Déjà Vu: Efficient Video-Language Query Engine with Learning-based Inter-Frame Computation Reuse

Jinwoo Hwang

Daeun Kim

Yoonsung Kim

Hojoon Kim

Tadiwos Meaza

Jeongseob Ahn‡

Sangyeop Lee

Guseul Heo

Yunseok Jeong

Eunhyeok Park†

Jongse Park

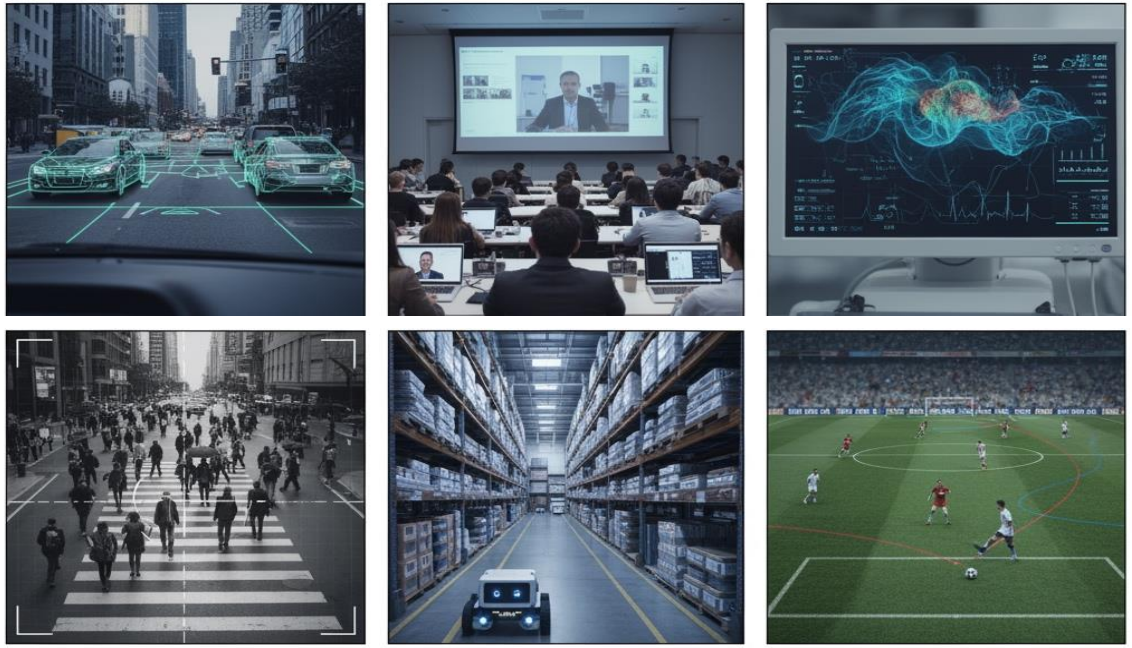
KAIST

† POSTECH

‡ Korea University

Video data is exploding!

Video data now makes up more than 54% the global IP traffic*.



Yet, they are underutilized,

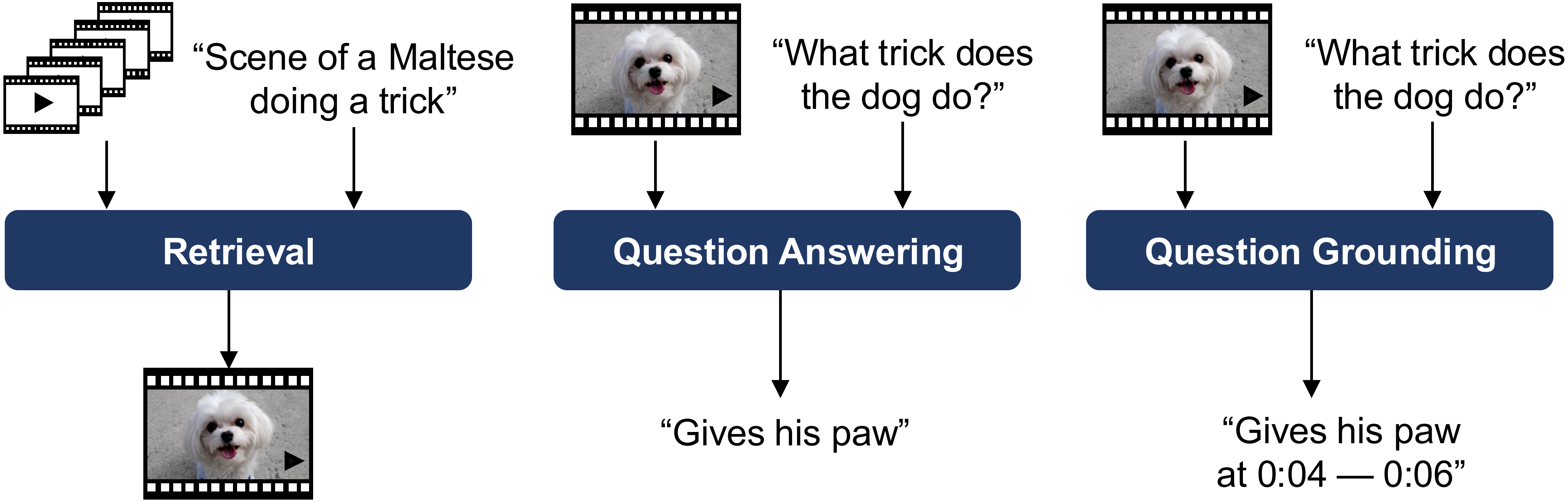
68% of such unstructured data remain unused.**

*Sandvine, The Global Internet Phenomena Report (2024)

**IDC & Seagate, Rethink Data (2020)

Video Language Models (VideoLMs)

Three representative VideoLM applications



“Scene of a Maltese doing a trick”

Retrieval

“What trick does the dog do?”

Question Answering

“Gives his paw”

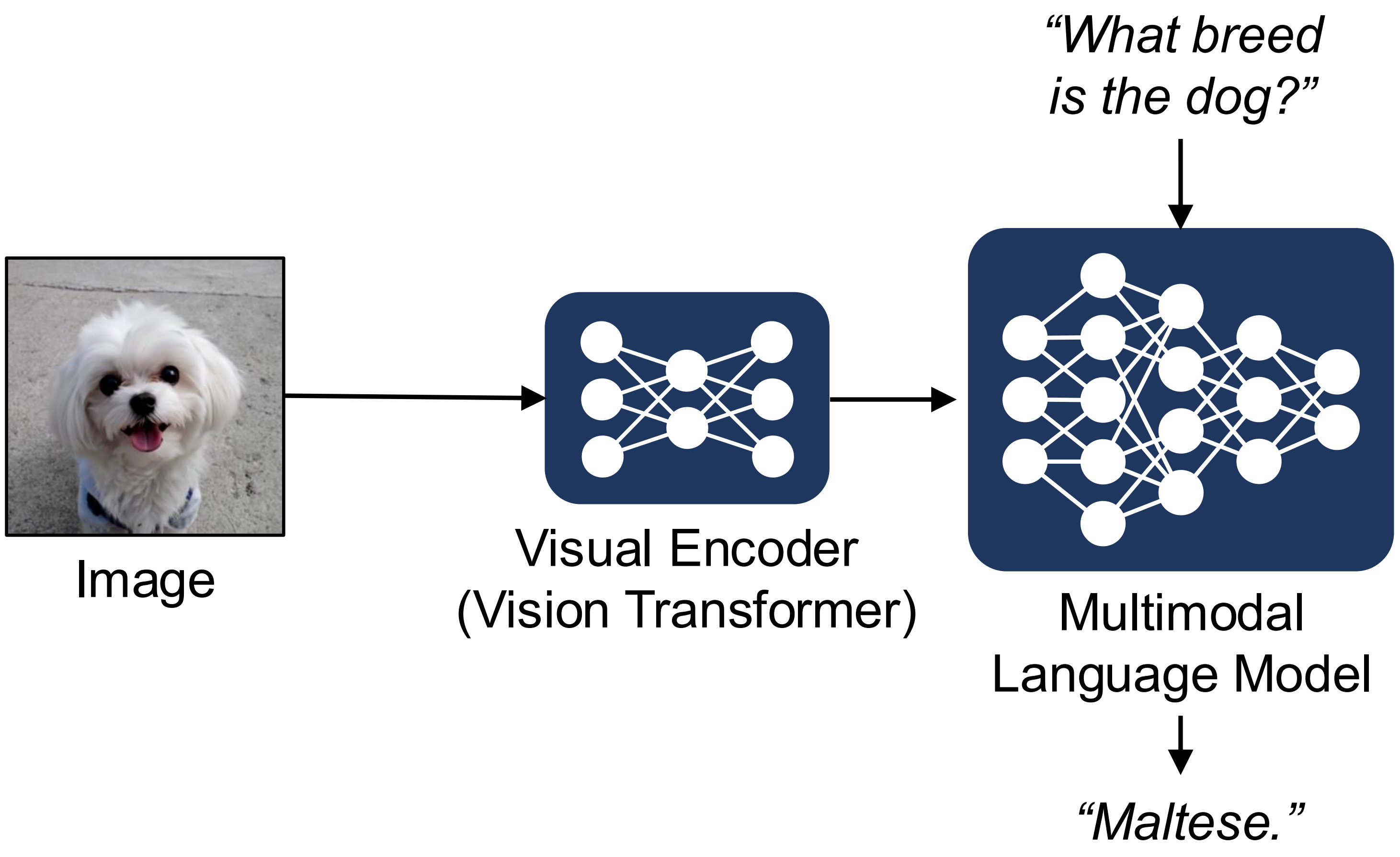
“What trick does the dog do?”

Question Grounding

“Gives his paw at 0:04 — 0:06”

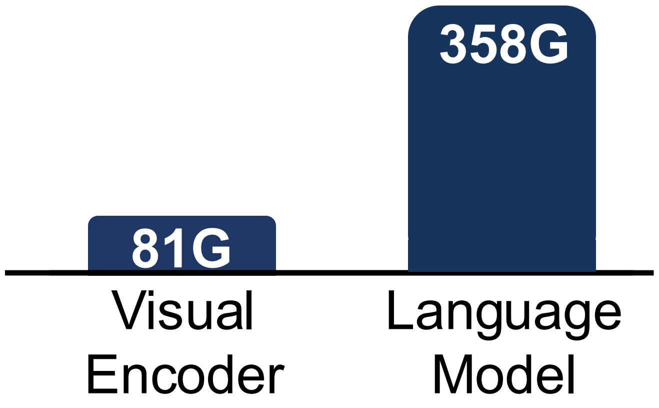
VideoLMs serve as a **new powerful interface** to video data.

Structure of Vision-Language Model



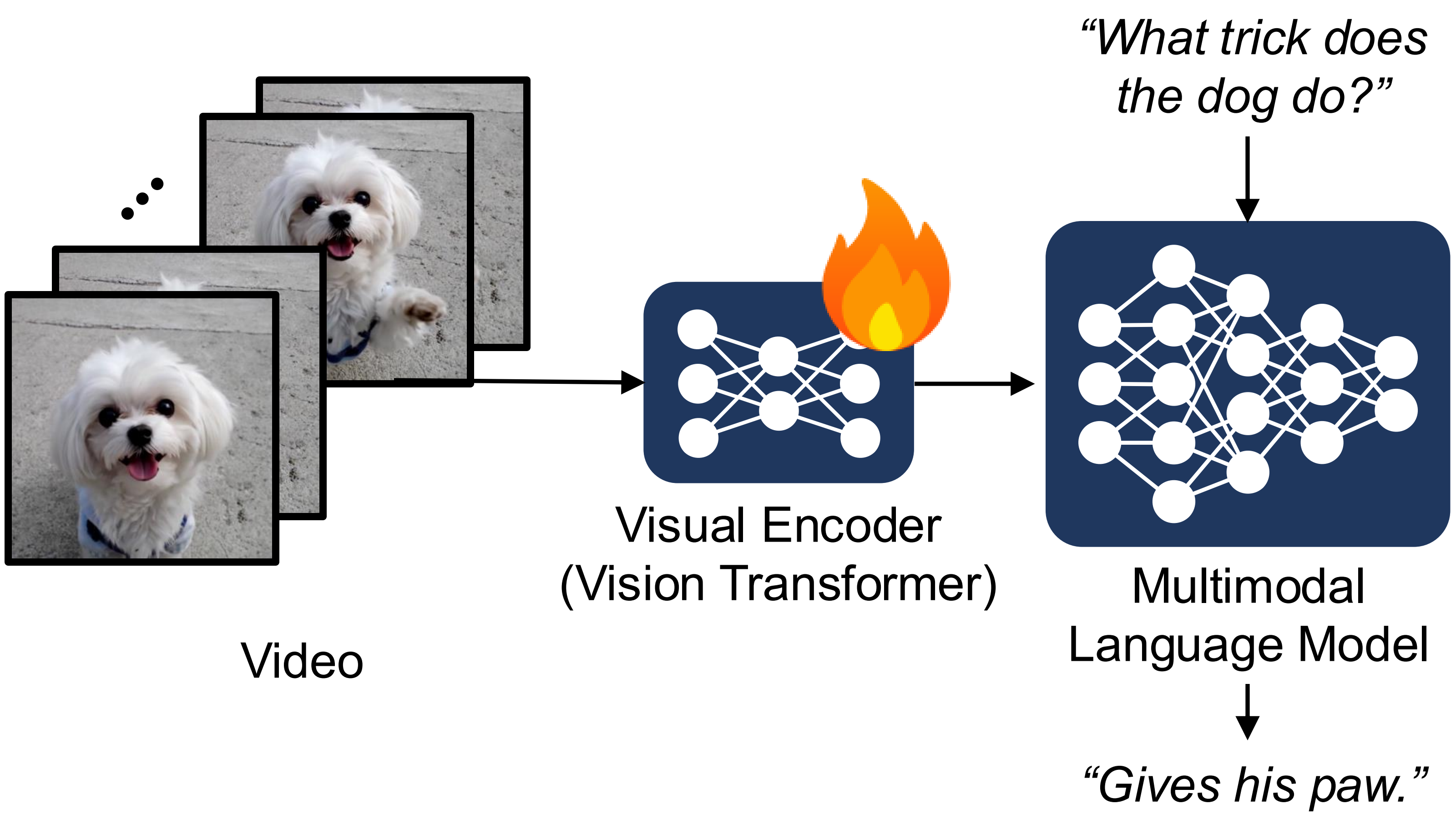
Question Answering Model
FLOPs Breakdown

