QTIP: Quantization with Trellises and Incoherence Processing

김승준, 권순현, 이재훈

Key contributions

- Incoherence Processing for Gaussian-Like Weights
    - Apply incoherence processing (RHT) to transform weights into independent, Gaussian-like distributions, enabling effective trellis coding for i.i.d. sources.
    - Ensures robust quantization by minimizing correlations in large-scale LLM weights.

- Bitshift and Tail-Biting Trellis for Ultra-High-Dimensional Quantization
    - Introduce Bitshift Trellis for lookup-free transitions (1 cycle) and Tail-Biting Trellis with Algorithm 4 (two Viterbi calls) to enable tractable quantization of ultra-high-dimensional sequences (>100 dimensions).
    - Achieve bit efficiency and reduced distortion, supporting fast inference (4ms/token) on GPUs.

- Hybrid Lookup-Computed Code for Efficient Code Generation
    - Develop a hybrid code that generates pseudorandom indices with a small LUT (cached in GPU memory), eliminating full codebook storage (8 bytes $\rightarrow$ 0).
    - Balances speed and quality with post-quantization fine-tuning, enhancing overall performance.

Key contributions

- Incoherence Processing for Gaussian-Like Weights
    - Apply incoherence processing (RHT) to transform weights into independent, Gaussian-like distributions, enabling effective trellis coding for i.i.d. sources.
    - Ensures robust quantization by minimizing correlations in large-scale LLM weights.

- Bitshift and Tail-Biting Trellis for Ultra-High-Dimensional Quantization
    - Introduce Bitshift Trellis for lookup-free transitions (1 cycle) and Tail-Biting Trellis with Algorithm 4 (two Viterbi calls) to enable tractable quantization of ultra-high-dimensional sequences (>100 dimensions).
    - Achieve bit efficiency and reduced distortion, supporting fast inference (4ms/token) on GPUs.

- Hybrid Lookup-Computed Code for Efficient Code Generation
    - Develop a hybrid code that generates pseudorandom indices with a small LUT (cached in GPU memory), eliminating full codebook storage (8 bytes → 0).
    - Balances speed and quality with post-quantization fine-tuning, enhancing overall performance.

Key contributions

- Incoherence Processing for Gaussian-Like Weights
  - Apply incoherence processing (RHT) to transform weights into independent, Gaussian-like distributions, enabling effective trellis coding for i.i.d. sources.
  - Ensures robust quantization by minimizing correlations in large-scale LLM weights.
- Bitshift and Tail-Biting Trellis for Ultra-High-Dimensional Quantization
  - Introduce Bitshift Trellis for lookup-free transitions (1 cycle) and Tail-Biting Trellis with Algorithm 4 (two Viterbi calls) to enable tractable quantization of ultra-high-dimensional sequences (>100 dimensions).
  - Achieve bit efficiency and reduced distortion, supporting fast inference (4ms/token) on GPUs.
- Hybrid Lookup-Computed Code for Efficient Code Generation
  - Develop a hybrid code that generates pseudorandom indices with a small LUT (cached in GPU memory), eliminating full codebook storage (8 bytes → 0).
  - Balances speed and quality with post-quantization fine-tuning, enhancing overall performance.

Key contributions

- Incoherence Processing for Gaussian-Like Weights
  - Apply incoherence processing (RHT) to transform weights into independent, Gaussian-like distributions, enabling effective trellis coding for i.i.d. sources.
  - Ensures robust quantization by minimizing correlations in large-scale LLM weights.

- Bitshift and Tail-Biting Trellis for Ultra-High-Dimensional Quantization
  - Introduce Bitshift Trellis for lookup-free transitions (1 cycle) and Tail-Biting Trellis with Algorithm 4 (two Viterbi calls) to enable tractable quantization of ultra-high-dimensional sequences (>100 dimensions).
  - Achieve bit efficiency and reduced distortion, supporting fast inference (4ms/token) on GPUs.

- Hybrid Lookup-Computed Code for Efficient Code Generation
  - Develop a hybrid code that generates pseudorandom indices with a small LUT (cached in GPU memory), eliminating full codebook storage (8 bytes → 0).
  - Balances speed and quality with post-quantization fine-tuning, enhancing overall performance.

Key contributions

Target

1. High dimension vector quantization (256)
2. Less memory access with more calculation

- Incoherence Processing for Gaussian-Like Weights
  - Apply incoherence processing (RHT) to transform weights into independent, Gaussian-like distributions, enabling effective trellis coding for i.i.d. sources.
  - Ensures robust quantization by minimizing correlations in large-scale LLM weights.
- Bitshift and Tail-Biting Trellis for Ultra-High-Dimensional Quantization
  - Introduce bitshift Trellis for lookup-free transitions (1 cycle) and Tail-biting Trellis with Algorithm 4 (two Viterbi calls) to enable tractable quantization of ultra-high-dimensional sequences (>100 dimensions).
  - Achieve bit efficiency and reduced distortion, supporting fast inference (4ms/token) on GPUs.
- Hybrid Lookup-Computed Code for Efficient Code Generation
  - Develop a hybrid code that generates pseudorandom indices with a small LUT (cached in GPU memory), eliminating full codebook storage (8 bytes → 0).
  - Balances speed and quality with post-quantization fine-tuning, enhancing overall performance.

Background (Contents)

1) Quantization

2) PTQ and QAT

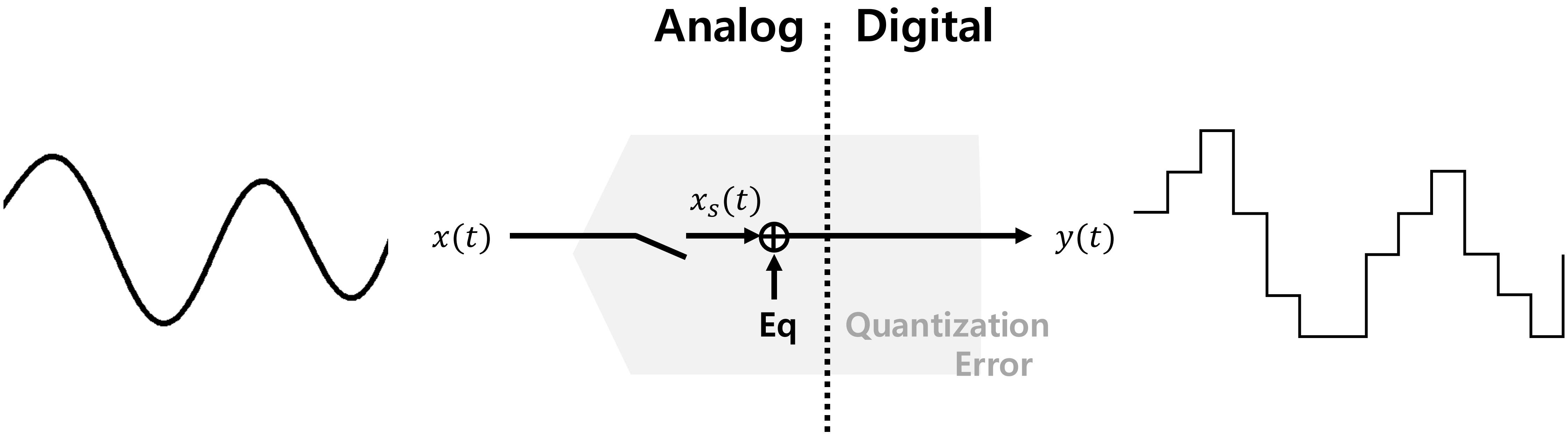
3) Scalar quantization and vector quantization

4) Incoherence processing

5) Trellis-coded quantization

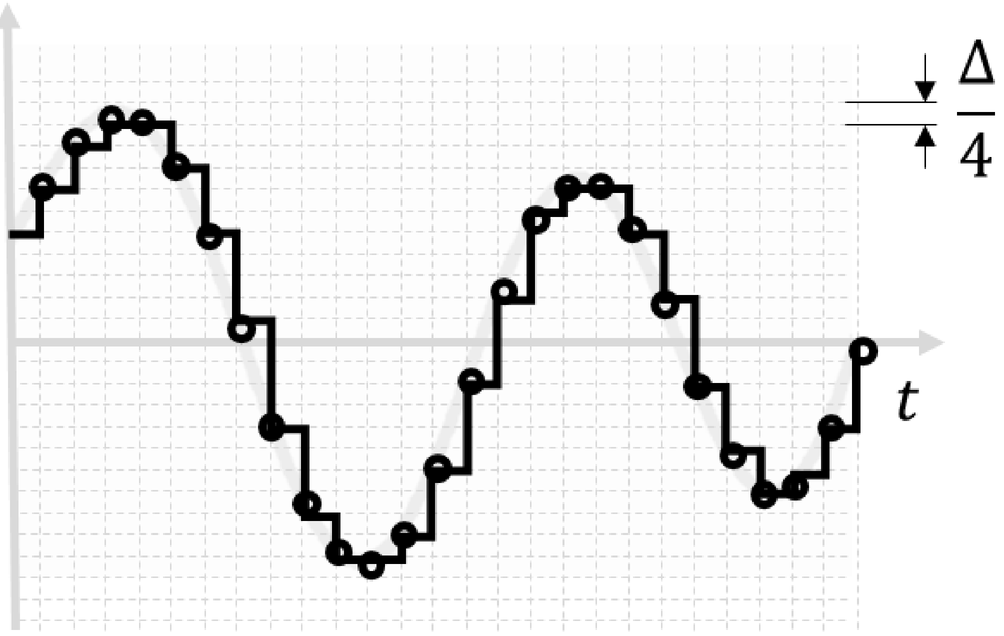
Background (Quantization)

- What is quantization?

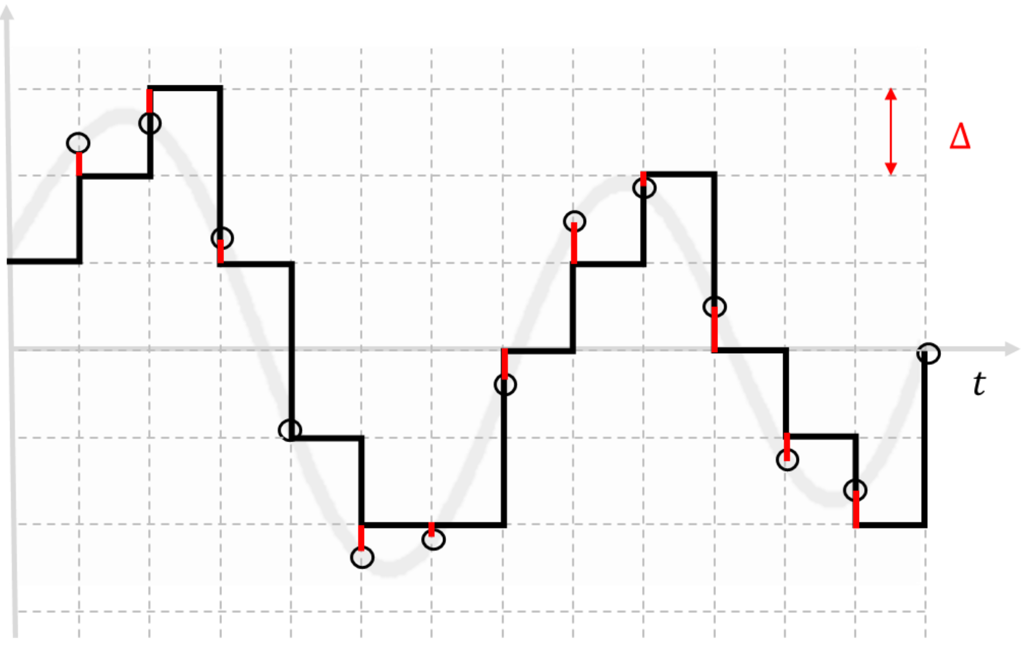


Background (Quantization)

- What is quantization?
  - More bits = higher accuracy
  - **In AI, quantization is reducing number of bit == less cost ➔ reducing accuracy ☹ (How to solve?)**



5bit quantization
4times accurate



3bit quantization

Background (PTQ and QAT)

- Post-training quantization (PTQ) and quantization aware training (QAT)

Criteria	**PTQ (Post-Training Quantization)**	**QAT (Quantization-Aware Training)**
Purpose	Quantize a pre-trained model witho ut retraining	Train the model with quantization const raints
Methodology	Apply quantization after training (e. g., weight scaling)	Simulate quantization during training (e. g., fake quant)
Advantages	- Fast and simple - No retraining needed	- Higher accuracy - Better handling of quantization errors
Disadvantages	- Potential accuracy drop - Limited flexibility	- Requires retraining - Higher computational cost
Use Case	- Quick deployment (e.g., QTIP)	- High-precision tasks

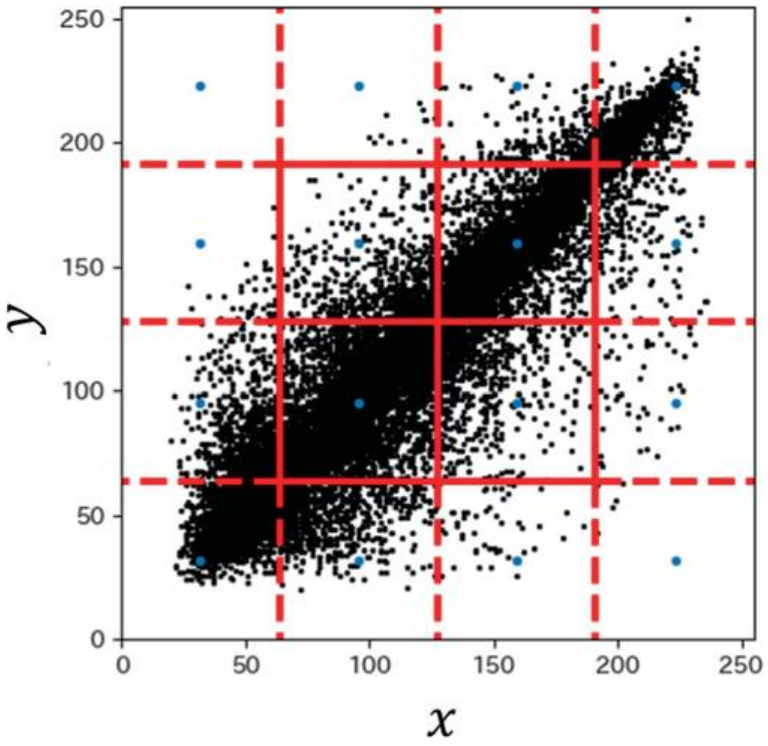
Background (SQ vs. VQ)

