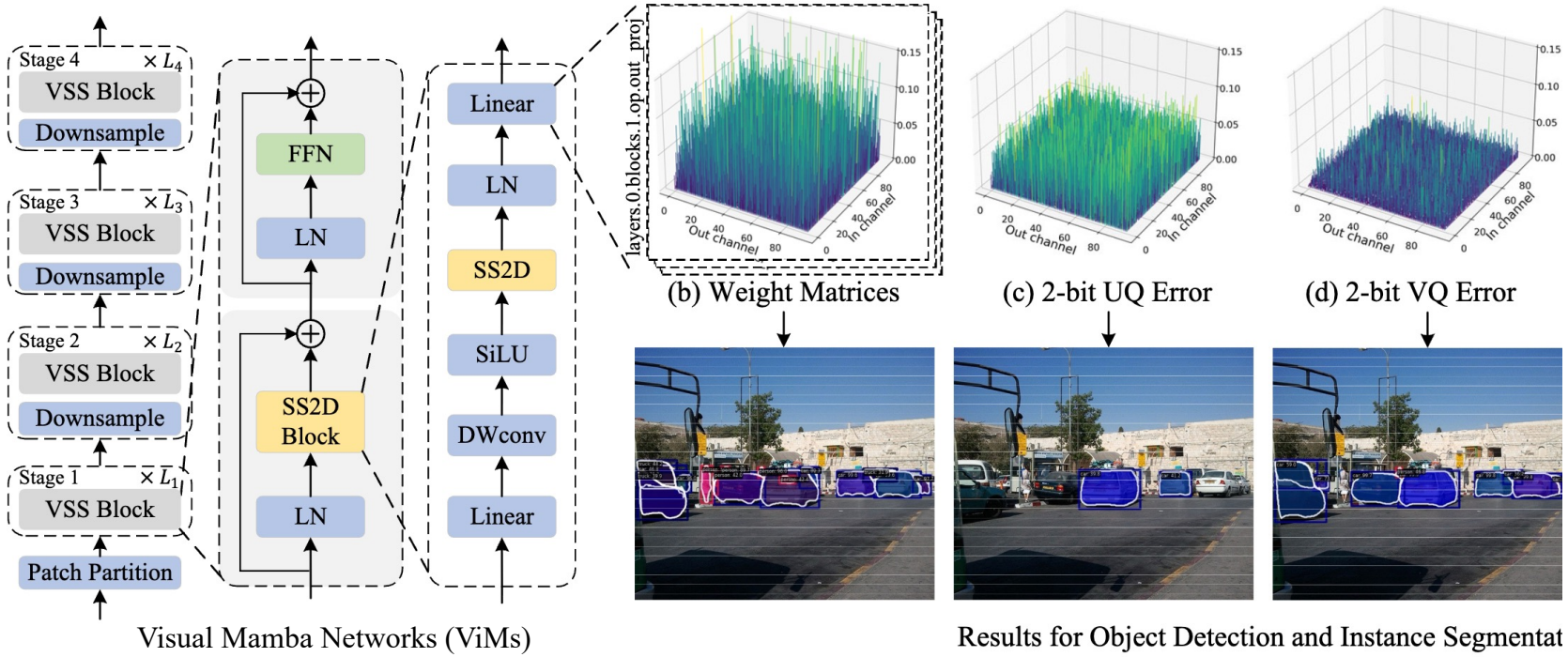
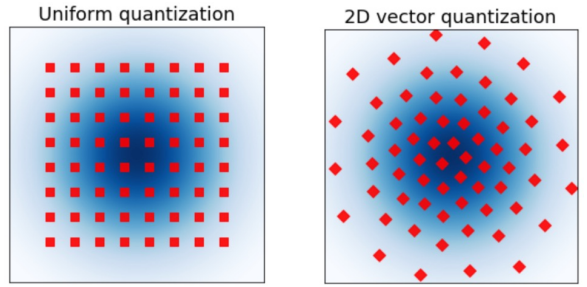
ViM-VQ: Efficient Post-Training Vector Quantization for Visual Mamba