*FrozenBiLM

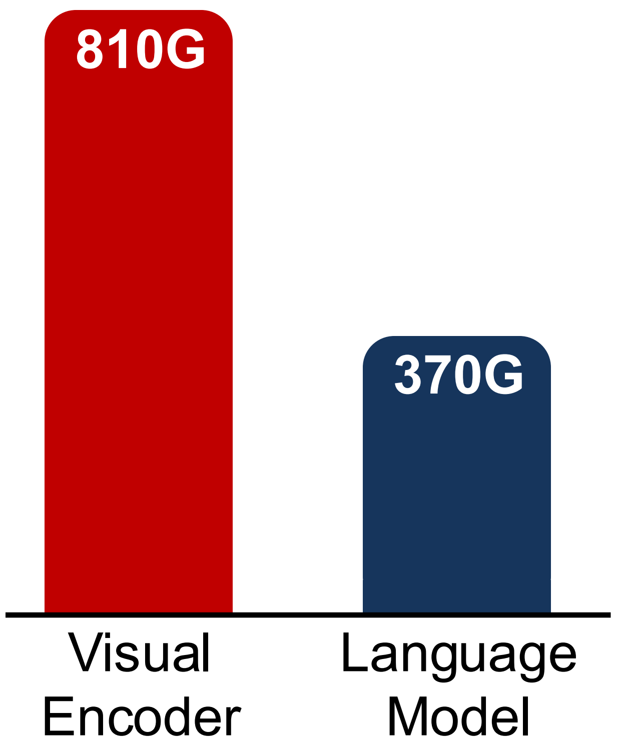


- Vision-language model has two parts: **visual encoder** and **language model**.

From Image to Video: Computational Shift



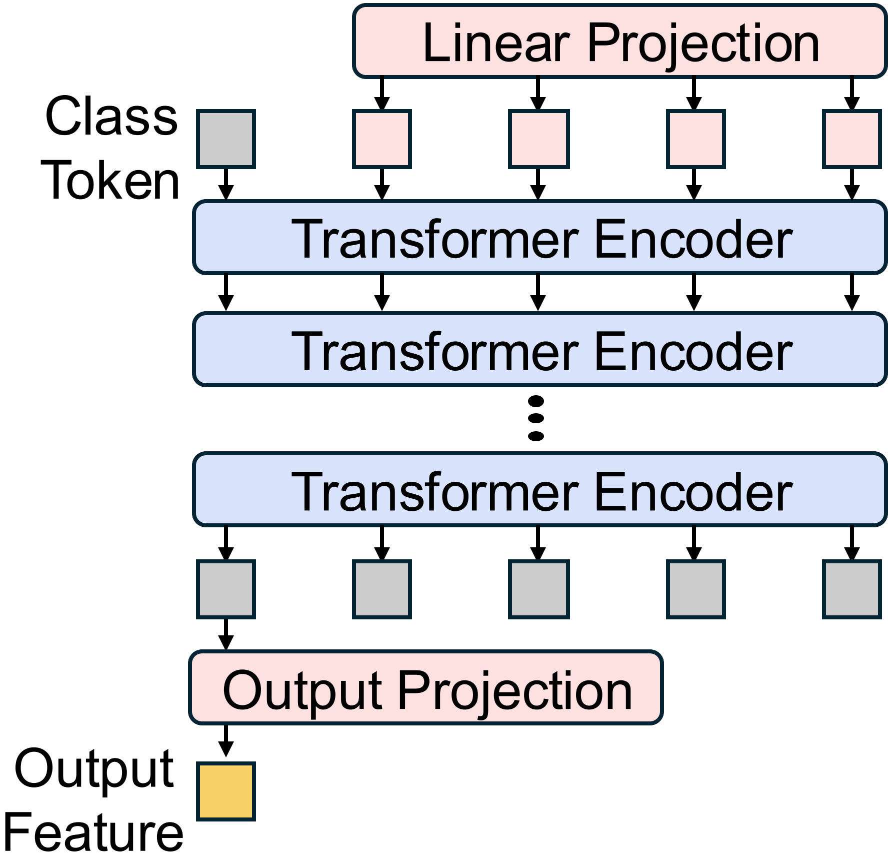
“What trick does the dog do?”
Video
Visual Encoder (Vision Transformer)
Multimodal Language Model
“Gives his paw.”


Question Answering Model FLOPs Breakdown

*FrozenBiLM

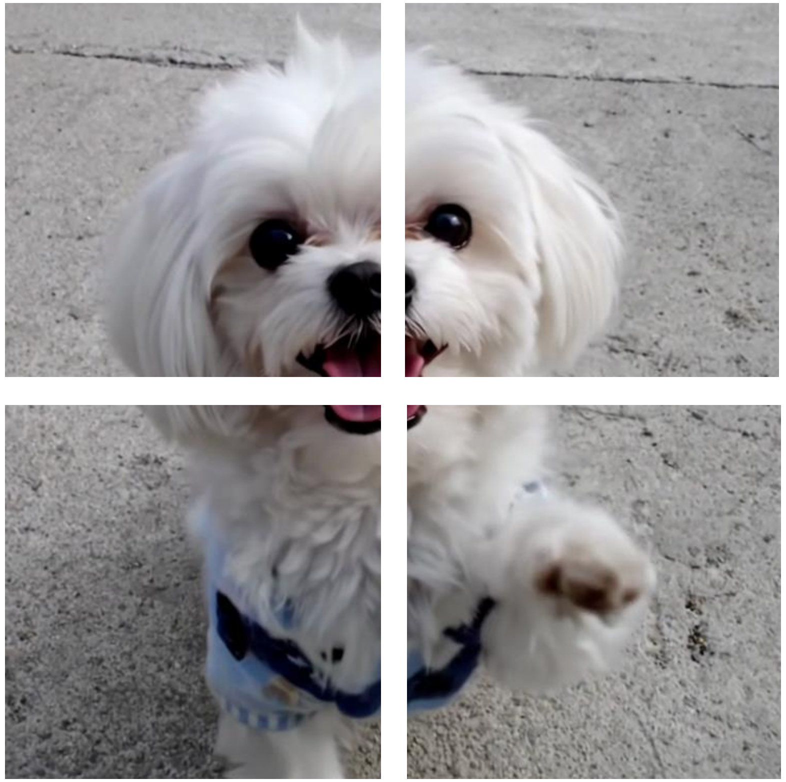
- As for the videos, the **visual encoder dominates** the computation.

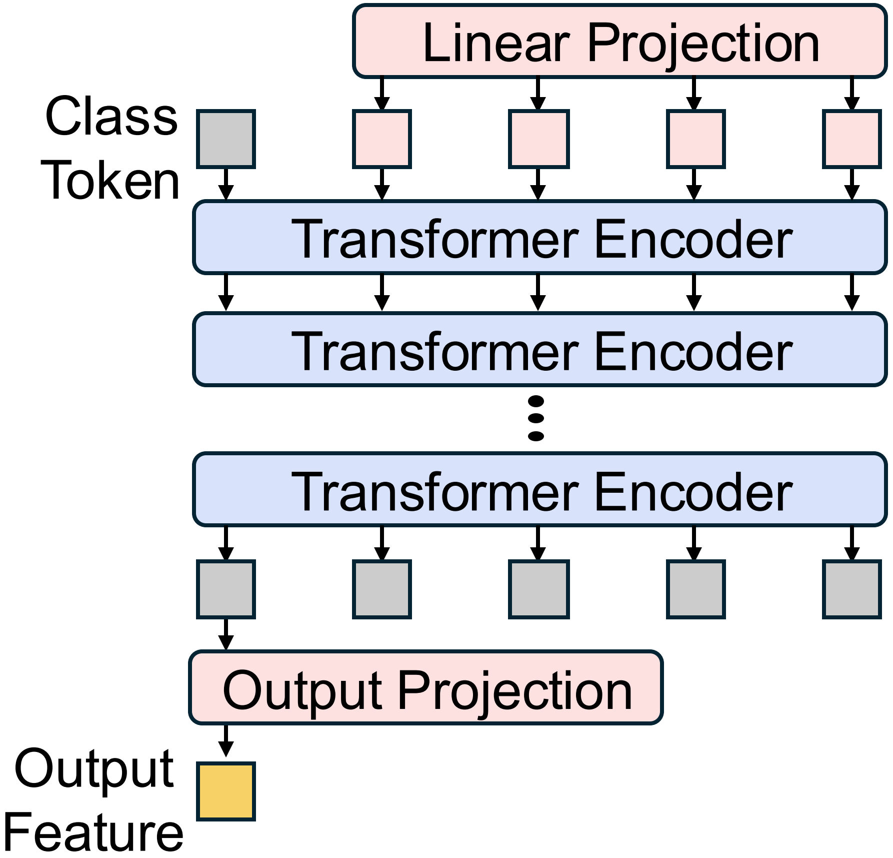
Vision Transformer (ViT) Architecture



- ViT works by splitting image into grid of patches and treating them as tokens.

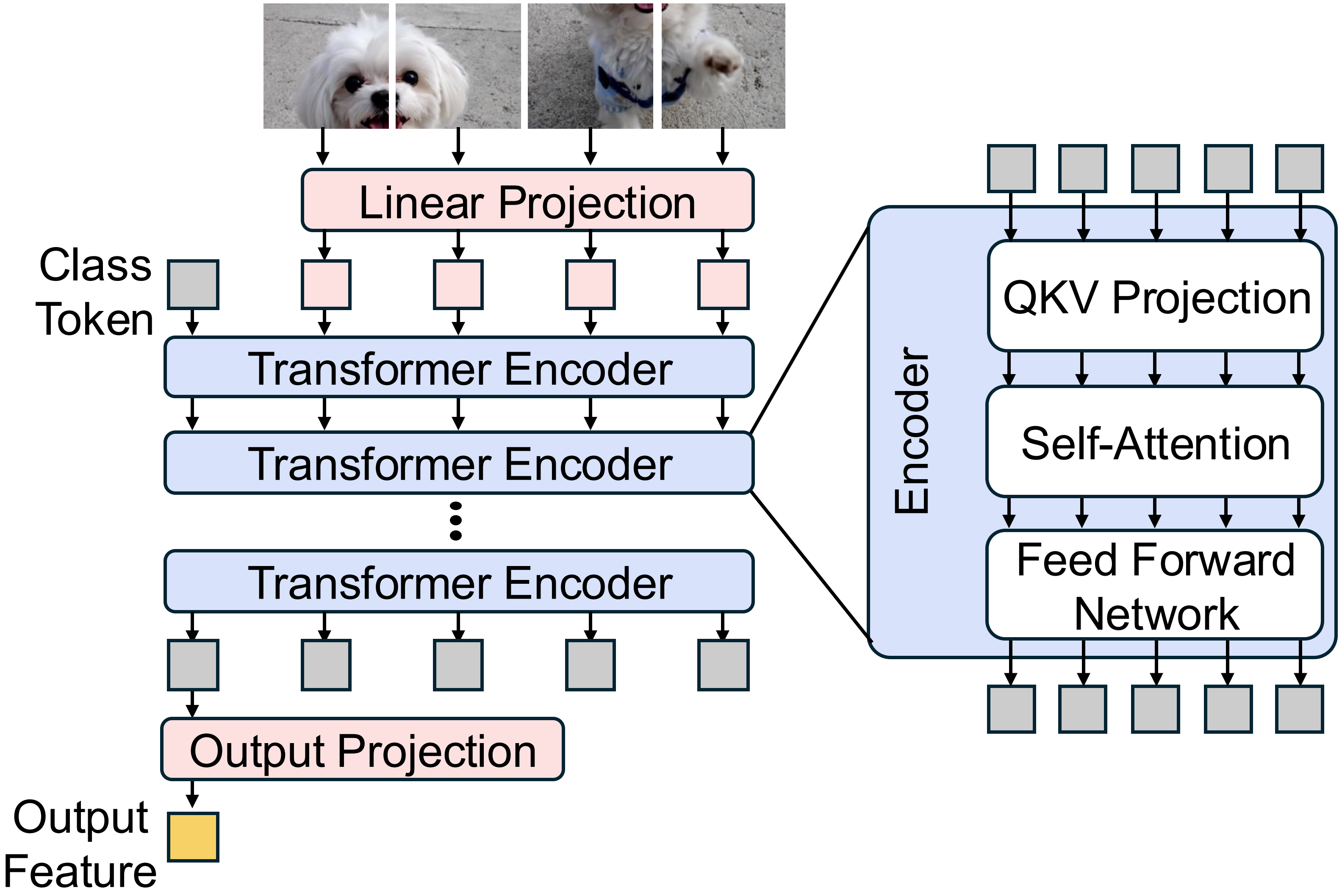
Vision Transformer (ViT) Architecture





- ViT works by splitting image into grid of patches and treating them as tokens.