- Scalar quantization vs Vector quantization

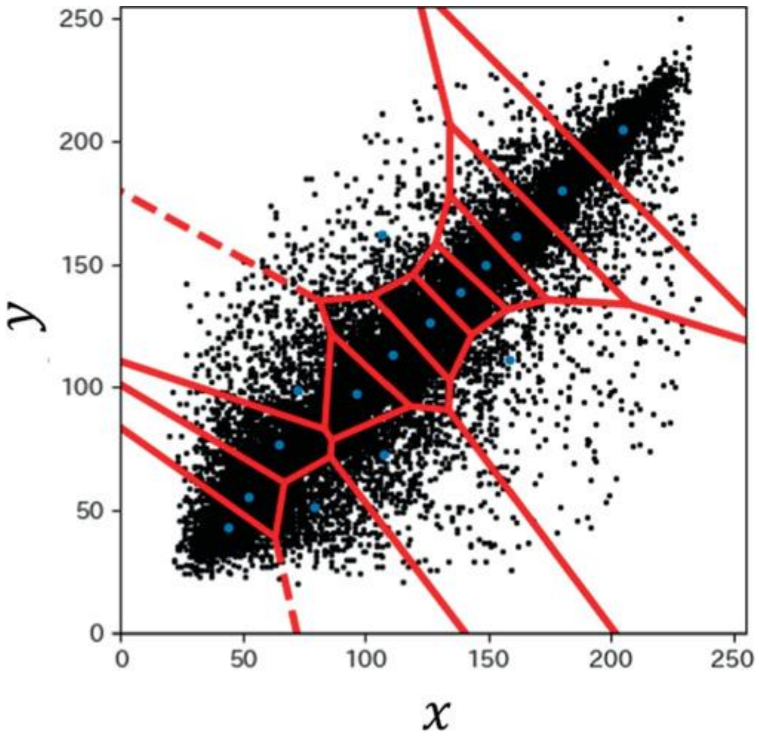
Criteria	Scalar Quantization (SQ)	Vector Quantization (VQ)
Definition	**Quantizes each element** (scalar) independently	**Quantizes groups of elements (vectors) together**
Granularity	Per element (e.g., each weight individually)	Per vector (e.g., group of weights as a single unit)
Complexity	Simple, low computational cost	More complex, higher computational cost
Memory Efficiency	Lower (stores more bits per element)	Higher (fewer bits per vector via codebook)
Quality	**Limited (ignores correlations between elements)**	**Better (captures correlations within vectors)**
Use Case	Basic compression (e.g., uniform quantization)	Advanced compression (e.g., QTIP's trellis coding)

Background (SQ vs. VQ)

- Scalar quantization vs Vector quantization



Scalar quantization



Vector quantization

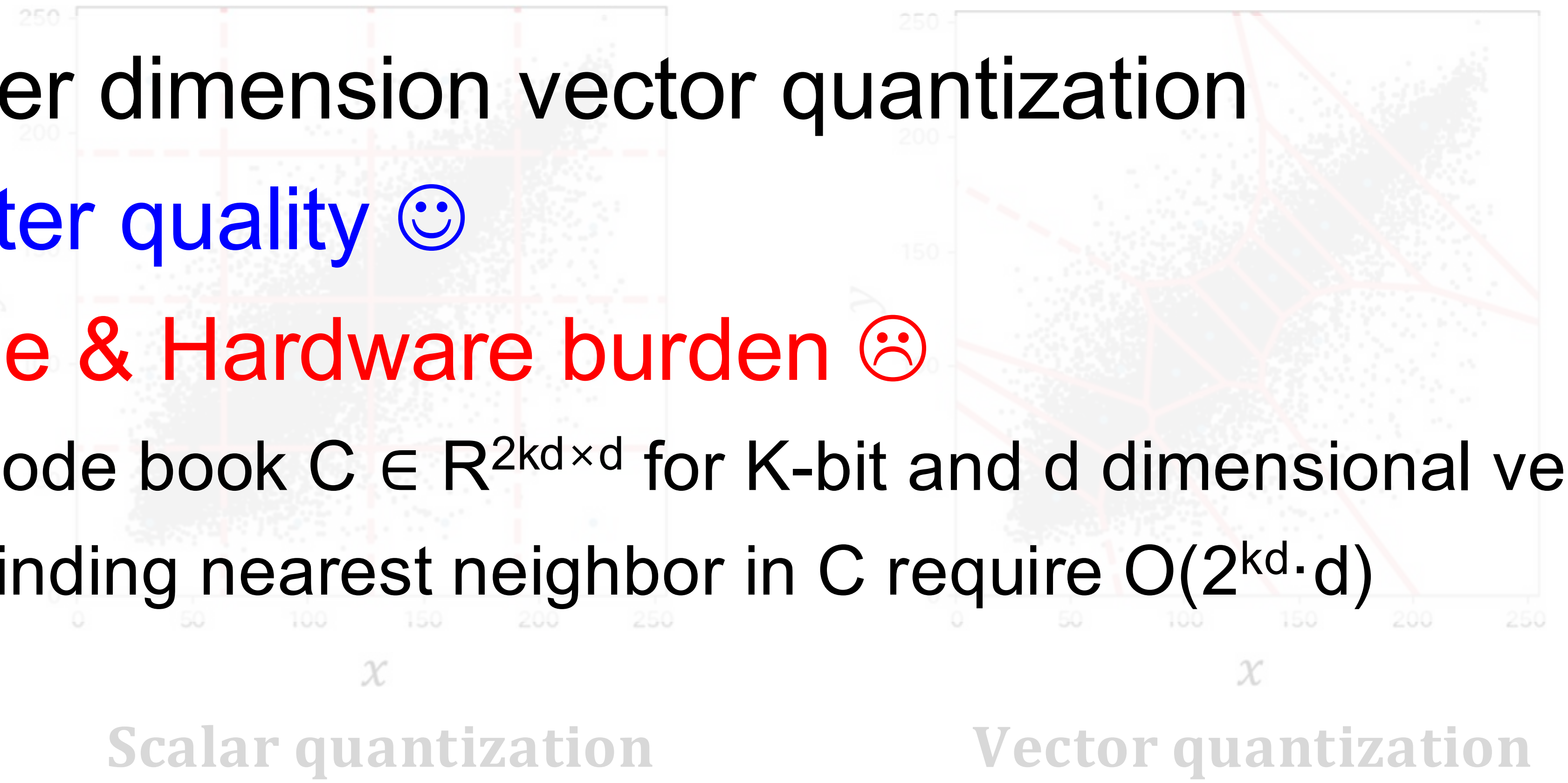
Reference: https://link.springer.com/chapter/10.1007/978-3-031-57840-3_40

Background (SQ vs. VQ)

- Scalar quantization vs Vector quantization

Higher dimension vector quantization

- Better quality ☺
- Time & Hardware burden ☹
  - Code book $C \in R^{2kd \times d}$ for K-bit and d dimensional vector S
  - Finding nearest neighbor in C require $O(2^{kd} \cdot d)$



Reference: https://link.springer.com/chapter/10.1007/978-3-031-57840-3_40

Related work (Incoherence processing)

- What is incoherence processing ?
  - Transforms LLM weights into independent, Gaussian-like distributions using Random Hadamard Transform (RHT).
  - Reduces correlations between weights, enabling effective trellis coding.

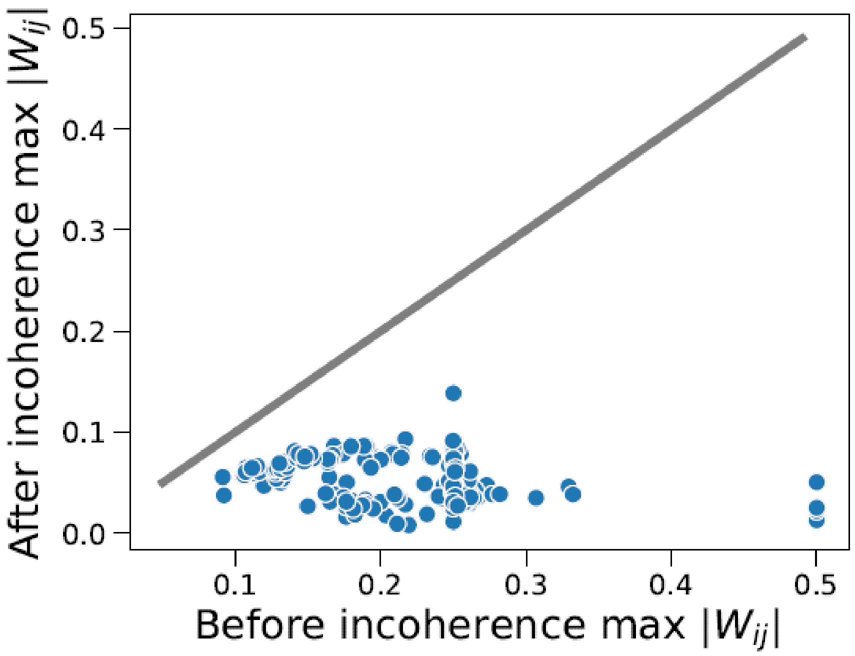
Definition 2.1 (Chee et al. [5]). A Hessian $H \in \mathbb{R}^{n \times n}$ is μ-incoherent if its eigendecomposition $H = Q\Lambda Q^T$ has $\max_{i,j} \ |Q_{ij}| = \max_{i,j} \ |e_i^T Q e_j| \leq \mu/\sqrt{n}$. A weight matrix $W \in \mathbb{R}^{m \times n}$ is μ-incoherent if $\max_{i,j} \ |W_{ij}| = \max_{i,j} \ \|e_i^T W e_j| \leq \mu \|W\|_F/\sqrt{mn}$.

- Why it matters in QTIP ?
  - Ensures i.i.d. Gaussian weights for high-quality quantization.
  - Supports ultra-high-dimensional (>100) quantization with minimal distortion.

Related work (Incoherence processing)

- What is incoherence processing ?
  - Transforms LLM weights into independent, Gaussian-like distributions using Random Hadamard Transform (RHT).
  - Reduces correlations between weights, enabling effective trellis coding.

- Why it r
  - Ens
  - Sup



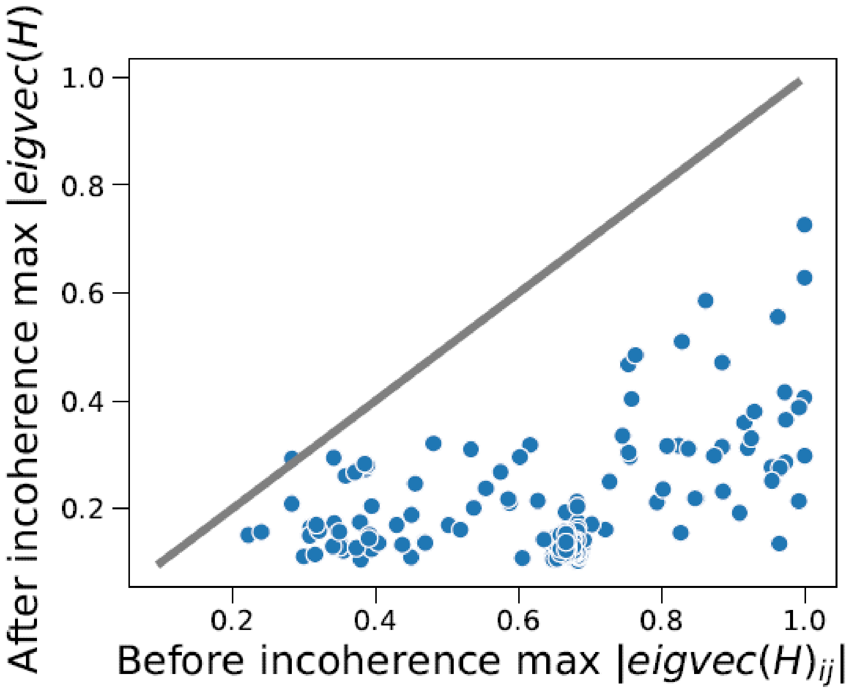


Figure Reference: QuIP, https://arxiv.org/abs/2307.13304

Related work (Incoherence processing)

- What is incoherence processing ?
  - Transforms LLM weights into independent, Gaussian-like distributions using Random Hadamard Transform (RHT).
  - Reduces correlations between weights, enabling effective trellis coding.

Definition 2.1 (Chee et al. [5]). A Hessian $H \in \mathbb{R}^{n \times n}$ is μ-incoherent if its eigendecomposition $H = Q \Lambda Q^T$ has $\max_{i,j} |Q_{ij}| = \max_{i,j} |e_i^T Q e_j| \leq \mu / \sqrt{n}$. A weight matrix $W \in \mathbb{R}^{m \times n}$ is μ-incoherent if $\max_{i,j} |W_{ij}| = \max_{i,j} \|e_i^T W e_j| \leq \mu \|W\|_F / \sqrt{mn}$.

- Why it matters in QTIP ?
  - **Ensures Gaussian [weights & eigenvector]** for high-quality quantization.
  - Supports **ultra-high-dimensional (>100) quantization** with minimal distortion.

Related work (TCQ)

- Trellis_Coded Quantization from Trellis code modulation (communication system)

Definition	Combines error-correcting coding with modulation to improve signal reliability over noisy channels
Key Features	- Uses a trellis structure to map coded bits to signal constellations - Increases coding gain without bandwidth expansion - Improves error performance (e.g., 3-6 dB gain) - Widely used in modems, satellite communications
Advantages	- Enhanced error correction - Efficient use of signal space - Robust against channel noise
Challenges	Scalability Issue : Infeasible for large-scale LLMs (e.g., Llama 3 405B, 810GB). Hardware Inefficiency : Lookup-heavy operations are not GPU-friendly, limiting parallelism. **= Low inference speed**

Related work (TCQ)

- Trellis_Coded Quantization from Trellis code modulation (communication system)

Definition	Combines error-correcting coding with modulation to improve signal reliability over noisy channels
Key Features	- **Uses a trellis structure to map coded bits to signal constellations** - Increases coding gain without bandwidth expansion - Improves error performance (e.g., 3-6 dB gain) - Widely used in modems, satellite communications
Advantages	- Enhanced error correction - Efficient use of signal space - Robust against channel noise
Challenges	Scalability Issue : Infeasible for large-scale LLMs (e.g., Llama 3 405B, 810GB). Hardware Inefficiency : Lookup-heavy operations are not GPU-friendly, limiting parallelism. **= Low inference speed**

Related work (TCQ)

- Trellis_Coded Quantization from Trellis code modulation (communication system)