Juncan Deng[1,2], Shuaiting Li[1,2], Zeyu Wang[1], Kedong Xu[2], Hong Gu[2], Kejie Huang[1]

1 Zhejiang University
2 VIVO Mobile

Vector Quantization for ViMs



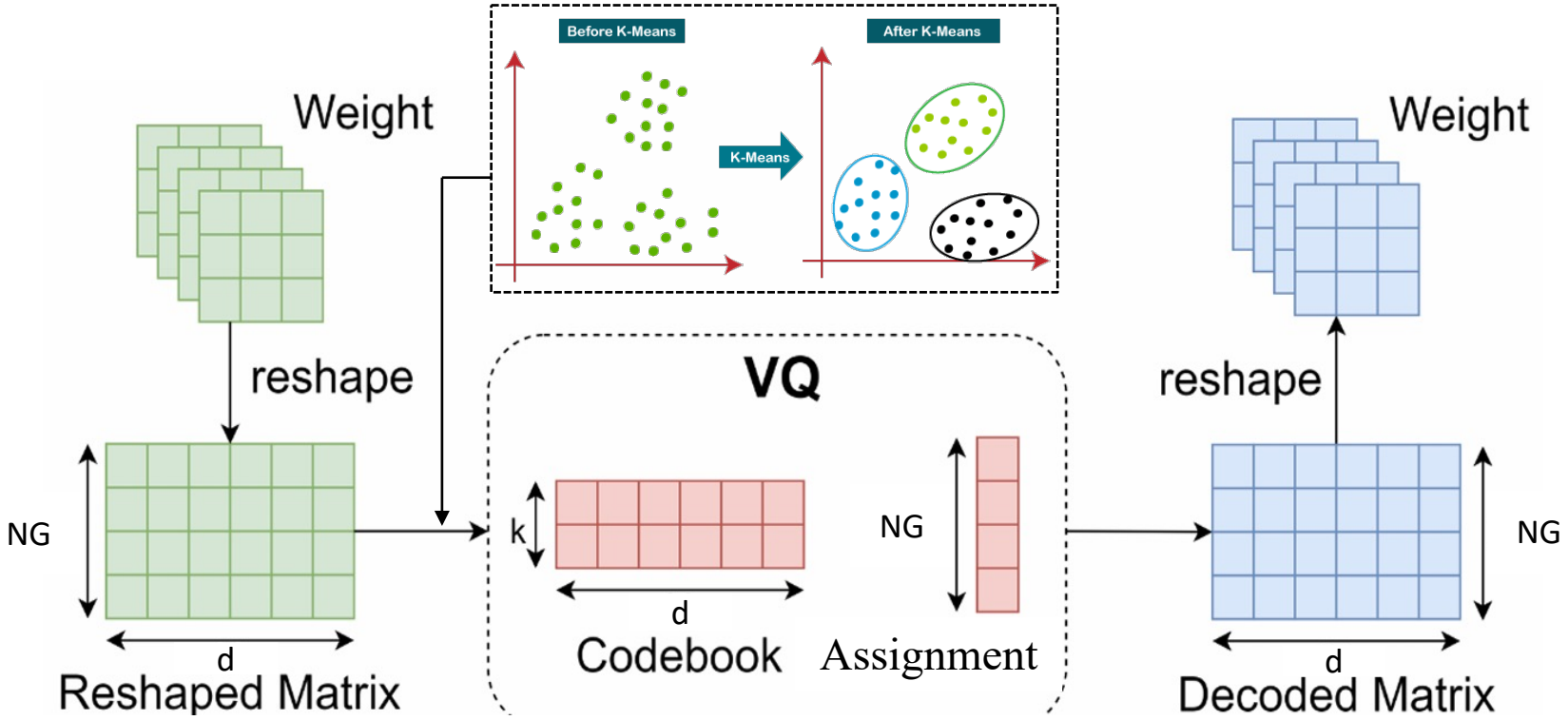


VQ has better quantization performance

- To reducing memory usage and computational latency, it is generally necessary to develop vector quantization algorithm for Visual Mamba networks (ViMs).

