Cache Me If You Must: Adaptive Key-Value Quantization for Large Language Models

Myeongjun Lee, Jeonghwan Ahn
Department of Electrical Engineering
POSTECH, Korea

Contents

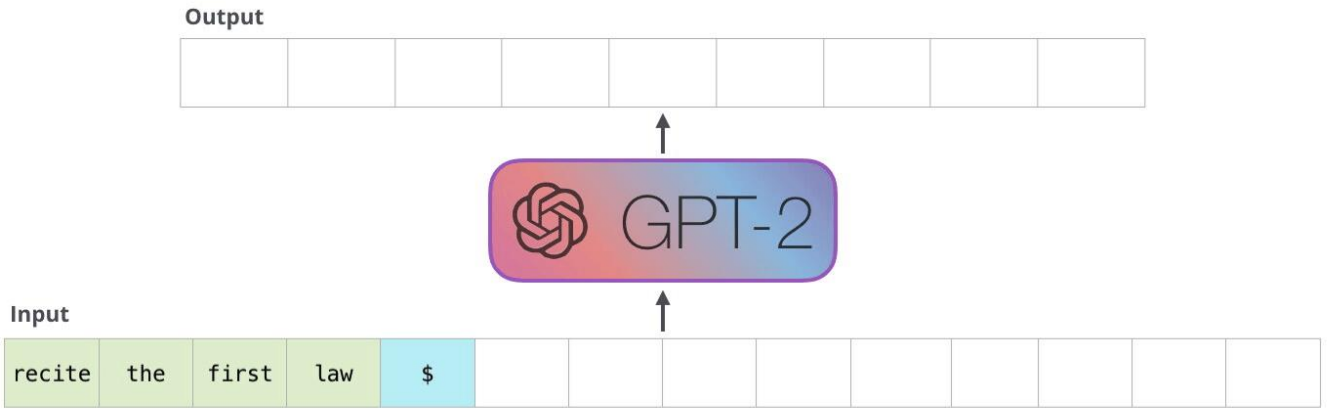
1. Preliminary study

- What is the KV Cache?
- Long Sequence
- Challenge in KV Cache compression

What is the KV-Cache?

- ✓ **Transformer-based models generate the next token (word) in an autoregressive manner based on previous inputs**

- ✓ **They predict the next word using the previously input text information.**



[1] https://medium.com/@joaolages/kv-caching-explained-276520203249

What is the KV-Cache?

- ✓ **Repeated computations are performed during the auto-regressive generation process**
  - **recalculating the same previous token attention at each generation step**
  - **Computational cost !!!**

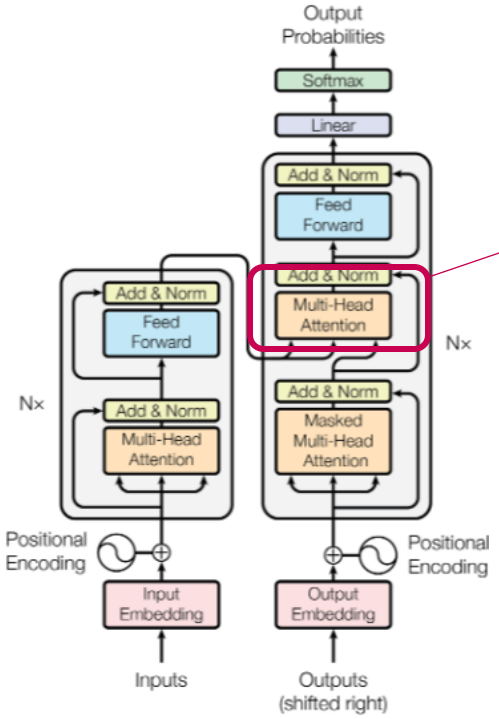
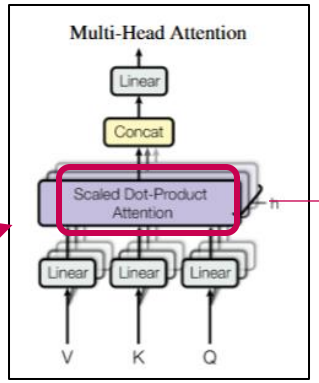
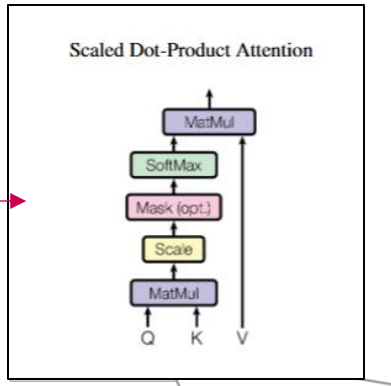
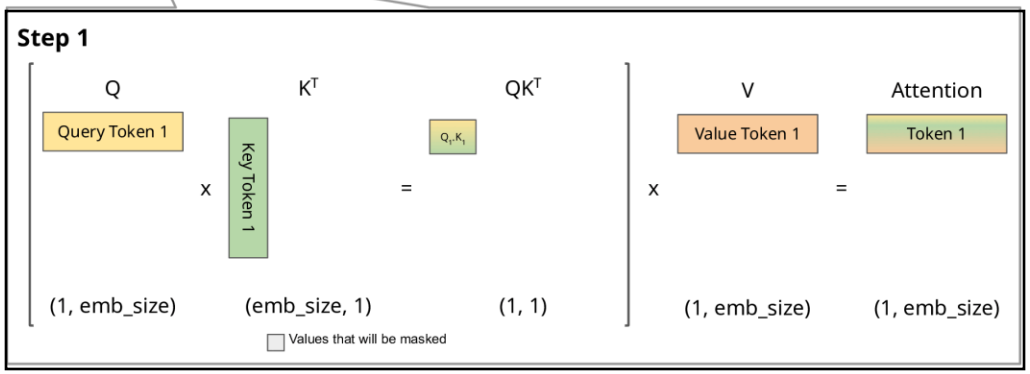


Figure 1: The Transformer - model architecture.





$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

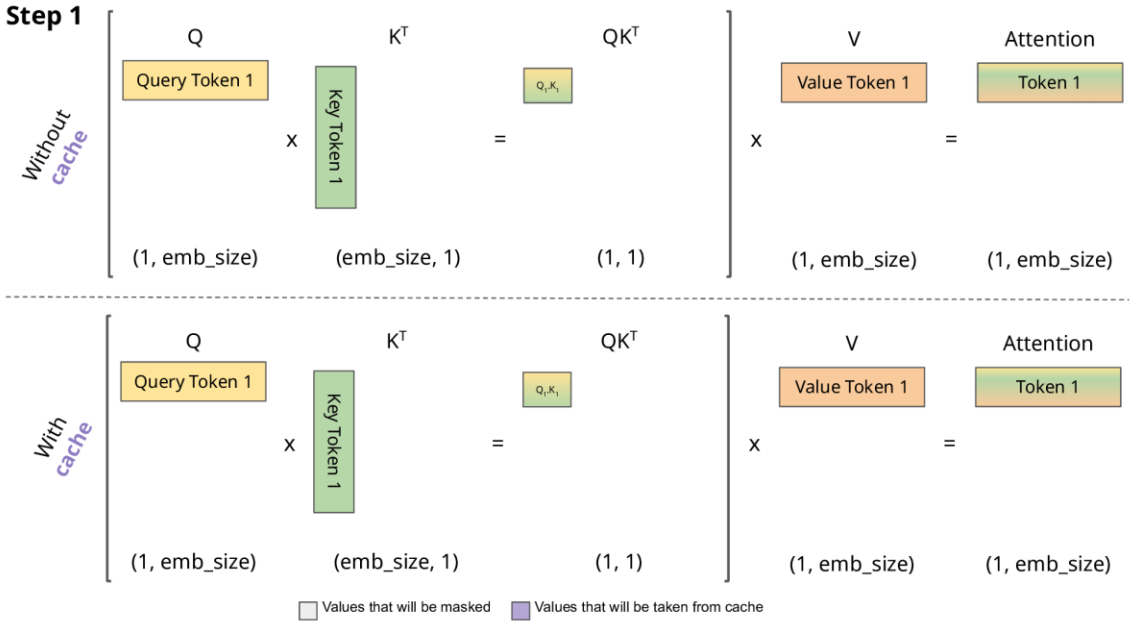


Zoom-in! (simplified without Scale and Softmax)

[1] Ashish Vaswani et. al., “Attention Is All You Need” , arxiv:1706.03762
[2] https://medium.com/@joaolages/kv-caching-explained-276520203249

What is the KV-Cache?