Vision Transformer (ViT) Architecture



Key Opportunity: Temporal Redundancy

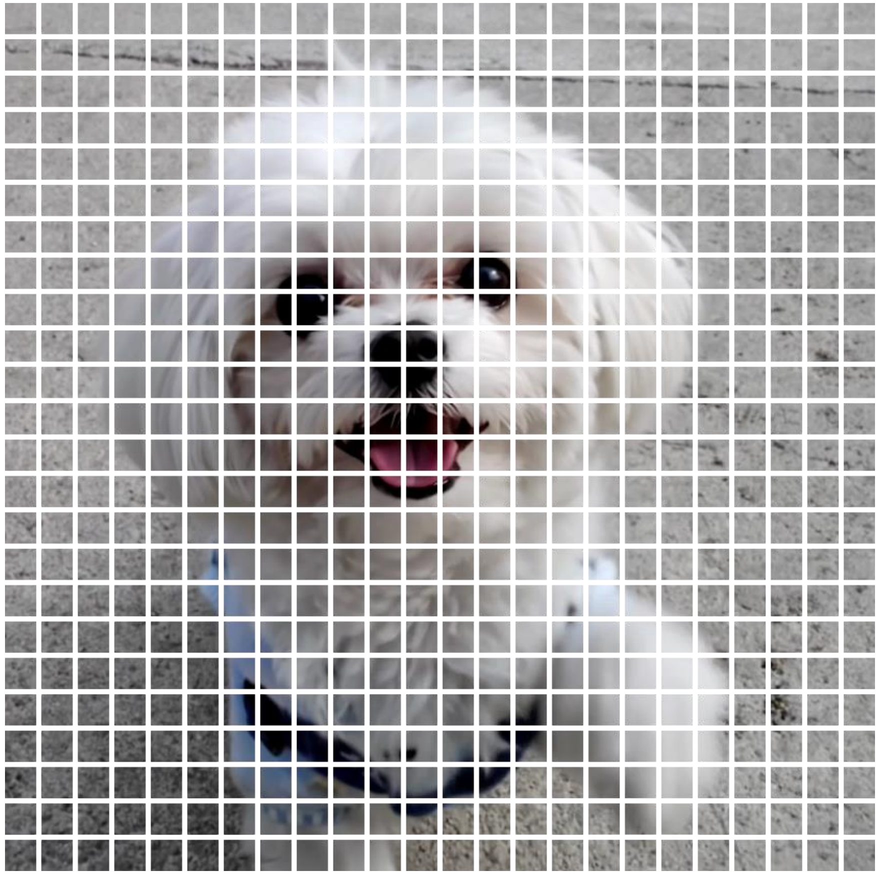
Previous Frame

Current Frame

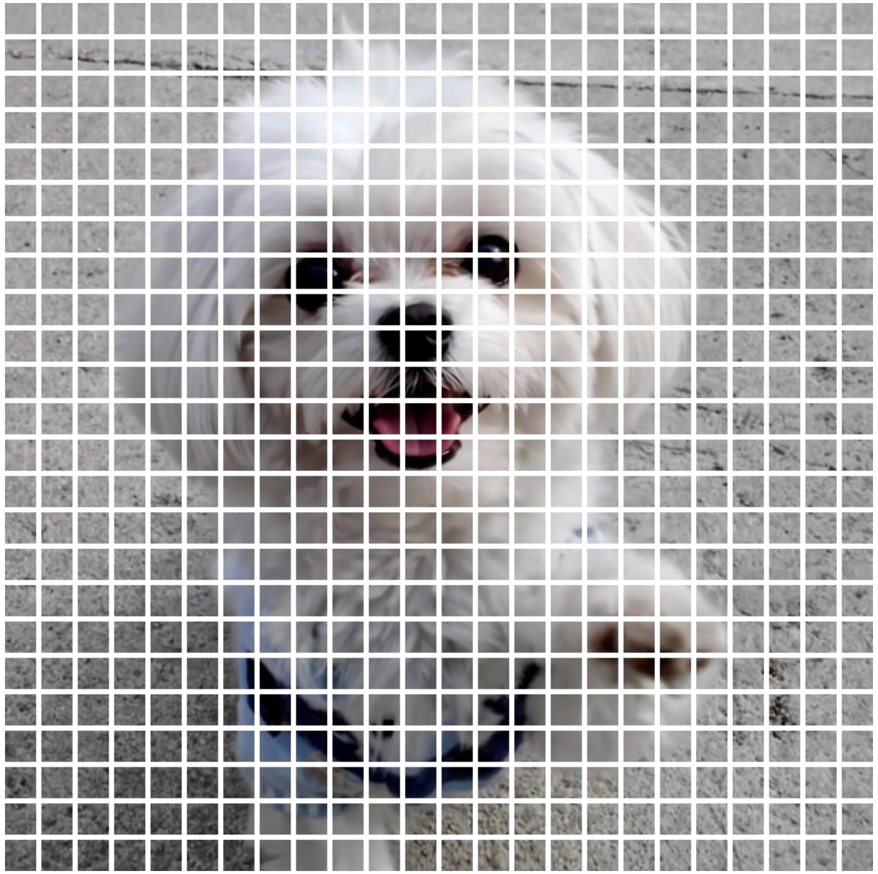
- Video data contains abundant **temporally redundancy**.