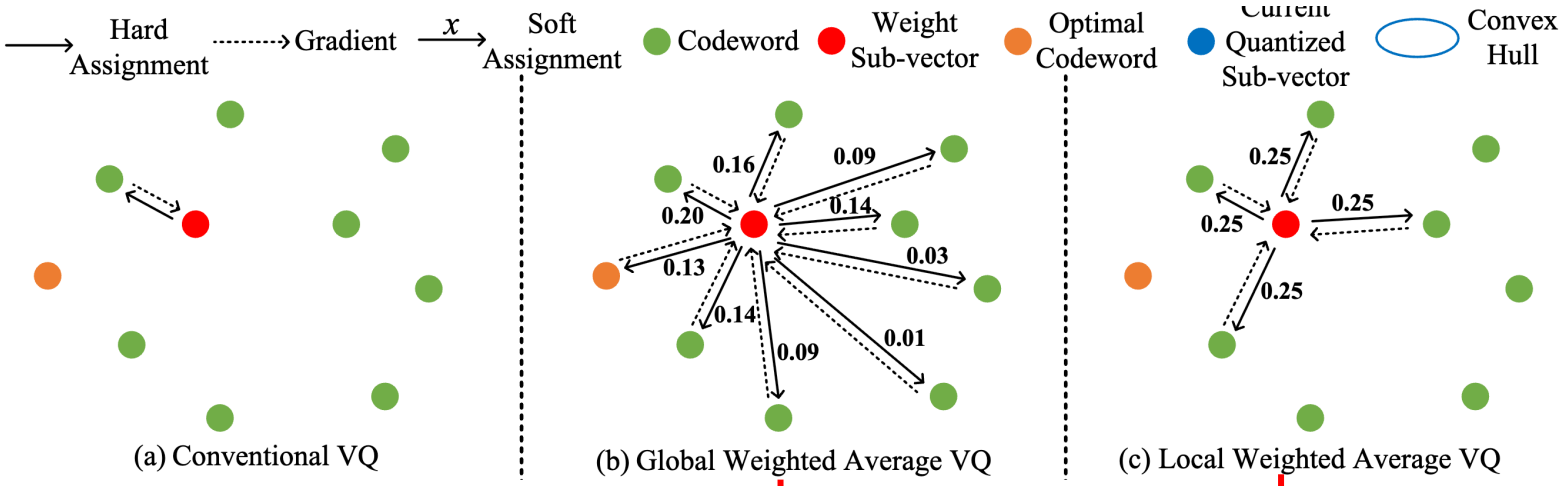
Background of Vector Quantization

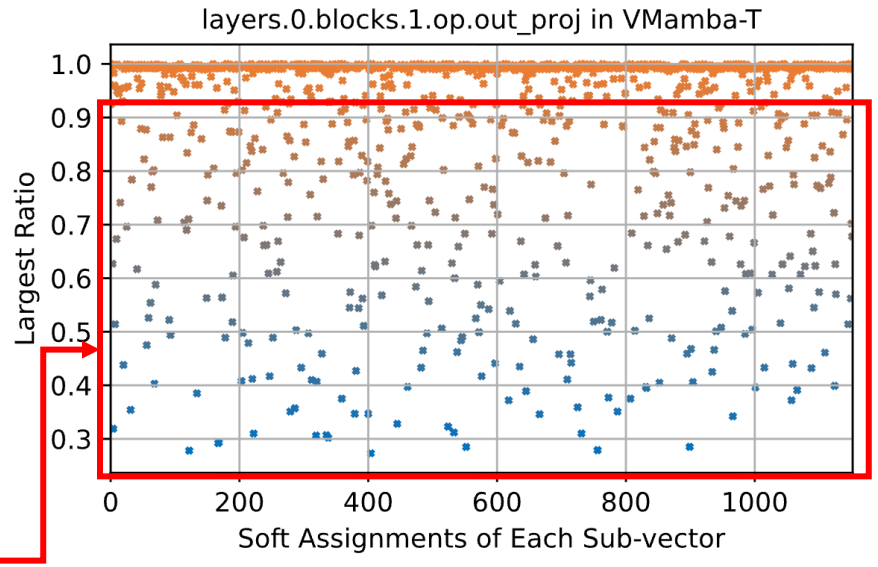
Vector Quantization (VQ) is a hardware-friendly compression technique.