- ✓ By caching the previous Keys an Values, it is possible to focus on only calculating the attention for the new token
- ✓ The matrices obtained with KV caching are small → fast matrix multiplications
- ✓ faster inference & to avoid wasteful re-computation

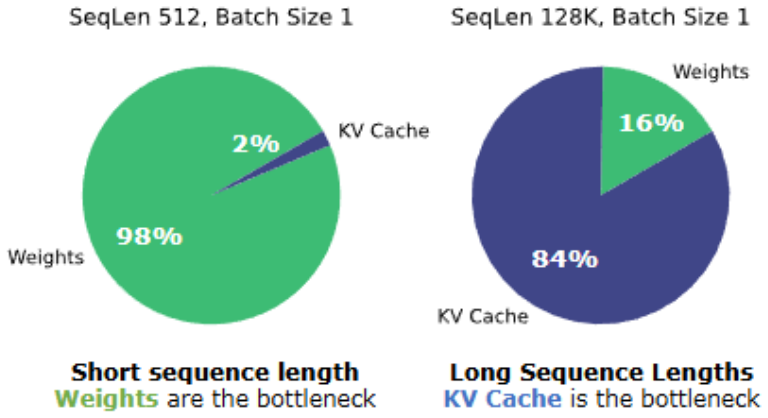


[1] https://medium.com/@joaolages/kv-caching-explained-276520203249

Long Sequence

- ✓ KV caching comes with its own pitfalls
- ✓ For longer context lengths, the KV cache becomes the dominant memory bottleneck

[Model size and activation memory size estimates for different LLaMA models]



BS	Model	Model Size (GB) 16 → 2-bit	KV Cache Size w/ Diff. Seq Len (GB) 32K	128K	1M	10M (16 → 2-bit)
1	7B	12.6 → 1.6	16	64	512	4883 → 610
	13B	24.1 → 3.0	25	100	800	7629 → 954
	30B	60.3 → 7.5	49	195	1560	14877 → 1860
	65B	121.1 → 15.1	80	320	2560	24414 → 3052
4	7B	12.6 → 1.6	64	256	2048	19531 → 2441
	13B	24.1 → 3.0	100	400	3200	30518 → 3815
	30B	60.3 → 7.5	195	780	6240	59509 → 7439
	65B	121.1 → 15.1	320	1280	10240	97656 → 12207

[1] Coleman Hopper et. al., “KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization” , NuerIPS 2024

2. Background and related work

- Challenge in KV Cache compression
- Approach to KV Cache quantization

Challenges in KV-Cache Compression

- ✓ **Both compression and decompression speeds**
  - Dynamically adding new entries to the caches as well as decoding them at inference

- ✓ **Inherent structure in caches**
  - The existence of attention sinks
  - Large outlier values

Approach to KV-Cache quantization

- ✓ **Quantization Granularity**
- ✓ **Error Handling**
- ✓ **Cross Layer merging**

Approach to KV-Cache quantization

✓ Quantization Granularity

- An approach to determining at what scope or level the scale factor should be set for quantization
- Examples include Global, Group-wise, Channel-wise, Token-wise, hybrid granularity

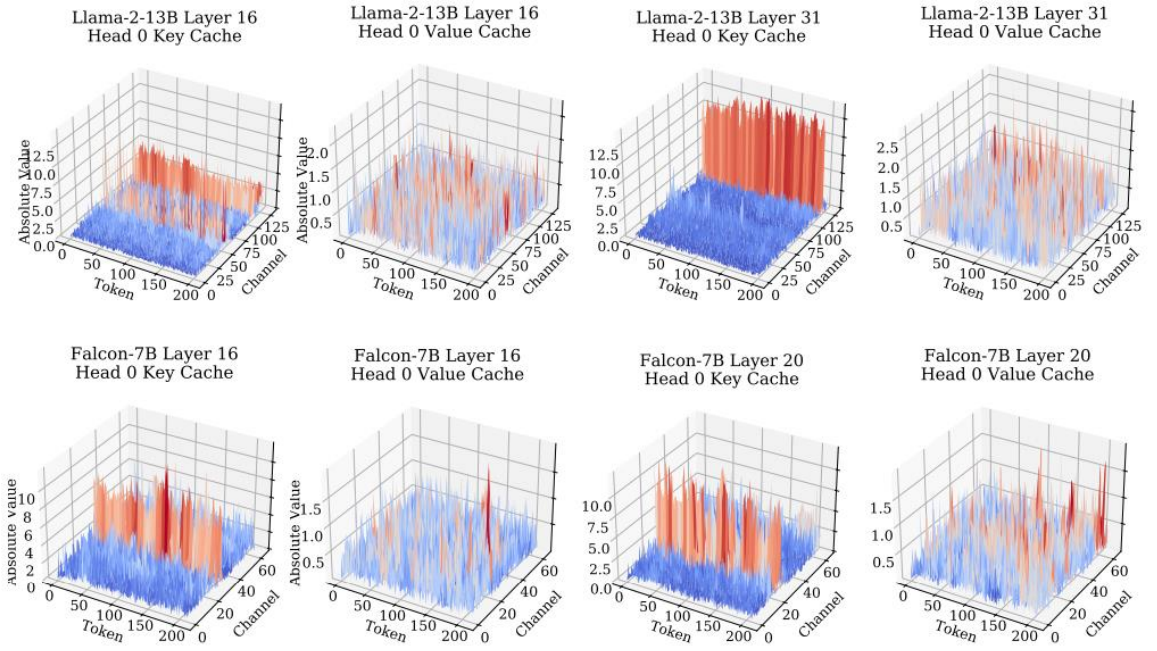


Figure 2: Magnitude of key and value cache for Llama-2-13B and Falcon-7B. We observe (1) for key cache, there are a few channels whose magnitudes are very large. (2) for value cache, there is no obvious outlier pattern.

[1] Zirui Liu et. al., “KIVI A Tuning-Free Asymmetric 2bit Quantization for KV Cache”, arxiv:2402.02750

Approach to KV-Cache quantization

✓ Error Handling

- Approaches to minimizing and correcting quantization error
- Most methods maintain a window of recent historical KV cache in full precision to preserve accuracy

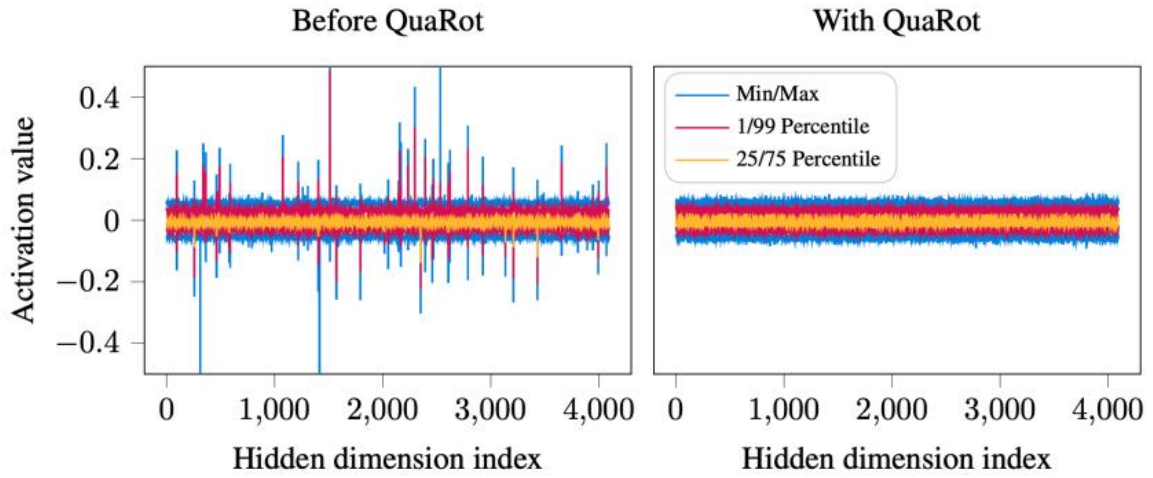


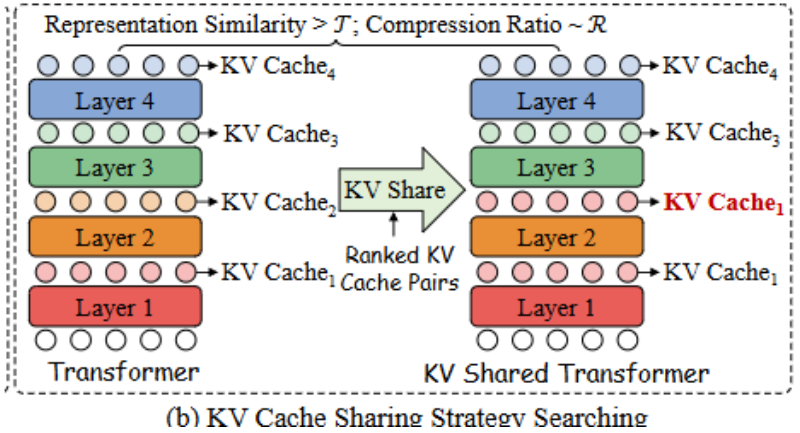
Figure 1: The distributions of activations at the input to the FFN block in LLAMA2-7B model, in the tenth layer. Left: using the default configuration as downloaded from Hugging Face. Right: after processing using QuaRot. The processed distribution has no outliers, leading to superior quantization.

