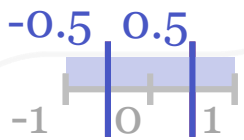


asymmetric data ranges

base integer grid

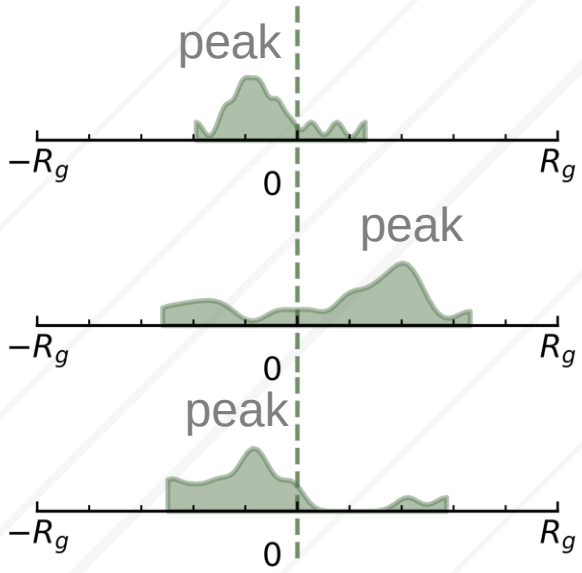


movable ½–step grid



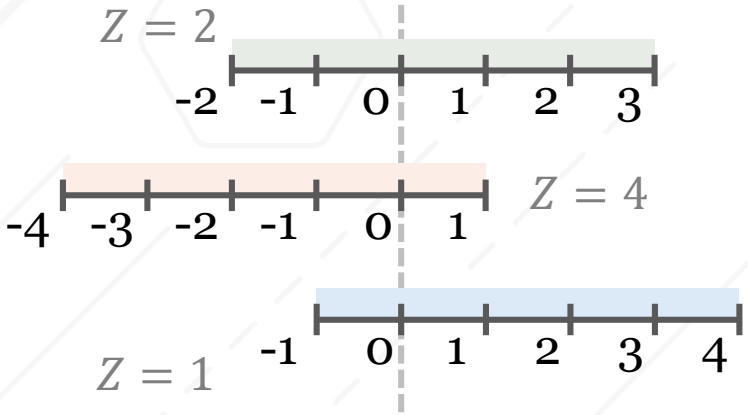
Numeric Type Design

diverse peak positions

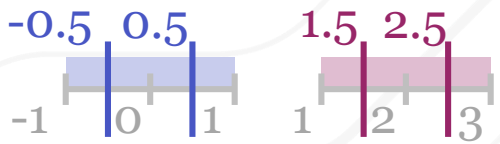


asymmetric data ranges

base integer grid

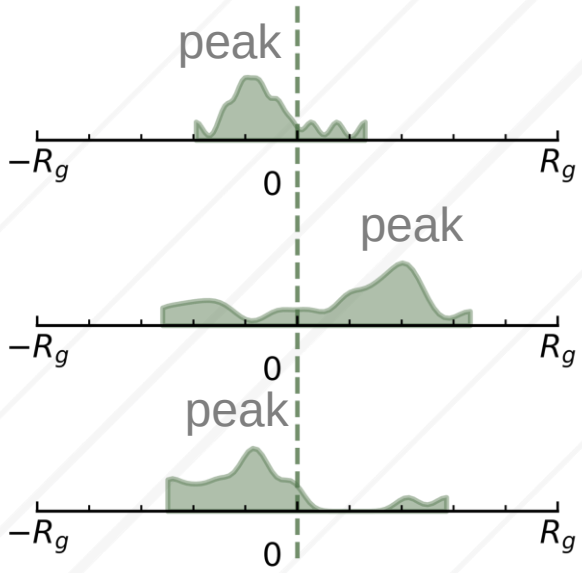


movable ½–step grid



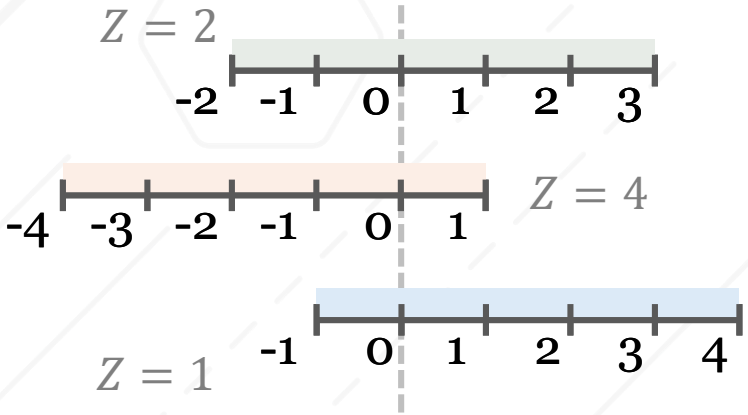
Numeric Type Design

diverse peak positions

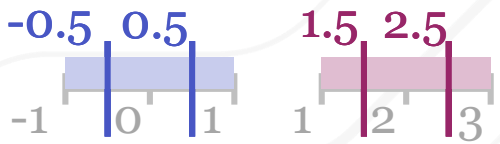


asymmetric data ranges

base integer grid

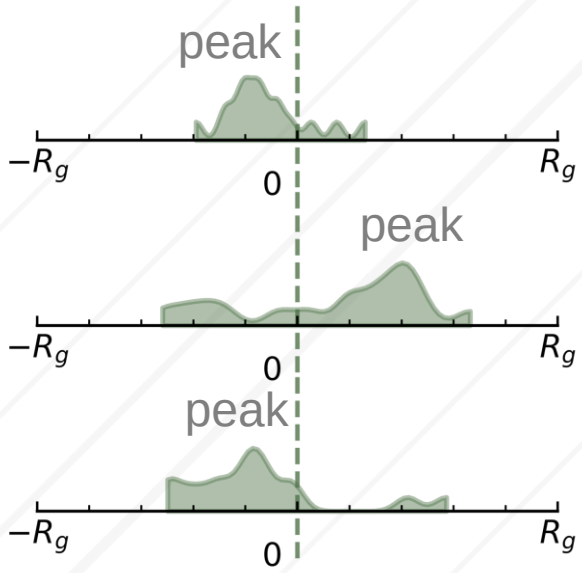


movable ½–step grid



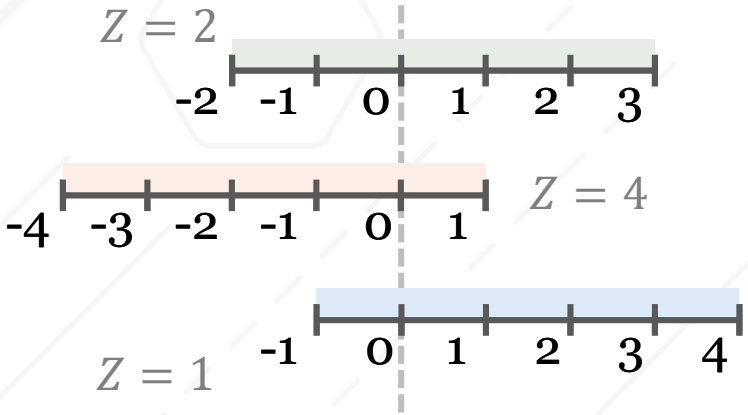
Numeric Type Design

diverse peak positions

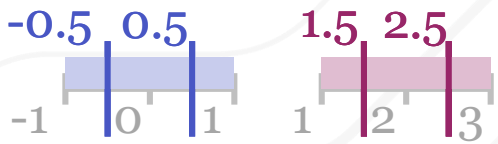


asymmetric data ranges

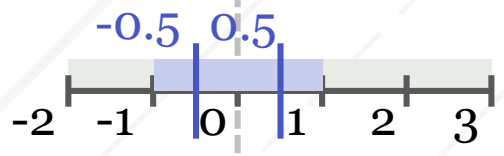
base integer grid



movable ½–step grid

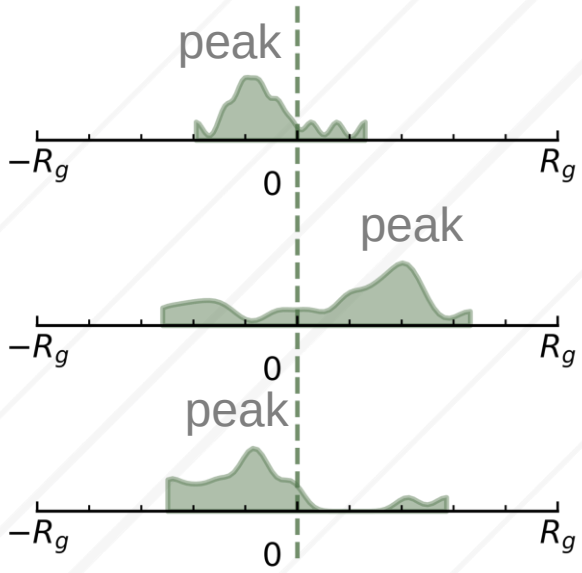


combine to form Adana grids



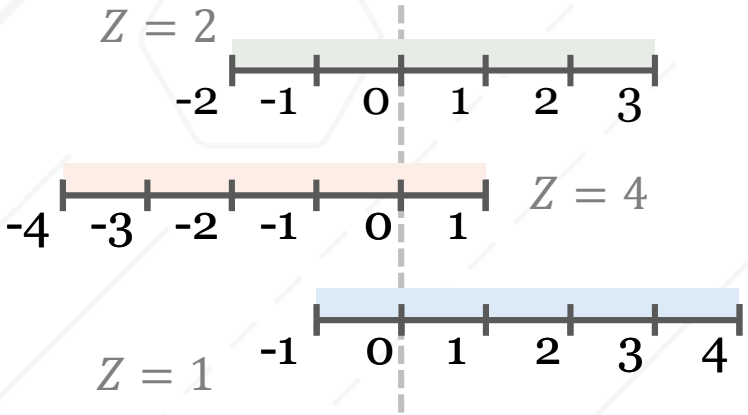
Numeric Type Design

diverse peak positions

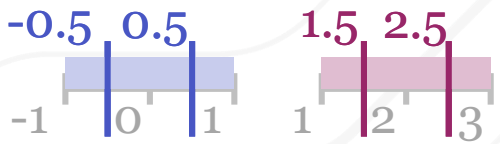


asymmetric data ranges

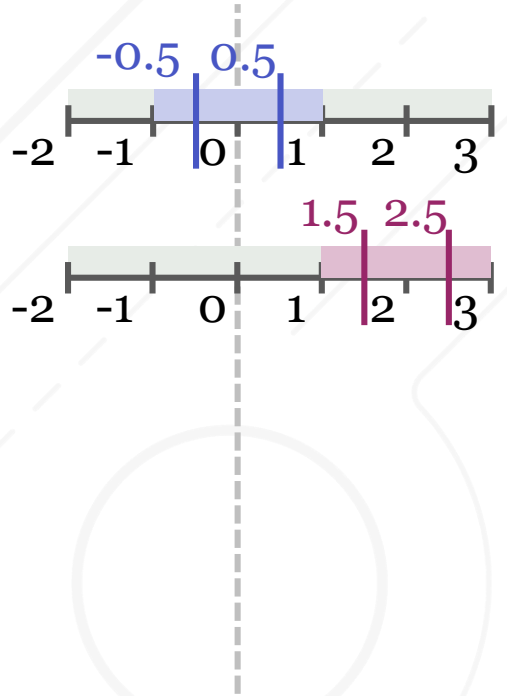
base integer grid



movable ½–step grid

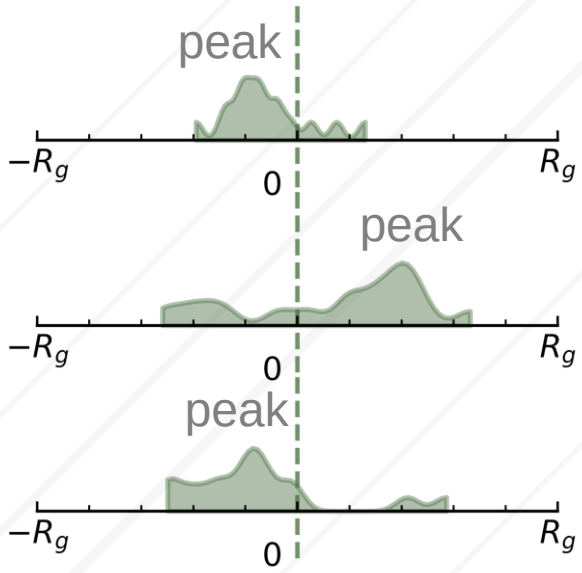


combine to form Adana grids



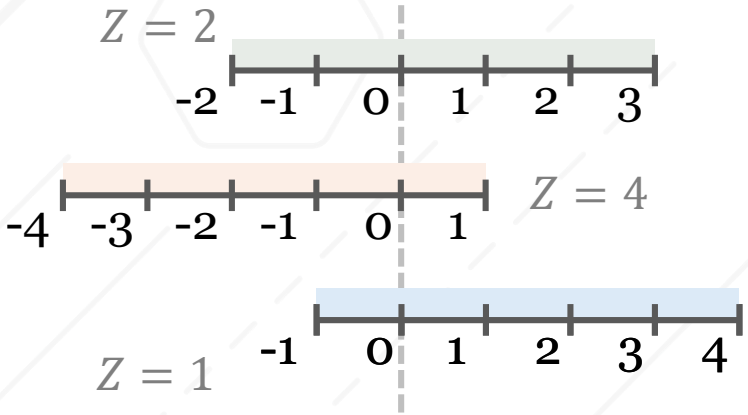
Numeric Type Design

diverse peak positions

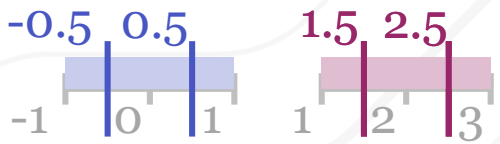


asymmetric data ranges

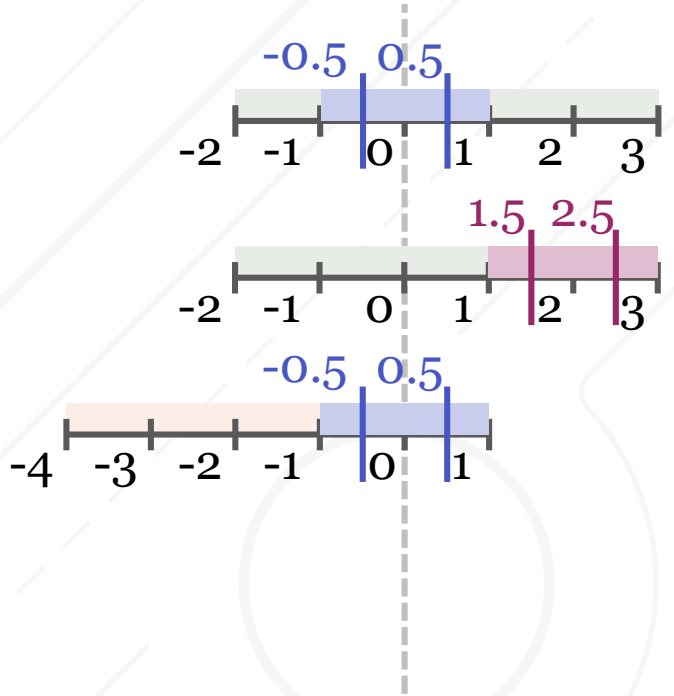
base integer grid



movable ½–step grid

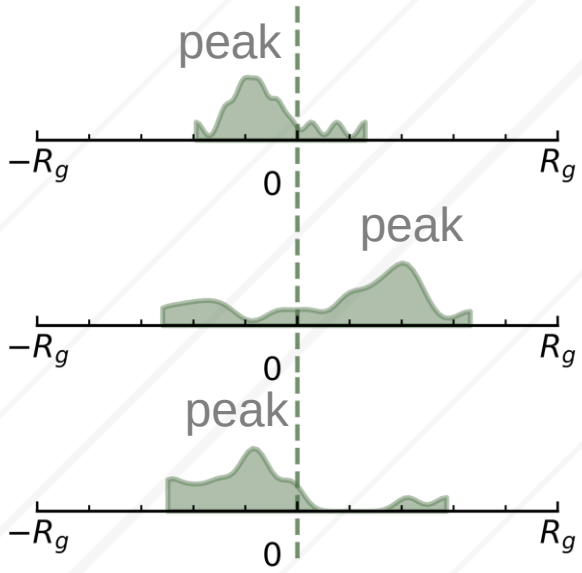


combine to form Adana grids



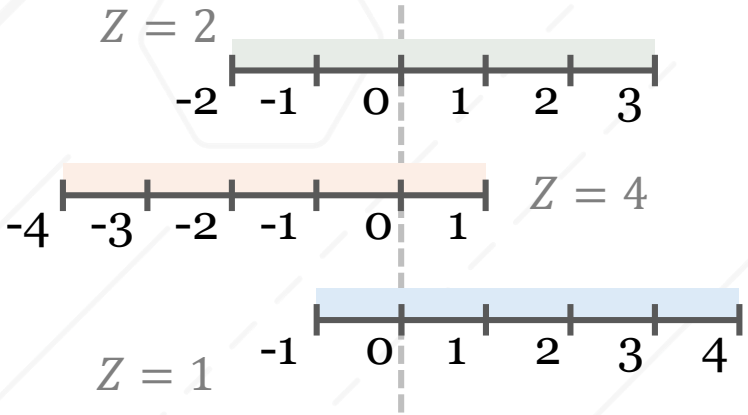
Numeric Type Design