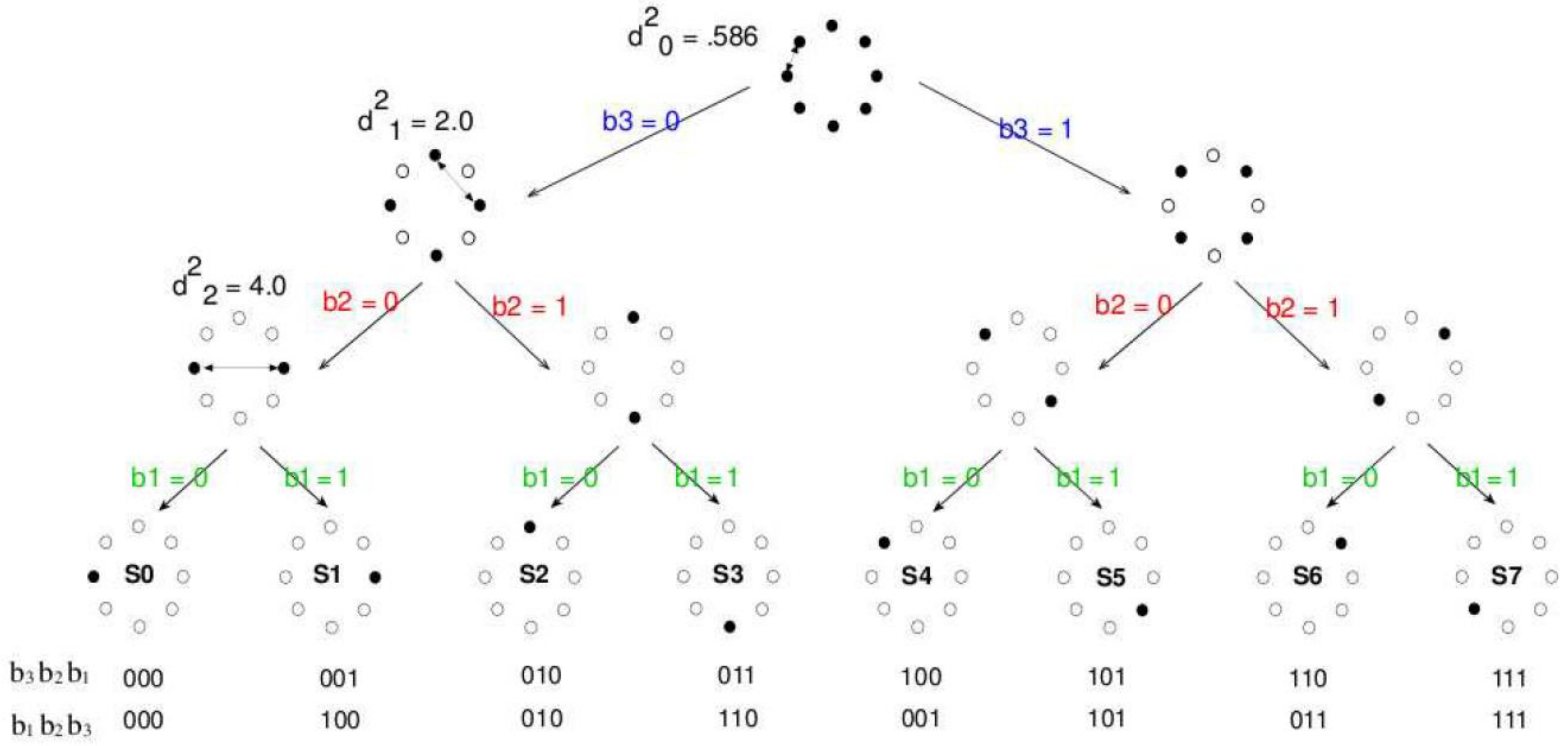
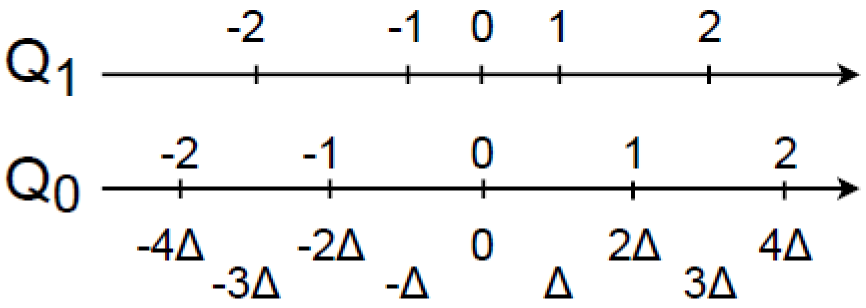


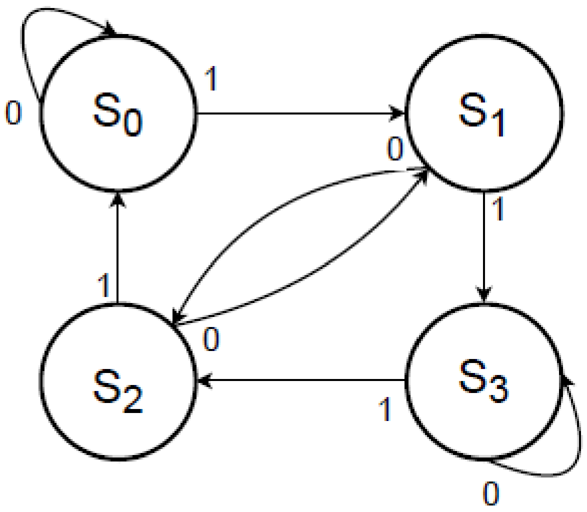
Figure Reference: Trellis Code Modulation
, https://www.telecom.tuc.gr/~liavas/Seminars/coding_theory/Docs/3_Trellis_Coded_Modulation.pdf

Related work (TCQ)

- Trellis_Coded Quantization (TCQ)



(a) Design of quantization intervals of quantizers Q_0 and Q_1



(b) State machine with four states for switching between quantizers Q_0 and Q_1

Figure Reference: Trellis-Coded Quantization for End-to-End Learned Image Compression
, https://ieeexplore.ieee.org/document/9897685

Related work (TCQ)

- Trellis-coded Quantization from Trellis code modulation (communication system)

Definition	Combines error-correcting coding with modulation to improve signal reliability over noisy channels
Key Features	- Uses a trellis structure to map coded bits to signal constellations - Increases coding gain without bandwidth expansion - Improves error performance (e.g., 3-6 dB gain) - Widely used in modems, satellite communications
Advantages	- Enhanced error correction - Efficient use of signal space - Robust against channel noise
Challenges	**Scalability Issue (Hugh book)** : Infeasible for large-scale LLMs (e.g., Llama 3 405B, 810GB). **Hardware Inefficiency (Slow)** : Lookup-heavy operations are not GPU-friendly, limiting parallelism. **= Low inference speed**

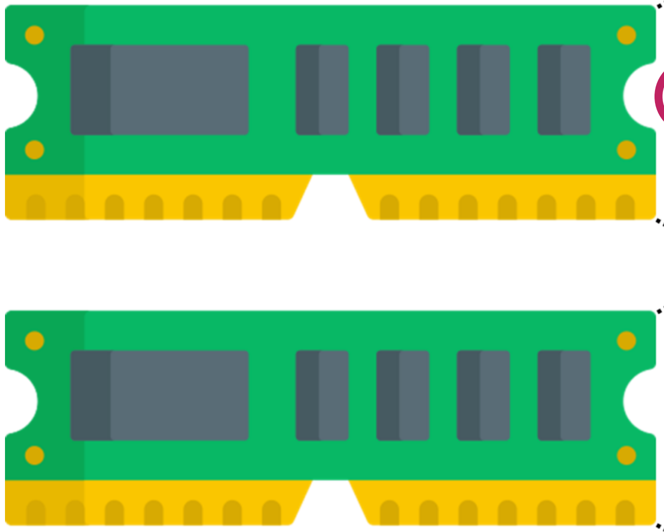
Problem Formulation

- Vector Quantization (VQ)
  - Hard to support high-dimension. → high quantization distortion.
  - Need to store codebook. → cache miss.
- Trellis Coded Quantization (TCQ)
  - Need to store trellis. → cache miss.
  - Need to store codebook. → cache miss.
  - Sequential decoding. → low parallelism & low throughput.

- QTIP – weight only PTQ
  - Incoherence Processing
  - Bitshift Trellis
  - Compute-based Codes

Problem Formulation

- Vector Quantization (VQ)
  - Hard to support high-dimension. → high quantization distortion.
  - Need to store codebook. → cache miss.



1

2

idx	*Value*
0	0.1, 1.3, 4.2, 0.6
1	1.4, 0.3, 1.2, 2.2
2	1.1, 2.1, 0.2, 1.1
3	1.4, 0.1, 1.1, 1.2
...	...
$2^{kd}-4$	0.1, 1.3, 4.2, 0.6
$2^{kd}-3$	1.4, 0.3, 1.2, 2.2
$2^{kd}-2$	0.1, 0.2, 0.3, 0.4
$2^{kd}-1$	2.1, 0.8, 2.5, 0.1

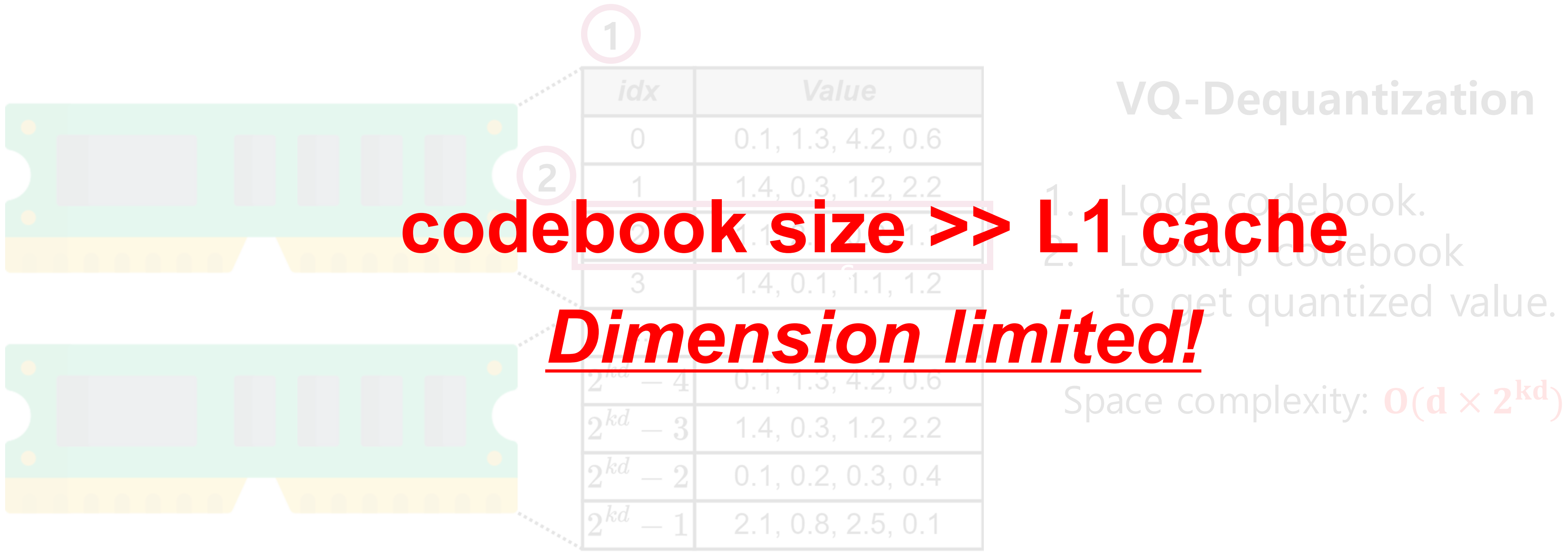
VQ-Dequantization

1. Lode codebook.
2. Lookup codebook to get quantized value.

Space complexity: $\mathbf{O(d \times 2^{kd})}$

Problem Formulation

- Vector Quantization (VQ)
  - Hard to support high-dimension. → high quantization distortion.
  - Need to store codebook. → cache miss.

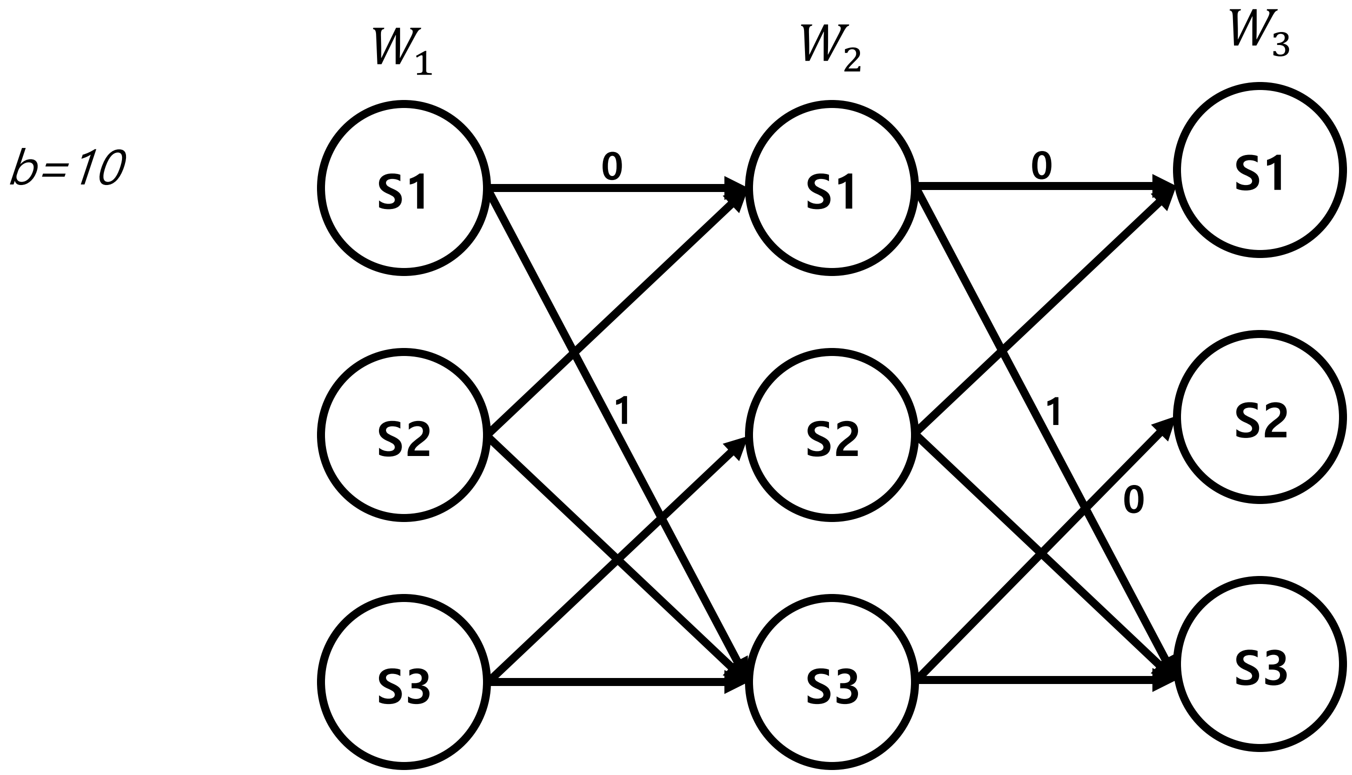


codebook size >> L1 cache

Dimension limited!

Problem Formulation

- Trellis Coded Quantization (TCQ)
  - Need to store trellis. → cache miss.
  - Need to store codebook. → cache miss.
  - Sequential decoding. → low parallelism & low throughput.