[1] Saleh Ashkboos et. al., “QuaRot: Outlier-Free 4Bit Inference in Rotated LLMs” , NuerIPS 2024

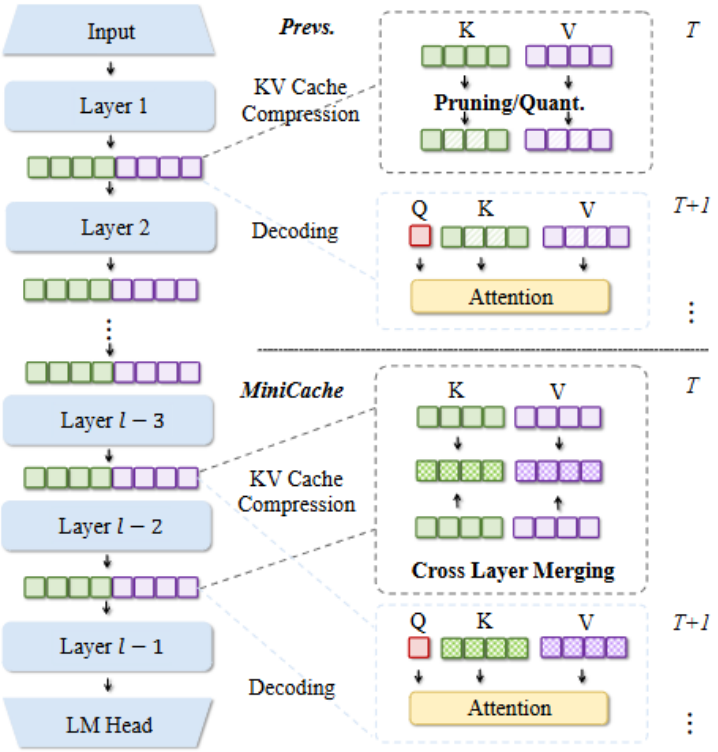
Approach to KV-Cache quantization

✓ Cross Layer merging

- Combining the KV caches from multiple layers into a single shared cache to reduce memory usage



(b) KV Cache Sharing Strategy Searching



(c) Comparison between **MiniCache** and previous methods

[1] Yifei Yang et. al., "KVSHARER: EFFICIENT INFERENCE VIA LAYER-WISE DISSIMILAR KV CACHE SHARING", arxiv:2410.18517
[2] Akide Liu et. al., "MiniCache: KV Cache Compression in Depth Dimension for Large Language Models", arxiv:2405.14366

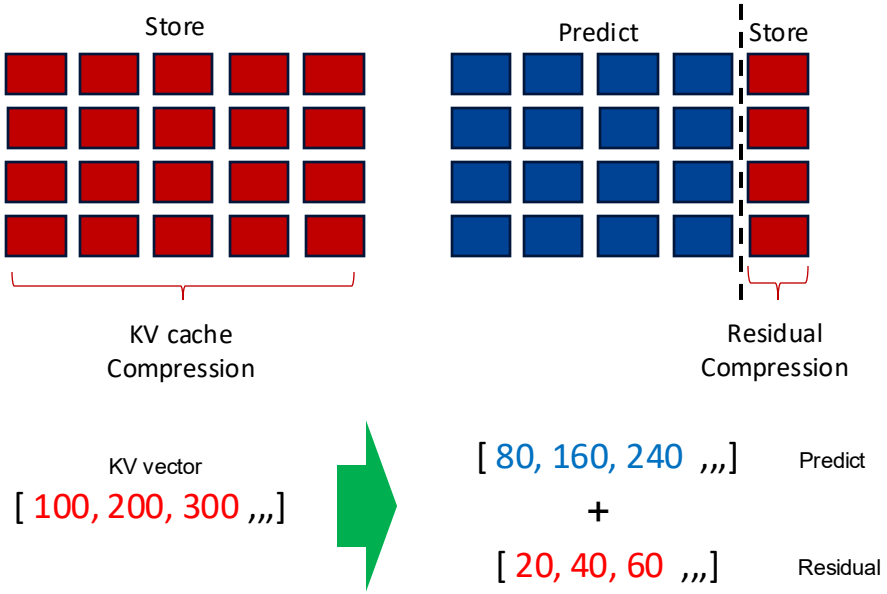
3. Method

- Core research idea
- Inter-Layer dependencies
- AQUA-KV Calibration
- AQUA-KV Inference

Core research idea

- ✓ **Remove predictable information from the KV cache and cache only the residuals after quantization**

- ✓ **Predict the current layer's KV cache based on the Key and Value of the previous layer**



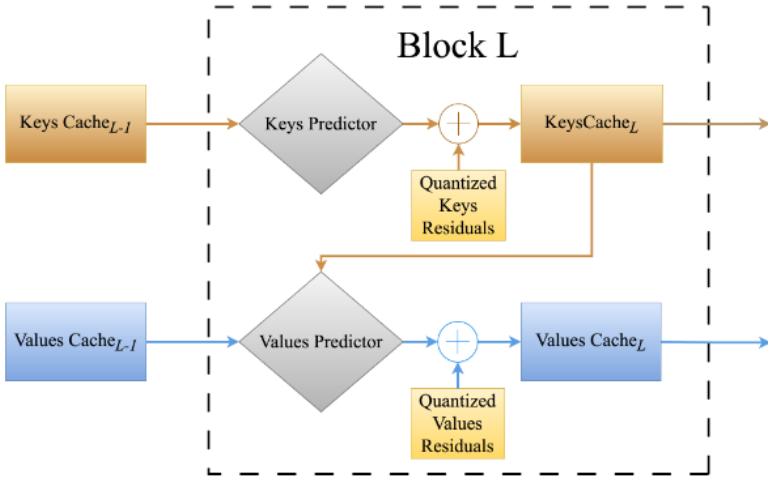
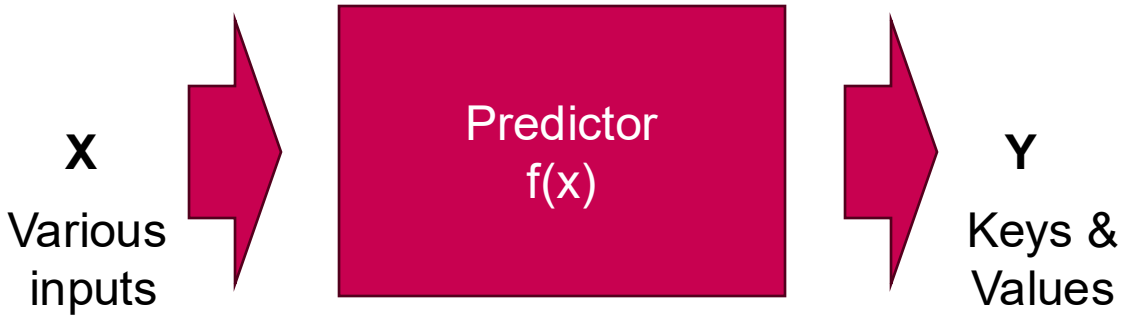


Figure 3. An intuitive scheme of the AQUA-KV inference. Only the quantized residuals are saved for each block.

Inter-Layer Dependencies

✓ Why Analyze Inter-Layer Dependencies?

- To build a predictor, the Key/Value of the current layer must be predictable
- AQUA-KV assumes a structure where the Key/Value is predicted linearly across layers
- To validate this hypothesis, the inter-layer dependency is experimentally verified across diverse inputs



Inter-Layer Dependencies

- ✓ **Linear probe model**
  - Train linear regression model
  - Prediction and interdependency analysis of i-th layer keys or values across various inputs

- ✓ **Various inputs**
  - Previous layer keys and values, adjacent tokens, and different vector types

- ✓ **Analysis of Layer Dependencies**
  - Inter Layer : cached vectors at different layer or tokens
  - Intra Layer : (key & value) vectors within the same layer
  - Extracting with the simple predictor models

Inter-Layer Dependencies

✓ Explained variance ratio

- A metric that quantifies how well the model explains the variance of the actual values.
- A higher value indicates better information reconstruction from the source input.
- Ensures fair comparison across layers with different scales (e.g., size and distribution of Keys and Values).

$$\mathrm{EVR} = 1 - \frac{\mathrm{Var}(y - \hat{y})}{\mathrm{Var}(y)}$$

- If a predictor captures 90% of the variance, it means that the subsequent quantization only needs to capture the remaining 10% of variance
- This would mean that the resulting quantization will also have roughly 10 times smaller error
- 1bit and 2bit quantizer usually explain 0.75 and 0.89 variance respectively

Inter-Layer Dependencies

✓ What Do the Results Tell Us?