Limitation in Previous VQ works

However, Existing VQ methods are ineffective.





Too simplistic, only fine-tunes codebook

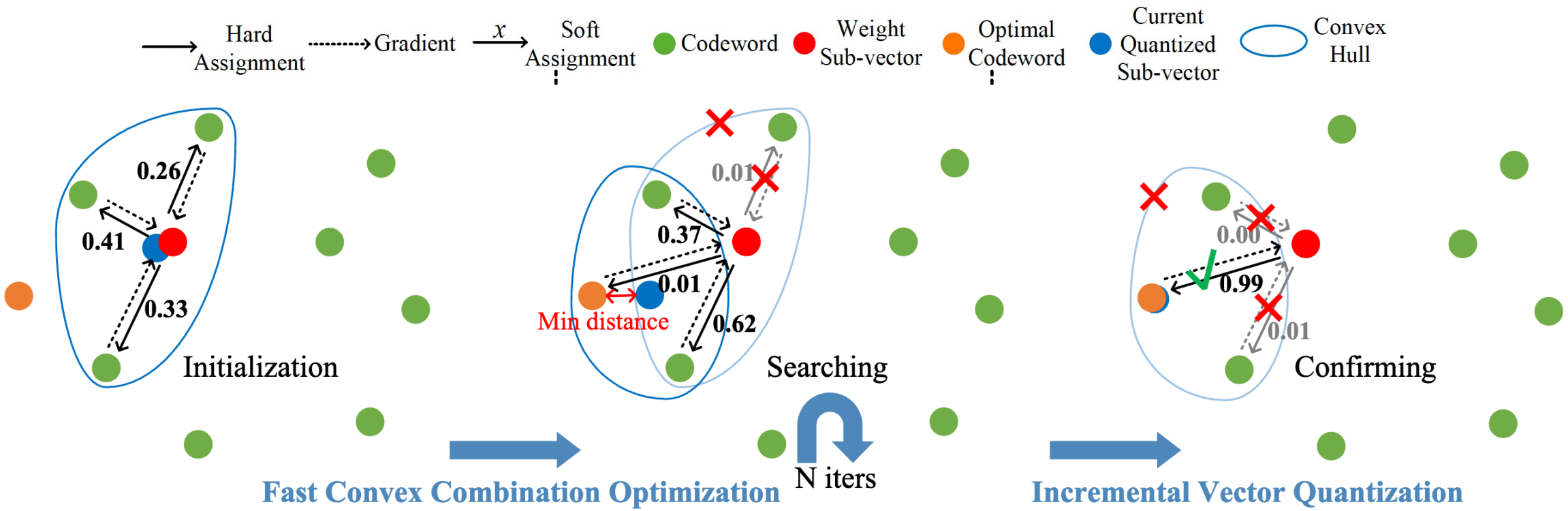
Prohibitive overhead of global soft assignments

Degraded performance of local soft assignments

Truncation error from soft-to-hard assignments

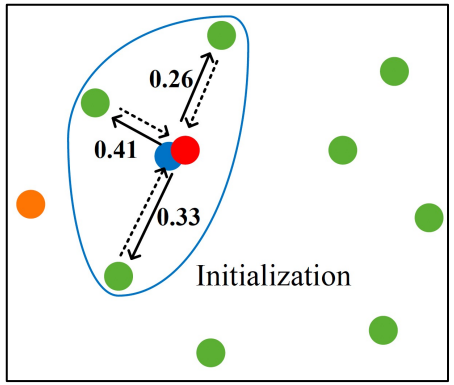
How can we boost VQ during finetuning?