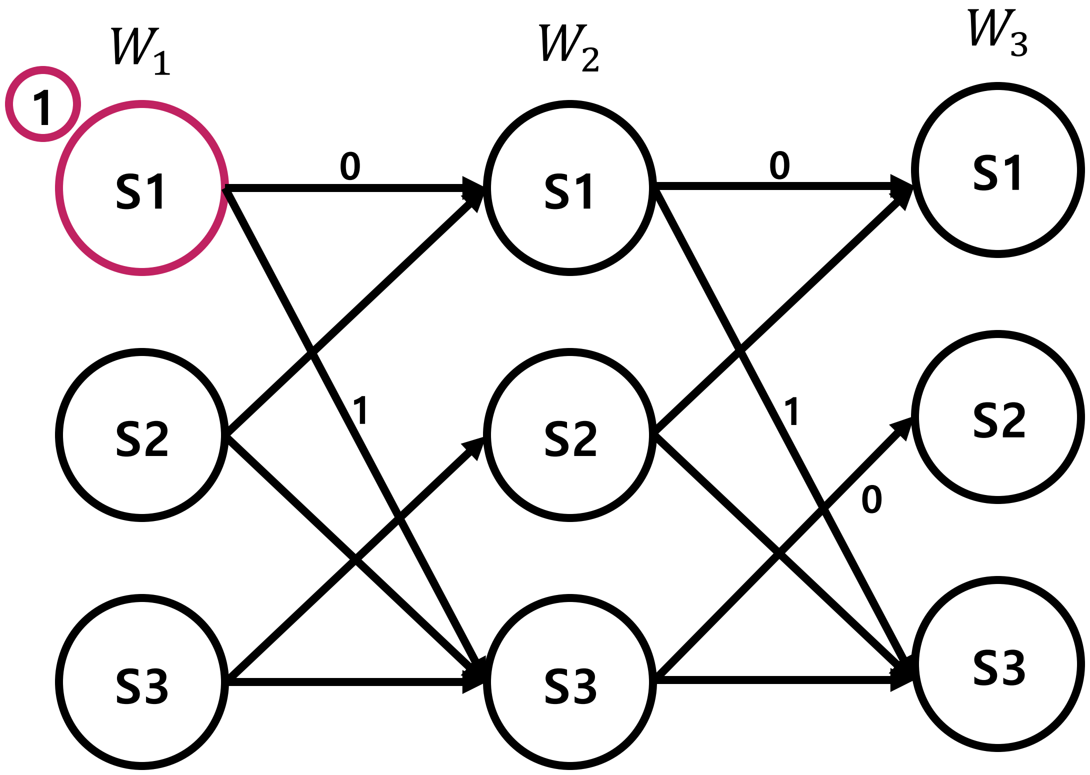
TCQ-Dequantization

Trellis walking!
But sequential decoding…

Problem Formulation

- Trellis Coded Quantization (TCQ)
  - Need to store trellis. → cache miss.
  - Need to store codebook. → cache miss.
  - Sequential decoding. → low parallelism & low throughput.

b=10

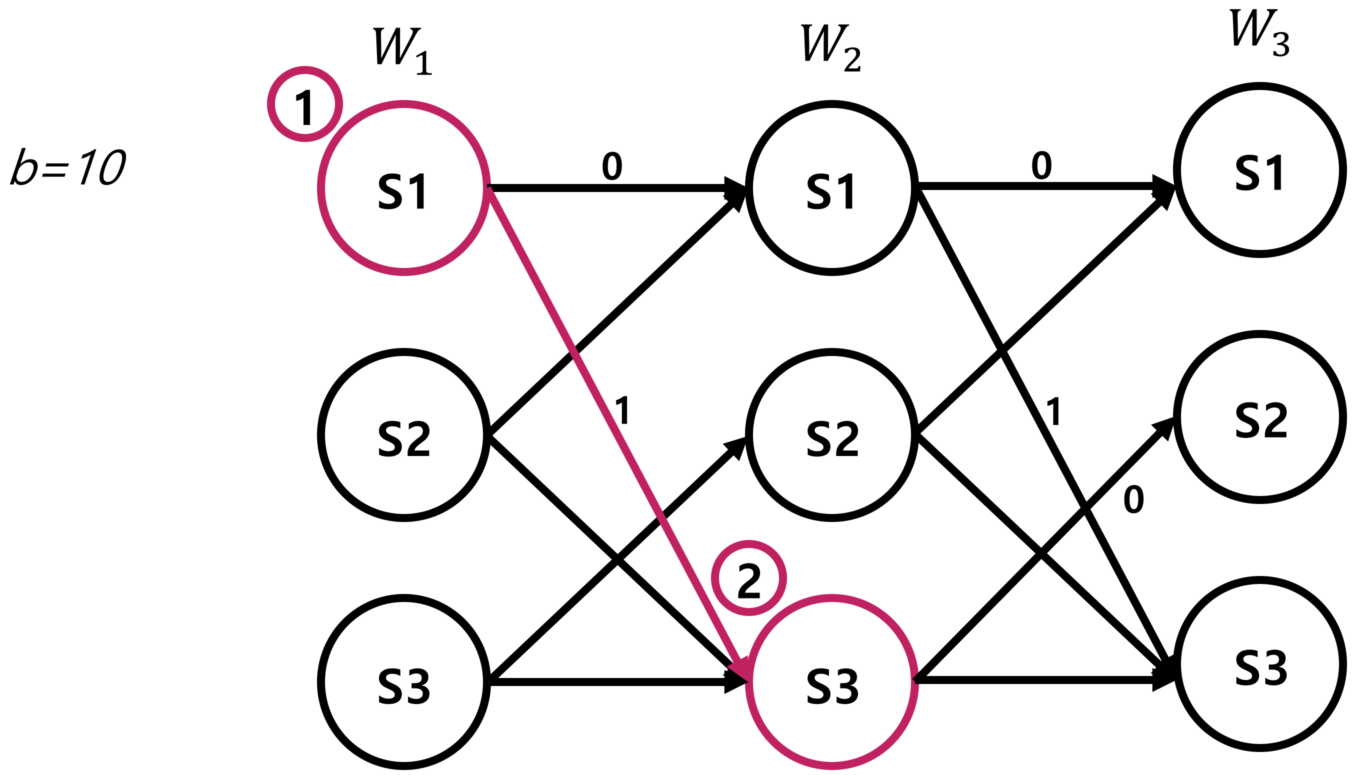


TCQ-Dequantization

Trellis walking!
But sequential decoding…

Problem Formulation

- Trellis Coded Quantization (TCQ)
  - Need to store trellis. → cache miss.
  - Need to store codebook. → cache miss.
  - Sequential decoding. → low parallelism & low throughput.

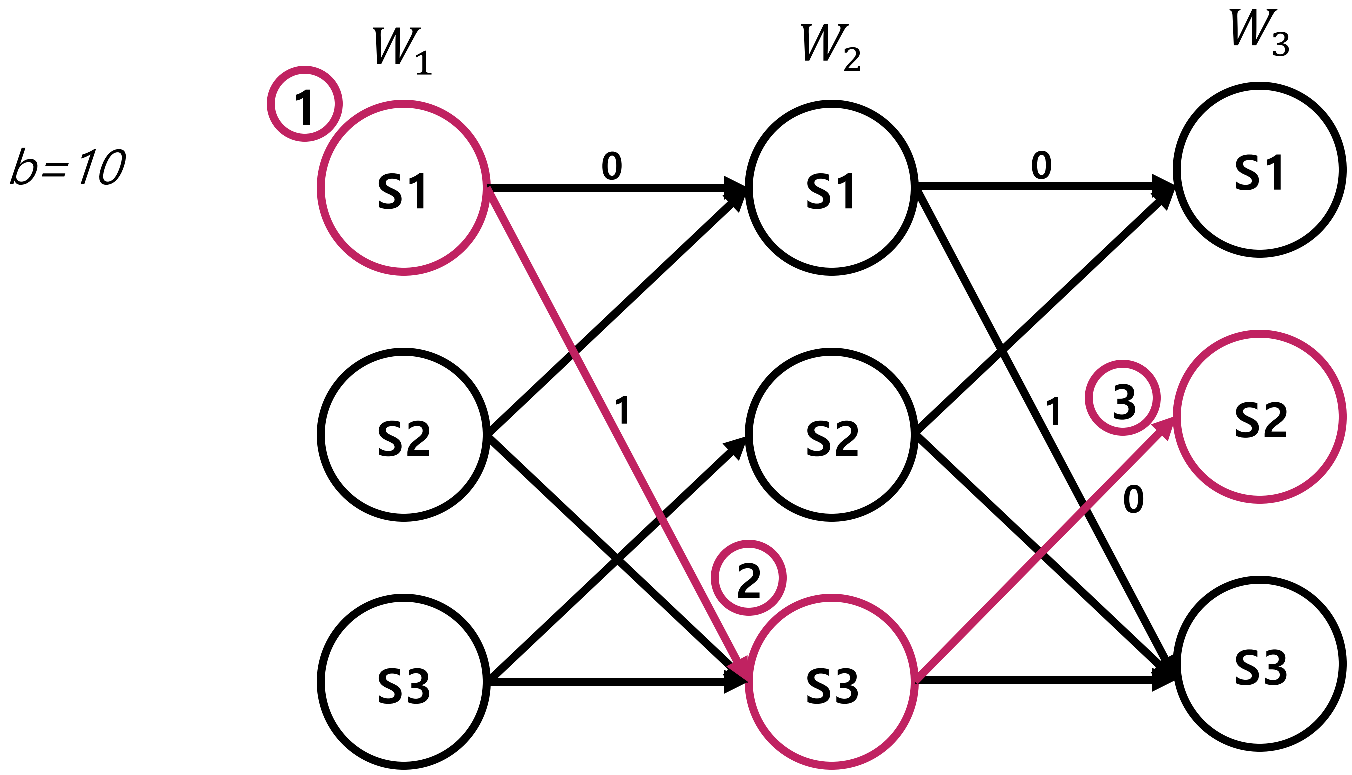


TCQ-Dequantization

Trellis walking!
But sequential decoding...

Problem Formulation

- Trellis Coded Quantization (TCQ)
  - Need to store trellis. → cache miss.
  - Need to store codebook. → cache miss.
  - Sequential decoding. → low parallelism & low throughput.



TCQ-Dequantization

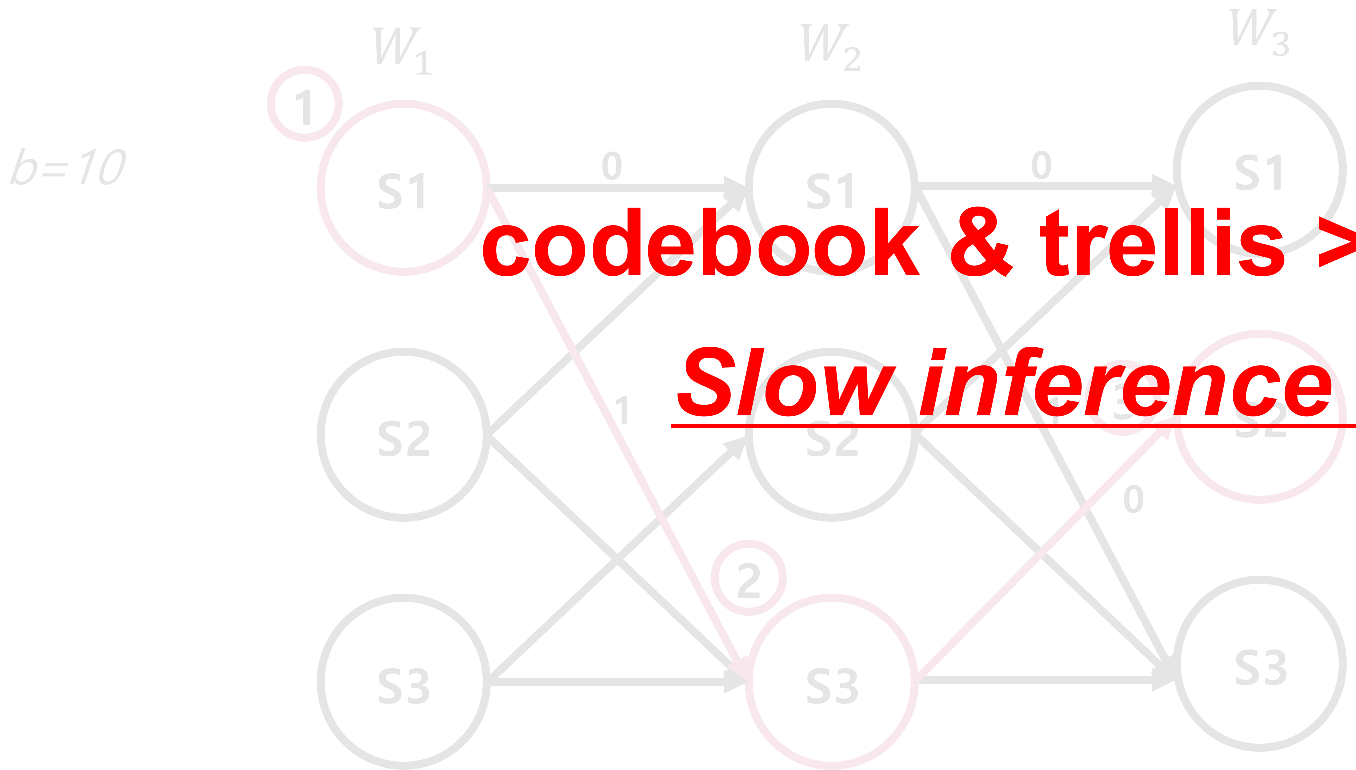
Trellis walking!
But sequential decoding...

Also, require lookup codebook!

$W_1 = codebook[S1]$
$W_2 = codebook[S3]$
$W_3 = codebook[S2]$

Problem Formulation

- Trellis Coded Quantization (TCQ)
  - Need to store trellis. → cache miss.
  - Need to store codebook. → cache miss.
  - Sequential decoding. → low parallelism & low throughput.



TCQ-Dequantization

Trellis walking!

But sequential decoding...

Also, require lookup codebook!

$W_1 = codebook[S1]$
$W_2 = codebook[S3]$
$W_3 = codebook[S2]$

codebook & trellis >> L1 cache

Slow inference speed!

QTIP

How to support parallel decoding with high dimension.

+ low quantization distortion & w/o codebook

QTIP

How to support parallel decoding with high dimension.

+ low quantization distortion & w/o codebook

1. ***Incoherence Processing***
2. ***Bitshift Trellis***
3. ***Compute-based Codes***

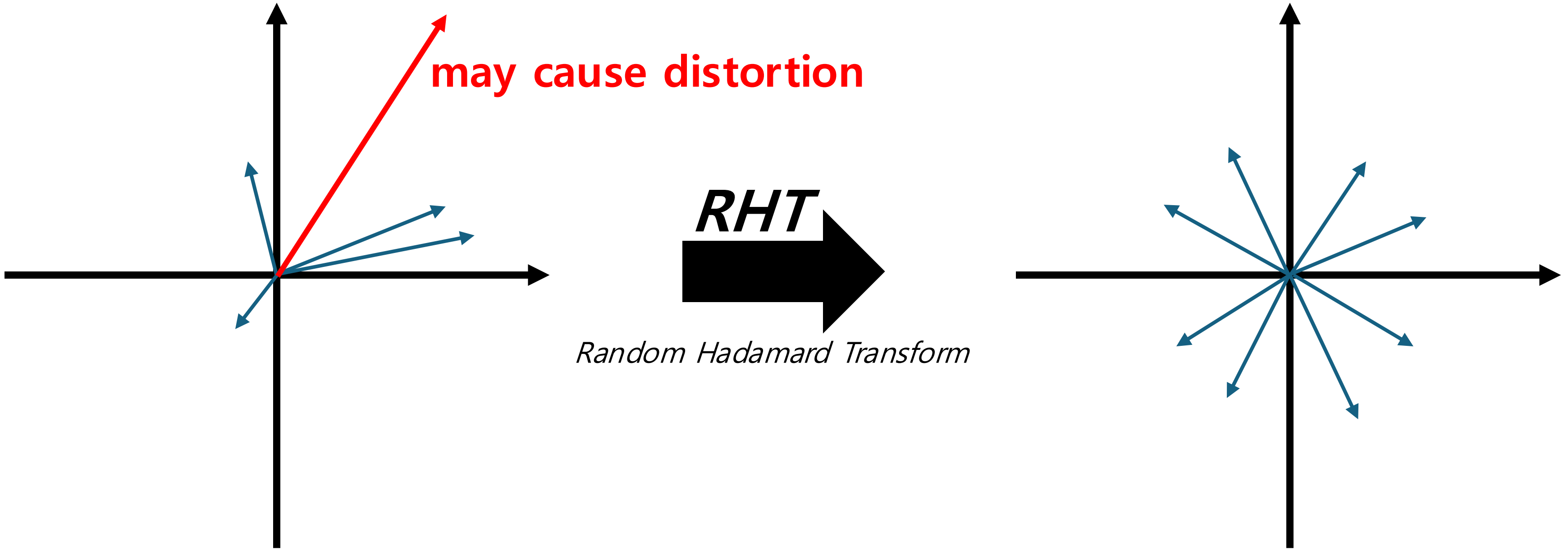
Proposed Method

- Incoherence Processing
  - outliers cause poor quantization quality.
  - weights and Hessian eigenvectors are not too large in any direction, aiding quantization.