diverse peak positions

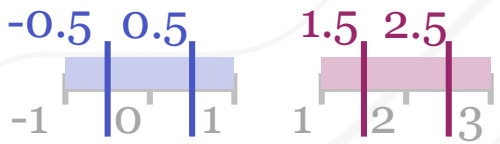


asymmetric data ranges

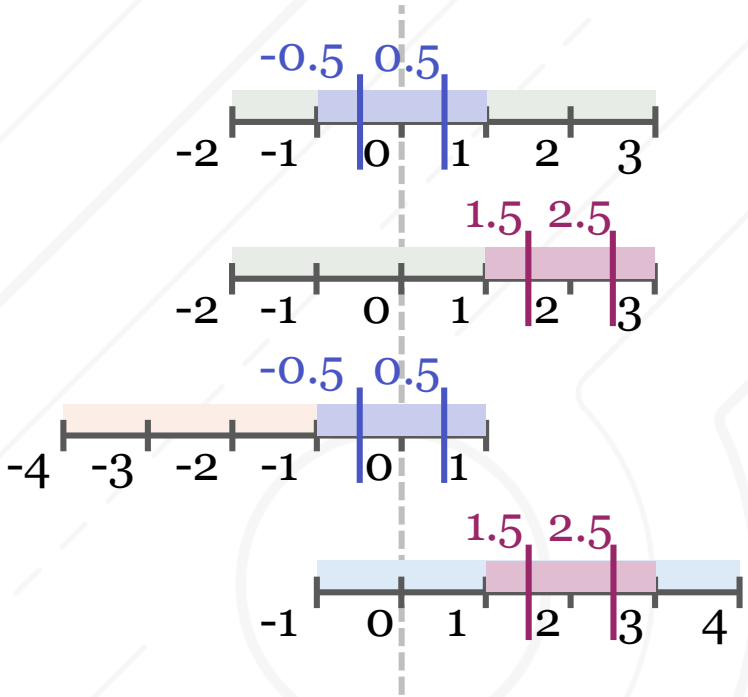
base integer grid



movable ½–step grid



combine to form Adana grids



Zero point adapts to asymmetric data ranges.

Movable ½–step grid fits diverse peak positions.

Algorithm & Architecture

Quantization Framework

offline weight quantization

weight group w_f

↓

Search sub-type T_j

↓

$\text{argmin}_j \| w_f - \Delta q(w_f, T_j) \|$

↓

select the closest point

GPU-friendly / done once

online activation quantization

activation group a_f

↓

integer rounding / half-step rounding

↓

update error table e_j for all modes

↓

$\text{argmin}_j e_j$

↓

encode by the selected mode

why it is efficient

- No floating-point MSE computation
- Shared base grid across sub-types
- Only rounding & error counting

Approximate metric:
count mismatches to the union grid
instead of full floating-point MSE.

MSE-based search | **lightweight rounding & error counting**

Algorithm & Architecture

Quantization Framework

offline weight quantization

weight group w_f

↓

Search sub-type T_j

↓

$\text{argmin}_j \|w_f - \Delta q(w_f, T_j)\|$

↓

select the closest point

GPU-friendly / done once

online activation quantization

activation group a_f

integer rounding | half-step rounding

update error table e_j for all modes

$\text{argmin}_j e_j$

encode by the selected mode

why it is efficient

No floating-point MSE computation

Shared base grid across sub-types

Only rounding & error counting

Approximate metric:
count mismatches to the union grid
instead of full floating-point MSE.