- Keys and Values can be predicted based on inter-layer dependency
- For attention keys, using just one previous already achieves errors similar to 2bit quantization
- For values, the dependency on previous layer is also strong
- Strong dependencies between key and values within the same layer

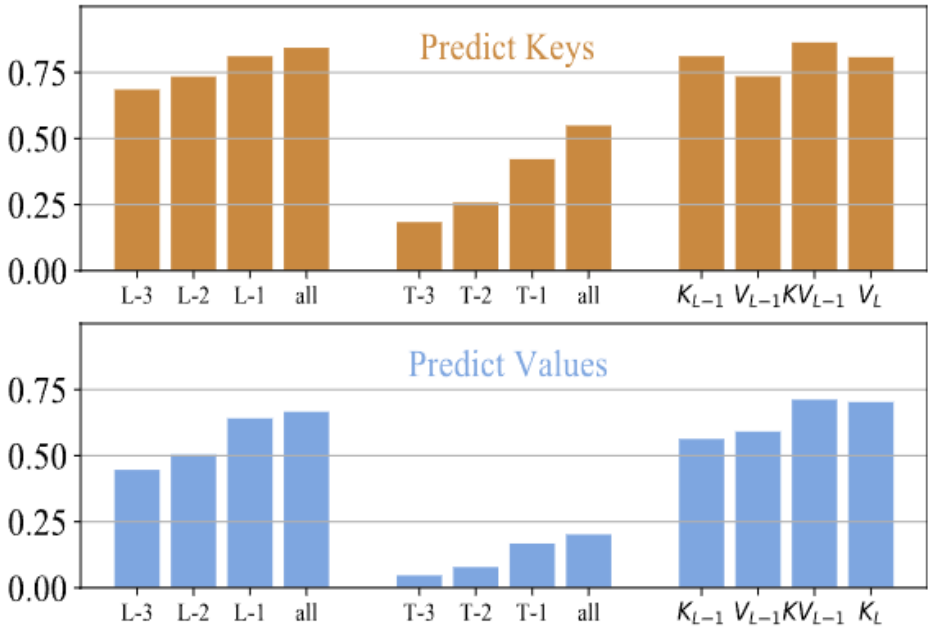


Figure 2. Mean Explained Variance Ratios by linear probes from previous blocks (L), tokens (T) and role on Llama-3.2-3B.

AQUA-KV Calibration

✓ AQUA-KV

- Exploits inter- and intra-layer dependencies to improve quantization accuracy

✓ Predictors

- Per layer
  - Key predictor, Value Predictor
- Sequential Prediction
  - Operate layer-by-layer
- Reflect the way predictors are used
  - For training: reconstructed cache entries as input
- Key Predictor
  - previous layer keys
- Value Predictor
  - previous layer keys + current layer keys
  - Cannot predict in both direction within same layer
    - $V_{L,rec} \rightarrow K_L,\ K_{L,rec} \rightarrow V_L$
    - Values are harder to predict

Algorithm 1 AQUA-KV Calibration

Require: model, data, quantization method Q

$\mathbf{X} \leftarrow$ `model.input_embeddings(data)`
$\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \emptyset$
`predictors` $\leftarrow \{\}$
for $i = 1, \ldots,$ `model.num_layers` **do**
  `block` $\leftarrow$ `model.transformer_layers`$[i]$
  $(\mathbf{K}, \mathbf{V}) \leftarrow$ `block.get_attention_kv`$(\mathbf{X})$
  $\mathbf{X} \leftarrow$ `block`$(\mathbf{X})$
  if $\mathbf{K}_{\text{old}} = \emptyset$ **and** $\mathbf{V}_{\text{old}} = \emptyset$ **then**
    $\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \mathbf{K}, \mathbf{V}$
    continue
  end if
  # Predict keys from past keys
  $f_{\text{key}} \leftarrow \arg\min_f \|f(\mathbf{K}_{\text{old}}) - \mathbf{K}\|_2^2$
  $\mathbf{K}_{\text{residue}} \leftarrow Q^{-1}(Q(\mathbf{K} - f_{\text{key}}(\mathbf{K}_{\text{old}})))$
  $\mathbf{K}_{\text{rec}} \leftarrow f_{\text{key}}(\mathbf{K}_{\text{old}}) + \mathbf{K}_{\text{residue}}$
  # Predict values from past values and current keys
  $f_{\text{value}} \leftarrow \arg\min_f \|f([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}]) - \mathbf{V}\|_2^2$
  $\mathbf{V}_{\text{residue}} \leftarrow Q^{-1}(Q(\mathbf{V} - f_{\text{value}}([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}])))$
  $\mathbf{V}_{\text{rec}} \leftarrow f_{\text{value}}([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}]) + \mathbf{V}_{\text{residue}}$
  `predictors[i]` $= (f_{\text{key}}, f_{\text{value}})$
  $\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \mathbf{K}_{\text{rec}}, \mathbf{V}_{\text{rec}}$
end for
return `predictors`

AQUA-KV Calibration

✓ AQUA-KV

- Exploits inter- and intra-layer dependencies to improve quantization accuracy

✓ **Predictors**

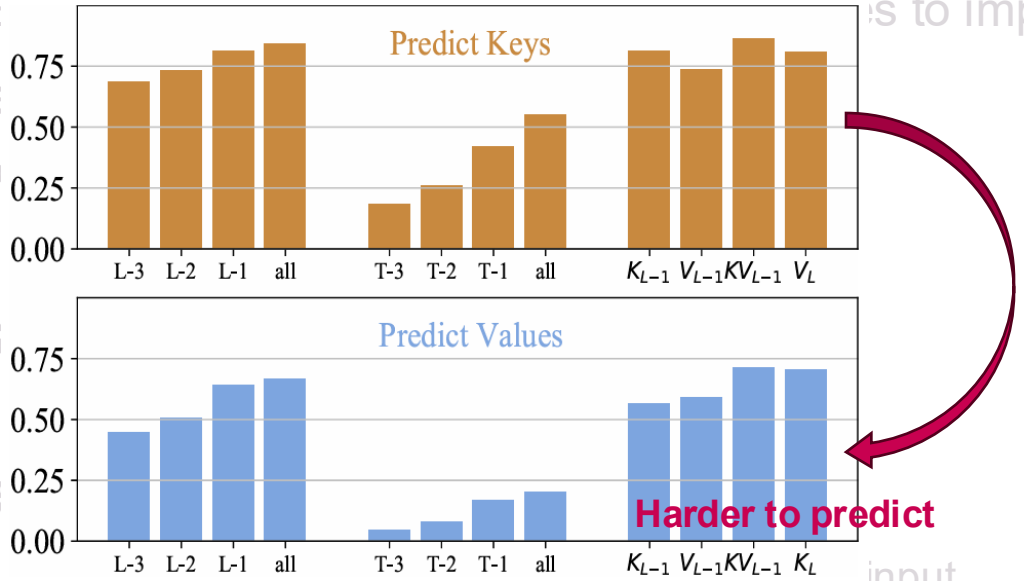
- Per layer
  - Key predictor, Value Predictor
- Sequential Prediction
  - Operate layer-by-layer
- Reflect the way predictors are used
  - For training: reconstructed cache entries as input
- Key Predictor
  - previous layer keys
- Value Predictor
  - previous layer keys + current layer keys
  - Cannot predict in both direction within same layer
    - $V_{L,rec} \rightarrow K_L,\ K_{L,rec} \rightarrow V_L$
    - Values are harder to predict

Algorithm 1 AQUA-KV Calibration

Require: model, data, quantization method Q

$\mathbf{X} \leftarrow$ `model.input_embeddings(data)`
$\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \emptyset$
`predictors` $\leftarrow \{\}$
for $i = 1, \ldots,$ `model.num_layers` **do**
  `block` $\leftarrow$ `model.transformer_layers`$[i]$
  $(\mathbf{K}, \mathbf{V}) \leftarrow$ `block.get_attention_kv`$(\mathbf{X})$
  $\mathbf{X} \leftarrow$ `block`$(\mathbf{X})$
  if $\mathbf{K}_{\text{old}} = \emptyset$ **and** $\mathbf{V}_{\text{old}} = \emptyset$ **then**
    $\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \mathbf{K}, \mathbf{V}$
    continue
  end if
  # Predict keys from past keys
  $f_{\text{key}} \leftarrow \arg\min_f \|f(\mathbf{K}_{\text{old}}) - \mathbf{K}\|_2^2$
  $\mathbf{K}_{\text{residue}} \leftarrow Q^{-1}(Q(\mathbf{K} - f_{\text{key}}(\mathbf{K}_{\text{old}})))$
  $\mathbf{K}_{\text{rec}} \leftarrow f_{\text{key}}(\mathbf{K}_{\text{old}}) + \mathbf{K}_{\text{residue}}$
  # Predict values from past values and current keys
  $f_{\text{value}} \leftarrow \arg\min_f \|f([\mathbf{V}_{\text{old}}\ \mathbf{K}_{\text{rec}}]) - \mathbf{V}\|_2^2$
  $\mathbf{V}_{\text{residue}} \leftarrow Q^{-1}(Q(\mathbf{V} - f_{\text{value}}([\mathbf{V}_{\text{old}}\ \mathbf{K}_{\text{rec}}])))$
  $\mathbf{V}_{\text{rec}} \leftarrow f_{\text{value}}([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}]) + \mathbf{V}_{\text{residue}}$
  `predictors[i]` $= (f_{\text{key}}, f_{\text{value}})$
  $\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \mathbf{K}_{\text{rec}}, \mathbf{V}_{\text{rec}}$
end for
return `predictors`