ViM-VQ Algorithm Overview



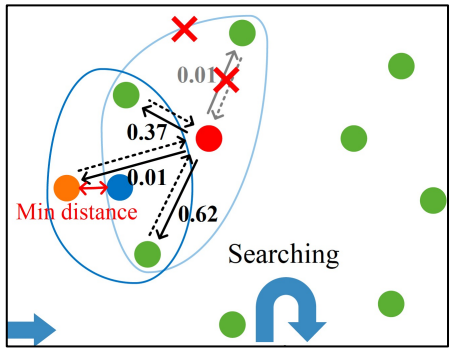
- The core idea is a local search with a dynamically updated scope to achieve progressive quantization.

ViM-VQ Algorithm



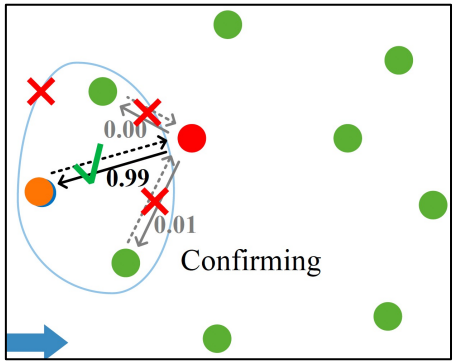
$$\widehat{w}_{o,i/d} = \sum \mathcal{C}_{o,i/d} \mathcal{R}_{o,i/d}$$

Step1: Construct convex combination for neighboring codewords, initializing them near the original weight.



$$\mathcal{C}_{o,i/d} \leftarrow \mathcal{C}_{o,i/d} - O(\nabla_{\mathcal{C}_{o,i/d}} \mathcal{L}_{\text{init}}, \theta_c),$$
$$\mathcal{R}_{o,i/d} \leftarrow \mathcal{R}_{o,i/d} - O(\nabla_{\mathcal{R}_{o,i/d}} \mathcal{L}_{\text{init}}, \theta_r),$$
$$c^m_{o,i/d} \leftarrow \underset{c(k) \in \mathbf{C} \setminus \mathcal{C}_{o,i/d}}{\operatorname{argmin}} \|\widehat{w}_{o,i/d} - c(k)\|_2^2,$$

Step2: Finetune the combinations and update the convex hulls towards a more optimal solution.