MSE-based search | **lightweight rounding & error counting**

Algorithm & Architecture

Quantization Framework

offline weight quantization

weight group w_f

↓

Search sub-type T_j

↓

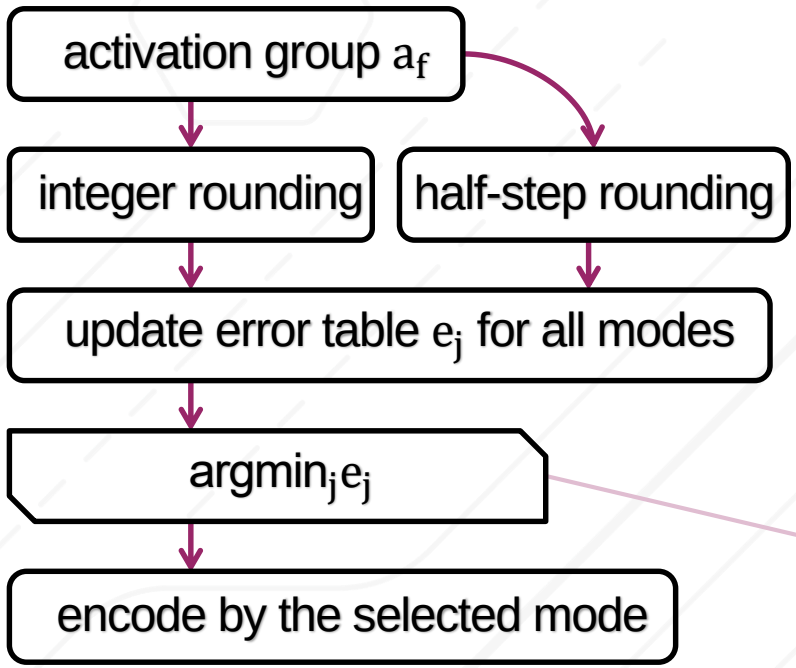
$\text{argmin}_j \| w_f - \Delta q(w_f, T_j) \|$

↓

select the closest point

GPU-friendly / done once

online activation quantization



why it is efficient

No floating-point MSE computation

Shared base grid across sub-types

Only rounding & error counting

Approximate metric:
count mismatches to the union grid
instead of full floating-point MSE.

MSE-based search | **lightweight rounding & error counting**

Algorithm & Architecture

Quantization Framework

offline weight quantization

- weight group w_f
- Search sub-type T_j
- $\text{argmin}_j \| w_f - \Delta q(w_f, T_j) \|$
- select the closest point

GPU-friendly / done once

online activation quantization

- activation group a_f
- integer rounding / half-step rounding
- update error table e_j for all modes
- $\text{argmin}_j e_j$
- encode by the selected mode

why it is efficient

- No floating-point MSE computation
- Shared base grid across sub-types
- Only rounding & error counting

Approximate metric:
count mismatches to the union grid
instead of full floating-point MSE.

MSE-based search | **lightweight rounding & error counting**

Algorithm & Architecture

Quantization Framework

offline weight quantization

weight group w_f

↓

Search sub-type T_j

↓

$\text{argmin}_j \| w_f - \Delta q(w_f, T_j) \|$

↓

select the closest point

GPU-friendly / done once

online activation quantization

activation group a_f

↓

integer rounding / half-step rounding

↓

update error table e_j for all modes

↓

$\text{argmin}_j e_j$

↓

encode by the selected mode

why it is efficient

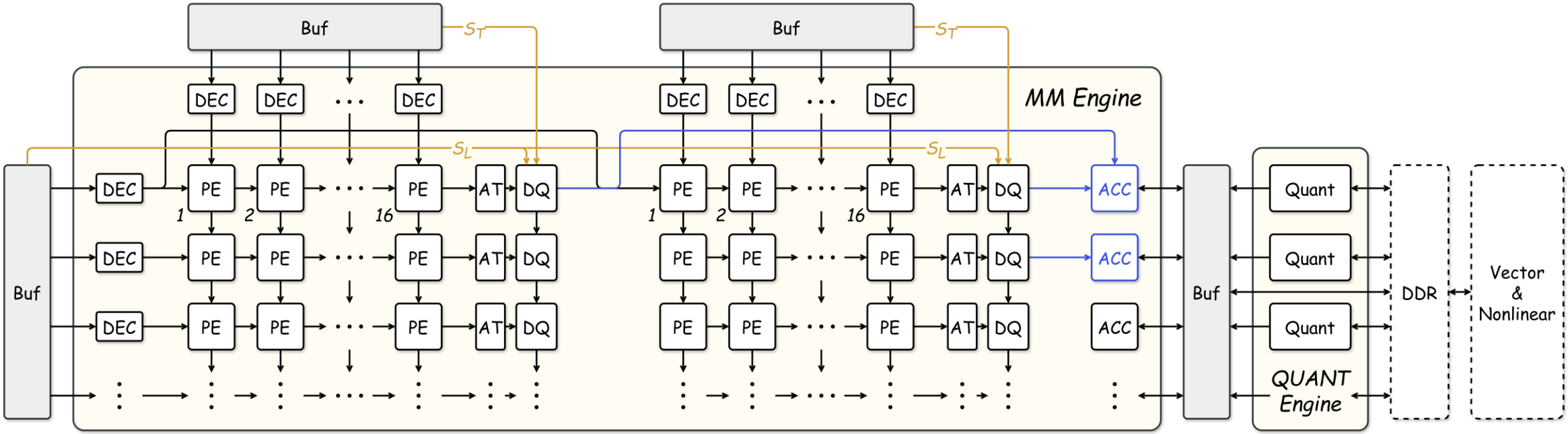
- No floating-point MSE computation
- Shared base grid across sub-types
- Only rounding & error counting

Approximate metric:
count mismatches to the union grid
instead of full floating-point MSE.