Key Opportunity: Temporal Redundancy

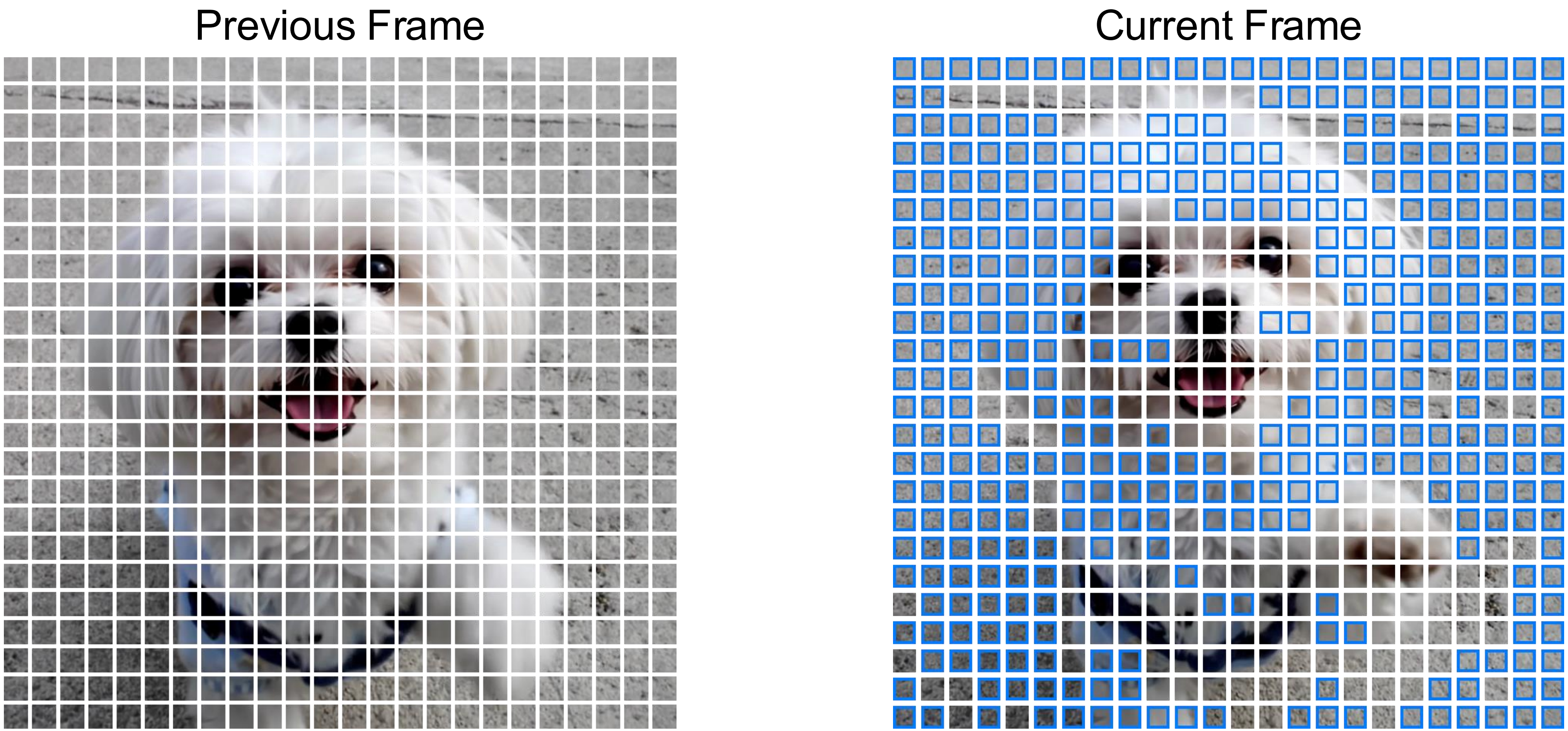
Previous Frame



Current Frame



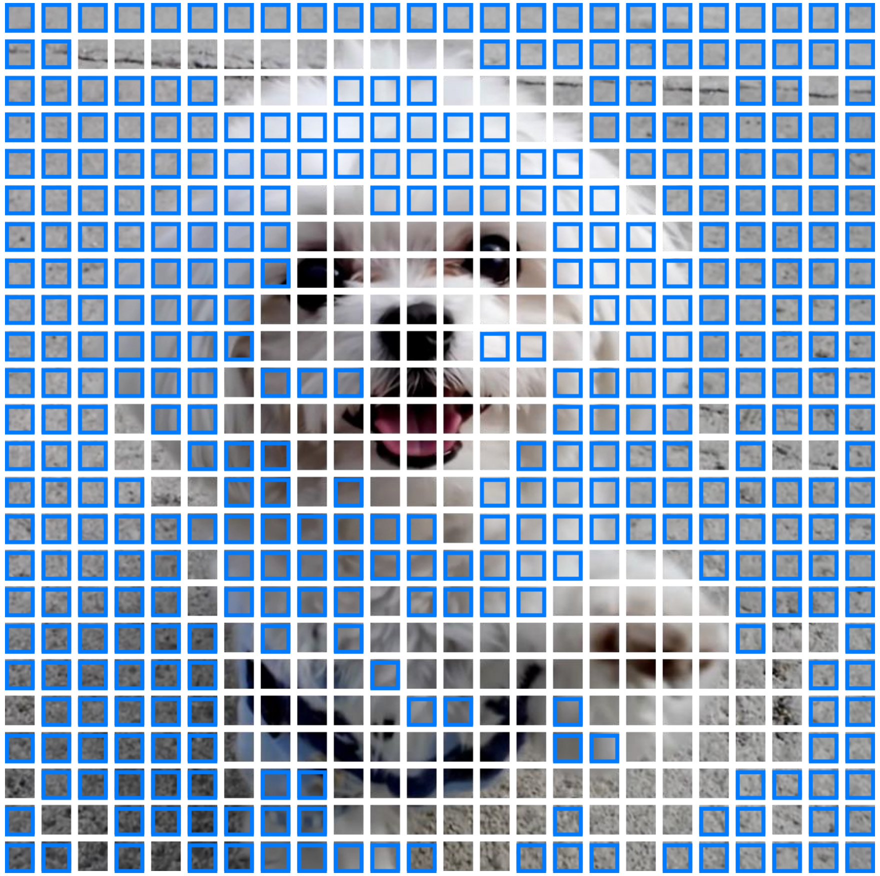
Key Opportunity: Temporal Redundancy



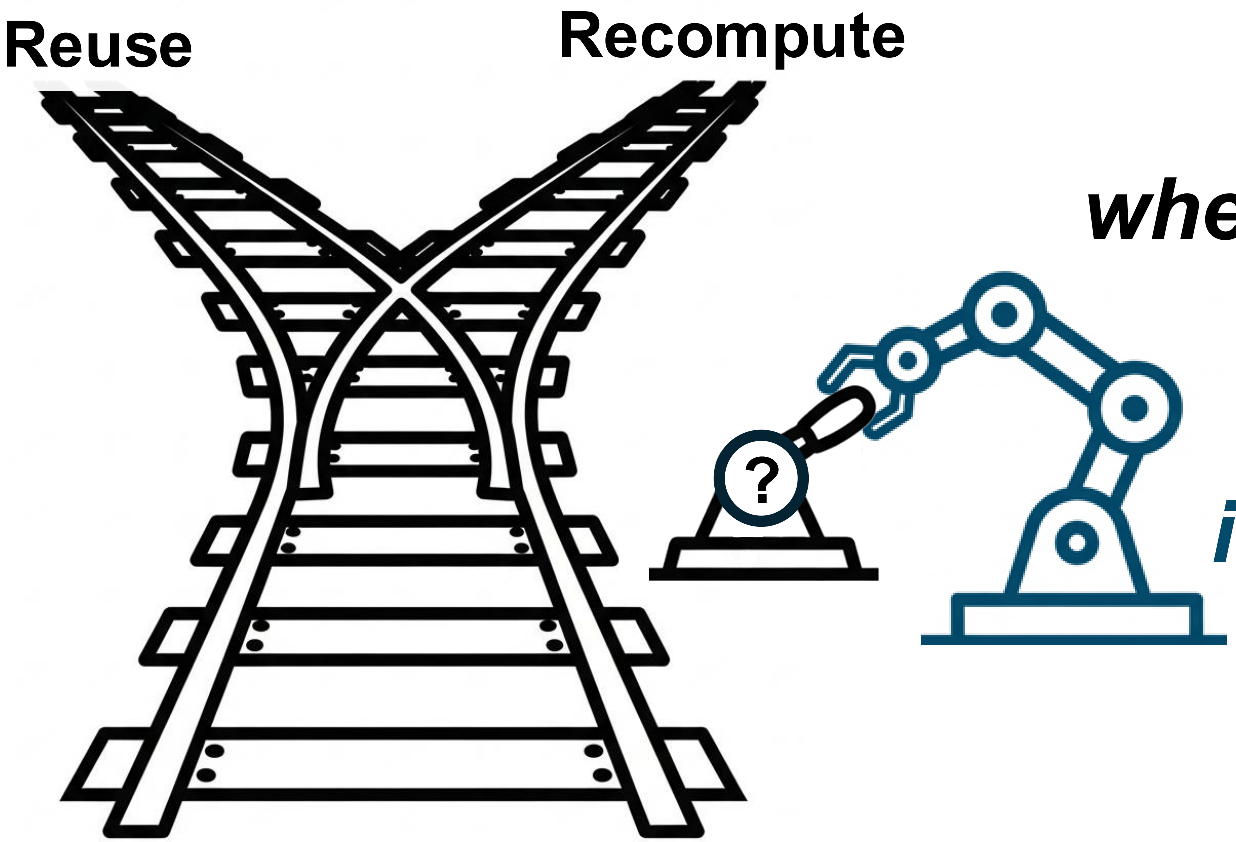
- Many patches persist across frames as highlighted in blue

Key Opportunity: Temporal Redundancy

Frame reconstructed with reused patches



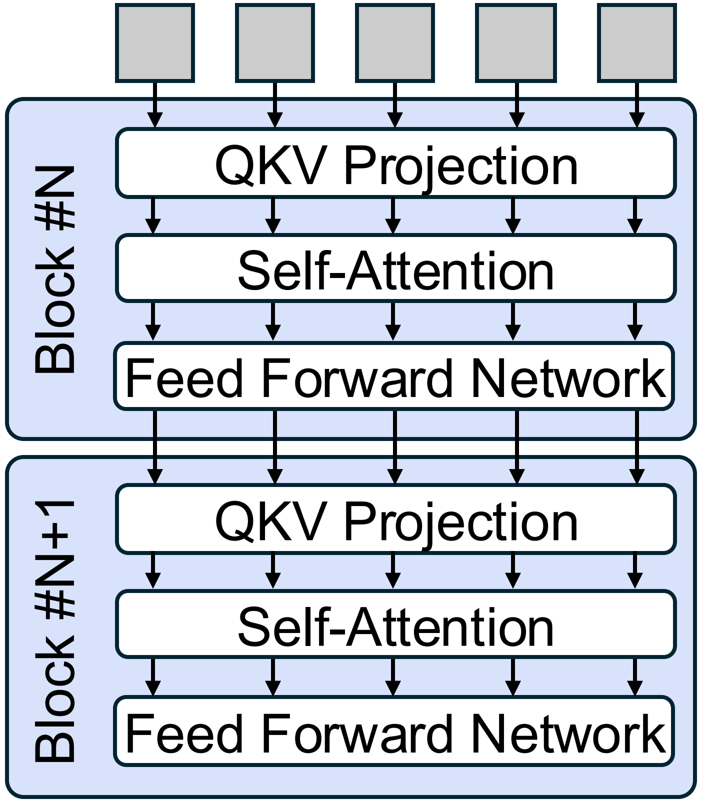
- Core Idea: **Reuse redundant computations** from previous frame within ViT



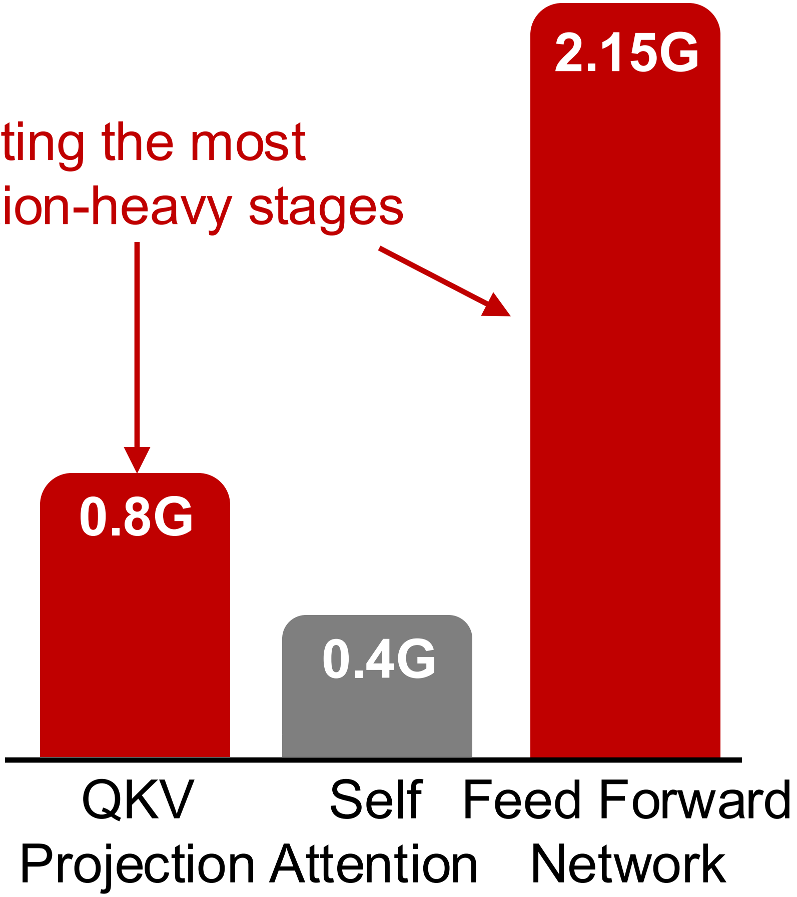
How do we decide when to reuse or recompute?

Let the model learn its own reuse decision.