AQUA-KV Calibration



Algorithm 1 AQUA-KV Calibration

Require: model, data, quantization method Q

$\mathbf{X} \leftarrow$ `model.input_embeddings(data)`
$\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \emptyset$
`predictors` $\leftarrow \{\}$
for $i = 1, \ldots,$ `model.num_layers` **do**
  `block` $\leftarrow$ `model.transformer_layers`$[i]$
  $(\mathbf{K}, \mathbf{V}) \leftarrow$ `block.get_attention_kv`$(\mathbf{X})$
  $\mathbf{X} \leftarrow$ `block`$(\mathbf{X})$
  if $\mathbf{K}_{\text{old}} = \emptyset$ **and** $\mathbf{V}_{\text{old}} = \emptyset$ **then**
    $\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \mathbf{K}, \mathbf{V}$
    continue
  end if
  # Predict keys from past keys
  $f_{\text{key}} \leftarrow \arg\min_f \|f(\mathbf{K}_{\text{old}}) - \mathbf{K}\|_2^2$
  $\mathbf{K}_{\text{residue}} \leftarrow Q^{-1}(Q(\mathbf{K} - f_{\text{key}}(\mathbf{K}_{\text{old}})))$
  $\mathbf{K}_{\text{rec}} \leftarrow f_{\text{key}}(\mathbf{K}_{\text{old}}) + \mathbf{K}_{\text{residue}}$
  # Predict values from past values and current keys
  $f_{\text{value}} \leftarrow \arg\min_f \|f([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}]) - \mathbf{V}\|_2^2$
  $\mathbf{V}_{\text{residue}} \leftarrow Q^{-1}(Q(\mathbf{V} - f_{\text{value}}([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}])))$
  $\mathbf{V}_{\text{rec}} \leftarrow f_{\text{value}}([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}]) + \mathbf{V}_{\text{residue}}$
  `predictors[i]` $= (f_{\text{key}}, f_{\text{value}})$
  $\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \mathbf{K}_{\text{rec}}, \mathbf{V}_{\text{rec}}$
end for
return `predictors`

- Key Predictor
  - previous layer keys
- Value Predictor
  - previous layer keys + current layer keys
  - Cannot predict in both direction within same layer
    - $V_{L,rec} \rightarrow K_L,\ K_{L,rec} \rightarrow V_L$
    - Values are harder to predict

AQUA-KV Calibration

✓ AQUA-KV

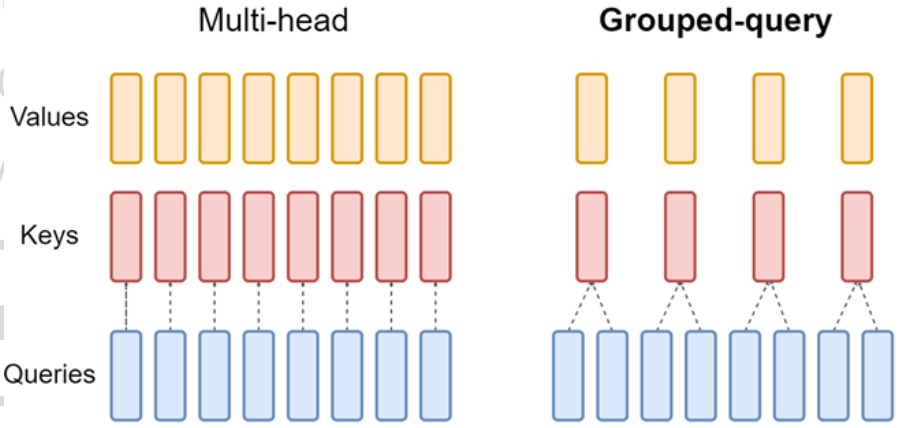
- Exploits inter- and intra-layer dependencies to improve quantization accuracy

✓ **Lightweight Linear Regressors**

- Fast training with closed-form solutions
  - $W^* = (X^T X)^{-1} X^T y$
  - ~ 4 hours training for Llama-3.1-70B
- Low compute & memory overhead
  - Modern LLMs: Grouped Query Attention (GQA)

✓ Quantization

- (
- /
- I
- Improves accuracy on recent tokens



icted

Algorithm 1 AQUA-KV Calibration

Require: model, data, quantization method Q

$\mathbf{X} \leftarrow$ `model.input_embeddings(data)`
$\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \emptyset$
`predictors` $\leftarrow \{\}$
for $i = 1, \ldots,$ `model.num_layers` **do**
  `block` $\leftarrow$ `model.transformer_layers`$[i]$
  $(\mathbf{K}, \mathbf{V}) \leftarrow$ `block.get_attention_kv`$(\mathbf{X})$
  $\mathbf{X} \leftarrow$ `block`$(\mathbf{X})$
  if $\mathbf{K}_{\text{old}} = \emptyset$ **and** $\mathbf{V}_{\text{old}} = \emptyset$ **then**
    $\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \mathbf{K}, \mathbf{V}$
    continue
  end if
  # Predict keys from past keys
  $f_{\text{key}} \leftarrow \arg\min_f \|f(\mathbf{K}_{\text{old}}) - \mathbf{K}\|_2^2$
  $\mathbf{K}_{\text{residue}} \leftarrow Q^{-1}(Q(\mathbf{K} - f_{\text{key}}(\mathbf{K}_{\text{old}})))$
  $\mathbf{K}_{\text{rec}} \leftarrow f_{\text{key}}(\mathbf{K}_{\text{old}}) + \mathbf{K}_{\text{residue}}$
  # Predict values from past values and current keys
  $f_{\text{value}} \leftarrow \arg\min_f \|f([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}]) - \mathbf{V}\|_2^2$
  $\mathbf{V}_{\text{residue}} \leftarrow Q^{-1}(Q(\mathbf{V} - f_{\text{value}}([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}])))$
  $\mathbf{V}_{\text{rec}} \leftarrow f_{\text{value}}([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}]) + \mathbf{V}_{\text{residue}}$
  `predictors[i]` $= (f_{\text{key}}, f_{\text{value}})$
  $\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \mathbf{K}_{\text{rec}}, \mathbf{V}_{\text{rec}}$
end for
return `predictors`

[1] Ainslie, Joshua, et al. "Gqa: Training generalized multi-query transformer models from multi-head checkpoints." *arXiv preprint arXiv:2305.13245* (2023).

AQUA-KV Calibration

✓ AQUA-KV

- Exploits inter- and intra-layer dependencies to improve quantization accuracy

✓ Lightweight Linear Regressors

- Fast training with closed-form solutions
  - $W^* = (X^T X)^{-1} X^T y$
  - ~ 4 hours training for Llama-3.1-70B
- Low compute & memory overhead
  - Modern LLMs: Grouped Query Attention (GQA)

✓ **Quantization**

- Only the residual parts that cannot be predicted
- Attention Sink
  - Keep the first 4 tokens uncompressed
- Recent token buffer
  - Up to 128 tokens uncompressed
  - Improves accuracy on recent tokens

Algorithm 1 AQUA-KV Calibration

Require: model, data, quantization method Q

$\mathbf{X} \leftarrow$ `model.input_embeddings(data)`
$\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \emptyset$
`predictors` $\leftarrow \{\}$
for $i = 1, \ldots,$ `model.num_layers` **do**
  `block` $\leftarrow$ `model.transformer_layers`$[i]$
  $(\mathbf{K}, \mathbf{V}) \leftarrow$ `block.get_attention_kv`$(\mathbf{X})$
  $\mathbf{X} \leftarrow$ `block`$(\mathbf{X})$
  if $\mathbf{K}_{\text{old}} = \emptyset$ **and** $\mathbf{V}_{\text{old}} = \emptyset$ **then**
    $\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \mathbf{K}, \mathbf{V}$
    continue
  end if
  # Predict keys from past keys
  $f_{\text{key}} \leftarrow \arg\min_f \|f(\mathbf{K}_{\text{old}}) - \mathbf{K}\|_2^2$
  $\mathbf{K}_{\text{residue}} \leftarrow Q^{-1}(Q(\mathbf{K} - f_{\text{key}}(\mathbf{K}_{\text{old}})))$
  $\mathbf{K}_{\text{rec}} \leftarrow f_{\text{key}}(\mathbf{K}_{\text{old}}) + \mathbf{K}_{\text{residue}}$
  # Predict values from past values and current keys
  $f_{\text{value}} \leftarrow \arg\min_f \|f([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}]) - \mathbf{V}\|_2^2$
  $\mathbf{V}_{\text{residue}} \leftarrow Q^{-1}(Q(\mathbf{V} - f_{\text{value}}([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}])))$
  $\mathbf{V}_{\text{rec}} \leftarrow f_{\text{value}}([\mathbf{V}_{\text{old}}; \mathbf{K}_{\text{rec}}]) + \mathbf{V}_{\text{residue}}$
  `predictors[i]` $= (f_{\text{key}}, f_{\text{value}})$
  $\mathbf{K}_{\text{old}}, \mathbf{V}_{\text{old}} \leftarrow \mathbf{K}_{\text{rec}}, \mathbf{V}_{\text{rec}}$
end for
return `predictors`

AQUA-KV Inference

✓ Inference

- Decode → Computation → Encode
  - Reconstruct KV-cache from a compressed representation