MSE-based search | **lightweight rounding & error counting**

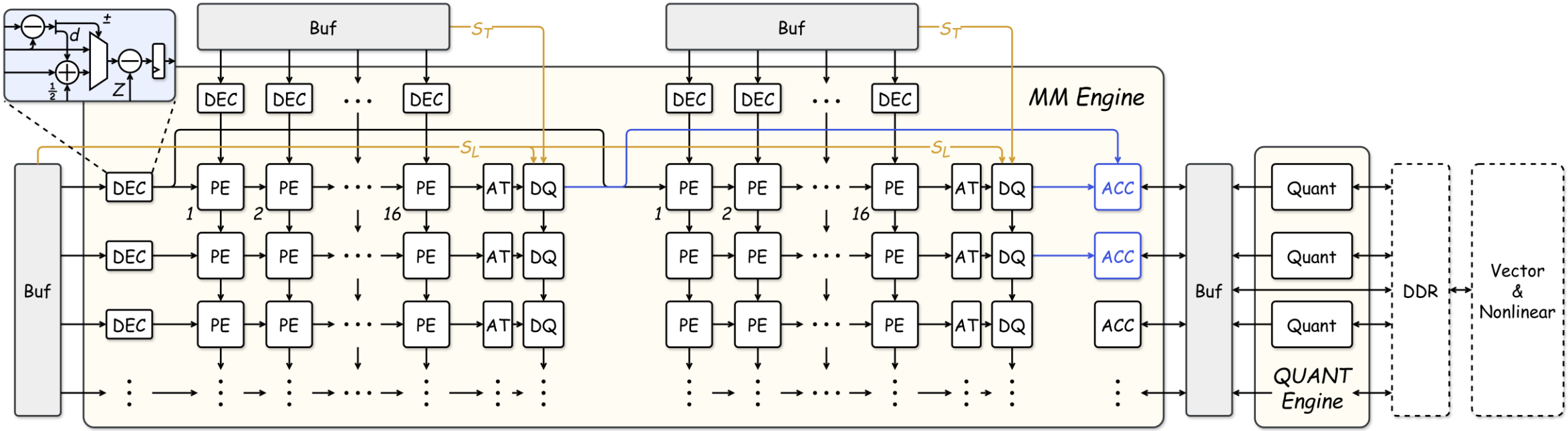
Algorithm & Architecture

Accelerator Implementation



Algorithm & Architecture

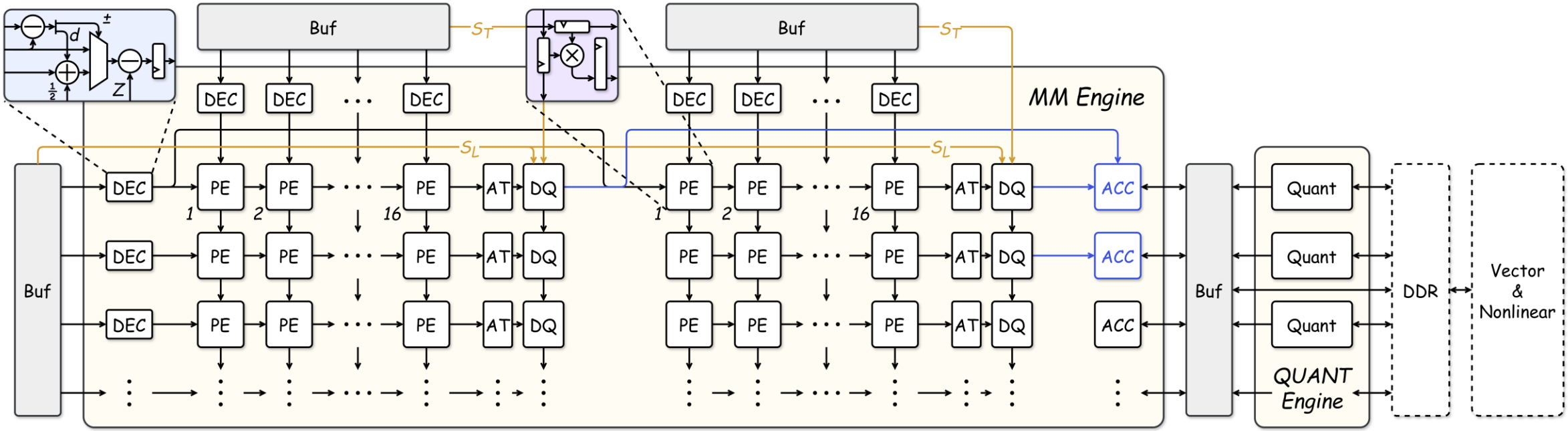
Accelerator Implementation



64 + 64 Decoders

Algorithm & Architecture

Accelerator Implementation

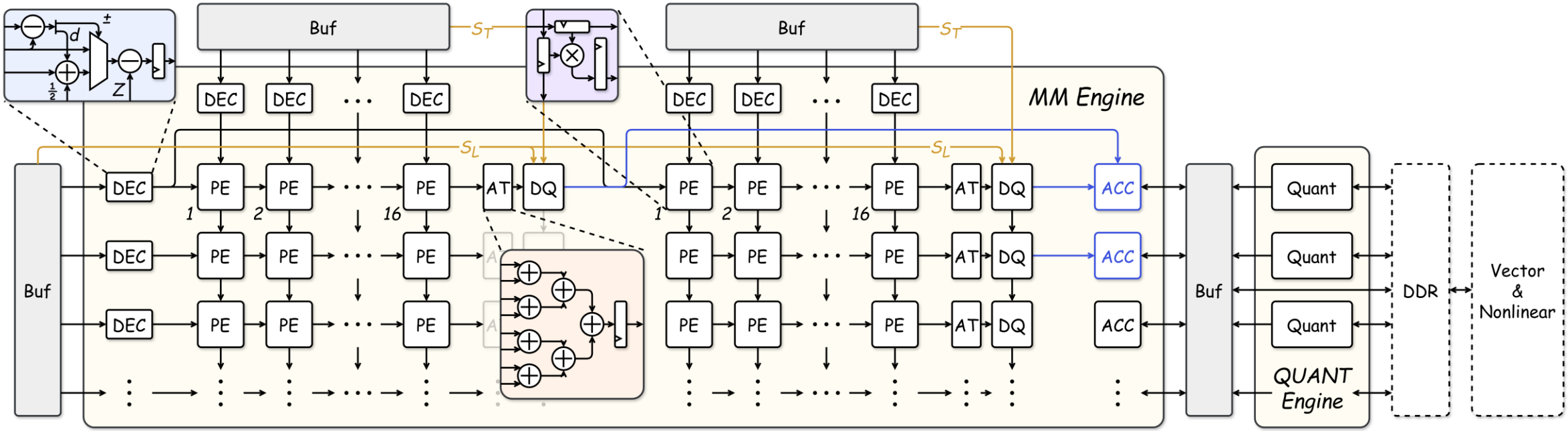


64 + 64 Decoders

4×64×16 Processing Elements → Low-bit INT Multiplication

Algorithm & Architecture

Accelerator Implementation



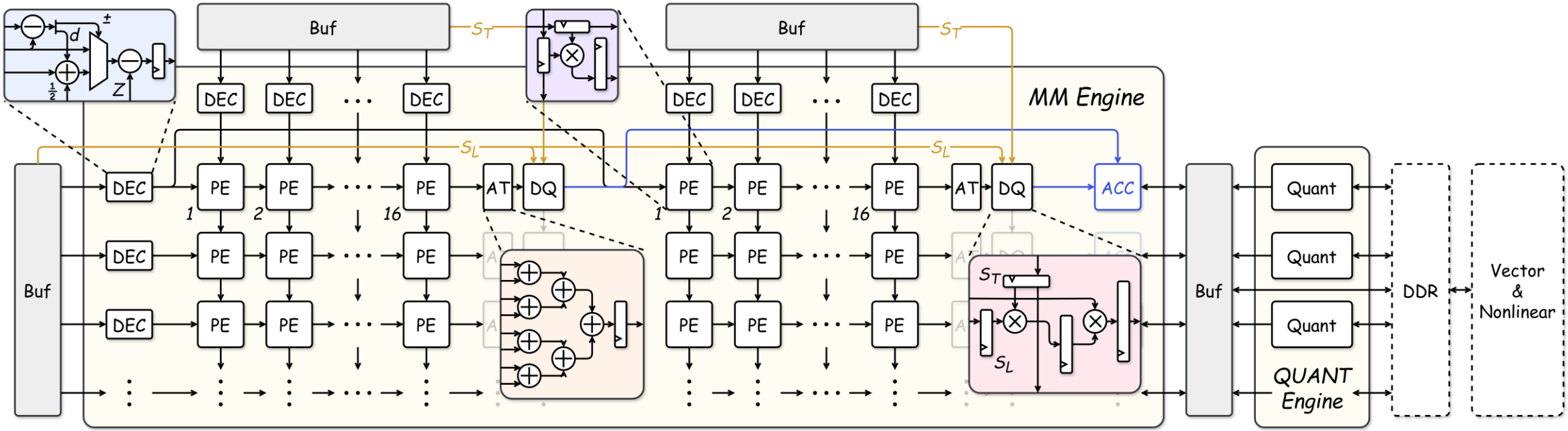
64 + 64 Decoders

4×64×16 Processing Elements → Low-bit INT Multiplication

4×64 Adder Trees

Algorithm & Architecture

Accelerator Implementation



64 + 64 Decoders

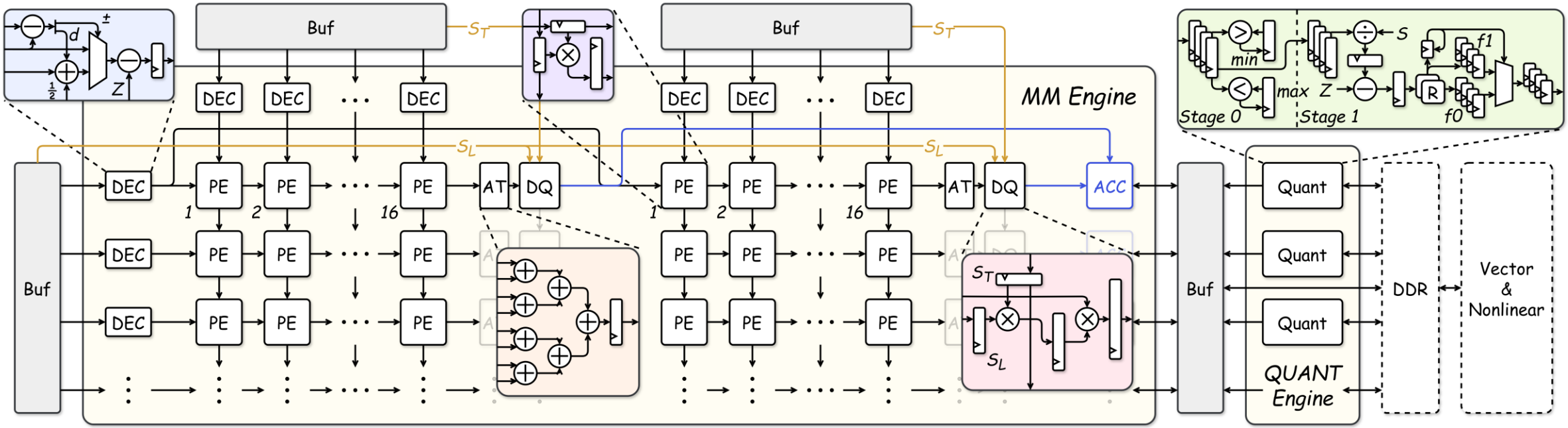
4×64×16 Processing Elements → Low-bit INT Multiplication