Reuse Target Identification



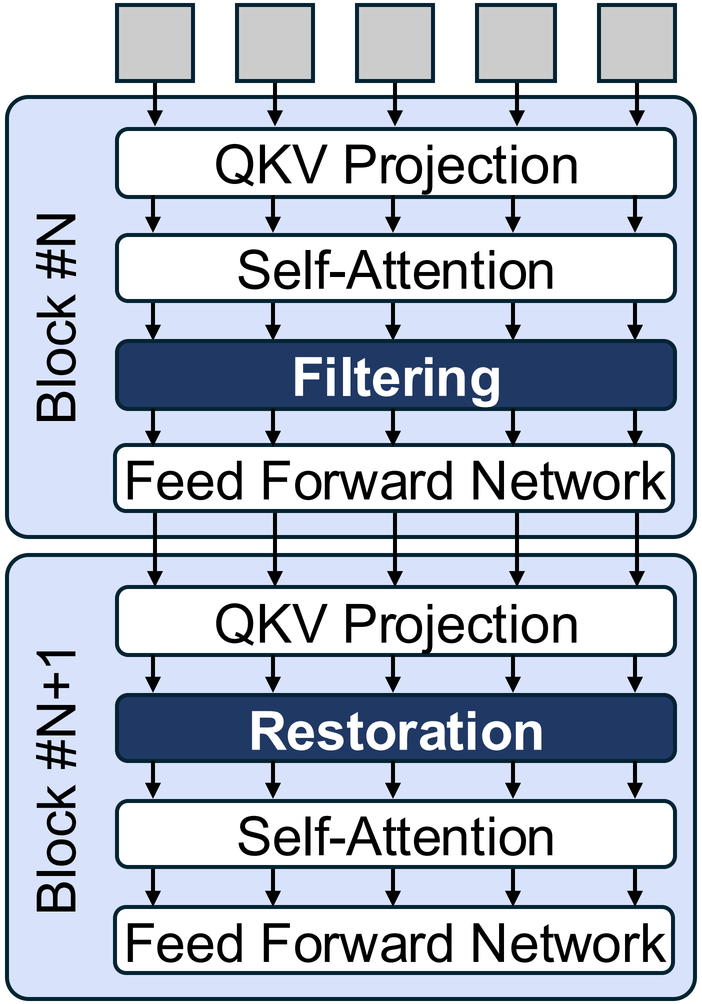
ViT FLOPs Breakdown



Targeting the most computation-heavy stages

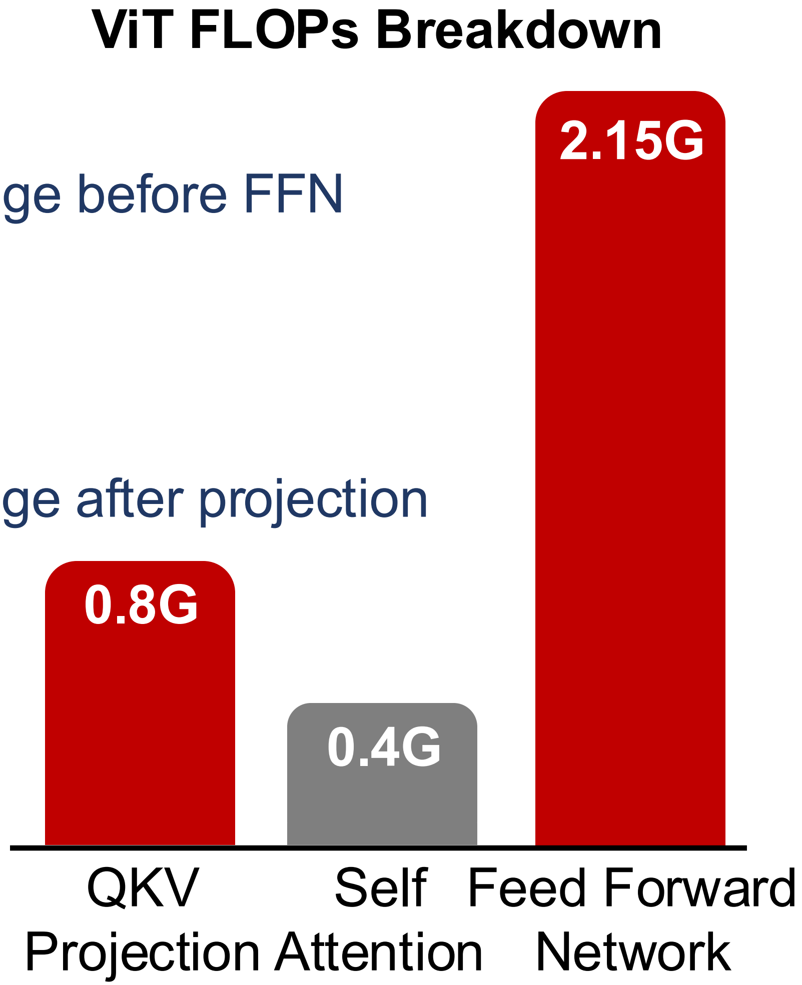
*ViT-large-patch14-336px, 80% reuse

Filtering and Restoration Stages



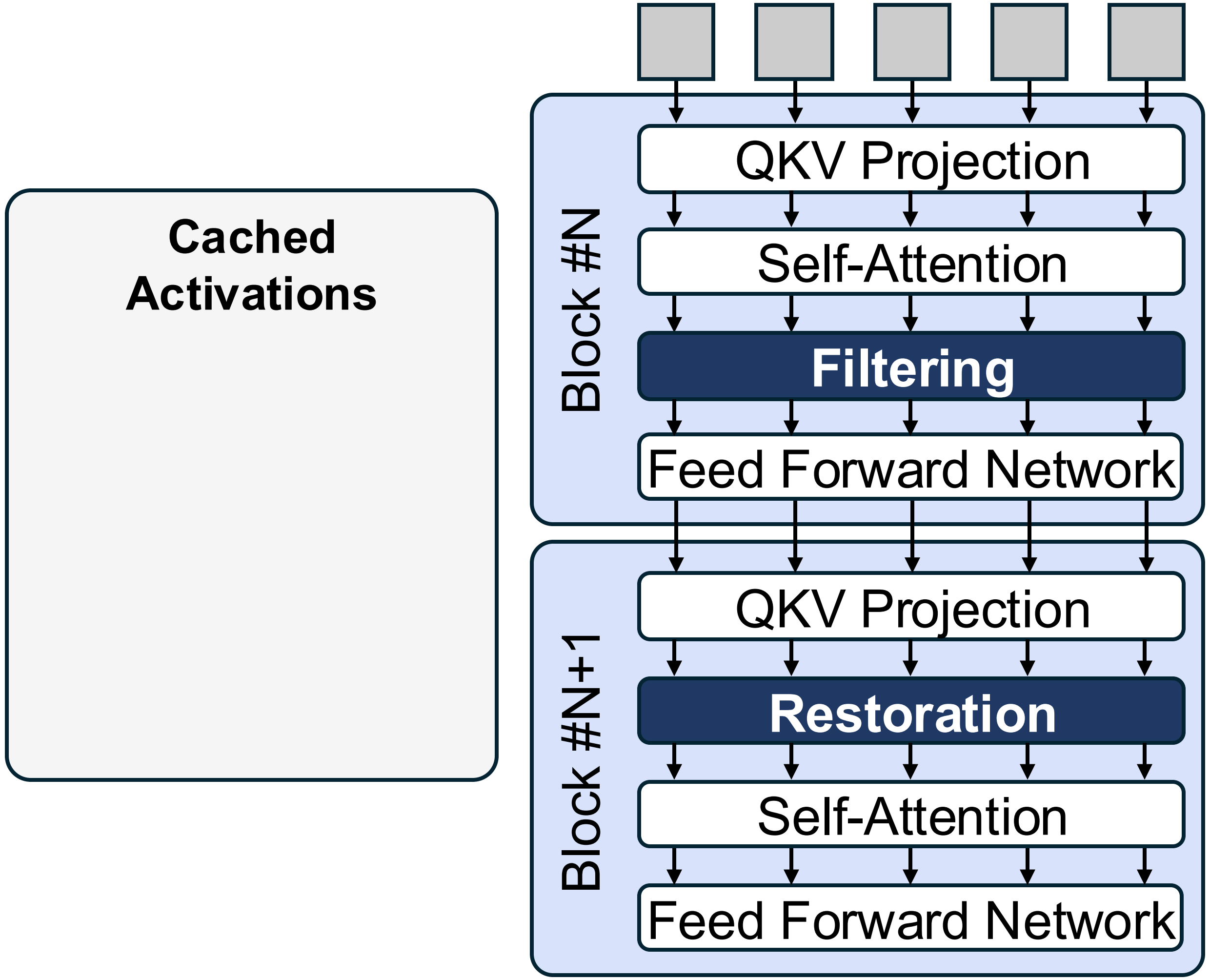
Filtering stage before FFN

Restoration stage after projection



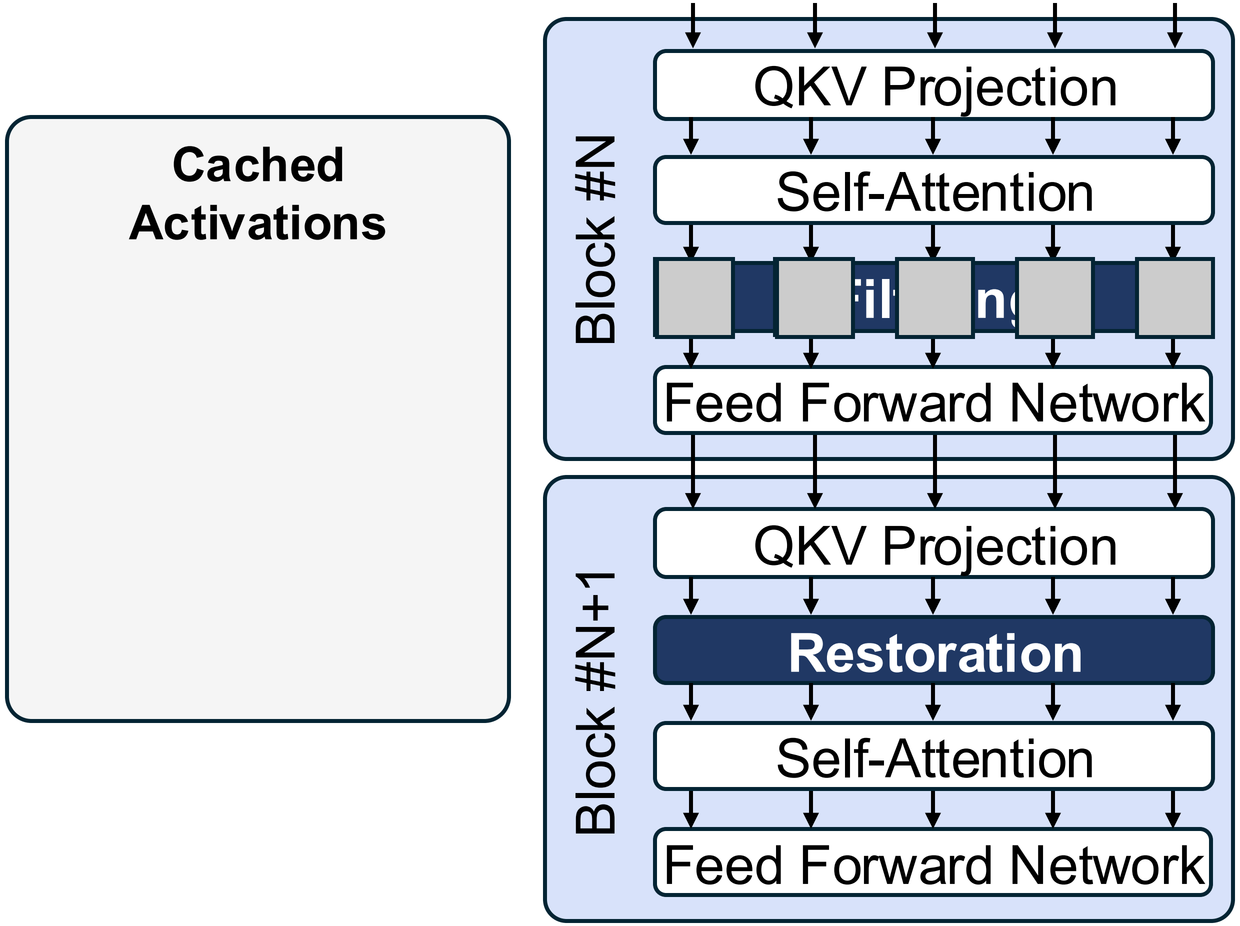
*ViT-large-patch14-336px, 80% reuse

Example Flow: First Frame without Reuse



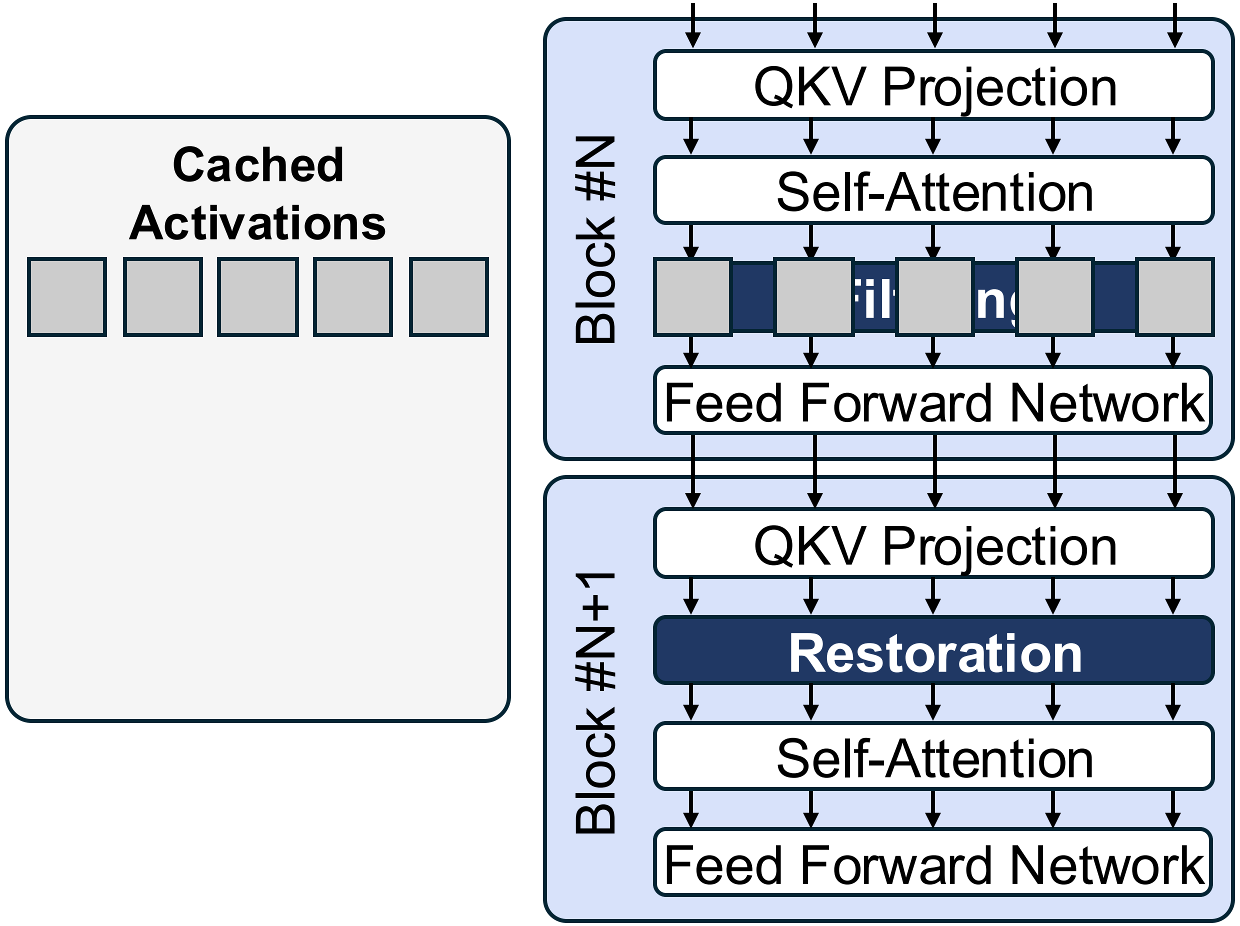
- Initial frame: compute everything from scratch, no reuse yet.

Example Flow: First Frame without Reuse



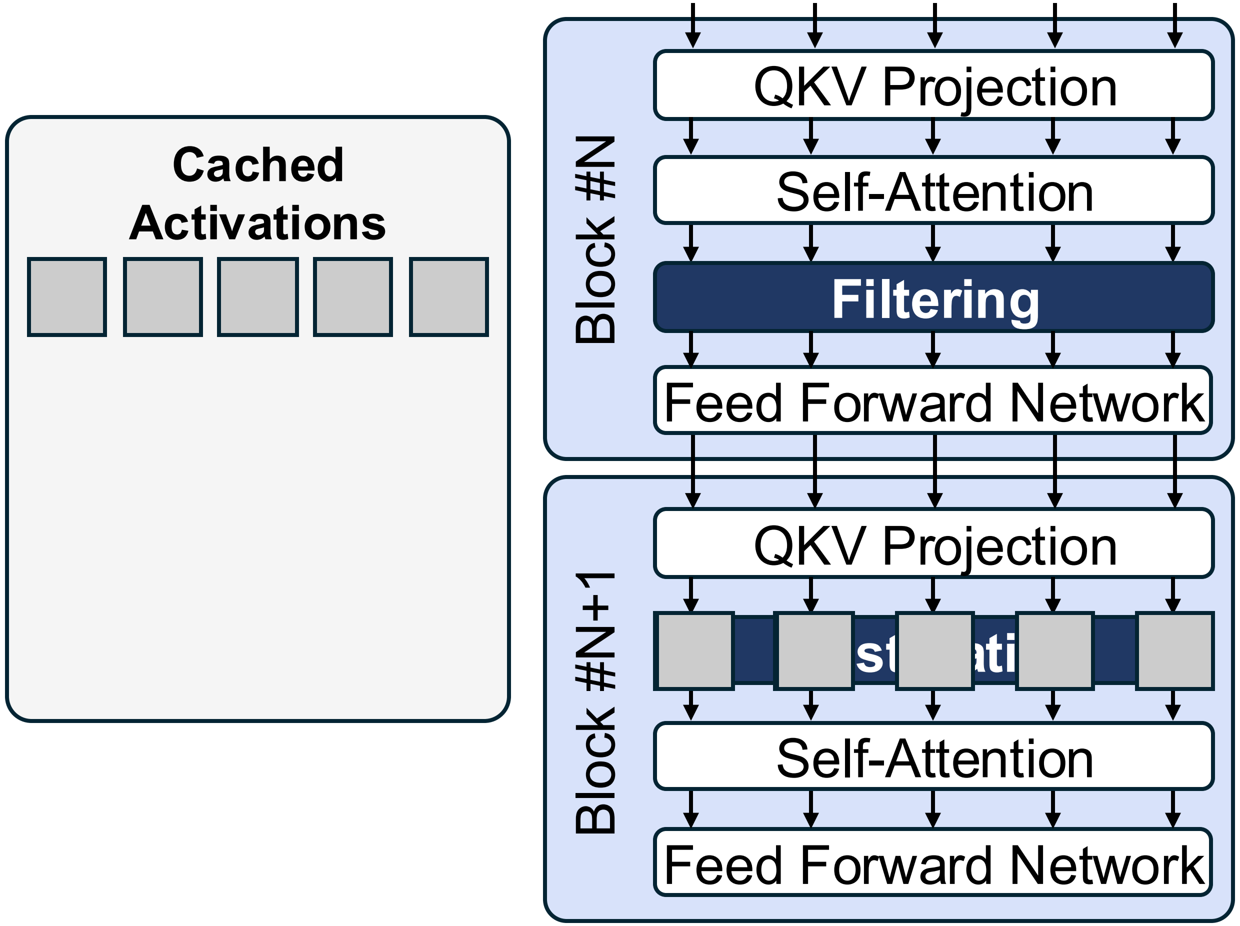
- Initial frame: compute everything from scratch, no reuse yet.