Algorithm 4 `Inference with AQUA-KV Predictors`

Require: `model, input, key_cache, value_cache, predictors`

$\hat{\mathbf{K}}_{\text{past}}, \hat{\mathbf{V}}_{\text{past}}, \hat{\mathbf{K}}_{\text{inp\_prev}}, \hat{\mathbf{V}}_{\text{inp\_prev}} \leftarrow \emptyset$
$\mathbf{X} \leftarrow$ `model.input_embeddings(input)`
for $i = 0, \ldots,$ `model.num_layers` $- 1$ **do**
  # Recover previously saved key-values
  $(\hat{\mathbf{K}}_{\text{past}}, \hat{\mathbf{V}}_{\text{past}}) \leftarrow$ `decode`$(i,$ `key_cache`$[i],$ `value_cache`$[i], \hat{\mathbf{K}}_{\text{past}}, \hat{\mathbf{V}}_{\text{past}},$ `predictors`$)$
  # Run forward pass
  `block` $\leftarrow$ `model.transformer_layers`$[i]$
  $(\mathbf{K}_{\text{inp}}, \mathbf{V}_{\text{inp}}) \leftarrow$ `block.get_attention_kv`$(\mathbf{X})$
  $\mathbf{X} \leftarrow$ `block`$(\mathbf{X}, K = [\hat{\mathbf{K}}_{\text{past}}; \mathbf{K}_{\text{inp}}], V = [\hat{\mathbf{V}}_{\text{past}}, \mathbf{V}_{\text{inp}}])$ {Concatenate past and input keys/values}
  # Compress new key-value entries
  $(\mathbf{K}^q_{\text{inp}}, \mathbf{V}^q_{\text{inp}}) \leftarrow$ `encode`$(i, \hat{\mathbf{K}}_{\text{inp}}, \mathbf{V}_{\text{inp}}, \hat{\mathbf{K}}_{\text{inp\_prev}}, \mathbf{V}_{\text{inp\_prev}},$ `predictors`$)$
  `key_cache`$[i] \leftarrow [$`key_cache`$[i]; \mathbf{K}^q_{\text{inp}}]$ {Concatenate key cache with new keys}
  `value_cache`$[i] \leftarrow [$`value_cache`$[i]; \mathbf{V}^q_{\text{inp}}]$ {Concatenate value cache with new values}
  $(\hat{\mathbf{K}}_{\text{inp\_prev}}, \hat{\mathbf{V}}_{\text{inp\_prev}}) \leftarrow$ `decode`$(i, \mathbf{K}^q_{\text{inp}}, \mathbf{V}^q_{\text{inp}}, \hat{\mathbf{K}}_{\text{inp\_prev}}, \hat{\mathbf{V}}_{\text{inp\_prev}},$ `predictors`$)$
end for
return `model.compute_logits`$(\mathbf{X})$

AQUA-KV Inference

✓ Inference

- Decode → Computation → Encode
  - Inference combines the current token's KV with the reconstructed cache from decode step

Algorithm 4 Inference with AQUA-KV Predictors

Require: model, input, key_cache, value_cache, predictors

$\hat{\mathbf{K}}_{\text{past}}, \hat{\mathbf{V}}_{\text{past}}, \hat{\mathbf{K}}_{\text{inp\_prev}}, \hat{\mathbf{V}}_{\text{inp\_prev}} \leftarrow \emptyset$
$\mathbf{X} \leftarrow$ `model.input_embeddings(input)`
for $i = 0, \ldots,$ `model.num_layers` $- 1$ **do**
  # Recover previously saved key-values
  $(\hat{\mathbf{K}}_{\text{past}}, \hat{\mathbf{V}}_{\text{past}}) \leftarrow$ `decode`$(i,$ `key_cache`$[i],$ `value_cache`$[i], \hat{\mathbf{K}}_{\text{past}}, \hat{\mathbf{V}}_{\text{past}},$ `predictors`$)$
  # Run forward pass
  `block` $\leftarrow$ `model.transformer_layers`$[i]$
  $(\mathbf{K}_{\text{inp}}, \mathbf{V}_{\text{inp}}) \leftarrow$ `block.get_attention_kv`$(\mathbf{X})$
  $\mathbf{X} \leftarrow$ `block`$(\mathbf{X}, K = [\hat{\mathbf{K}}_{\text{past}}; \mathbf{K}_{\text{inp}}], V = [\hat{\mathbf{V}}_{\text{past}}, \mathbf{V}_{\text{inp}}])$ {Concatenate past and input keys/values}
  # Compress new key-value entries
  $(\mathbf{K}^q_{\text{inp}}, \mathbf{V}^q_{\text{inp}}) \leftarrow$ `encode`$(i, \hat{\mathbf{K}}_{\text{inp}}, \mathbf{V}_{\text{inp}}, \hat{\mathbf{K}}_{\text{inp\_prev}}, \mathbf{V}_{\text{inp\_prev}},$ `predictors`$)$
  `key_cache`$[i] \leftarrow [$`key_cache`$[i]; \mathbf{K}^q_{\text{inp}}]$ {Concatenate key cache with new keys}
  `value_cache`$[i] \leftarrow [$`value_cache`$[i]; \mathbf{V}^q_{\text{inp}}]$ {Concatenate value cache with new values}
  $(\hat{\mathbf{K}}_{\text{inp\_prev}}, \hat{\mathbf{V}}_{\text{inp\_prev}}) \leftarrow$ `decode`$(i, \mathbf{K}^q_{\text{inp}}, \mathbf{V}^q_{\text{inp}}, \hat{\mathbf{K}}_{\text{inp\_prev}}, \hat{\mathbf{V}}_{\text{inp\_prev}},$ `predictors`$)$
end for
return `model.compute_logits`$(\mathbf{X})$

AQUA-KV Inference

✓ Inference

- Decode → Computation → Encode
  - Only residual part that the predictor cannot predict is compressed and stored

Algorithm 4 Inference with AQUA-KV Predictors

Require: model, input, key_cache, value_cache, predictors

$\hat{\mathbf{K}}_{\text{past}}, \hat{\mathbf{V}}_{\text{past}}, \hat{\mathbf{K}}_{\text{inp\_prev}}, \hat{\mathbf{V}}_{\text{inp\_prev}} \leftarrow \emptyset$
$\mathbf{X} \leftarrow$ model.input_embeddings(input)
for $i = 0, \ldots,$ model.num_layers $- 1$ **do**
  # Recover previously saved key-values
  $(\hat{\mathbf{K}}_{\text{past}}, \hat{\mathbf{V}}_{\text{past}}) \leftarrow \text{decode}(i, \text{key\_cache}[i], \text{value\_cache}[i], \hat{\mathbf{K}}_{\text{past}}, \hat{\mathbf{V}}_{\text{past}}, \text{predictors})$
  # Run forward pass
  block $\leftarrow$ model.transformer_layers$[i]$
  $(\mathbf{K}_{\text{inp}}, \mathbf{V}_{\text{inp}}) \leftarrow$ block.get_attention_kv$(\mathbf{X})$
  $\mathbf{X} \leftarrow \text{block}(\mathbf{X}, K = [\hat{\mathbf{K}}_{\text{past}}; \mathbf{K}_{\text{inp}}], V = [\hat{\mathbf{V}}_{\text{past}}, \mathbf{V}_{\text{inp}}])$ {Concatenate past and input keys/values}
  # Compress new key-value entries
  $(\mathbf{K}^q_{\text{inp}}, \mathbf{V}^q_{\text{inp}}) \leftarrow \text{encode}(i, \hat{\mathbf{K}}_{\text{inp}}, \mathbf{V}_{\text{inp}}, \hat{\mathbf{K}}_{\text{inp\_prev}}, \mathbf{V}_{\text{inp\_prev}}, \text{predictors})$
  key_cache$[i] \leftarrow [\text{key_cache}[i]; \mathbf{K}^q_{\text{inp}}]$ {Concatenate key cache with new keys}
  value_cache$[i] \leftarrow [\text{value_cache}[i]; \mathbf{V}^q_{\text{inp}}]$ {Concatenate value cache with new values}
  $(\hat{\mathbf{K}}_{\text{inp\_prev}}, \hat{\mathbf{V}}_{\text{inp\_prev}}) \leftarrow \text{decode}(i, \mathbf{K}^q_{\text{inp}}, \mathbf{V}^q_{\text{inp}}, \hat{\mathbf{K}}_{\text{inp\_prev}}, \hat{\mathbf{V}}_{\text{inp\_prev}}, \text{predictors})$
end for
return model.compute_logits$(\mathbf{X})$

4. Evaluation

- Evaluation Settings
- Detailed Evaluation
- Large-Scale Evaluation

Evaluation Settings

- ✓ **Predictor Calibration**
  - Dataset: RedPajama
    - Random 256 sequences with 8192 tokens
    - Use 32 of sequences as holdout for hyperparameter selection
    - Use remaining 224 for training the predictors
- ✓ **Perplexity evaluation**
  - Account for the effect of recent token buffers and attention sinks
- ✓ **LongBench v1**
  - 14 English-language tasks
    - SamSum, 2WikiMQ, TREC, HotpotAQ, MultiNews, TriviaQA, QMSum, PsgCount, MFQA_en, Musique, Qasper, PsgRetr, NarrativeQA, GovReport
  - Without restricting the input 8192 tokens
    - To better explore the effectiveness of AQUA-KV on longer sequences

Detailed Evaluation

✓ Alternative Quantizer

- Addition of AQUA-KV to Quanto
  - Group size: 64, per token quantization
  - A significant improvement in both PPL and LongBench avg scores
- Addition of AQUA-KV to HIGGS
  - $d = 2, n \in \{16,64,256\}$, 2-, 3-, 4-bit for each
  - Outperforms in both PPL and LongBench avg scores