4×64 Adder Trees

4×64 De-quantizers

Algorithm & Architecture

Accelerator Implementation



64 + 64 Decoders

4×64×16 Processing Elements → Low-bit INT Multiplication

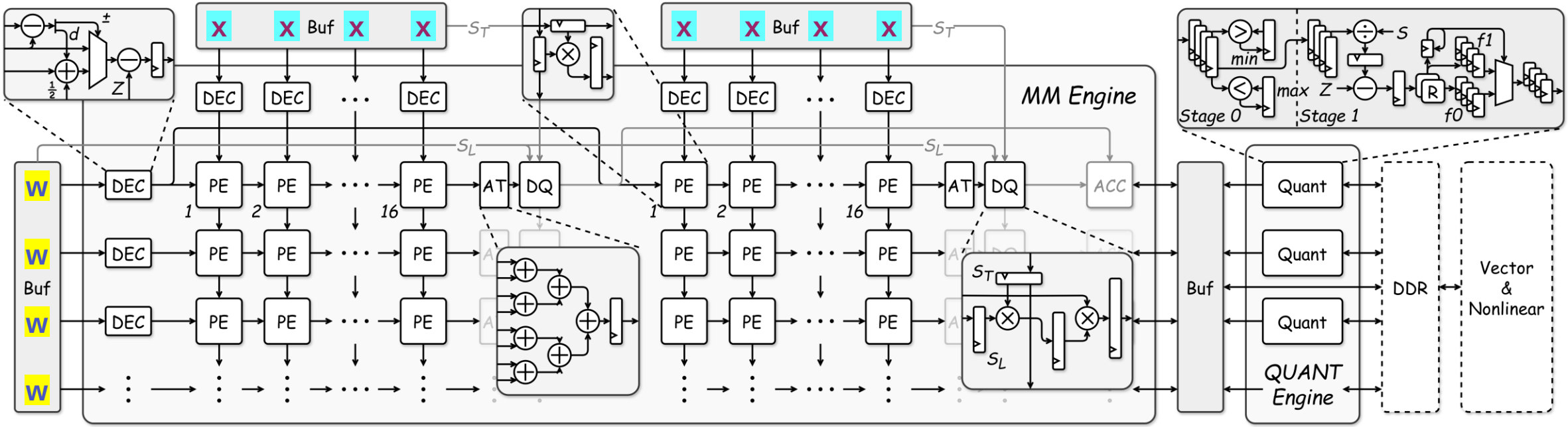
4×64 Adder Trees

4×64 De-quantizers

64 Quantizers

Algorithm & Architecture

Accelerator Implementation

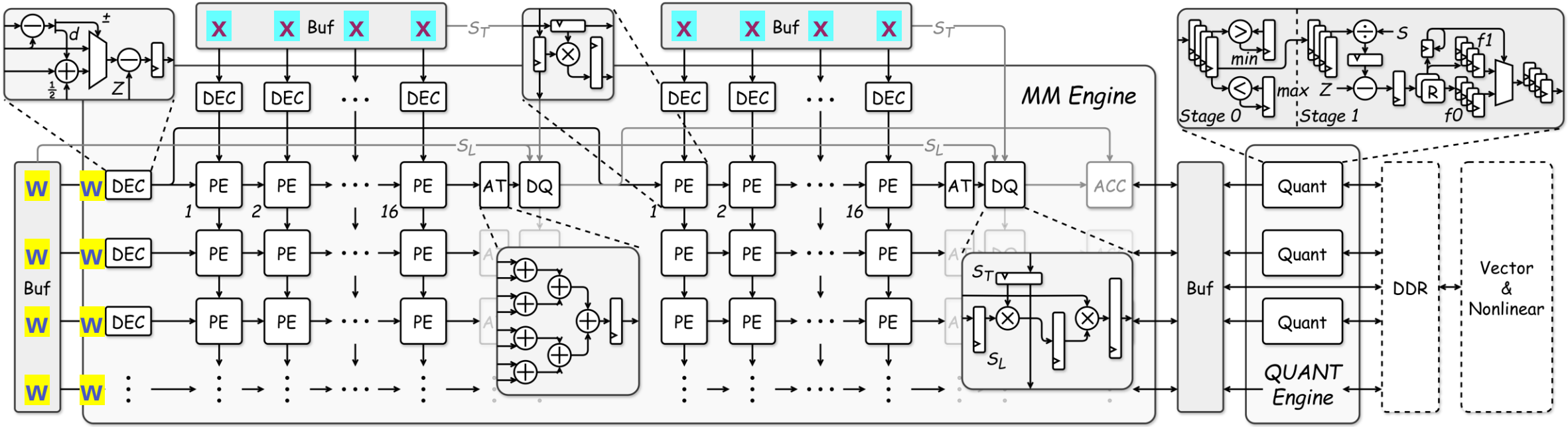


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

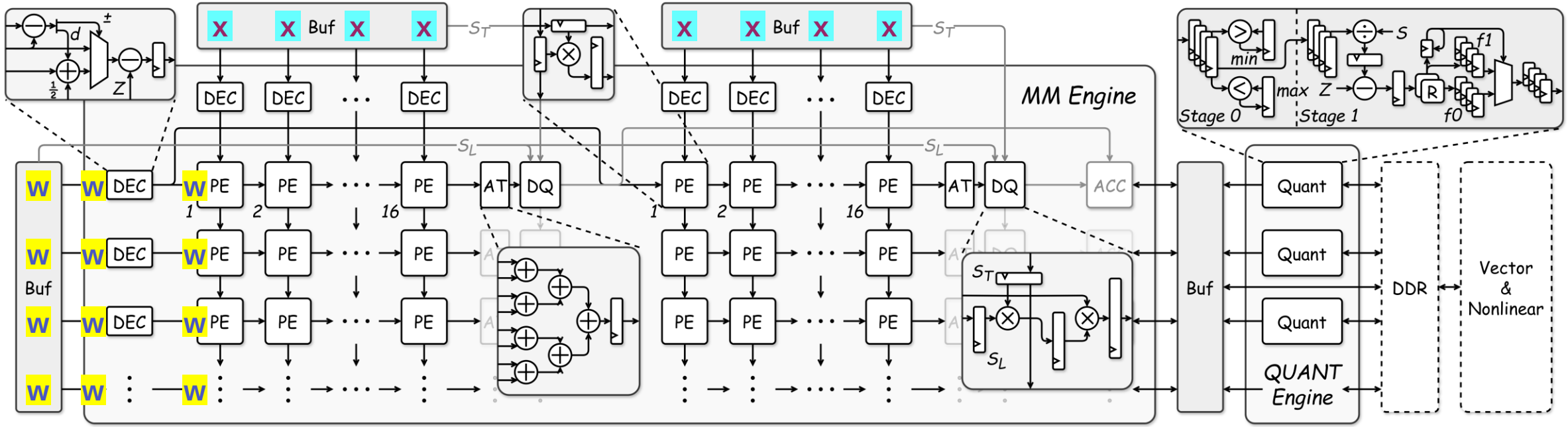


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

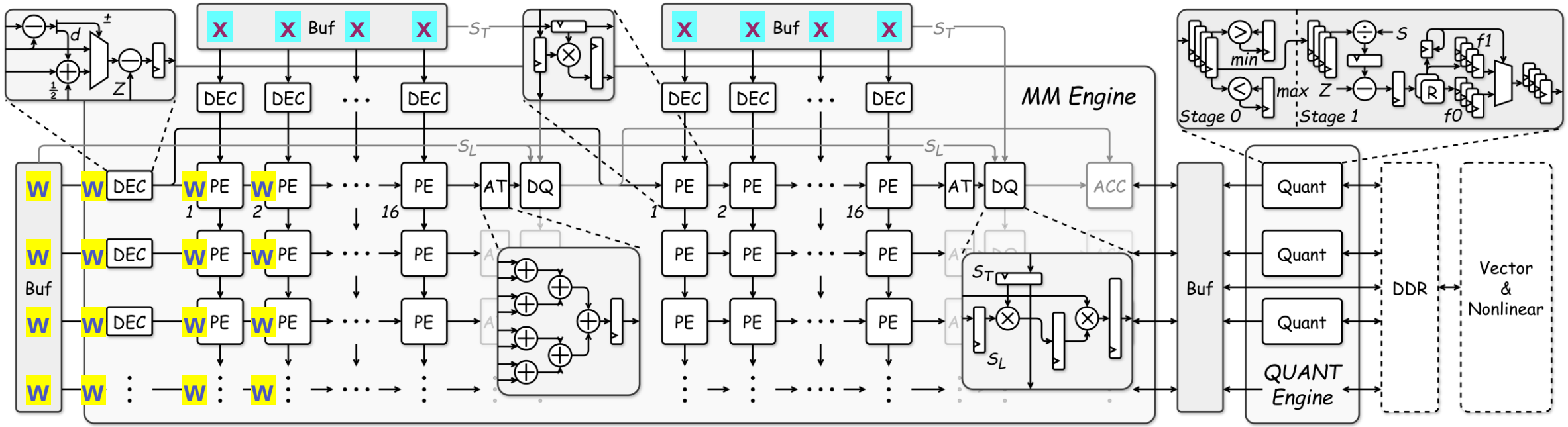


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

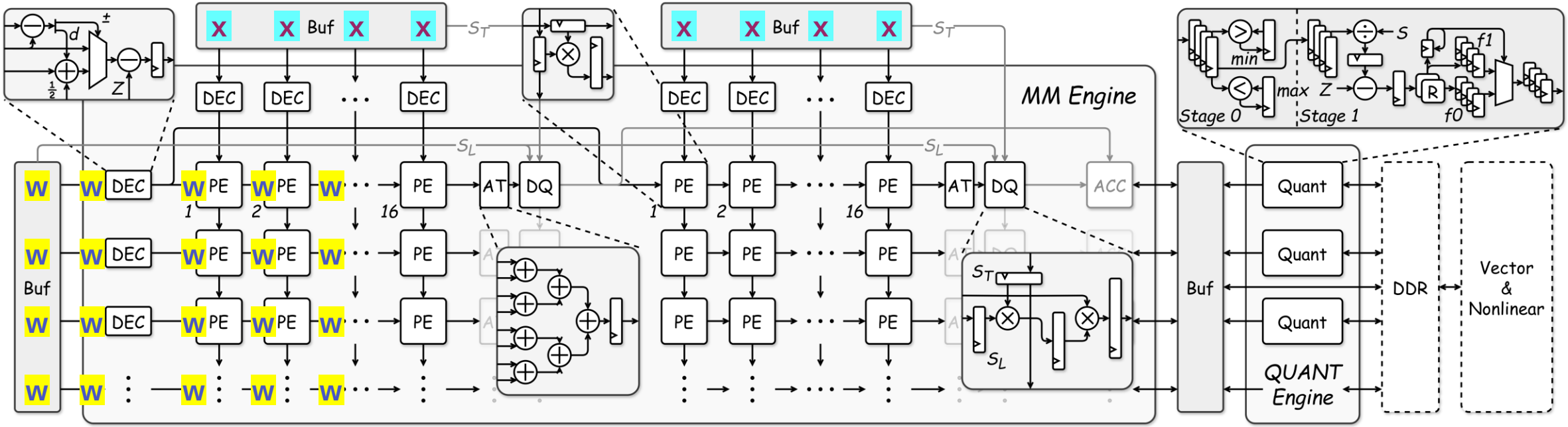


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

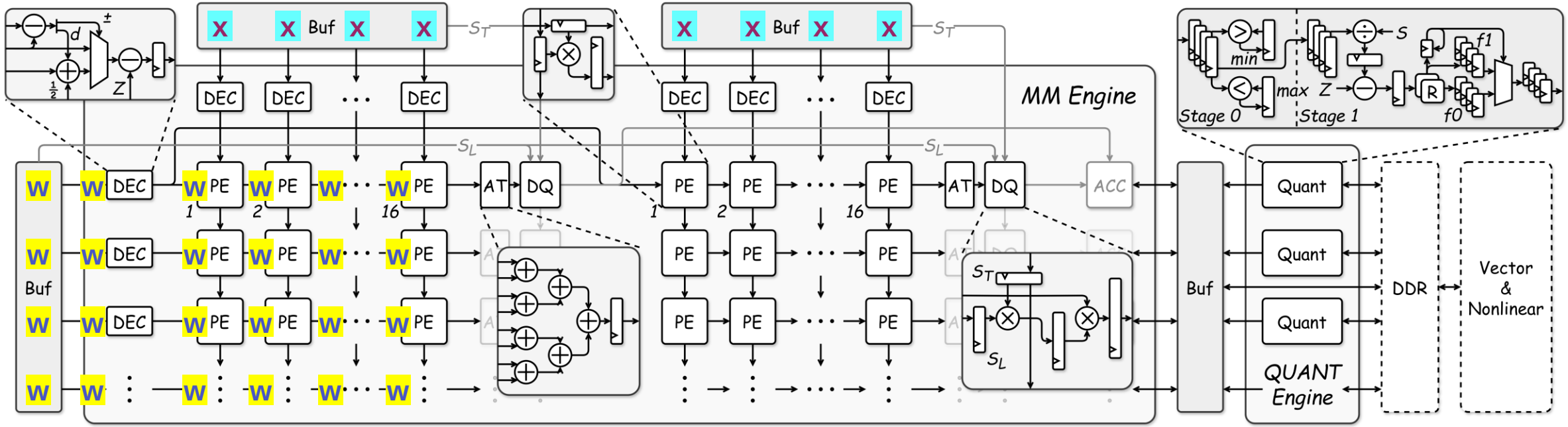


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

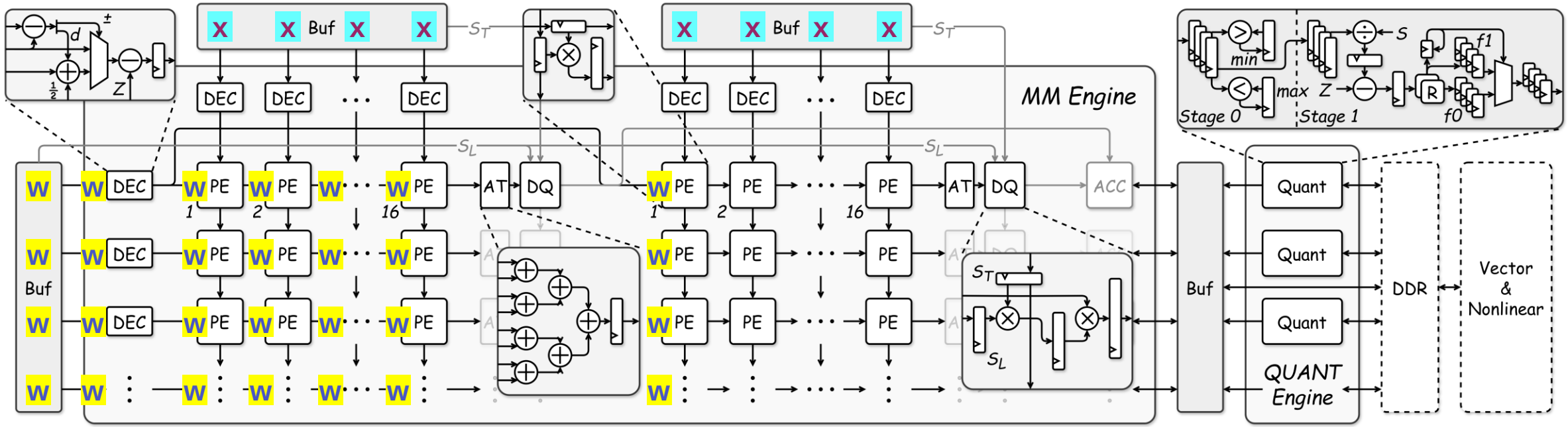


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

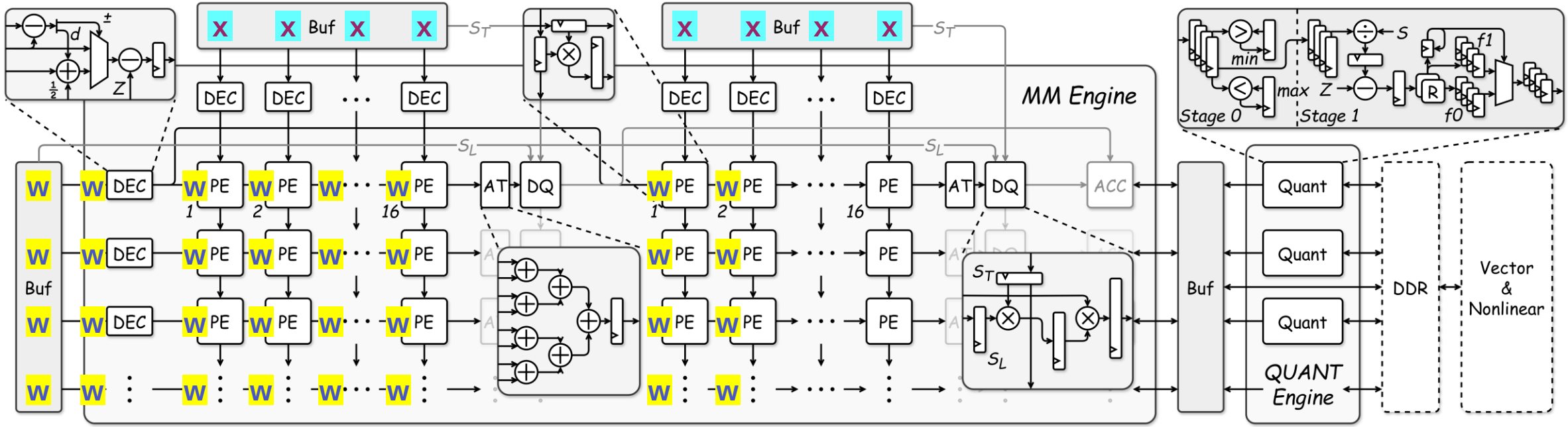


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

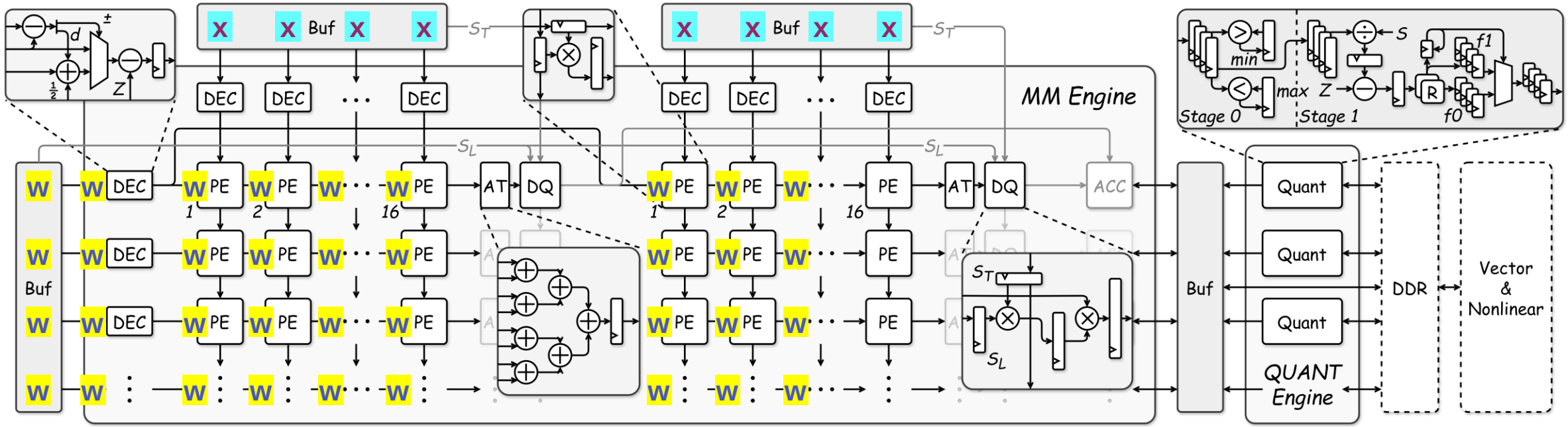


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

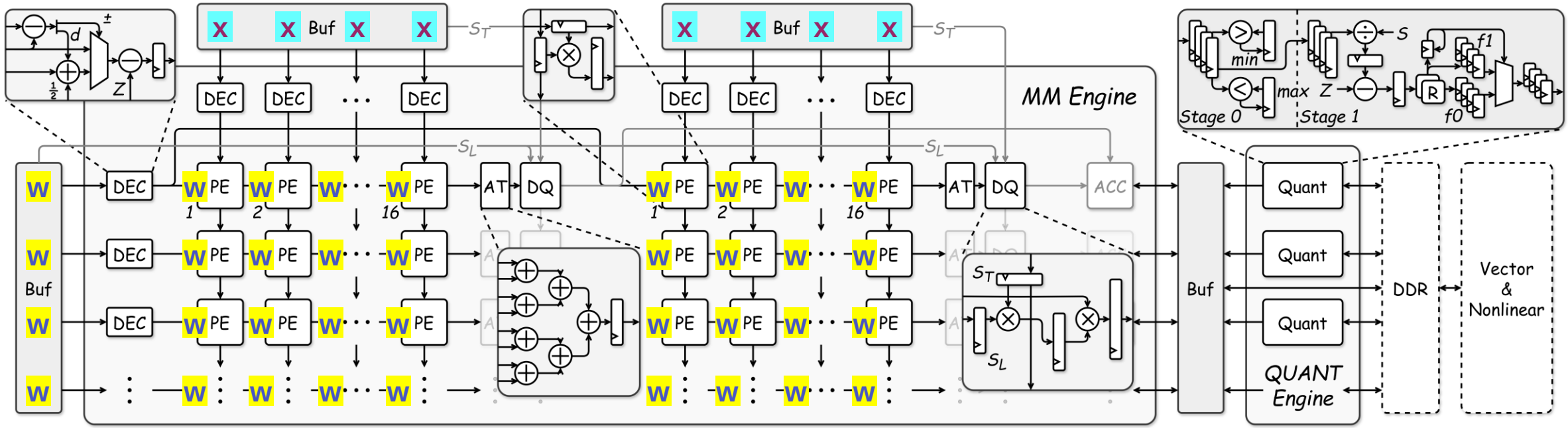


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

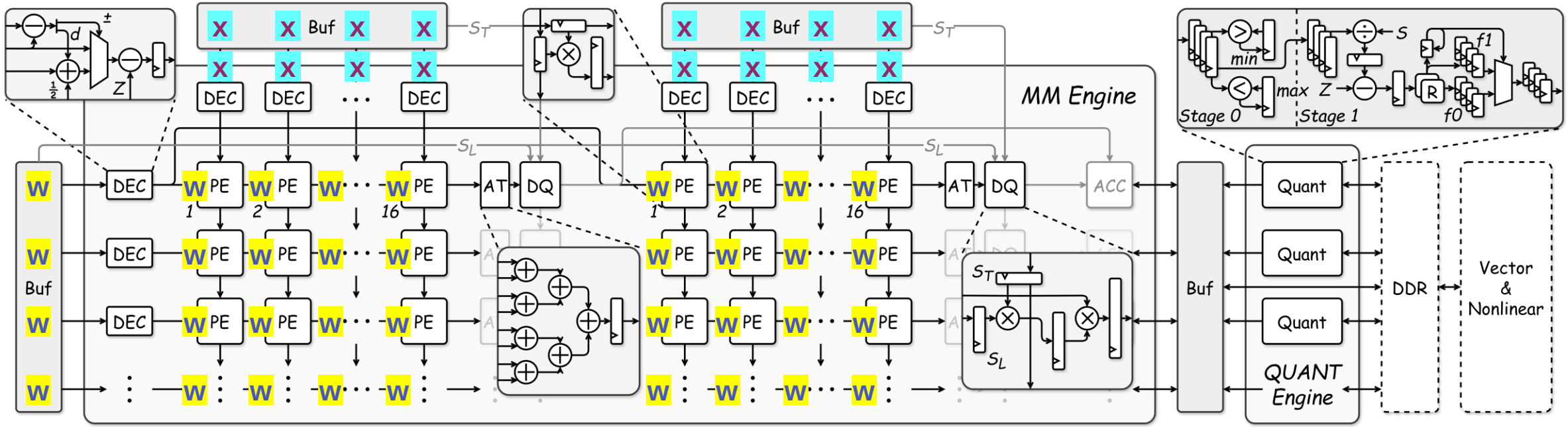


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

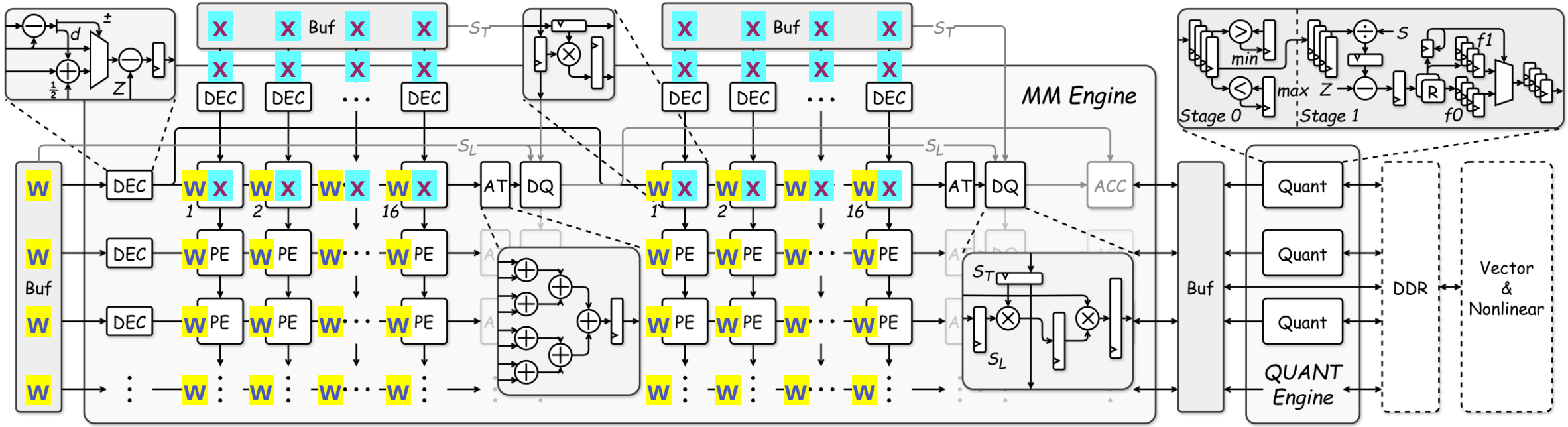
Accelerator Implementation



Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Accelerator Implementation

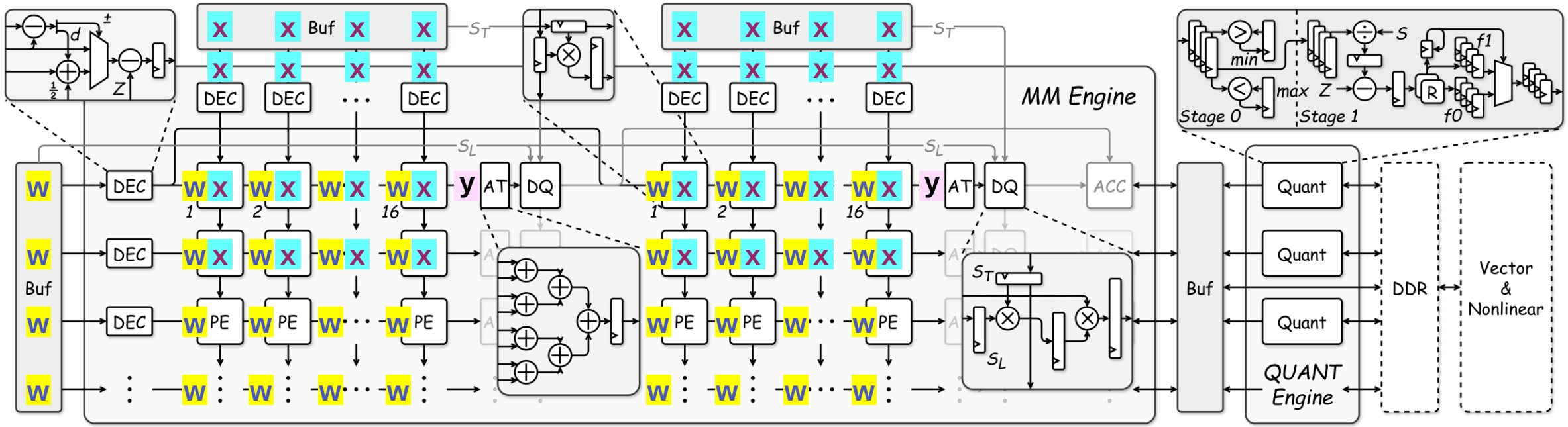