Example Flow: First Frame without Reuse



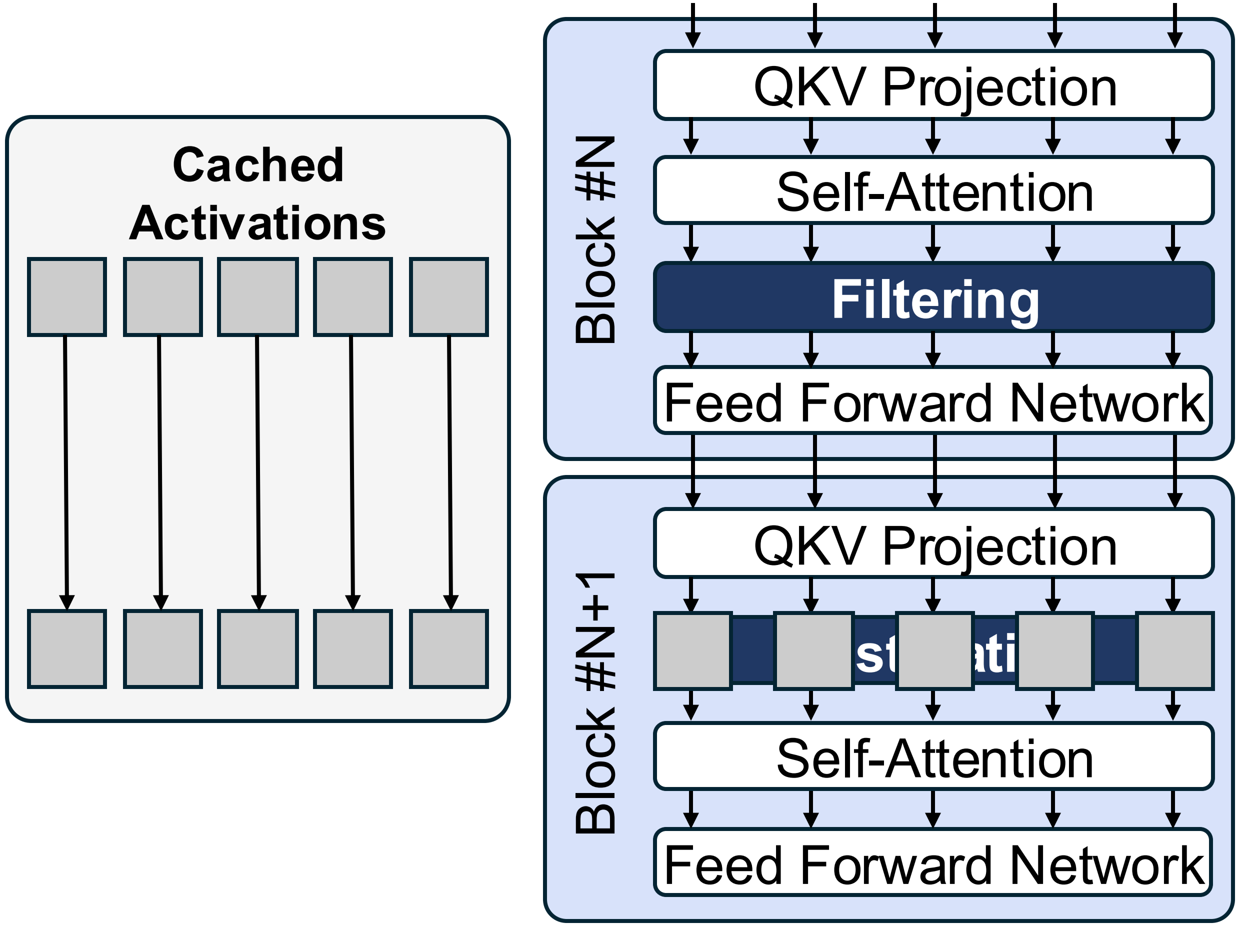
- Initial frame: compute everything from scratch, no reuse yet.
- Cache input activations before the FFN.

Example Flow: First Frame without Reuse



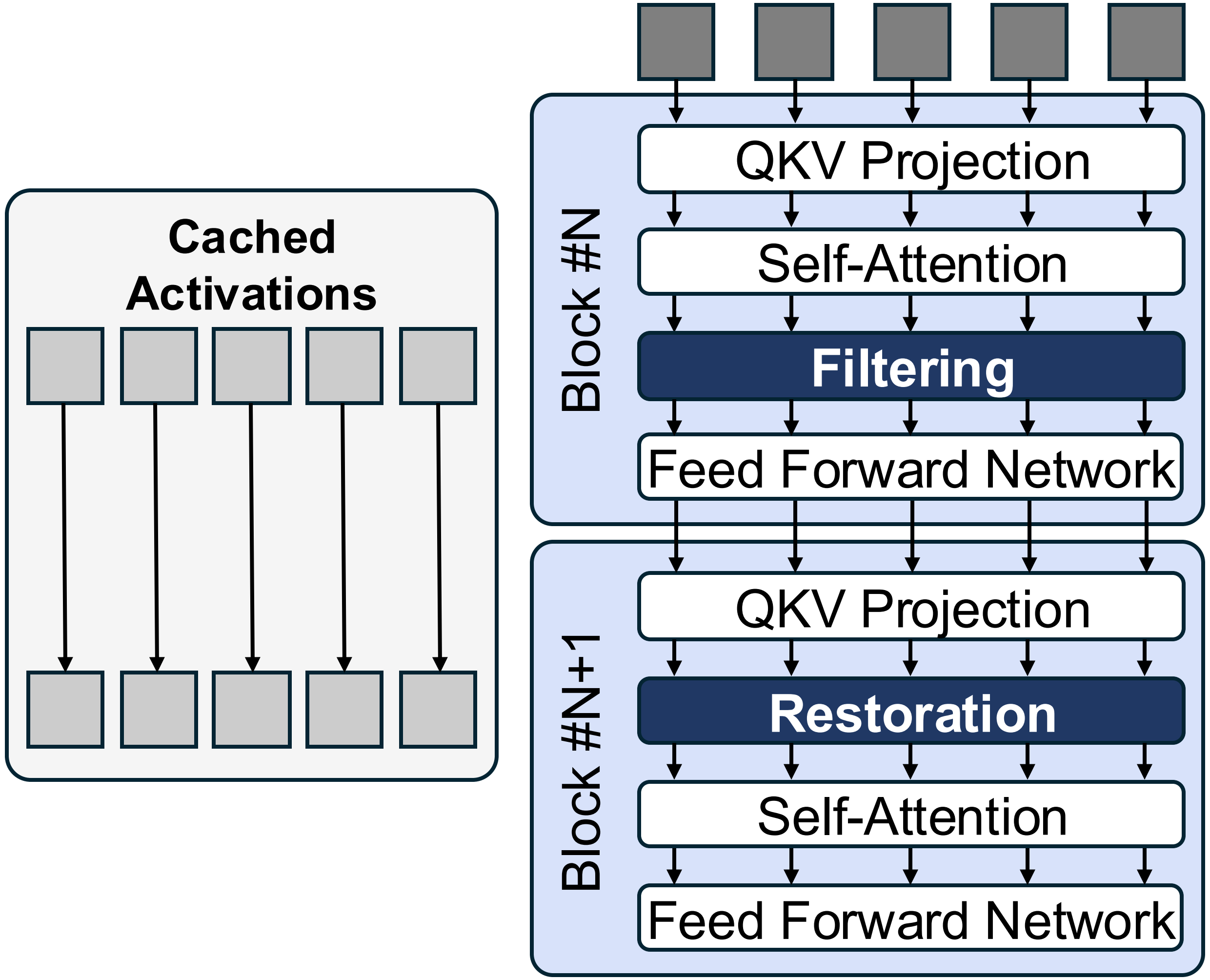
- Initial frame: compute everything from scratch, no reuse yet.
- Cache input activations before the FFN.
- Cache output activations after QKV.

Example Flow: First Frame without Reuse



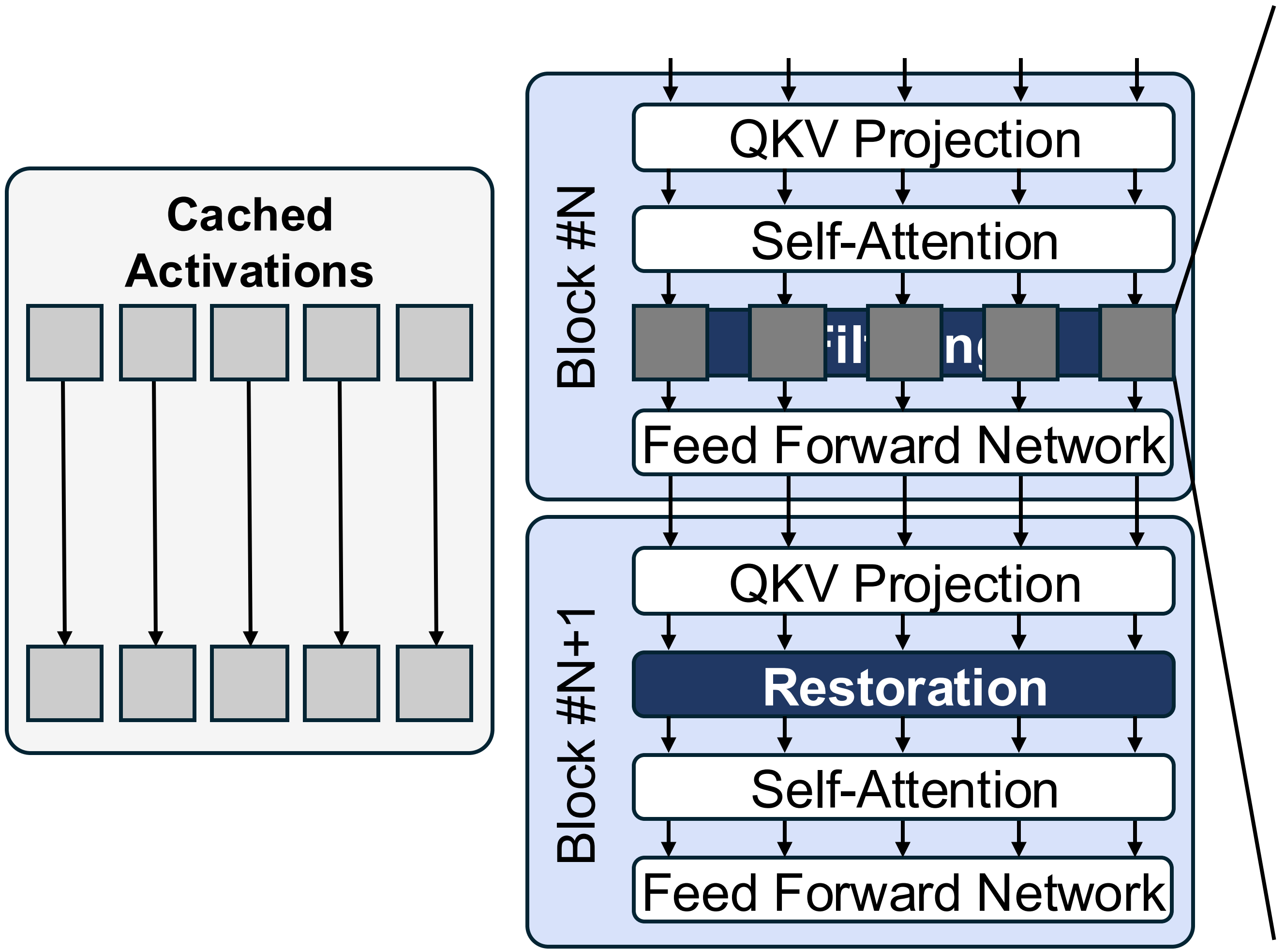
- Initial frame: compute everything from scratch, no reuse yet.
- Cache input activations before the FFN.
- Cache output activations after QKV.