✓ Layer Sharing

- Sharing multiple layer pairs causes major accuracy drops

Config	Quant. Bits	Wiki2 PPL↓ (base model)	LongBench Avg.↑ (instruct model)
Uncompressed	16	6.98	44.47
Quanto-2b-gs64	2.50	21.56	33.59
KIVI-2b-gs128-r128	2.25	9.33	39.63
KVQuant-2b-s1%	2.33	9.43	20.56
HIGGS-2b-gs1024	2.02	7.47	43.25
KVSharer (1 pair)	15.43	7.45	36.81
KVSharer (4 pairs)	13.71	9.60	29.82
AQUA-KV (Quanto)	2.64	10.33	43.64
AQUA-KV (HIGGS)	2.16	**7.03**	**44.26**

Detailed Evaluation

✓ Predictor Architecture

- Linear regression (main proposal)
- Reduce-Rank Regression(with rank=256)
  - Slight increase in PPL😢
- MLP with two layers
  - Double hidden dimension + Layer normalization
  - Slightly better performance😁
  - A significant cost in size and speed😢
- Linear regressor + GPTQ
  - Effective for scenarios where minimal model size is crucial

Config	Quant. Bits	Wiki2 PPL↓ (base model)	LongBench Avg.↑ (instruct model)
AQUA-KV Predictor Architecture			
Linear (162 MiB)	2.16	7.03	44.26
MLP (540 MiB)	2.16	7.03	44.61
RRR (68 MiB)	2.16	7.22	44.30
GPTQ (41 MiB)	2.16	7.03	44.19

Detailed Evaluation

✓ First Layer Keys & Values

- Use 4-bit HIGGS quantization
  - Since the first layer is not compressed by AQUA-KV predictors

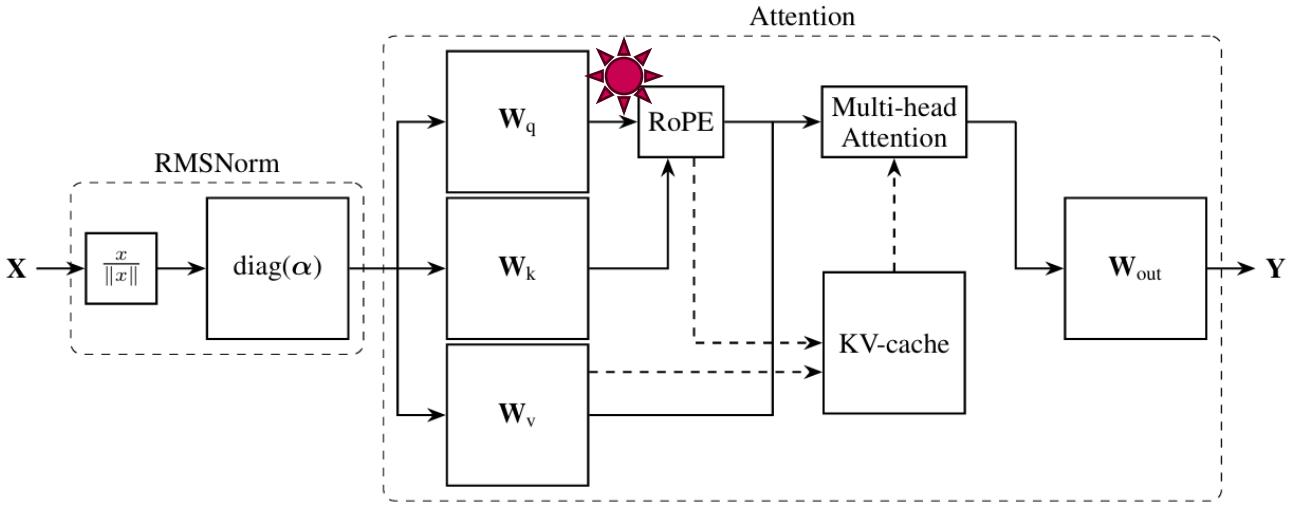
Config	Quant. Bits	Wiki2 PPL↓ (base model)	LongBench Avg.↑ (instruct model)
AQUA-KV 1st Layer Keys & Values			
Keep in BF16	2.59	7.03	44.26
HIGGS 4 bit	2.16	7.03	44.26
HIGGS 3 bit	2.13	7.03	44.28
HIGGS 2 bit	2.09	7.05	44.41

Detailed Evaluation

✓ Rotary Positional Embedding (RoPE)

- Natural dilemma about whether to apply predictors and quantizers
  - Better in Pre-RoPE
  - Simple linear models are not rotation-equivariant

Config	Quant. Bits	Wiki2 PPL↓ (base model)	LongBench Avg.↑ (instruct model)
Before RoPE HIGGS	2.16	7.03	44.26
After RoPE HIGGS	2.16	7.05	44.13
Before RoPE Quanto gs64	2.64	10.90	44.54
After RoPE Quanto gs64	2.64	10.33	43.64



Detailed Evaluation

✓ Ablation Analysis

Config	Quant. Bits	Wiki2 PPL↓ (base model)	LongBench Avg.↑ (instruct model)
Ablation Analysis			
AQUA-KV (default)	2.16	7.03	44.26
w/o 16-bit Attn. Sink	2.16	7.15	44.13
w/o V predictor	2.09	7.06	43.91
w/o K predictor	2.09	7.50	42.92
w/o pre-RoPE	2.16	7.05	44.13

Large-Scale Evaluation

✓ Evaluation of AQUA-KV with HIGGS backbone

- Particularly noticeable for extreme 2-bit compression

Method	Quant. bits	Cache size GiB, 70B	WikiText-2 PPL↓ (base model) Llama 3.x 3B	Llama 3.x 8B	Llama 3.x 70B	Qwen 2.5 3B	Qwen 2.5 7B	LongBench Average↑ (instruct) Llama 3.x 3B	Llama 3.x 8B	Llama 3.x 70B	Qwen 2.5 3B	Qwen 2.5 7B
Uncompressed	16	40	6.98	5.61	2.54	7.14	6.13	44.61	48.13	52.92	38.80	46.82
AQUA-KV	2.09	5.7	**7.03**	**5.72**	**2.62**	**7.20**	**6.17**	**44.30**	**47.77**	**52.79**	**38.31**	**46.43**
HIGGS	2.02	5.1	7.47	5.89	2.77	7.93	8.08	42.80	47.37	52.18	30.92	25.97
KIVI	2.25	5.6	9.34	7.37	3.06	9.05	7.02	39.64	46.28	52.45	28.66	32.78
KVQuant	2.33	5.6	9.43	6.64	3.28	—	—	20.56	37.17	46.14	—	—
AQUA-KV	3.06	8.1	**6.98**	**5.64**	**2.55**	**7.15**	**6.14**	44.37	**48.10**	52.81	**38.77**	**46.81**
HIGGS	3.02	7.6	7.05	5.66	2.57	7.26	7.20	**44.41**	47.86	52.56	31.85	14.61
KIVI	3.05	7.7	7.87	6.04	2.87	7.63	6.37	41.40	46.98	**52.87**	30.37	32.63
KVQuant	3.33	8.3	7.26	5.84	2.75	—	—	41.40	46.42	50.74	—	—
AQUA-KV	4.02	10.9	**6.98**	**5.61**	**2.54**	**7.15**	**6.14**	**44.48**	**48.10**	52.95	**38.92**	**46.77**
HIGGS	4.02	10.6	7.01	5.62	2.55	7.16	6.88	44.41	48.07	**52.97**	32.11	11.54
KIVI	4.25	10.6	7.03	5.64	2.61	7.17	**6.14**	43.11	47.57	52.88	31.50	33.40
KVQuant	4.33	10.8	7.04	5.65	2.58	—	—	43.62	47.77	52.89	—	—

Large-Scale Evaluation

✓ Evaluation of AQUA-KV with HIGGS backbone

- Particularly noticeable for extreme 2-bit compression

Method	Quant. bits	Cache size GiB, 70B	WikiText-2 PPL↓ (base model) Llama 3.x 3B	Llama 3.x 8B	Llama 3.x 70B	Qwen 2.5 3B	Qwen 2.5 7B	LongBench Average↑ (instruct) Llama 3.x 3B	Llama 3.x 8B	Llama 3.x 70B	Qwen 2.5 3B	Qwen 2.5 7B
Uncompressed	16	40	6.98	5.61	2.54	7.14	6.13	44.61	48.13	52.92	38.80	46.82
AQUA-KV	2.09	5.7	**7.03**	**5.72**	**2.62**	**7.20**	**6.17**	**44.30**	**47.77**	**52.79**	**38.31**	**46.43**
HIGGS	2.02	5.1	7.47	5.89	2.77	7.93	8.08	42.80	47.37	52.18	30.92	25.97
KIVI												
KVQuant												
AQUA-KV	3.06	8.1	**6.98**	**5.64**	**2.55**	**7.15**	**6.14**	44.37	**48.10**	52.81	**38.77**	**46.81**
HIGGS	3.02	7.6	7.05	5.66	2.57	7.26	7.20	**44.41**	47.86	52.56	31.85	14.61
KIVI	3.05	7.7	7.87	6.04	2.87	7.63	6.37	41.40	46.98	**52.87**	30.37	32.63
KVQuant	3.33	8.3	7.26	5.84	2.75	—	—	41.40	46.42	50.74	—	—
AQUA-KV	4.02	10.9	**6.98**	**5.61**	**2.54**	**7.15**	**6.14**	**44.48**	**48.10**	52.95	**38.92**	**46.77**
HIGGS	4.02	10.6	7.01	5.62	2.55	7.16	6.88	44.41	48.07	**52.97**	32.11	11.54
KIVI	4.25	10.6	7.03	5.64	2.61	7.17	**6.14**	43.11	47.57	52.88	31.50	33.40
KVQuant	4.33	10.8	7.04	5.65	2.58	—	—	43.62	47.77	52.89	—	—