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

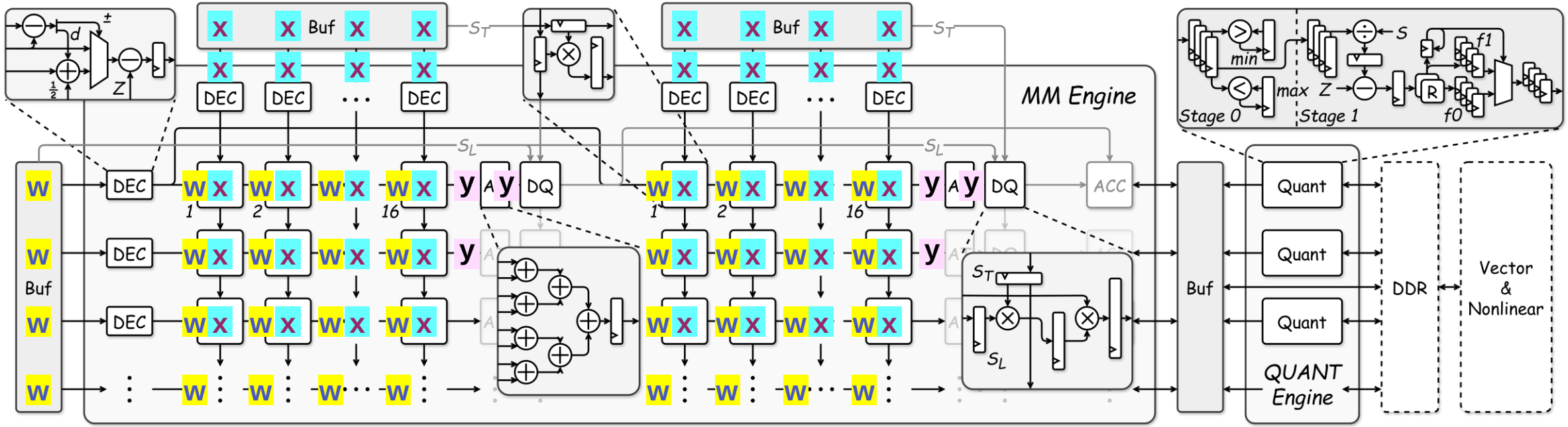


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

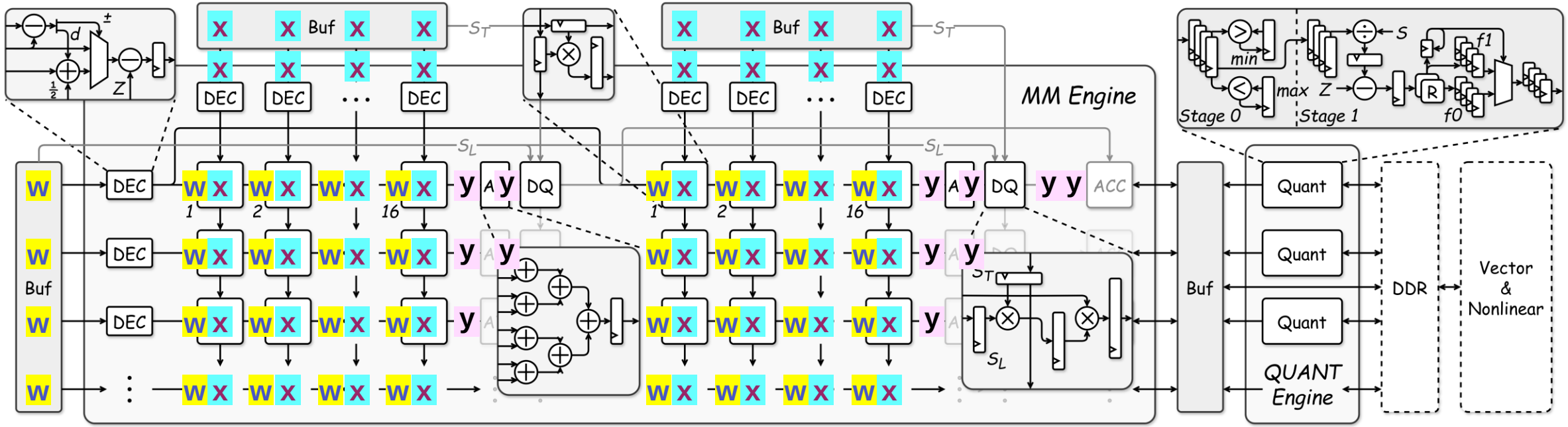


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

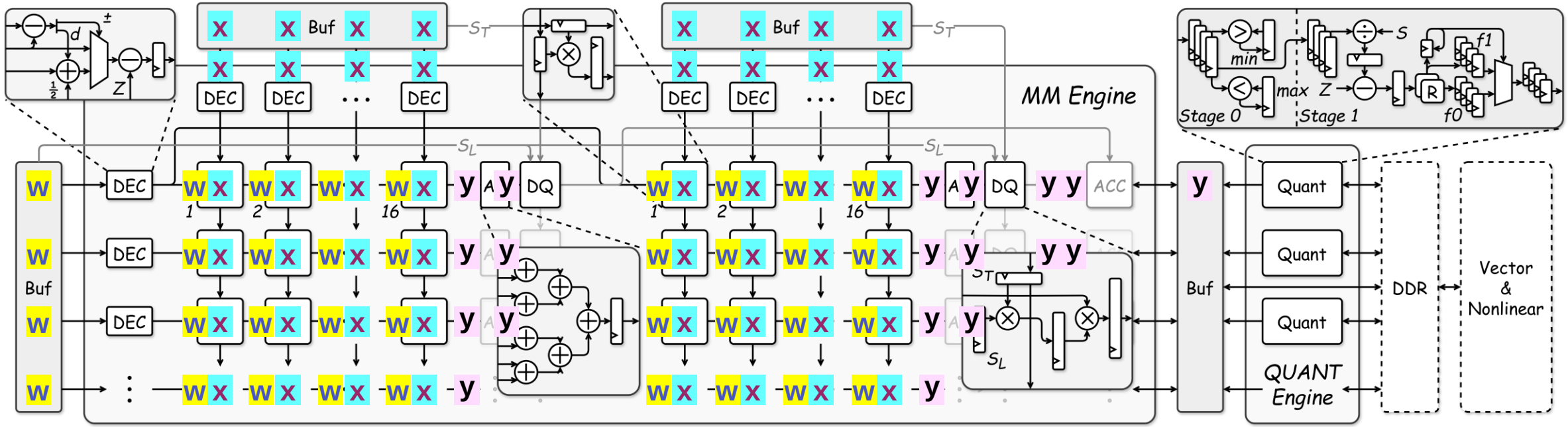


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation

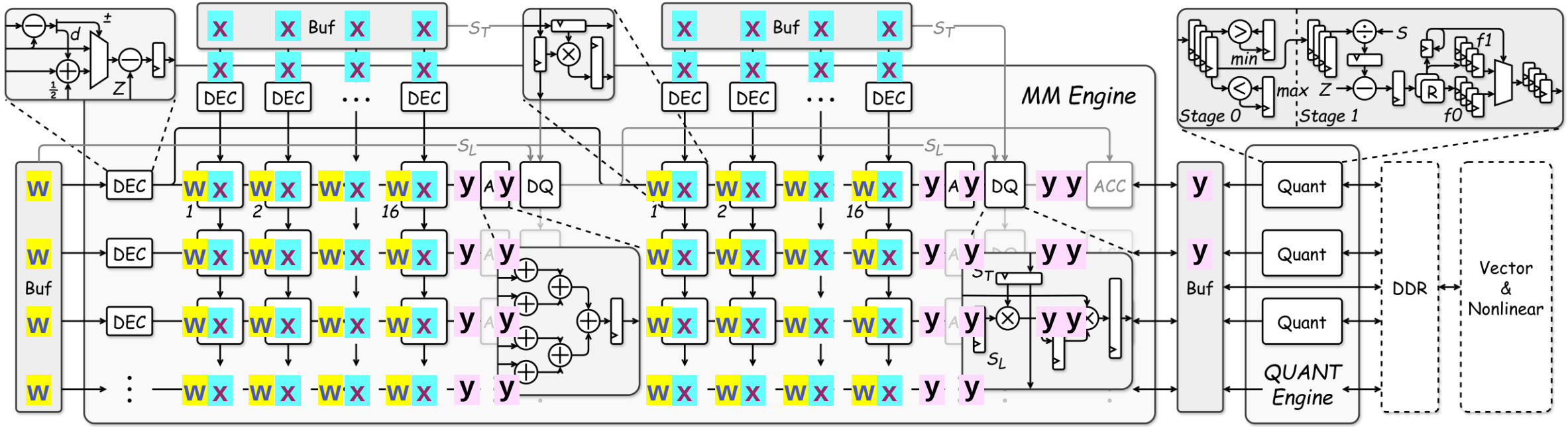


Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Algorithm & Architecture

Accelerator Implementation



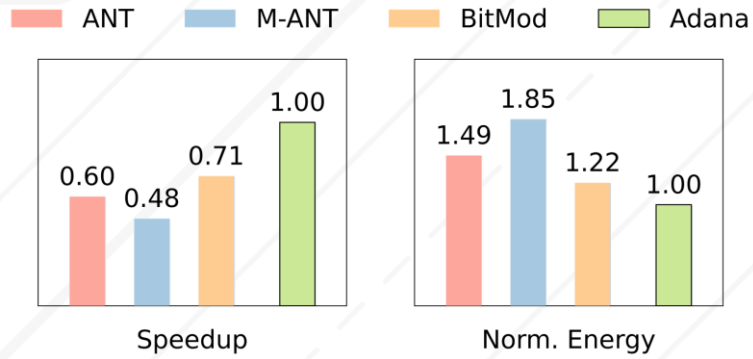
Decoded values enter INT-style multipliers; scaling is deferred until the completion of dot-product sum.

This dataflow keeps most computation in low precision and avoids unnecessary high-precision operations.

Results & Conclusion

Comparison of LLM Quantization Accelerators

	Prior works			This work
Venue	MICRO'22	HPCA'25	HPCA'25	DAC'26
Method	ANT	M-ANT	BitMod	Adana
Avg. 4-bit PPL↓	9.35	9.45	9.28	9.00
Avg. 3-bit PPL↓	60.39	103.76	19.79	15.17
QU Area (μm²)	14,328.1	7,828.9	14,931.3	5,165.9
QU Power (mW)	8.22	3.90	8.42	2.80



Avg. Speedup: **1.68× / 2.10× / 1.42×**

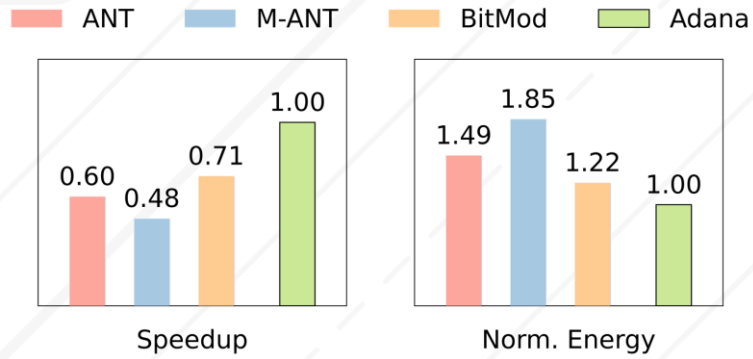
Avg. Energy Reduction: **32.81% / 45.85% / 17.91%**

- *Avg. PPL is averaged over 15 LLMs: Llama-3.1/3.2-1B/3B/8B, Phi-1.5/2/3-1.3B/2.7B/4B, Yi-1.5-6B/9B/34B, Gemma-2/3-1B/2B/4B, and Qwen-3-4B/8B/14B.*
- *Avg. Speedup and energy results are evaluated on the full accelerator and averaged over four representative models: Llama-3.1-8B, Phi-2-2.7B, Gemma-3-4B, and Qwen-3-14B.*