Example Flow: Other Frames with Reuse



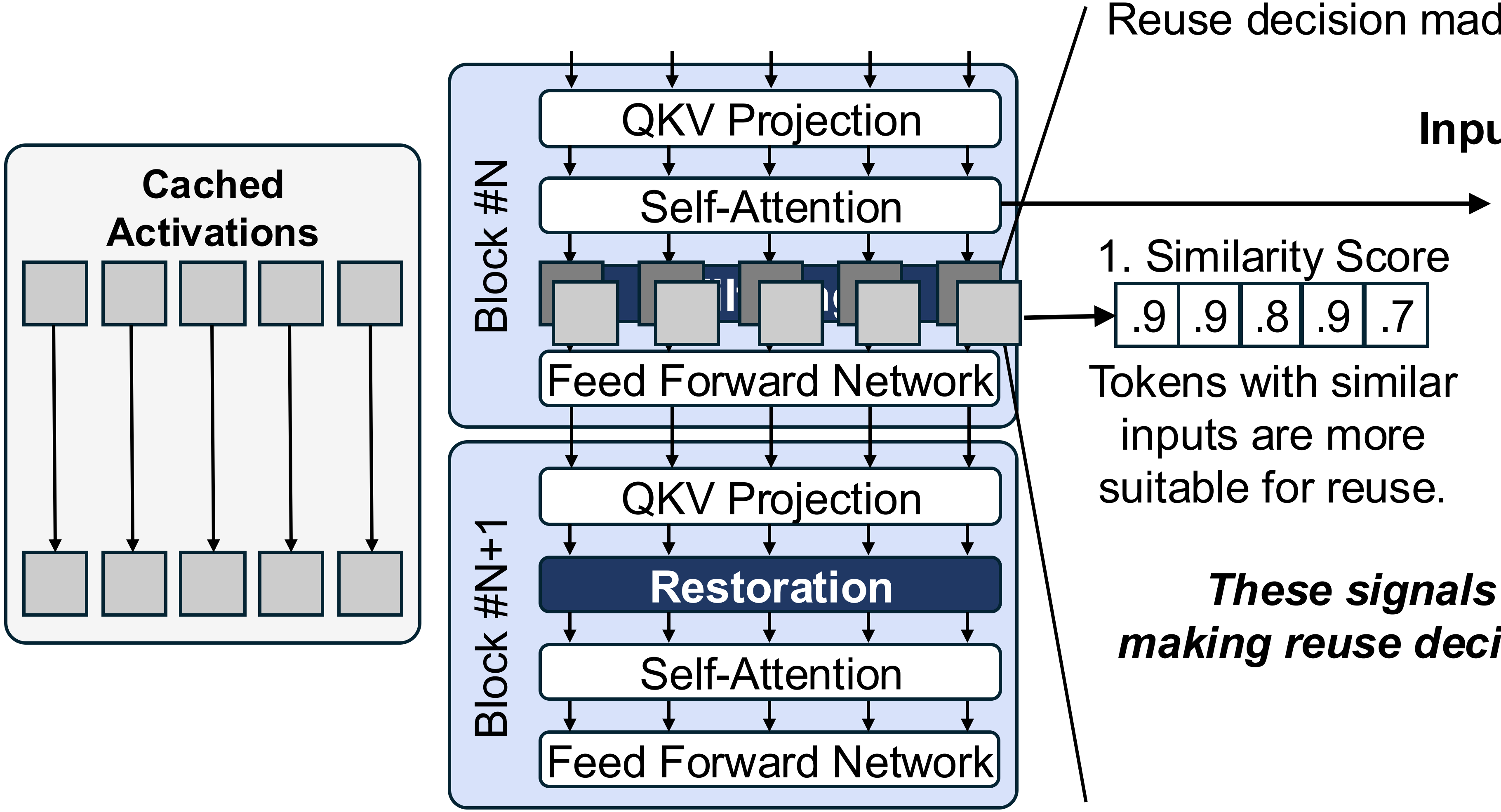
- Second frame: reuse cached activation to reduce computation.

Example Flow: Other Frames with Reuse



Reuse decision made at filtering stage

Example Flow: Other Frames with Reuse



Reuse decision made at filtering stage

Inputs

1. Similarity Score

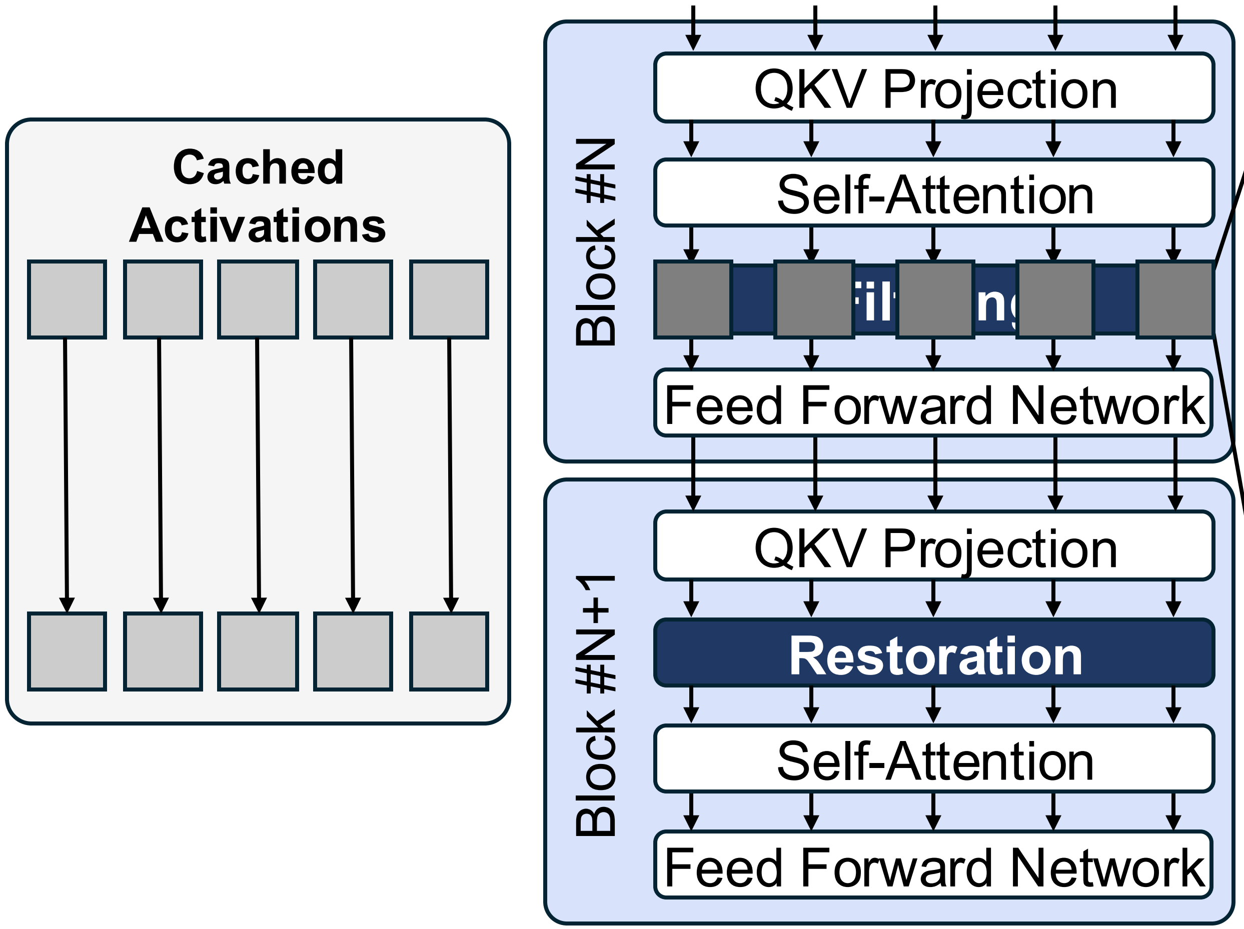
Tokens with similar inputs are more suitable for reuse.

2. Attention Score

Tokens with high attention are more likely to be recomputed.

These signals can conflict, making reuse decisions non-trivial.

Example Flow: Other Frames with Reuse



Reuse decision made at filtering stage

Inputs

1. Similarity Score
2. Attention Score

*Details for other inputs omitted from this talk.

Decision Layer

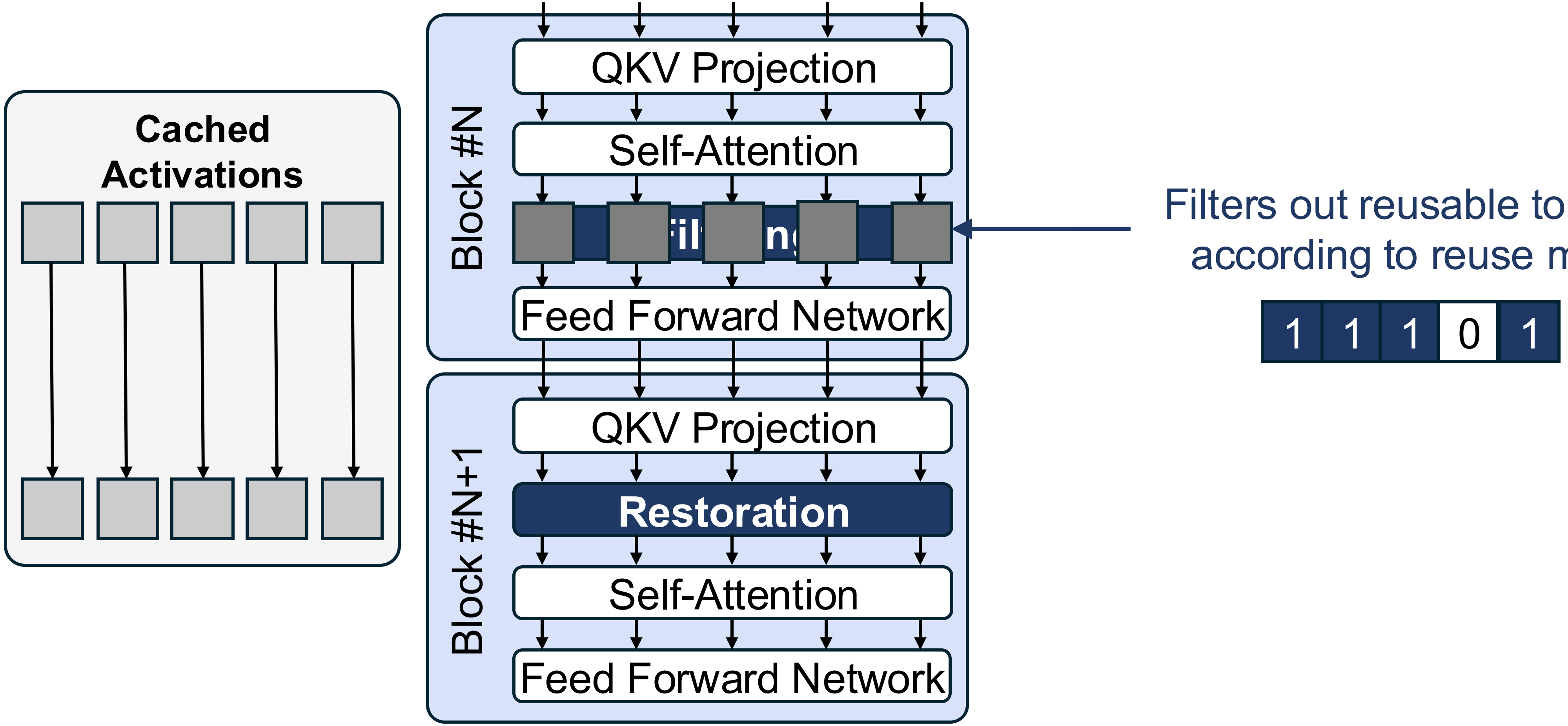
Simple three-layer MLP learns how to weigh these factors

Output

1	1	1	0	1

Binary decision we call “Reuse Map”