  → *enables good shaping.*

high coherent
(most weights align in few dominant directions)

pseudo-Gaussian
(weights are spread uniformly in all directions)



Proposed Method

- Incoherence Processing
  - outliers cause poor quantization quality.
  - weights and Hessian eigenvectors are not too large in any direction, aiding quantization.

  → *enables good shaping.*

Low distortion ☺

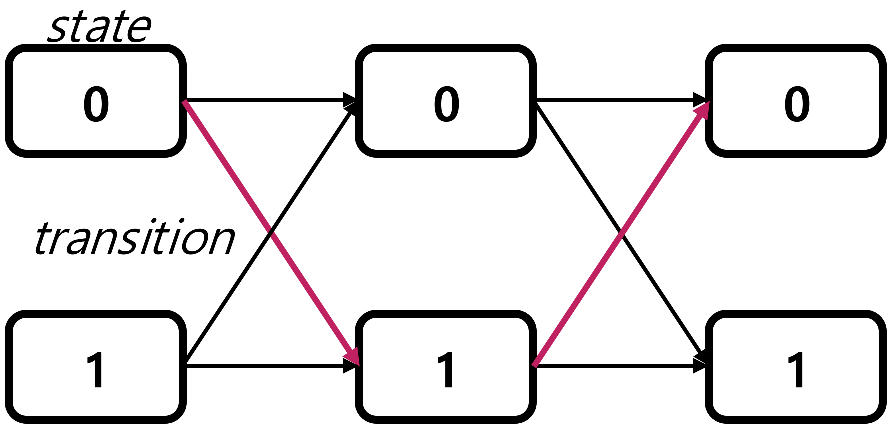
Better shaping ☺

Proposed Method

- Bitshift Trellis
  - TCQ requires sequential decoding, limiting parallelism.
  - Using Viterbi algorithm.

  → *enables parallel decoding.*

- Viterbi algorithm
  - Finding the most likely sequence(minimize metric) of hidden states.
  - Used in communication systems, such as error correction codes etc.

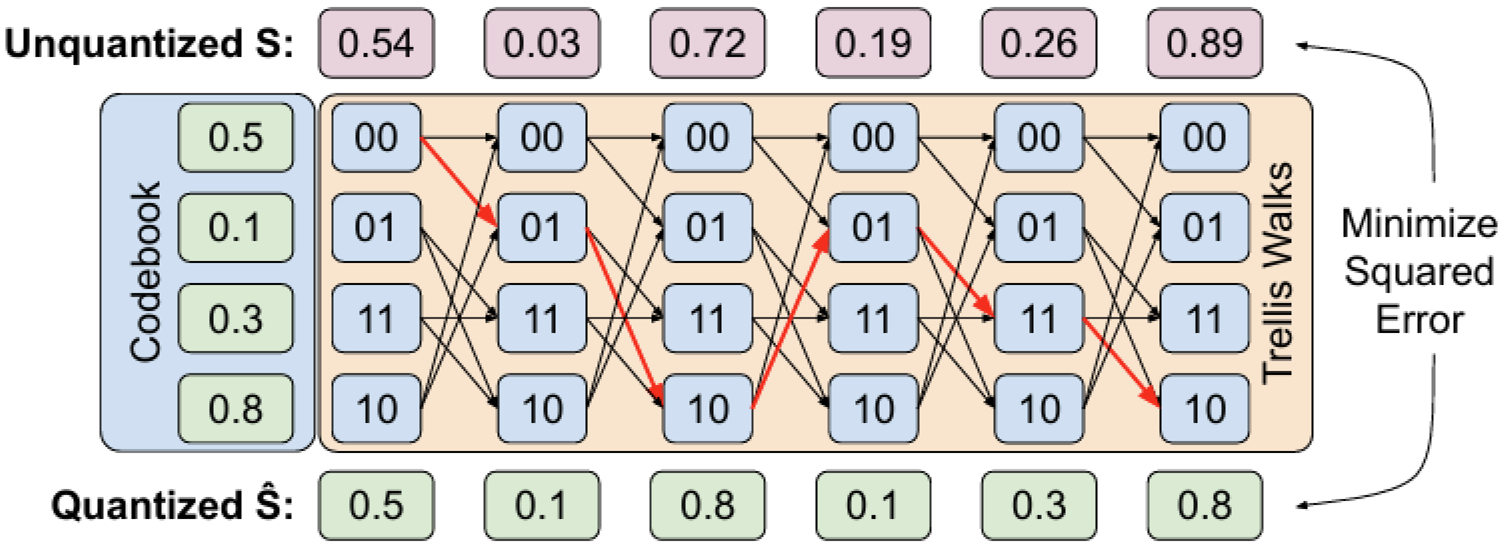


1. **Recursive step**
   calculate metric (MSE) for all possible path.
2. **Storage of back pointers**
   store previous path's pointer.
3. **Traceback**
   find optimal path which make best metric.

Proposed Method

- Bitshift Trellis **Encoding**
  - $2^L\ nodes\ with\ j\ edges. j = \left(i2^{kV} mod 2^L\right) + c, 0 \le c < 2^{kV}.$
  - *top L-kV bits of node j equal the bottom L-kV bits of node i. If node i → node j.*
  - *where, L = log2(# of codebook), k = bitrate, V = # of weight in groups.*

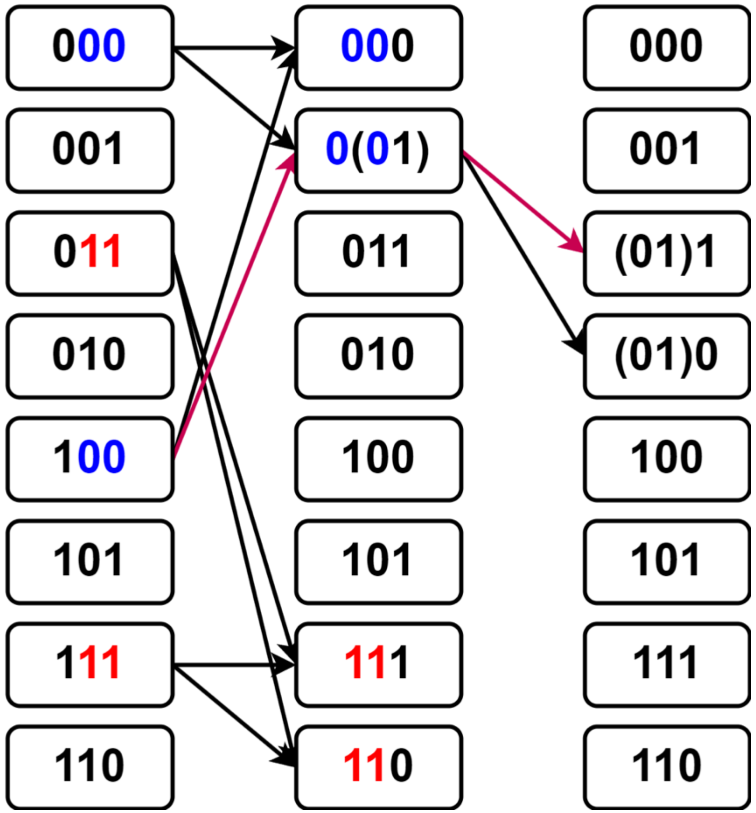
$$L = 2, k = 1, V = 1$$



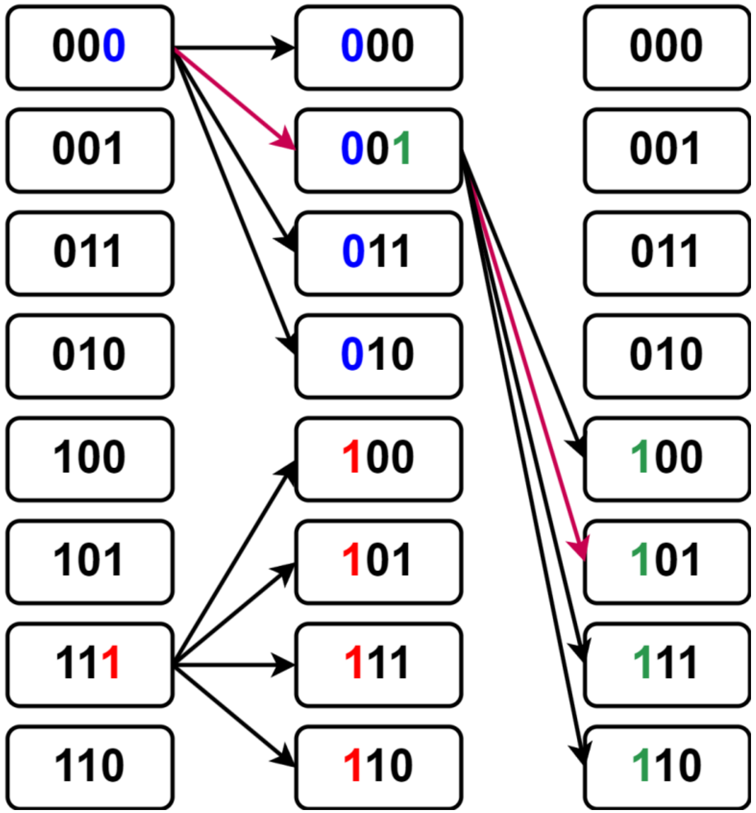
Proposed Method

- Bitshift Trellis **Encoding**
  - 2^L *nodes with j edges.* $j = \left(i2^{kV} mod 2^L\right) + c, 0 \leq c < 2^{kV}$.
  - *top L-kV bits of node j equal the bottom L-kV bits of node i. If node i → node j.*