Results & Conclusion

Comparison of LLM Quantization Accelerators

	Prior works			This work
Venue	MICRO'22	HPCA'25	HPCA'25	DAC'26
Method	ANT	M-ANT	BitMod	Adana
Avg. 4-bit PPL↓	9.35	9.45	9.28	9.00
Avg. 3-bit PPL↓	60.39	103.76	19.79	15.17
QU Area (μm²)	14,328.1	7,828.9	14,931.3	5,165.9
QU Power (mW)	8.22	3.90	8.42	2.80



Avg. Speedup: **1.68× / 2.10× / 1.42×**

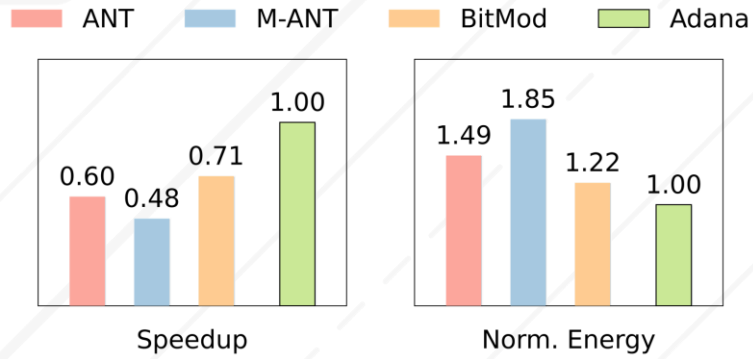
Avg. Energy Reduction: **32.81% / 45.85% / 17.91%**

- *Avg. PPL is averaged over 15 LLMs: Llama-3.1/3.2-1B/3B/8B, Phi-1.5/2/3-1.3B/2.7B/4B, Yi-1.5-6B/9B/34B, Gemma-2/3-1B/2B/4B, and Qwen-3-4B/8B/14B.*
- *Avg. Speedup and energy results are evaluated on the full accelerator and averaged over four representative models: Llama-3.1-8B, Phi-2-2.7B, Gemma-3-4B, and Qwen-3-14B.*

Results & Conclusion

Comparison of LLM Quantization Accelerators

	Prior works			This work
Venue	MICRO'22	HPCA'25	HPCA'25	DAC'26
Method	ANT	M-ANT	BitMod	Adana
Avg. 4-bit PPL↓	9.35	9.45	9.28	9.00
Avg. 3-bit PPL↓	60.39	103.76	19.79	15.17
QU Area (μm²)	14,328.1	7,828.9	14,931.3	5,165.9
QU Power (mW)	8.22	3.90	8.42	2.80



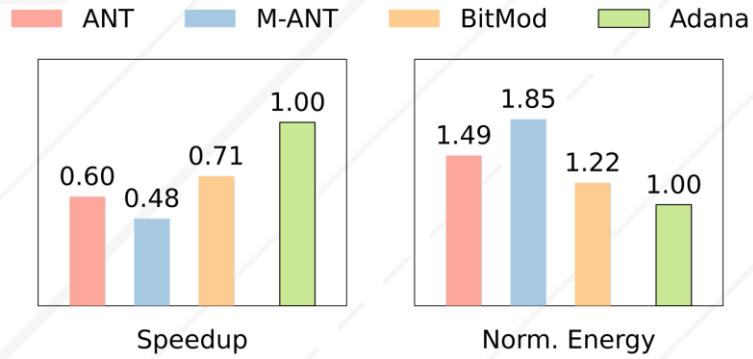
Avg. Speedup: **1.68× / 2.10× / 1.42×**

Avg. Energy Reduction: **32.81% / 45.85% / 17.91%**

- *Avg. PPL is averaged over 15 LLMs: Llama-3.1/3.2-1B/3B/8B, Phi-1.5/2/3-1.3B/2.7B/4B, Yi-1.5-6B/9B/34B, Gemma-2/3-1B/2B/4B, and Qwen-3-4B/8B/14B.*