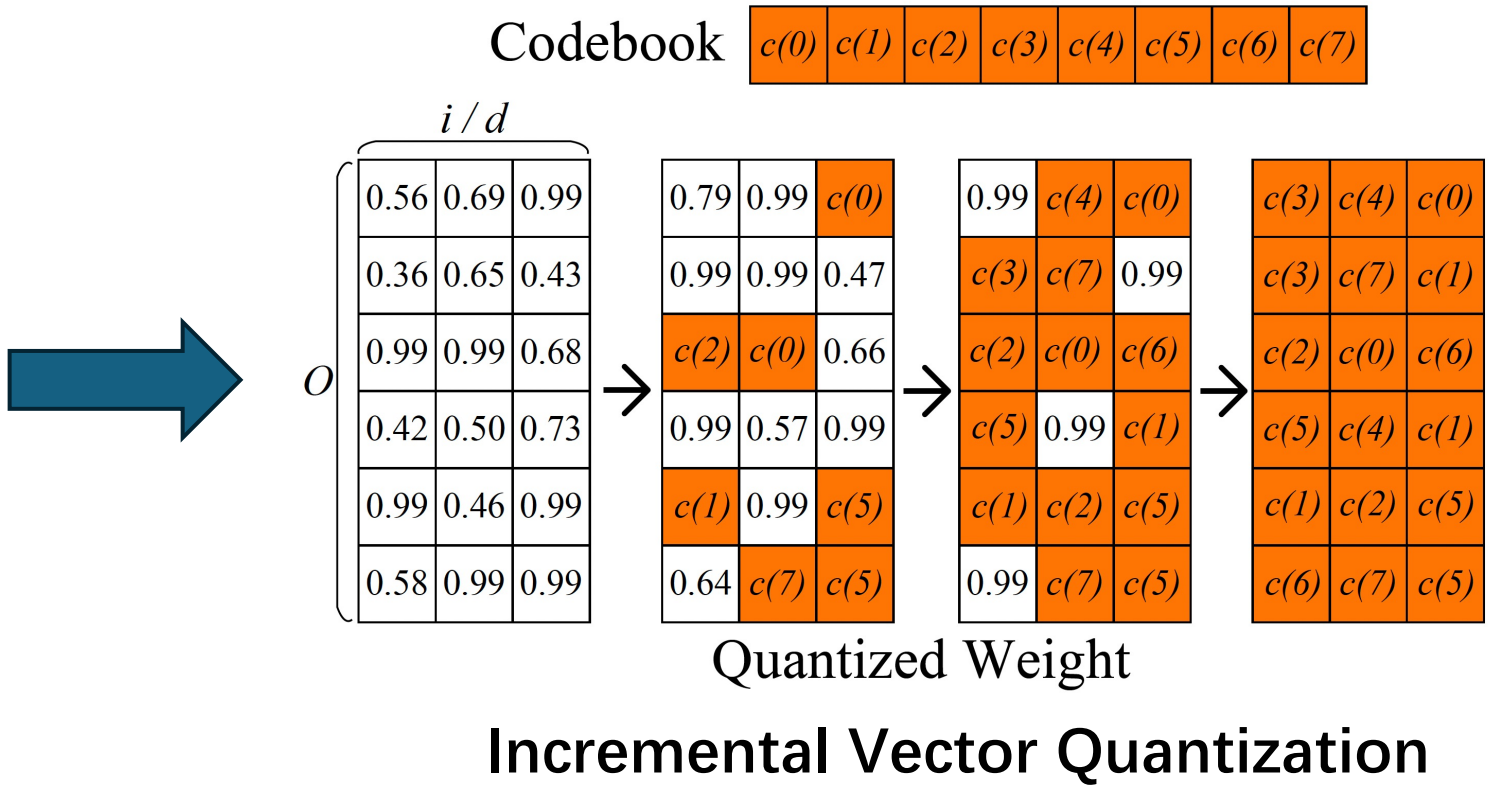
$$\widehat{w}_{o,i/d} \leftarrow c^m_{o,i/d},$$
$$a_{o,i/d} \leftarrow \text{index}(c^m_{o,i/d}).$$

Step3: Confirm high-confidence codeword as result of incremental quantization.

ViM-VQ Algorithm

Algorithm 1 ViM-VQ

Input: Weight matrix $\mathbf{W}$, calibration data $(\mathbf{x}, \mathbf{y})$
Output: Codebook $\mathbf{C}$, assignments $\mathbf{A}$
$\mathbf{C}, \mathbf{A} = \text{K-Means}(\mathbf{W}, k)$
▷ Phase 1: Fast Convex Combination Optimization
for each sub-vector $w_{o,i/d}$ in $\mathbf{W}$ **do**
  $\mathcal{C}_{o,i/d} = \text{Top}_n(-\|w_{o,i/d} - c(k)\|_2^2)$
  $\mathcal{R}_{o,i/d} = \text{softmax}(z_{o,i/d})$
  Convex combination: $(\mathcal{C}_{o,i/d}, \mathcal{R}_{o,i/d})$
end for
$\widehat{\mathbf{W}} = \sum \mathcal{C} \odot \mathcal{R}$
▷ Initialize codewords and ratios
while not converged **do**
  $\mathcal{L}_{\text{init}} = \|\mathbf{W} - \widehat{\mathbf{W}}\|_2^2$
  $\mathcal{C} \leftarrow \mathcal{C} - O(\nabla_c \mathcal{L}_{\text{init}}, \theta_c), \quad \mathcal{R} \leftarrow \mathcal{R} - O(\nabla_r \mathcal{L}_{\text{init}}, \theta_r)$
end while
▷ Phase 2: Incremental Vector Quantization
for calibration step $t = 1$ to T **do**
  $\widehat{\mathbf{y}} = \epsilon_q(\mathbf{x})$
  $\mathcal{L} = \mathcal{L}_t + \mathcal{L}_{\text{bkd}} + \mathcal{L}_r$
  $\mathcal{C} \leftarrow \mathcal{C} - O(\nabla_c \mathcal{L}, \theta_c), \quad \mathcal{R} \leftarrow \mathcal{R} - O(\nabla_r \mathcal{L}, \theta_r)$
  ▷ Confirm high-ratio assignments
  for each sub-vector with $r_{o,i/d}^m > \tau$ **do**
    $\widehat{w}_{o,i/d} \leftarrow c_{o,i/d}^m$
    $a_{o,i/d} \leftarrow \text{index}(c_{o,i/d}^m)$
  end for
  ▷ Adaptive candidate replacement
  for each candidate with $r_{o,i/d}^m < \lambda$ **do**
    $c_{o,i/d}^m \leftarrow \underset{c(k) \in \mathbf{C} \setminus \mathcal{C}_{o,i/d}}{\text{argmin}} \|\widehat{w}_{o,i/d} - c(k)\|_2^2$
  end for
end for
return $\mathbf{C}$, $\mathbf{A}$