$L = 3, k = 1, V = 1$



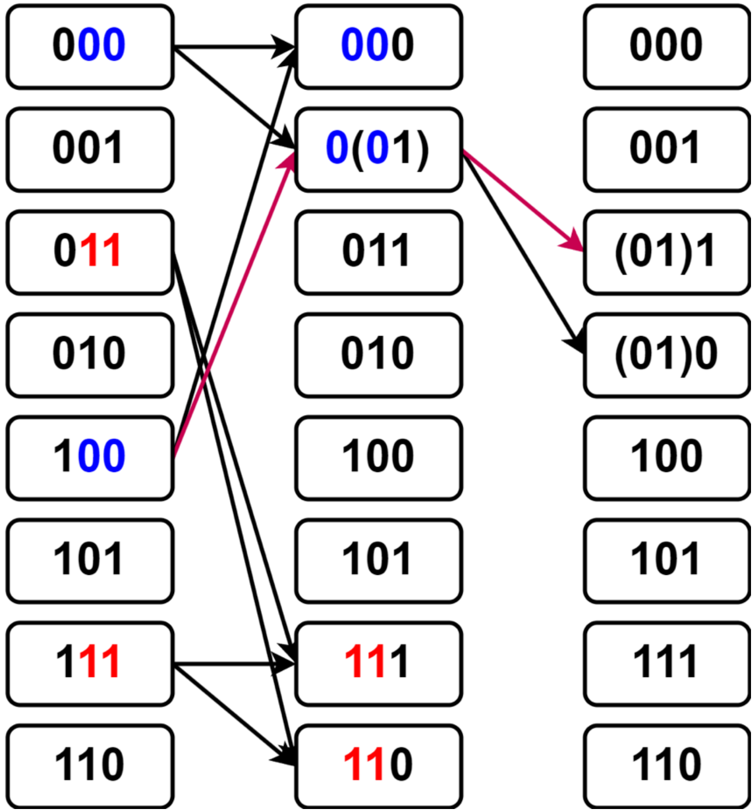
$L = 3, k = 2, V = 1$



Proposed Method

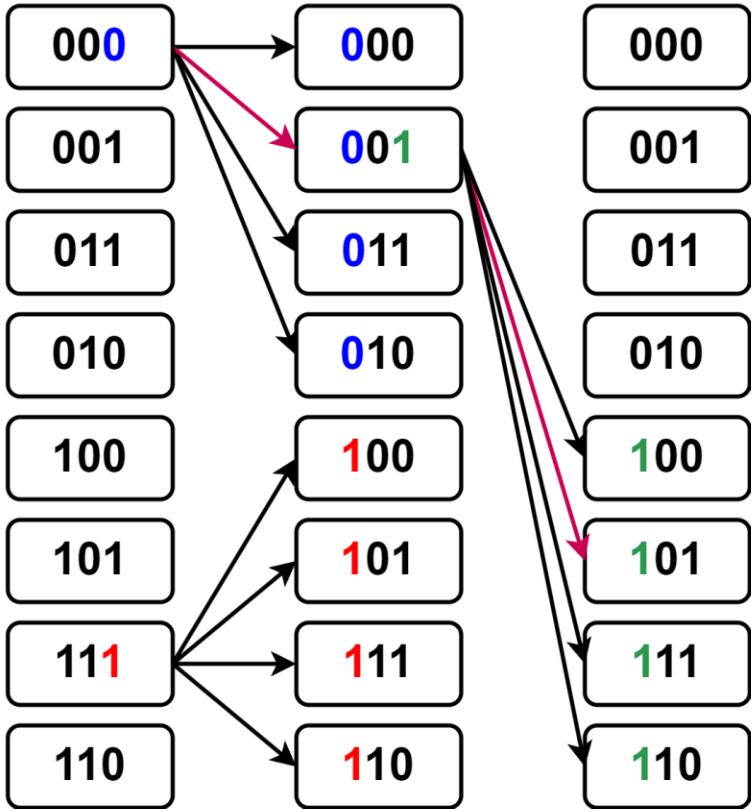
- Bitshift Trellis **Encoding**

$L = 3, k = 1, V = 1$



$S = 10011 \rightarrow k = 1.67$

$L = 3, k = 2, V = 1$

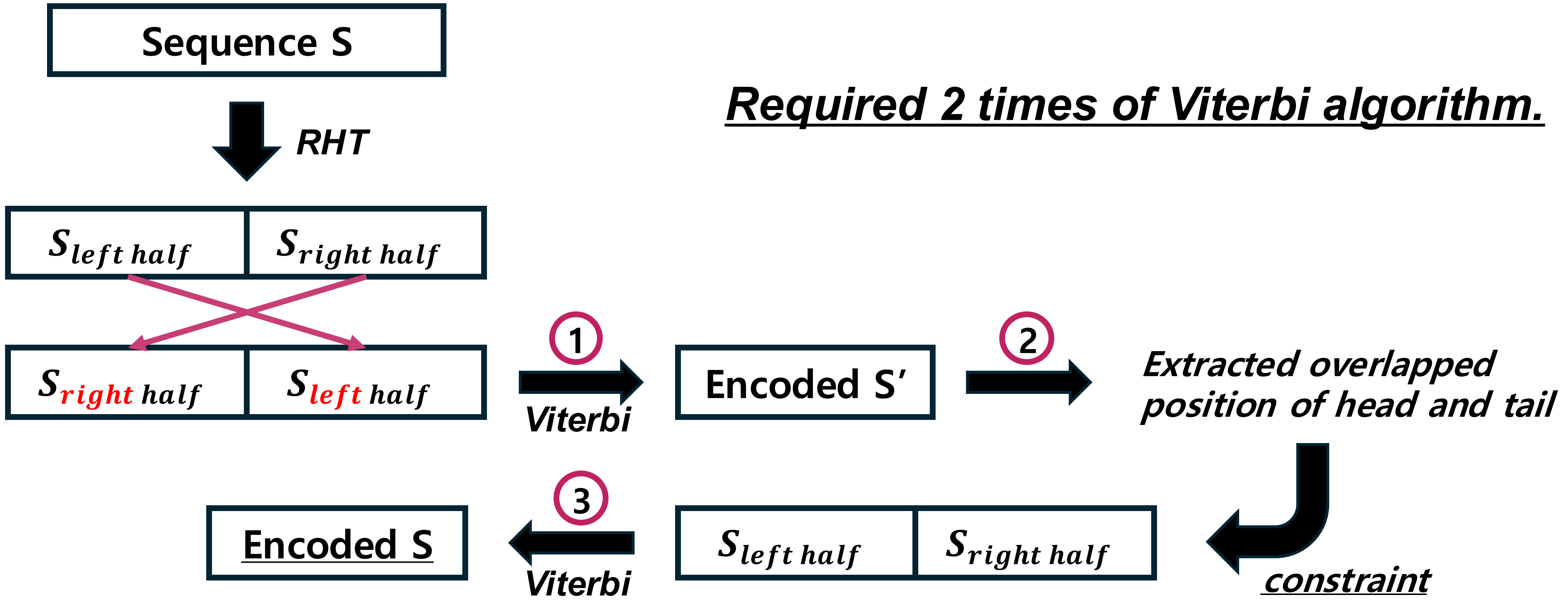


$S = 0000101 \rightarrow k = 2.33$

Proposed Method

- Bitshift Trellis **Encoding with *tail-biting***
  - Force head and tail to have the same bit. → near-optimal.

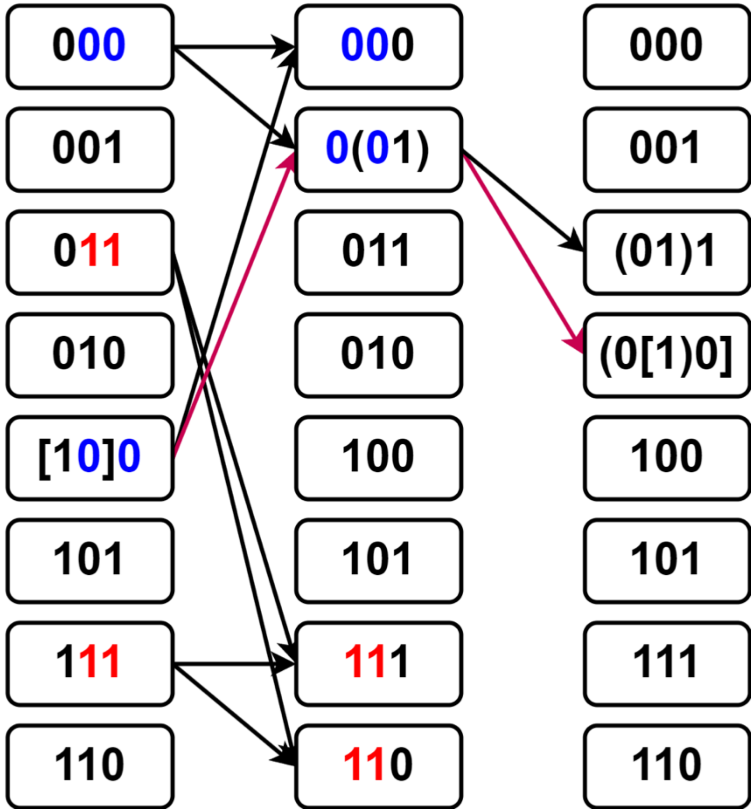
Required 2 times of Viterbi algorithm.



Proposed Method

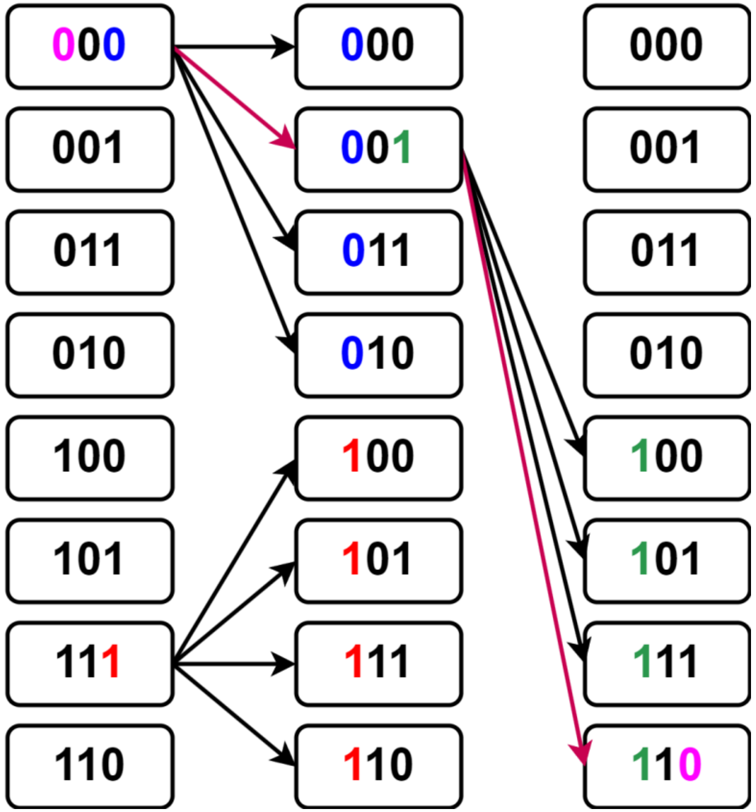
- Bitshift Trellis **Encoding with *tail-biting***

$L = 3, k = 1, V = 1$



$S = 100 \rightarrow k = 1$

$L = 3, k = 2, V = 1$



$S = 000011 \rightarrow k = 2$

Proposed Method

- Bitshift Trellis **Encoding with *tail-biting***

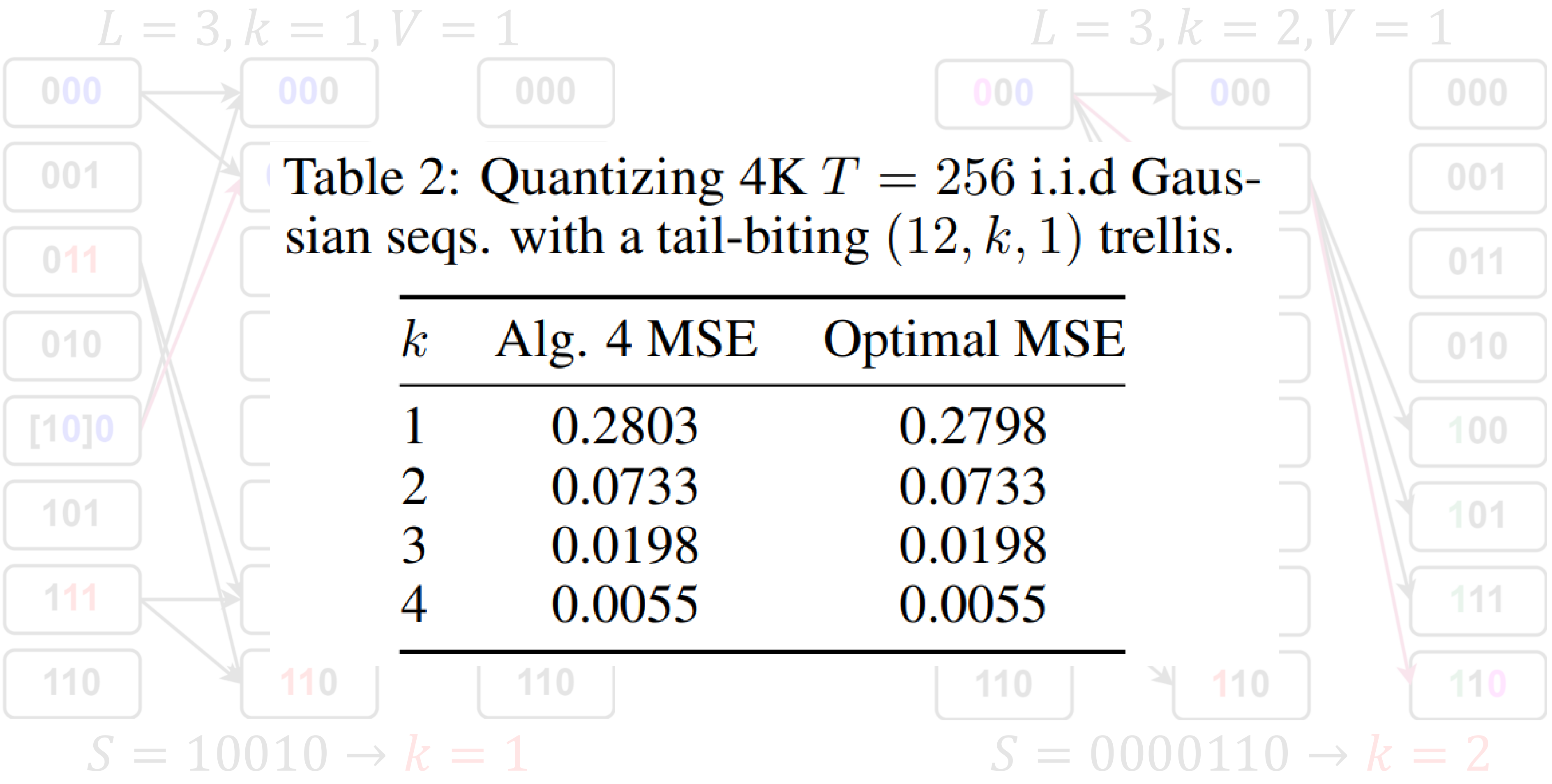


Table 2: Quantizing 4K $T = 256$ i.i.d Gaussian seqs. with a tail-biting $(12, k, 1)$ trellis.

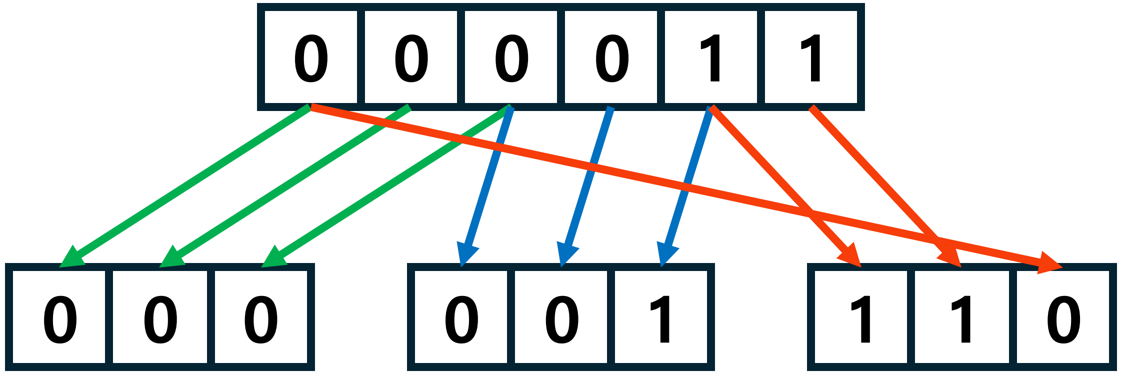
k	Alg. 4 MSE	Optimal MSE
1	0.2803	0.2798
2	0.0733	0.0733
3	0.0198	0.0198
4	0.0055	0.0055

Proposed Method

- Decoding of Bitshift Trellis
  - shifted-windowing of sequence.

$$L = 3, k = 2, V = 1$$

→1bit share



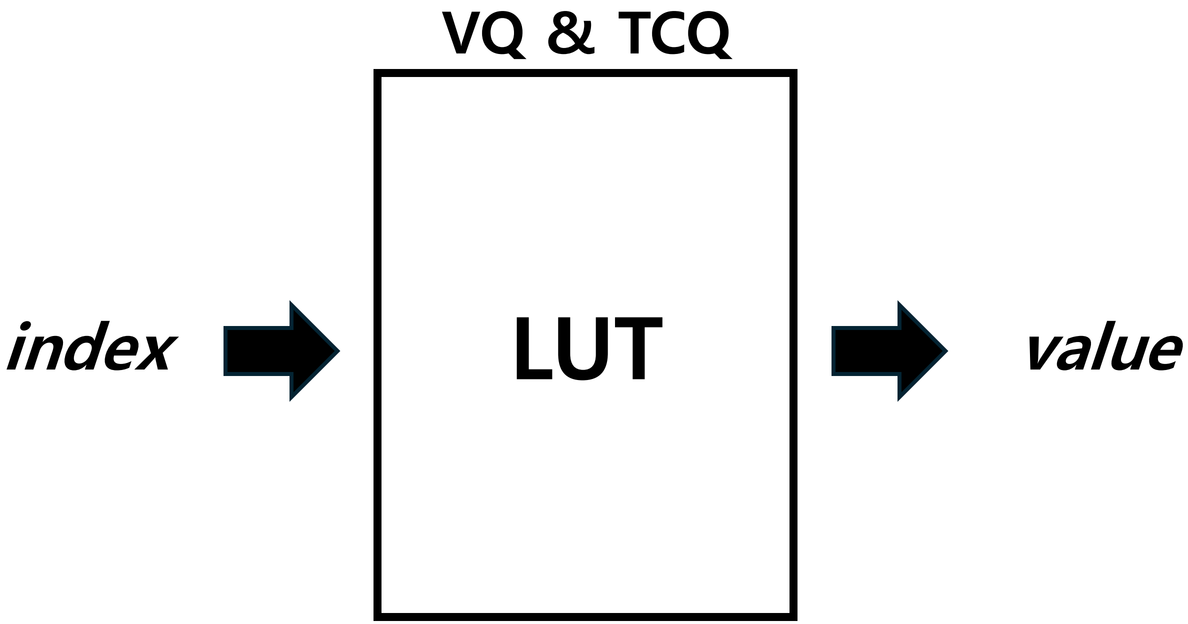
<u>But, this is not a true quantized value.</u>

<u>Need to convert to float value.</u>

Proposed Method

- Lookup-Free Computed Codes
  - 1MAD, 3INST
  - generate pseudorandom approximately Gaussian number from a L bit word.