- *Avg. Speedup and energy results are evaluated on the full accelerator and averaged over four representative models: Llama-3.1-8B, Phi-2-2.7B, Gemma-3-4B, and Qwen-3-14B.*

Comparison of LLM Quantization Accelerators

	Prior works			This work
Venue	MICRO'22	HPCA'25	HPCA'25	DAC'26
Method	ANT	M-ANT	BitMod	Adana
Avg. 4-bit PPL↓	9.35	9.45	9.28	9.00
Avg. 3-bit PPL↓	60.39	103.76	19.79	15.17
QU Area (μm²)	14,328.1	7,828.9	14,931.3	5,165.9
QU Power (mW)	8.22	3.90	8.42	2.80



Avg. Speedup: **1.68× / 2.10× / 1.42×**

Avg. Energy Reduction: **32.81% / 45.85% / 17.91%**

- *Avg. PPL is averaged over 15 LLMs: Llama-3.1/3.2-1B/3B/8B, Phi-1.5/2/3-1.3B/2.7B/4B, Yi-1.5-6B/9B/34B, Gemma-2/3-1B/2B/4B, and Qwen-3-4B/8B/14B.*
- *Avg. Speedup and energy results are evaluated on the full accelerator and averaged over four representative models: Llama-3.1-8B, Phi-2-2.7B, Gemma-3-4B, and Qwen-3-14B.*

Conclusion

- **Adana Numeric Type**
  - A novel numeric type that effectively captures nonuniformity and asymmetry in small groups.
- **Quantization Framework**
  - An efficient online algorithm that uses rounding-based error counting, avoiding full MSE search.
- **Dedicated Microarchitecture**
  - A scalable architecture with MM and QUANT engines, tailored for group-wise quantization.
- **Superior Performance**
  - Adana delivers 1.42× - 2.10× speedups, and 17.9% - 45.9% power savings with consistently higher accuracy compared to SOTA works.

Adana: Accelerating Large Language Models via Adaptive Nonuniform Asymmetric Quantization

Thank you for your attention.