Incremental Vector Quantization

Algorithm Evaluation

- ViM-VQ demonstrate its superiority over VQ and UQ.

Model (Params)	Method	Top-1 Accuracy (%) FP	3-bit	2-bit	1-bit
Vim-T (7M)	UQ†	76.07	52.84	0.16	-
	GPTQ		56.74	0.17	-
	MambaQuant		57.21	0.18	-
	PTQ4VM		57.29	0.18	-
	VQ†		67.56	33.18	0.22
	PQF		71.71	61.84	4.20
	DKM		73.64	71.25	68.29
	VQ4DiT		73.57	71.04	67.72
	ViM-VQ (Ours)		**74.79**	**72.17**	**69.93**
Vim-S (26M)	UQ†	80.48	72.31	0.16	-
	GPTQ		74.45	0.20	-
	MambaQuant		74.57	0.24	-
	PTQ4VM		74.68	0.25	-
	VQ†		77.59	65.96	0.37
	PQF		78.46	73.28	10.93
	DKM		79.70	77.96	70.38
	VQ4DiT		79.37	77.87	69.73
	ViM-VQ (Ours)		**80.10**	**78.66**	**72.02**
Vim-B (98M)	UQ†	81.88	76.17	2.06	-
	GPTQ		76.81	2.47	-
	MambaQuant		77.01	3.49	-
	PTQ4VM		77.17	3.62	-
	VQ†		77.78	70.88	5.28
	PQF		78.76	76.02	27.40
	DKM		80.02	79.18	74.57
	VQ4DiT		79.73	78.86	74.37
	ViM-VQ (Ours)		**80.34**	**79.46**	**75.58**

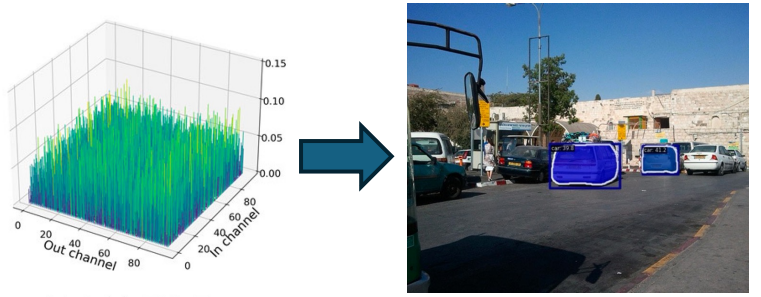
classification

Bit	Method	mIoU (SS)	mIoU (MS)	mAcc	aAcc
FP	Swin-T	44.5	45.8	-	-
	ConvNeXt-T	46.0	46.7	-	-
	VMamba-T	47.9	48.8	59.2	82.4
2	PQF	42.7	43.6	58.2	79.7
	DKM	44.1	44.8	58.7	80.6
	VQ4DiT	43.9	44.7	58.4	80.6
	ViM-VQ (Ours)	**44.4**	**45.4**	**58.9**	**81.0**
1	PQF	28.5	29.0	39.9	72.6
	DKM	37.2	37.6	52.8	76.6
	VQ4DiT	36.2	36.8	51.9	76.2
	ViM-VQ (Ours)	**41.1**	**41.8**	**56.0**	**77.3**