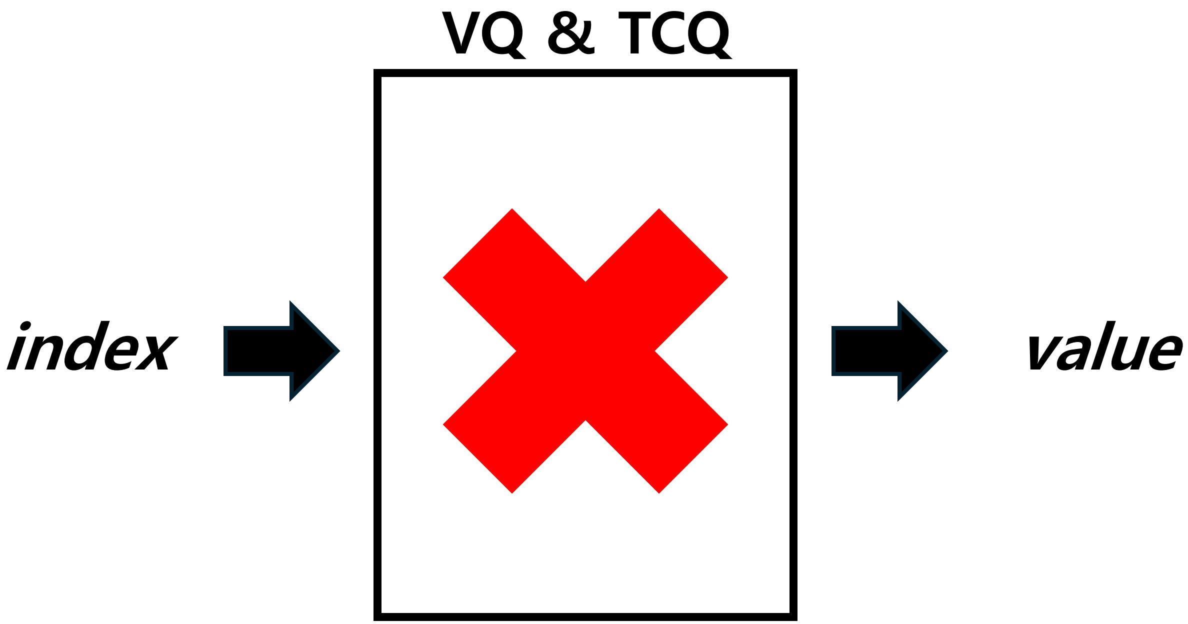
  → *enable fast decoding on cache-limited hardware.*



Proposed Method

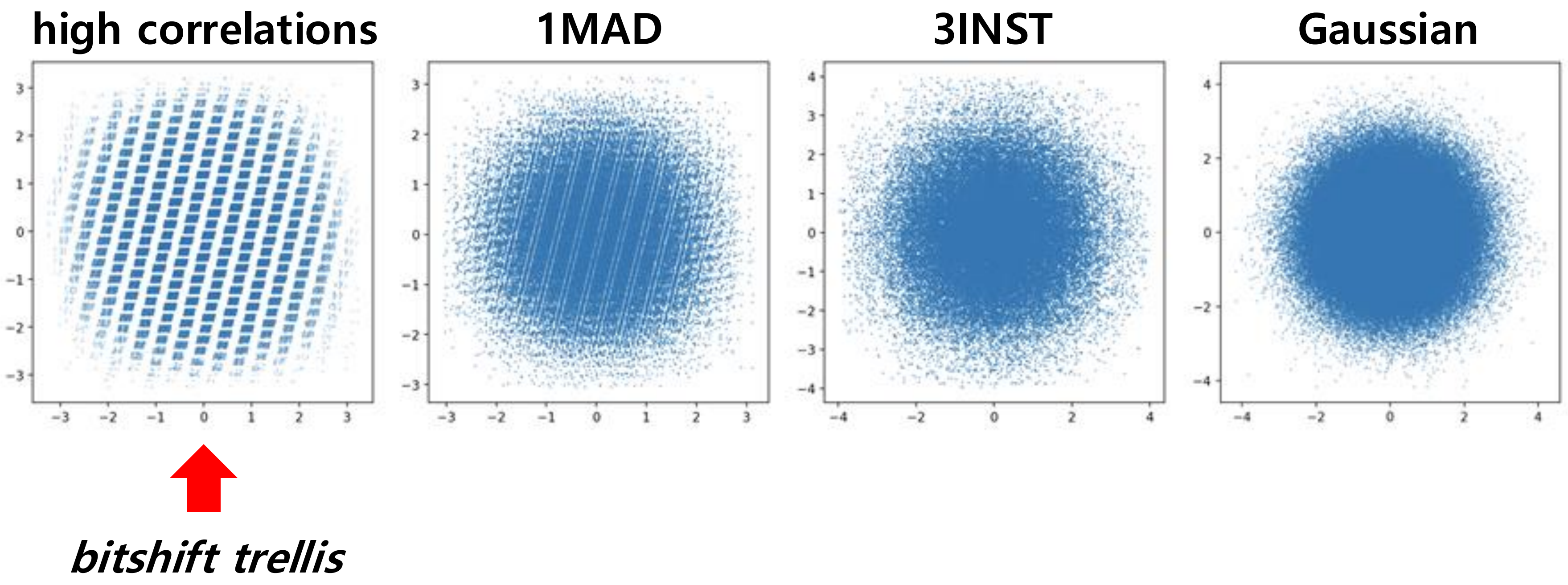
- Lookup-Free Computed Codes
  - 1MAD, 3INST
  - generate pseudorandom approximately Gaussian number from a L bit word.

  → *enable fast decoding on cache-limited hardware.*



Proposed Method

- Lookup-Free Computed Codes
  - 1MAD, 3INST
  - generate pseudorandom approximately Gaussian number from a L bit word.
  - → *enable fast decoding on cache-limited hardware.*



Proposed Method

- Lookup-Free Computed Codes
  - 1MAD, 3INST

Algorithm 1 Computed Gaussian Code "1MAD"

input L-bit 0 left-padded integer x, `uint32` a, b.

$x \leftarrow (ax + b) \bmod 2^{32}$ {run LCG to get uniform random x}

{sum x as four 8-bit unsigned integers, this is approximately Gaussian}

$x \leftarrow (x \texttt{ \& } 255) + ((x \texttt{ >> } 8) \texttt{ \& } 255) + ((x \texttt{ >> } 16) \texttt{ \& } 255) + ((x \texttt{ >> } 24) \texttt{ \& } 255)$

$x \leftarrow (x - 510)/147.8$

output Pseudorandom approximate Gaussian x.

Algorithm 2 Computed Gaussian Code "3INST"

input L-bit 0 left-padded integer x, `uint32` a, b, `float16` m.

$x \leftarrow (ax + b) \bmod 2^{32}$ {run LCG to get uniform random x}

{modify sign, mantissa, and bottom 2 exponent bits of m and sum, this is approximately Gaussian}

$m \leftarrow \texttt{reinterpret}(m, \texttt{uint32}) \texttt{ << } 16 + \texttt{reinterpret}(\mathrm{m}, \texttt{uint32})$

$x \leftarrow (x \texttt{ \& b10001111111111111000111111111111}) \texttt{ XOR } m$

$x \leftarrow \texttt{reinterpret}(x \texttt{ \& } 2^{16} - 1, \texttt{float16}) + \texttt{reinterpret}((x \texttt{ >> } 16) \texttt{ \& } 2^{16} - 1, \texttt{float16})$

output Pseudorandom approximate Gaussian x.

Proposed Method

- Lookup Computed Codes
  - HYB

Algorithm 3 Hybrid Computed-Lookup 2D Gaussian Code "HYB"

input L-bit 0 left-padded integer x, codebook $C \in \mathbb{R}^{2^Q \times (V=2)}$.
  $x \leftarrow x \cdot x + x \bmod 2^{32}$ {calculate hash}
  $v \in \mathbb{R}^2 \leftarrow C[(x \texttt{ >> } (15 - Q)) \texttt{ \& } 2^Q - 1]$ {lookup from symmetric codebook}
  $v \leftarrow v \texttt{ XOR } (x \texttt{ \& } (1 \texttt{ << } 15))$ {apply sign flip}
output Pseudorandom approximate Gaussian vector v.

- Computed Codes Results

	SQ	VQ	1D TCQ			2D TCQ		
QUANT.	LLOYD-MAX	QUIP# E8P	1MAD	3INST	RPTC	HYB	RPTC	D_R
DIM.	1	8	256	256	256	256	256	∞
MSE.	0.118	0.089	0.069	0.069	0.068	0.071	0.069	0.063

Experiments

Model

- Llama family (Llama 1, 2, 3)

VQ-based PTQ Methods

- QuIP#
- AQLM (Llama 2)
- GPTVQ-2D (Llama 1)

QTIP Methods

- Lookup-free computated codes
- Hybrid Lookup-computated codes

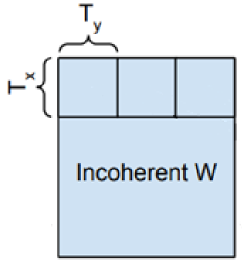
✓ <u>No fine-tuning for Lookup-free codes</u>, comparison with 2, 3, 4 quantization bit

Lookup-Free Computated Codes

QTIP Code Generation Algorithms
- 1MAD, 3INST

Perplexity (PPL)
- dataset : Wikitext2(W2), C4

Bitshift Trellis parameters
- L=16, V=1
- T_x=T_y=16 (16x16 tensor core MMA optimization



			4 Bit No FT			≈4 Bit		3 Bit No FT			≈3 Bit		2 Bit No FT			≈2 Bit	
		FP16	1MAD	3INST	QuIP#	QuIP#	AQLM	1MAD	3INST	QuIP#	QuIP#	AQLM	1MAD	3INST	QuIP#	QuIP#	AQLM
2-7	W2	5.12	**5.17**	**5.17**	5.22	5.19	5.21	**5.38**	5.40	5.60	5.41	5.38	7.05	**6.82**	8.22	6.19	6.14
	C4	6.63	**6.71**	**6.71**	6.79	6.75	6.75	**6.99**	7.01	7.34	7.04	7.01	9.14	**8.96**	11.0	8.16	8.09
2-13	W2	4.57	**4.62**	**4.62**	4.65	4.63	4.64	**4.74**	**4.74**	4.90	4.78	4.78	5.59	**5.52**	6.06	5.35	5.33
	C4	6.05	**6.10**	**6.10**	6.15	6.13	6.14	**6.28**	**6.28**	6.50	6.35	6.33	7.46	**7.39**	8.07	7.20	7.19
2-70	W2	3.12	**3.16**	**3.16**	3.18	3.18	3.19	**3.27**	**3.27**	3.41	3.35	3.36	**3.87**	3.90	4.16	3.91	3.83
	C4	4.97	**5.00**	**5.00**	5.02	5.02	5.03	**5.09**	**5.09**	5.20	5.15	5.17	5.70	**5.69**	6.01	5.71	5.62

Lookup-Free Computated Codes

QTIP Code Generation Algorithms

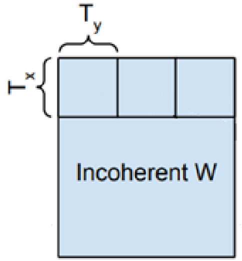
- 1MAD, 3INST

Perplexity (PPL)

- dataset : Wikitext2(W2), C4

Bitshift Trellis parameters

- L=16, V=1
- T_x=T_y=16 (16x16 tensor core MMA optimization



			4 Bit No FT			≈4 Bit		3 Bit No FT			≈3 Bit		2 Bit No FT			≈2 Bit	
		FP16	1MAD	3INST	QuIP#	QuIP#	AQLM	1MAD	3INST	QuIP#	QuIP#	AQLM	1MAD	3INST	QuIP#	QuIP#	AQLM
2-7	W2	5.12	**5.17**	**5.17**	5.22	5.19	5.21	**5.38**	5.40	5.60	5.41	5.38	7.05	**6.82**	8.22	6.19	6.14
	C4	6.63	**6.71**	**6.71**	6.79	6.75	6.75	**6.99**	7.01	7.34	7.04	7.01	9.14	**8.96**	11.0	8.16	8.09
2-13	W2	4.57	**4.62**	**4.62**	4.65	4.63	4.64	**4.74**	**4.74**	4.90	4.78	4.78	5.59	**5.52**	6.06	5.35	5.33
	C4	6.05	**6.10**	**6.10**	6.15	6.13	6.14	**6.28**	**6.28**	6.50	6.35	6.33	7.46	**7.39**	8.07	7.20	7.19
2-70	W2	3.12	**3.16**	**3.16**	3.18	3.18	3.19	**3.27**	**3.27**	3.41	3.35	3.36	**3.87**	3.90	4.16	3.91	3.83
	C4	4.97	**5.00**	**5.00**	5.02	5.02	5.03	**5.09**	**5.09**	5.20	5.15	5.17	5.70	**5.69**	6.01	5.71	5.62

better than fine-tuned QuIP#, AQLM

Lookup-Free Computated Codes

QTIP Code Generation Algorithms

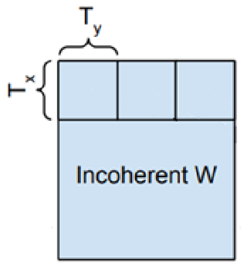
- 1MAD, 3INST

Perplexity (PPL)

- dataset : Wikitext2(W2), C4

Bitshift Trellis parameters

- L=16, V=1
- T_x=T_y=16 (16x16 tensor core MMA optimization



			4 Bit No FT			≈4 Bit		3 Bit No FT			≈3 Bit		2 Bit No FT			≈2 Bit	
		FP16	1MAD	3INST	QuIP#	QuIP#	AQLM	1MAD	3INST	QuIP#	QuIP#	AQLM	1MAD	3INST	QuIP#	QuIP#	AQLM
2-7	W2	5.12	**5.17**	**5.17**	5.22	5.19	5.21	**5.38**	5.40	5.60	5.41	5.38	7.05	**6.82**	8.22	6.19	6.14
	C4	6.63	**6.71**	**6.71**	6.79	6.75	6.75	**6.99**	7.01	7.34	7.04	7.01	9.14	**8.96**	11.0	8.16	8.09
2-13	W2	4.57	**4.62**	**4.62**	4.65	4.63	4.64	**4.74**	**4.74**	4.90	4.78	4.78	5.59	**5.52**	6.06	5.35	5.33
	C4	6.05	**6.10**	**6.10**	6.15	6.13	6.14	**6.28**	**6.28**	6.50	6.35	6.33	7.46	**7.39**	8.07	7.20	7.19
2-70	W2	3.12	**3.16**	**3.16**	3.18	3.18	3.19	**3.27**	**3.27**	3.41	3.35	3.36	**3.87**	3.90	4.16	3.91	3.83
	C4	4.97	**5.00**	**5.00**	5.02	5.02	5.03	**5.09**	**5.09**	5.20	5.15	5.17	5.70	**5.69**	6.01	5.71	5.62