AQUA-KV for 2 bit compression is roughly equivalent to the 3-bit HIGGS baseline quantizer

5. Discussion

- Summary
- Limitations & further research
- Our Observed Limitations

Conclusion

✓ Summary

- Analysis of KV cache structure
  - Identified compression opportunities via mutual information
- AQUA-KV compression framework
  - Exploit inter- and intra-layer dependencies
  - Compatible with any quantization

✓ Limitations & further research

- Inference speed
  - Depends heavily on the backbone quantizer
  - Cannot be faster than the baseline
  - For Llama-70B
    - HIGSS: 5.91 tokens/s
    - AQUA-KV: 5.76 tokens/s
- Unoptimized pytorch can be improved with specialized libraries.

Algorithm 2 `encode`

Require: `layer_index`, $\mathbf{K}$, $\mathbf{V}$, reconstructed keys and values of the previous layer $\hat{\mathbf{K}}_{\text{prev}}$, $\hat{\mathbf{V}}_{\text{prev}}$, `predictors`

1: **if** `layer_index` $= 0$ **then**
2: **return** $\mathbf{K}, \mathbf{V}$
3: **end if**
4: $f_{\text{key}}, f_{\text{value}} \leftarrow$ `predictors[layer_index]`
5: $\mathbf{K}_{\text{pred}} \leftarrow f_{\text{key}}(\hat{\mathbf{K}}_{\text{prev}})$
6: $\mathbf{K}_q \leftarrow Q(\mathbf{K} - \mathbf{K}_{\text{pred}})$
7: $\hat{\mathbf{K}} \leftarrow Q^{-1}(\mathbf{K}_q) + \mathbf{K}_{\text{pred}}$
8: $\mathbf{V}_{\text{pred}} \leftarrow f_{\text{value}}([\hat{\mathbf{V}}_{\text{prev}}; \hat{\mathbf{K}}])$ {Concatenate $\hat{\mathbf{V}}_{\text{prev}}$ and $\hat{\mathbf{K}}$}
9: $\mathbf{V}_q \leftarrow Q(\mathbf{V} - \mathbf{V}_{\text{pred}})$
10: **return** $\mathbf{K}_q, \mathbf{V}_q$

Redundant computations

Algorithm 3 `decode`

Require: `layer_index`, $\mathbf{K}_q$, $\mathbf{V}_q$, reconstructed keys and values of the previous layer $\hat{\mathbf{K}}_{\text{prev}}$, $\hat{\mathbf{V}}_{\text{prev}}$, `predictors`

1: **if** `layer_index` $= 0$ **then**
2: **return** $\mathbf{K}_q, \mathbf{V}_q$
3: **end if**
4: $f_{\text{key}}, f_{\text{value}} \leftarrow$ `predictors[layer_index]`
5: $\hat{\mathbf{K}} \leftarrow Q^{-1}(\mathbf{K}_q) + f_{\text{key}}(\hat{\mathbf{K}}_{\text{prev}})$
6: # value predictor expects both previous reconstructed $\hat{\mathbf{V}}_{\text{prev}}$ and current reconstructed $\hat{\mathbf{K}}$.
7: $\hat{\mathbf{V}} \leftarrow Q^{-1}(\mathbf{V}_q) + f_{\text{value}}([\hat{\mathbf{V}}_{\text{prev}}; \hat{\mathbf{K}}])$
8: **return** $\hat{\mathbf{K}}, \hat{\mathbf{V}}$

Conclusion

✓ Our Observed Limitations

- **Predictor overhead**
  - Additional memory and compute costs
  - No evaluation is provided for non-GQA architectures
    - Although claimed to be minimal due to GQA
  - Suggesting the justification may be incidental rather than fundamental.

Method	Quant. bits	Cache size GiB, 70B
Uncompressed	16	40
AQUA-KV	2.09	5.7
HIGGS	2.02	5.1
KIVI	2.25	5.6
KVQuant	2.33	5.6

- **Limited Parallelism**
  - Current key usage for value predictor restricts parallel execution
  - Unclear whether the performance gain justifies the added cost
- **The Lack of Evalutaion**
  - There is no EVR results for $[V_{L-1}; K_L]$
  - only $[KV_{L-1}; K_L]$ and $[KV_{L-1}]$ are evaluated
  - What if using $[KV_{L-1}]$ yields comparable performance with only minor inference-time overhead?

Thank you

AQUA-KV Inference

✓ Encode

- Only residual part that the predictor cannot predict is compressed and stored

Algorithm 2 `encode`

Require: `layer_index`, $\mathbf{K}$, $\mathbf{V}$, reconstructed keys and values of the previous layer $\hat{\mathbf{K}}_{\text{prev}}$, $\hat{\mathbf{V}}_{\text{prev}}$, `predictors`

if `layer_index` $= 0$ **then**
  return $\mathbf{K}, \mathbf{V}$
end if
$f_{\text{key}}, f_{\text{value}} \leftarrow$ `predictors[layer_index]`
$\mathbf{K}_{\text{pred}} \leftarrow f_{\text{key}}(\hat{\mathbf{K}}_{\text{prev}})$
$\mathbf{K}_q \leftarrow Q(\mathbf{K} - \mathbf{K}_{\text{pred}})$
$\hat{\mathbf{K}} \leftarrow Q^{-1}(\mathbf{K}_q) + \mathbf{K}_{\text{pred}}$
$\mathbf{V}_{\text{pred}} \leftarrow f_{\text{value}}([\hat{\mathbf{V}}_{\text{prev}}; \hat{\mathbf{K}}])$ {Concatenate $\hat{\mathbf{V}}_{\text{prev}}$ and $\hat{\mathbf{K}}$}
$\mathbf{V}_q \leftarrow Q(\mathbf{V} - \mathbf{V}_{\text{pred}})$
return $\mathbf{K}_q, \mathbf{V}_q$

✓ Decode

- Reconstruct KV-cache from a compressed representation

Algorithm 3 `decode`

Require: `layer_index`, $\mathbf{K}_q$, $\mathbf{V}_q$, reconstructed keys and values of the previous layer $\hat{\mathbf{K}}_{\text{prev}}$, $\hat{\mathbf{V}}_{\text{prev}}$, `predictors`

if `layer_index` $= 0$ **then**
  return $\mathbf{K}_q, \mathbf{V}_q$
end if
$f_{\text{key}}, f_{\text{value}} \leftarrow$ `predictors[layer_index]`
$\hat{\mathbf{K}} \leftarrow Q^{-1}(\mathbf{K}_q) + f_{\text{key}}(\hat{\mathbf{K}}_{\text{prev}})$
Value predictor expects both previous reconstructed $\hat{\mathbf{V}}_{\text{prev}}$ and current reconstructed $\hat{\mathbf{K}}$.
$\hat{\mathbf{V}} \leftarrow Q^{-1}(\mathbf{V}_q) + f_{\text{value}}([\hat{\mathbf{V}}_{\text{prev}}; \hat{\mathbf{K}}])$
return $\hat{\mathbf{K}}, \hat{\mathbf{V}}$

Auto-regressive PPL evaluation


```
for sequence_index, input_ids in enumerate(tqdm(inps, desc="Evaluating perplexity")):
    input_ids = input_ids.to(device)
    with torch.amp.autocast("cuda", enabled=amp_dtype is not None, dtype=amp_dtype or torch.float32):
        if cache_factory is None:
            cache = DynamicCache()
        else:
            cache = cache_factory()
        dtype = amp_dtype or next(model.parameters()).dtype
        lm_logits = torch.zeros(
                (input_ids.shape[0], input_ids.shape[1], model.get_output_embeddings().out_features), device=device, dtype=dtype)
        for i in range(0, input_ids.shape[1], step_size):
            out = model(input_ids[:, i: i + step_size], use_cache=True, past_key_values=cache)
            assert out.past_key_values is cache
            lm_logits[:, i: i + step_size, ...] = out.logits

    if sequence_index < num_sequences_without_padding:
        shift_logits = lm_logits[:, :-1, :].contiguous()
        shift_labels = input_ids[:, 1:]
        loss_fct = nn.CrossEntropyLoss()
        loss = loss_fct(shift_logits.view(-1, shift_logits.size(-1)), shift_labels.view(-1))
        total_nll += loss.float() * shift_labels.numel()
        total_tokens += shift_labels.numel()
    else:
        raise RuntimeError

ppl = torch.exp(total_nll / total_tokens)
return ppl.item()
```