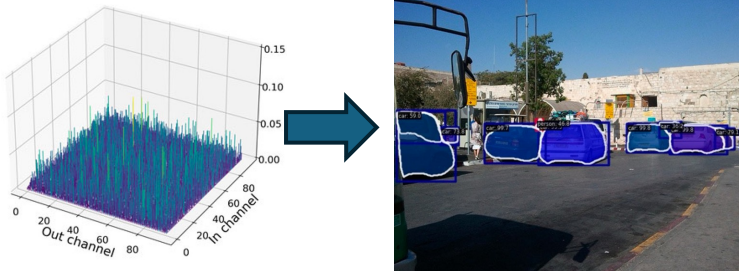
segmentation

Bit	Method	AP^{box}	AP^{box}_{50}	AP^{mask}	AP^{mask}_{50}
FP	Swin-T	42.7	-	39.3	-
	ConvNeXt-T	44.2	-	40.1	-
	VMamba-T	47.3	69.3	42.7	66.4
2	PQF	44.6	66.5	40.6	63.7
	DKM	45.7	68.2	41.5	64.8
	VQ4DiT	45.5	67.9	41.4	64.7
	ViM-VQ (Ours)	**46.0**	**68.8**	**41.8**	**65.7**
1	PQF	33.1	53.0	31.2	50.5
	DKM	37.5	58.0	35.4	55.4
	VQ4DiT	36.6	56.9	34.0	54.3
	ViM-VQ (Ours)	**40.9**	**60.7**	**35.8**	**57.9**

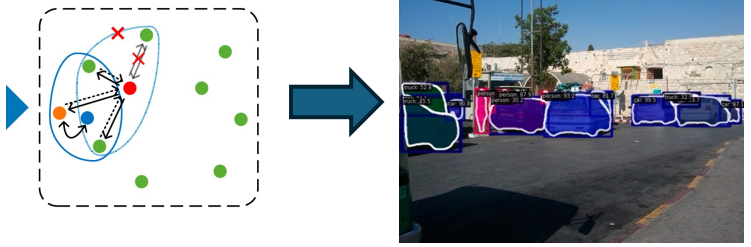
detection



(c) 2-bit UQ Error



(d) 2-bit VQ Error



(e) Our 2-bit ViM-VQ

Algorithm Evaluation

Method	Setting	Weight Storage Size	Ratio	Time
FP	-	360MB	1.0×	-
GPTQ AWQ OmniQ	per-group (g128)	Qweight + S + Z = 22.5MB + 6.0MB	12.6×	35m 1m 48m
MambaQ PTQ4VM	per-channel	Qweight + S + Z = 22.5MB + 1.3MB	15.1×	16m 92m
DKM VQ4DiT ViM-VQ	256 × 4	Qweight + C = 22.5MB + **0.5MB**	**15.7**×	1 day 104m 88m

Compression Ratio

C	Method	n	E	Mem	Acc-C	Acc-I
256 × 4	DKM	256	14	25 GB	71.98	71.25
	VQ4DiT	4	2	11 GB	71.33	71.04
	ViM-VQ	**4**	**2**	**11 GB**	**72.17**	**72.17**
	ViM-VQ	16	4	12 GB	**72.31**	**72.31**
	ViM-VQ	64	8	14 GB	**72.34**	**72.34**

Computation Overhead

- **vs. Uniform Quantization (UQ):**
  - Achieves superior compression rates.
- **vs. Vector Quantization (VQ):**
  - Reduces computational overhead.
  - Maintains accuracy consistency between calibration and inference.

Summary

In this work,

- We propose ViM-VQ, an efficient post-training vector quantization method tailored for Visual Mamba networks (ViMs).
- We introduce a fast convex combination optimization to search for optimal codewords, and an incremental quantization to mitigate truncation errors.
- ViM-VQ achieves superior performance in low-bit quantization across various tasks.

Contact Me by E-mail:
dengjuncan@zju.edu.cn