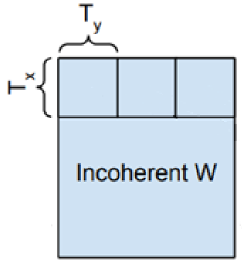
better than fine-tuned QuIP#, AQLM

Lookup-Free Computated Codes

QTIP Code Generation Algorithms
- 1MAD, 3INST

Perplexity (PPL)
- dataset : Wikitext2(W2), C4

Bitshift Trellis parameters
- L=16, V=1
- T_x=T_y=16 (16x16 tensor core MMA optimization



			4 Bit No FT			≈4 Bit		3 Bit No FT			≈3 Bit		2 Bit No FT			≈2 Bit	
		FP16	1MAD	3INST	QuIP#	QuIP#	AQLM	1MAD	3INST	QuIP#	QuIP#	AQLM	1MAD	3INST	QuIP#	QuIP#	AQLM
2-7	W2	5.12	**5.17**	**5.17**	5.22	5.19	5.21	**5.38**	5.40	5.60	5.41	5.38	7.05	**6.82**	8.22	6.19	6.14
	C4	6.63	**6.71**	**6.71**	6.79	6.75	6.75	**6.99**	7.01	7.34	7.04	7.01	9.14	**8.96**	11.0	8.16	8.09
2-13	W2	4.57	**4.62**	**4.62**	4.65	4.63	4.64	**4.74**	**4.74**	4.90	4.78	4.78	5.59	**5.52**	6.06	5.35	5.33
	C4	6.05	**6.10**	**6.10**	6.15	6.13	6.14	**6.28**	**6.28**	6.50	6.35	6.33	7.46	**7.39**	8.07	7.20	7.19
2-70	W2	3.12	**3.16**	**3.16**	3.18	3.18	3.19	**3.27**	**3.27**	3.41	3.35	3.36	**3.87**	3.90	4.16	3.91	3.83
	C4	4.97	**5.00**	**5.00**	5.02	5.02	5.03	**5.09**	**5.09**	5.20	5.15	5.17	5.70	**5.69**	6.01	5.71	5.62

worse than fine-tuned QuIP#, AQLM (except 2-bits)

Hybrid Lookup-Computated Codes (PPL)

Hybrid Lookup code parameters

- L=16, V=2, Q=9 (codebook size 2KiB with FP16)
- $T_x=T_y=16$

Fine-tuned

- codebook element block-wise fine-tuning

Llama1 → GPTVQ

		CTX. 2048, X = GPTVQ, Y = 0.13								CTX. 4096, X = AQLM, Y ≈ 0					
		WIKTEXT2				C4				WIKITEXT2			C4		
METHOD	BITS	1-7	1-13	1-30	1-65	1-7	1-13	1-30	1-65	2-7	2-13	2-70	2-7	2-13	2-70
FP16	16.0	5.68	5.09	4.10	3.53	7.04	6.61	5.98	5.62	5.12	4.57	3.12	6.63	6.05	4.97
X	4+Y	5.94	5.20	4.18	3.64	–	–	–	–	5.21	4.65	3.19	6.75	6.14	5.03
QUIP#	4.00	5.76	5.17	4.18	3.60	7.18	6.67	6.03	5.66	5.19	4.63	3.18	6.75	6.13	5.02
QTIP	4.00	**5.72**	**5.15**	**4.15**	**3.58**	**7.13**	**6.65**	**6.01**	**5.64**	**5.17**	**4.61**	**3.16**	**6.69**	**6.09**	**5.00**
X	3+Y	6.32	5.31	4.38	3.79	–	–	–	–	5.38	4.78	3.36	7.01	6.33	5.17
QUIP#	3.00	5.98	5.31	4.36	3.70	7.39	6.83	6.17	5.77	5.41	4.78	3.35	7.04	6.35	5.15
QTIP	3.00	**5.85**	**5.24**	**4.26**	**3.68**	**7.26**	**6.74**	**6.09**	**5.71**	**5.28**	**4.69**	**3.26**	**6.87**	**6.22**	**5.08**
X	2+Y	9.64	6.58	5.63	4.91	–	–	–	–	6.14	5.33	3.83	8.09	7.19	5.62
QUIP#	2.00	6.86	5.97	5.02	4.36	8.36	7.48	6.71	6.19	6.19	5.35	3.91	8.16	7.20	5.71
QTIP	2.00	**6.52**	**5.80**	**4.83**	**4.21**	**7.99**	**7.31**	**6.56**	**6.08**	**5.86**	**5.11**	**3.70**	**7.73**	**6.85**	**5.48**

outperforms all cases, especially at 2-bits (high dimensionality)

Hybrid Lookup-Computated Codes (PPL)

Hybrid Lookup code parameters

- L=16, V=2, Q=9 (codebook size 2KiB with FP16)
- T_x=T_y=16

Fine-tuned

- codebook element block-wise fine-tuning

Llama2 → AQLM

		CTX. 2048, X = GPTVQ, Y = 0.13								CTX. 4096, X = AQLM, Y ≈ 0					
		WIKTEXT2				C4				WIKITEXT2			C4		
METHOD	BITS	1-7	1-13	1-30	1-65	1-7	1-13	1-30	1-65	2-7	2-13	2-70	2-7	2-13	2-70
FP16	16.0	5.68	5.09	4.10	3.53	7.04	6.61	5.98	5.62	5.12	4.57	3.12	6.63	6.05	4.97
X	4+Y	5.94	5.20	4.18	3.64	–	–	–	–	5.21	4.65	3.19	6.75	6.14	5.03
QUIP#	4.00	5.76	5.17	4.18	3.60	7.18	6.67	6.03	5.66	5.19	4.63	3.18	6.75	6.13	5.02
QTIP	4.00	**5.72**	**5.15**	**4.15**	**3.58**	**7.13**	**6.65**	**6.01**	**5.64**	**5.17**	**4.61**	**3.16**	**6.69**	**6.09**	**5.00**
X	3+Y	6.32	5.31	4.38	3.79	–	–	–	–	5.38	4.78	3.36	7.01	6.33	5.17
QUIP#	3.00	5.98	5.31	4.36	3.70	7.39	6.83	6.17	5.77	5.41	4.78	3.35	7.04	6.35	5.15
QTIP	3.00	**5.85**	**5.24**	**4.26**	**3.68**	**7.26**	**6.74**	**6.09**	**5.71**	**5.28**	**4.69**	**3.26**	**6.87**	**6.22**	**5.08**
X	2+Y	9.64	6.58	5.63	4.91	–	–	–	–	6.14	5.33	3.83	8.09	7.19	5.62
QUIP#	2.00	6.86	5.97	5.02	4.36	8.36	7.48	6.71	6.19	6.19	5.35	3.91	8.16	7.20	5.71
QTIP	2.00	**6.52**	**5.80**	**4.83**	**4.21**	**7.99**	**7.31**	**6.56**	**6.08**	**5.86**	**5.11**	**3.70**	**7.73**	**6.85**	**5.48**

Hybrid Lookup-Computated Codes (Zeroshot)

Evaluation Tasks

- ARCC, ARCE, PIQA, WINO

	2-70					2-13					2-7				
Mthd.	Bits	ArcC	ArcE	PiQA	Wino	Bits	ArcC	ArcE	PiQA	Wino	Bits	ArcC	ArcE	PiQA	Wino
FP16	16	51.1	77.7	81.1	77.0	16	45.6	73.3	73.5	69.6	16	40.0	69.3	78.5	67.3
AQLM	4.14	50.7	77.3	81.5	76.5	3.94	**44.8**	73.3	78.4	**69.9**	4.04	41.0	70.2	78.2	67.3
QuIP#	4	50.5	77.7	81.4	**77.3**	4	43.6	71.3	78.7	69.6	4	40.4	68.6	**78.5**	**67.4**
QTIP	4	50.0	**77.8**	81.3	76.9	4	44.8	73.6	**78.9**	69.9	4	40.0	**68.9**	78.4	67.1
AQLM	3.01	50.3	78.0	80.7	75.3	3.03	42.8	72.9	78.5	68.8	3.04	38.5	66.8	77.3	65.4
QuIP#	3	**50.9**	77.6	**81.4**	76.1	3	**44.0**	72.5	78.4	69.1	3	**39.2**	**68.4**	77.3	66.5
QTIP	3	50.3	**78.2**	80.6	77.0	3	**44.0**	72.8	78.0	**69.5**	3	38.9	68.1	**78.1**	**66.9**
AQLM	2.07	47.9	**77.7**	80.4	75.9	1.97	38.8	69.3	75.9	68.8	2.02	32.8	63.7	74.8	65.7
QuIP#	2	47.6	77.1	79.5	74.6	2	39.6	69.0	**77.3**	67.4	2	35.2	65.3	75.4	**64.9**
QTIP	2	**48.0**	76.3	80.2	75.1	2	**41.4**	**70.8**	**77.3**	**67.6**	2	**35.7**	**65.6**	**75.9**	64.7

QTIP not always outperform even at 2-bits

Hybrid Lookup-Computated Codes (Zeroshot)

PPL (ctx=8192), Zero-shot w/ Llama 3

- Llama 3 is known to be more difficult to apply PTQ than Llama 2

		3-70 PPL (↓)		3-70 ZEROSHOT ACC (↑)					3-8 PPL (↓)		3-8 ZEROSHOT ACC (↑)				
MTHD.	BITS	W2	C4	ARCC	ARCE	BOOLQ	PIQA	WINO	W2	C4	ARCC	ARCE	BOOLQ	PIQA	WINO
BF16	16.0	2.59	5.78	60.5	86.9	85.3	82.4	80.3	5.54	7.10	50.2	80.1	81.0	79.7	72.9
QUIP#	4.00	2.99	5.96	35.0	67.3	84.7	71.9	76.7	5.81	7.32	50.2	**79.7**	**81.3**	**79.7**	73.1
QTIP	4.00	**2.75**	5.83	**56.1**	**83.9**	**85.8**	**81.3**	**80.6**	**5.67**	**7.20**	50.2	79.6	79.5	79.4	**73.4**
QUIP#	3.00	3.59	6.18	31.1	36.6	**85.7**	58.8	76.4	6.27	7.71	46.4	77.4	79.9	77.9	72.9
QTIP	3.00	**3.18**	5.98	**48.6**	**77.8**	85.0	**77.8**	**79.7**	**6.01**	**7.48**	**49.2**	**79.3**	**80.0**	**79.2**	**74.5**
QUIP#	2.00	5.77	7.46	18.3	32.2	82.1	54.7	68.9	7.84	9.06	39.2	72.9	76.6	75.6	68.2
QTIP	2.00	**4.97**	**6.80**	**28.0**	**35.2**	**83.6**	**57.1**	**72.6**	**7.33**	**8.62**	**44.2**	**75.2**	**76.7**	**77.6**	**70.7**

For Llama3, outperforms QuIP# because of high dimensionality (TCQ > VQ)

Hybrid Lookup-Computated Codes (Zeroshot)

Llama 3-1, instruct-tuned

- PV-Tuning : fine-tuning focused quantization method

			PPL. (↓)	ZEROSHOT (↑)			
		BITS	W2	ARCC	ARCE	HSWAG	PIQA
3.1 405B INST.	META "FP8"	16 ATTN. / 8 MLP	1.70	61.6	81.4	67.1	83.8
	QTIP	4	1.79	61.3	80.9	66.7	84.2
	QTIP	3	2.05	61.5	81.4	66.8	83.5
	QTIP	2	3.29	60.7	81.1	65.4	82.2
3.1 70B INST.	BF16	16	3.52	56.7	75.6	61.5	82.8
	QTIP	4	3.73	56.3	75.8	61.4	83.0
	QTIP	3	4.12	55.1	75.1	60.8	82.6
	QTIP	2	5.08	54.4	72.6	59.4	82.5
	PV-TUNING	2.01	5.70	52.7	72.2	60.2	82.6
3.1 8B INST.	BF16	16	6.50	51.6	77.8	57.7	80.0
	QTIP	4	6.61	50.7	78.0	57.5	80.1
	QTIP	3	6.80	50.4	77.7	56.9	79.3
	QTIP	2	7.82	45.1	75.6	54.5	79.0
	PV-TUNING	2.07	8.45	46.2	75.4	54.4	78.7

lower PPL, similar zeroshot

Llama 3-2, instruct-tuned

- 2.5-3X compression

			PPL (↓)	ZEROSHOT (↑)			
		SIZE (GB)	W2	ARCC	ARCE	HSWAG	PIQA
3B	BF16	6	9.58	43.3	74.3	52.2	75.7
	QTIP	2.1	9.77	43.5	74.3	51.9	75.1
1B	BF16	2.4	11.57	36.0	68.5	45.2	74.2
	QTIP	0.97	11.93	34.8	68.4	44.5	73.3

4bit quantization for decoding layer (not embedding)

Inference Speed

Batch 1 inference speed

- AQLM : large codebook size
- QuIP# : dimension 8, VQ search
- QTIP : dimension 256, parallel quantization decoding

Table 4: Batch size 1 decoding throughput on a RTX6000 Ada (960GB/s mem. BW).

Method	Bits	2-7B Tok/s	2-70B Tok/s
FP16	16	55.9	OOM
AQLM	2	81.5	8.78
QuIP#	2	186	22.2
QTIP	2	188	23.5
QTIP	3	161	19.1
QTIP	4	140	16.3

strong performance with fast inference

Conclusion

Contribution

- High-dimensional vector quantization based on a trellis code
- L1 cache-friendly lookup-free or hybrid lookup computated codes

Limitation

- Weight-only PTQ method
- Low zeroshot accuracy relative to PPL

Appendix

Ablation on Trellis Size (L)

- Large L reduces PPL, but codebook size ↑

L	Trellis Size	CB size	total size	W2	C4
QuIP#	-	8Kb	8Kb	8.22	11.0
8	8.19 Kb	4.10 Kb	12.29 Kb	7.83	10.3
10	40.96 Kb	16.38 Kb	57.34 Kb	7.49	9.67
12	196.61 Kb	65.54 Kb	262.14 Kb	6.97	9.21
16	4.19 Mb	1.05 Mb	5.24 Mb	6.83	8.92
16	Bitshift	3INST	0Kb	6.82	8.96

Ablation on Trellis Size (V)

- Large V enhances quantization efficiency, but increases PPL
- Larger L can cover effect of large V

Codebook	L	V	W2	C4
LUT	12	1	6.97	9.21
LUT	12	2	7.09	9.24
LUT	12	4	7.55	9.88
LUT	16	1	6.83	8.92
LUT	16	2	6.79	8.97
QTIP HYB (no FT)	16	2	6.83	8.97
LUT	16	4	6.92	9.07

Appendix

Decoding Speed on Different GPUs

GPU Model	Model	2-bit tok/s	3-bit tok/s	4-bit tok/s	FP16 tok/s
RTX 3090	2-7	127	119	109	52.5
RTX 3090	2-70	15.3	OOM	OOM	OOM
RTX A6000 Ampere	2-7	116	106	95	43.5
RTX A6000 Ampere	2-70	15.0	13.1	11.7	OOM
RTX 6000 Ada	2-7	188	161	140	55.9
RTX 6000 Ada	2-70	23.5	19.1	16.3	OOM