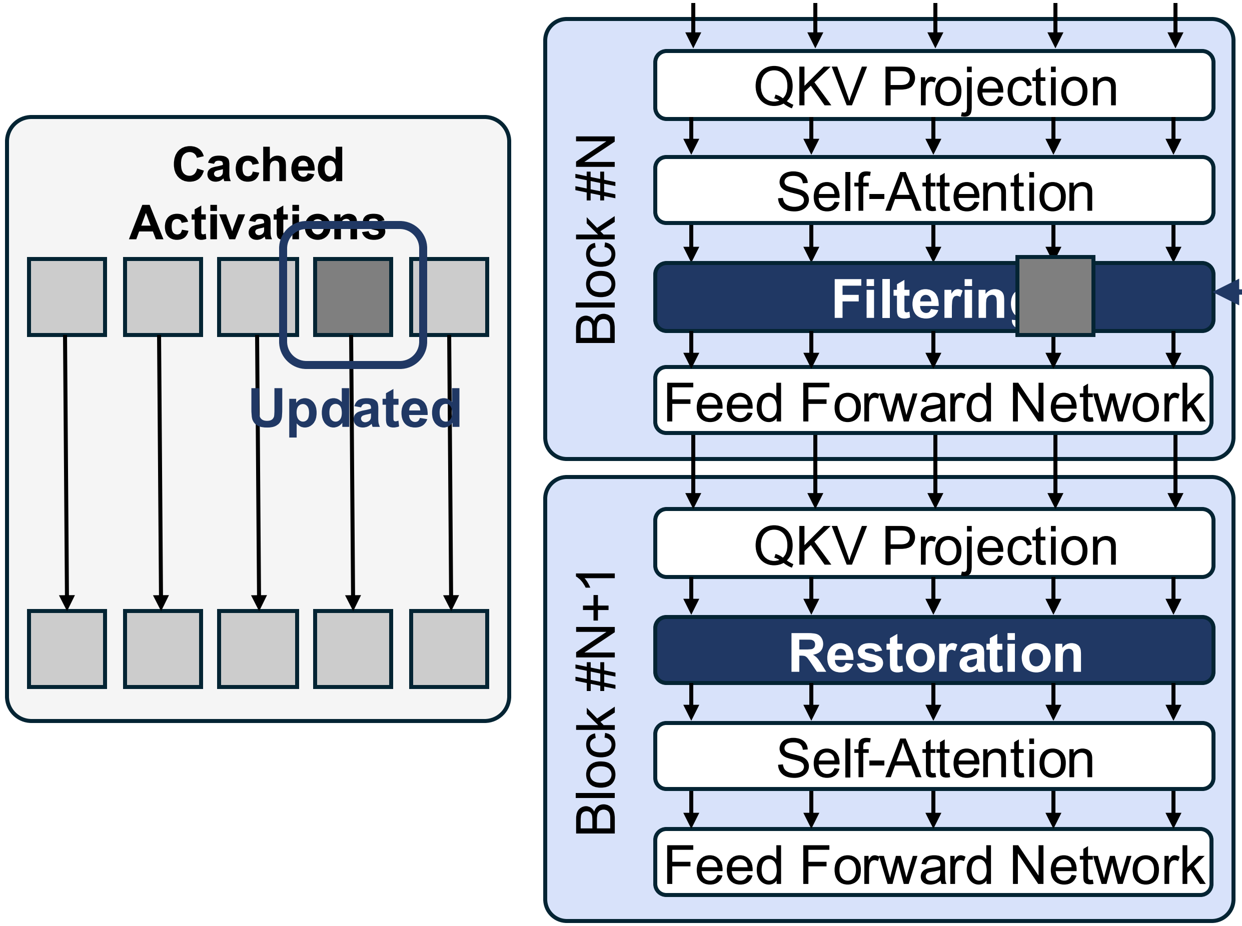
Example Flow: Other Frames with Reuse



Filters out reusable tokens according to reuse map

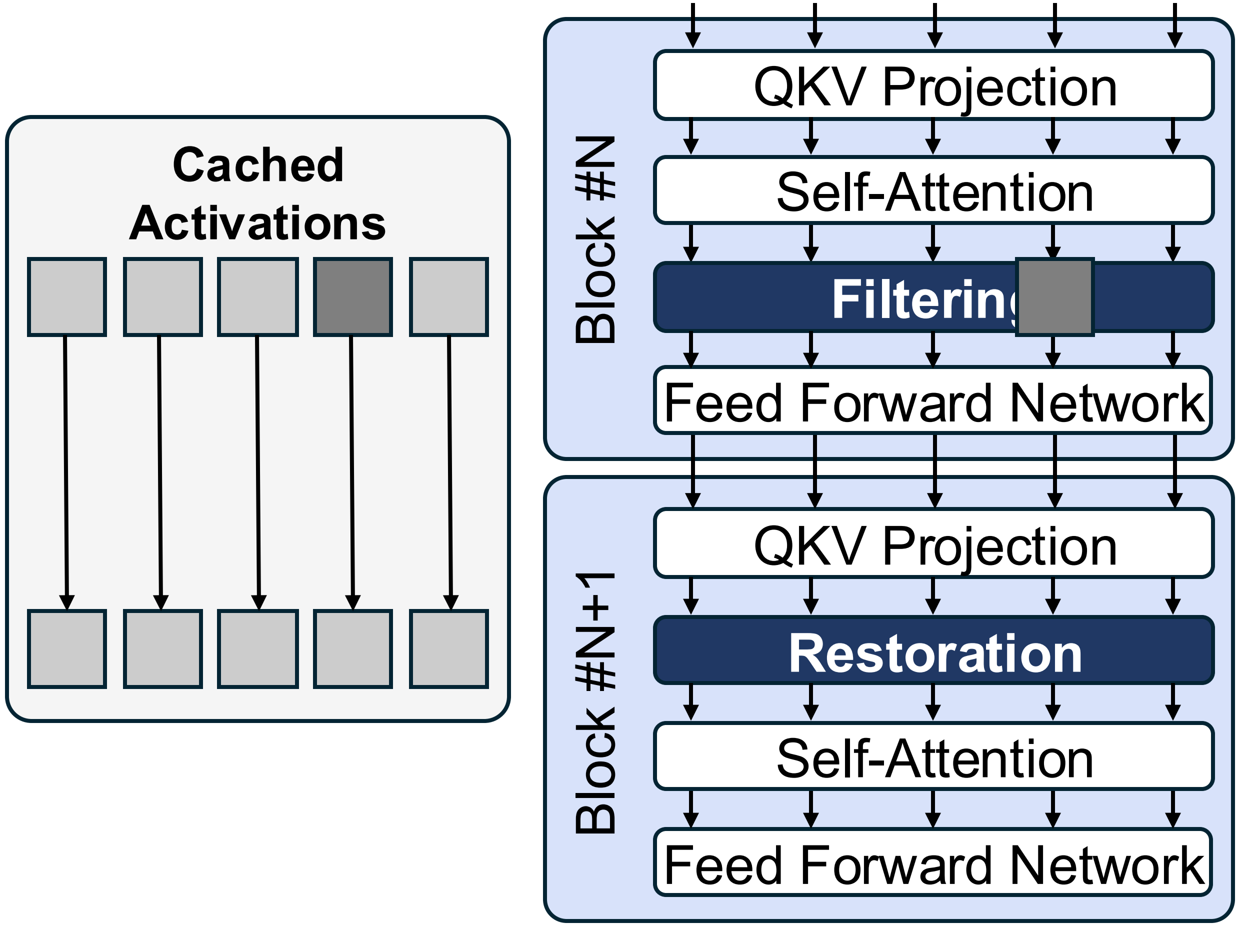
1 1 1 0 1

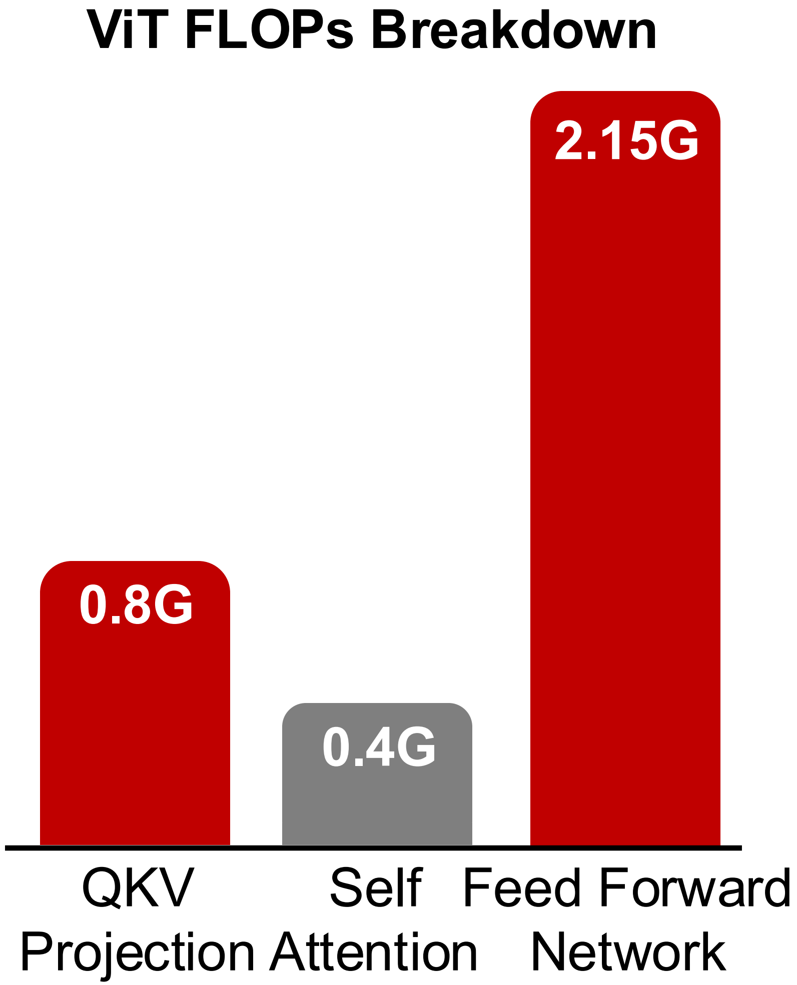
Example Flow: Other Frames with Reuse



Update the caches
for future reference

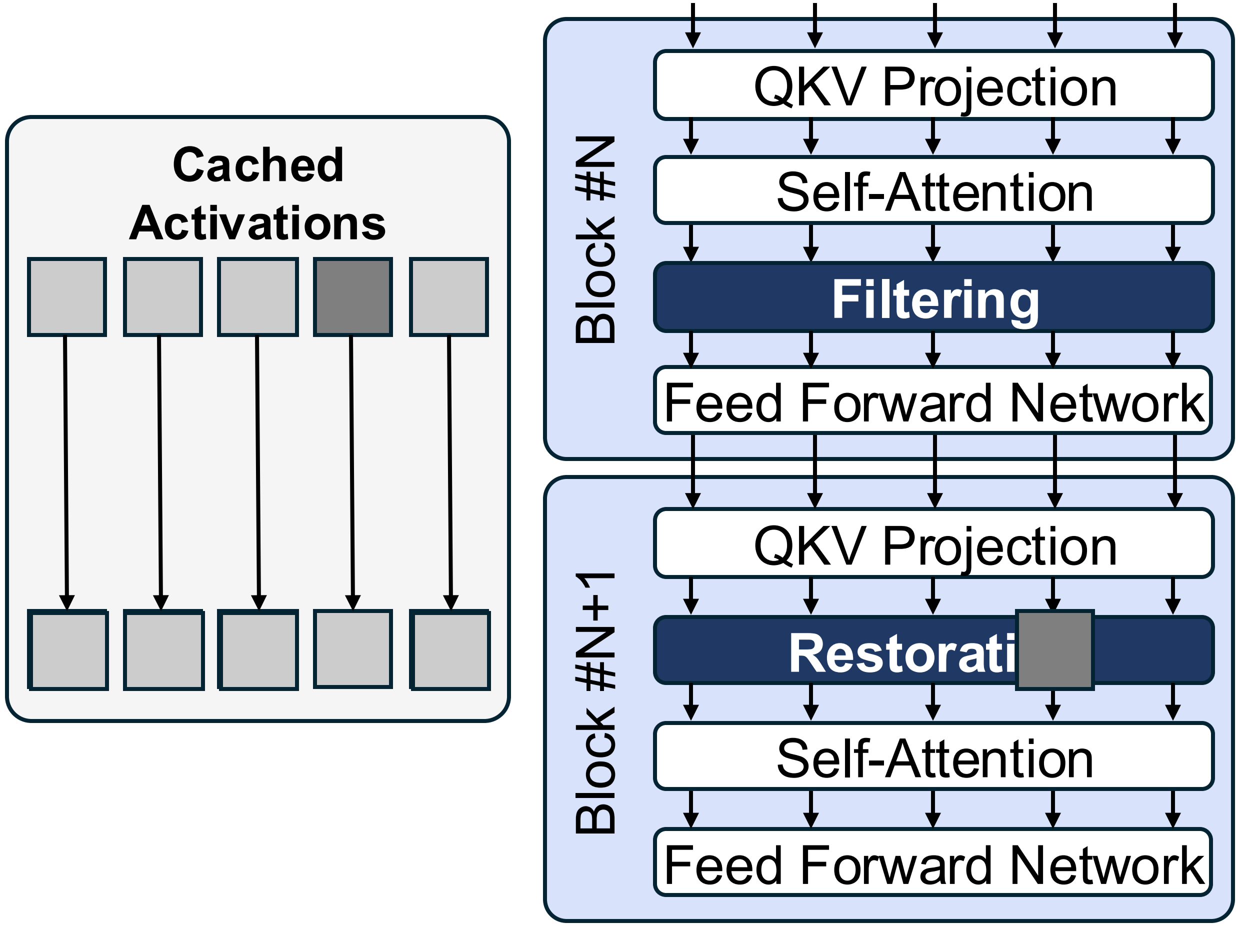
Example Flow: Other Frames with Reuse

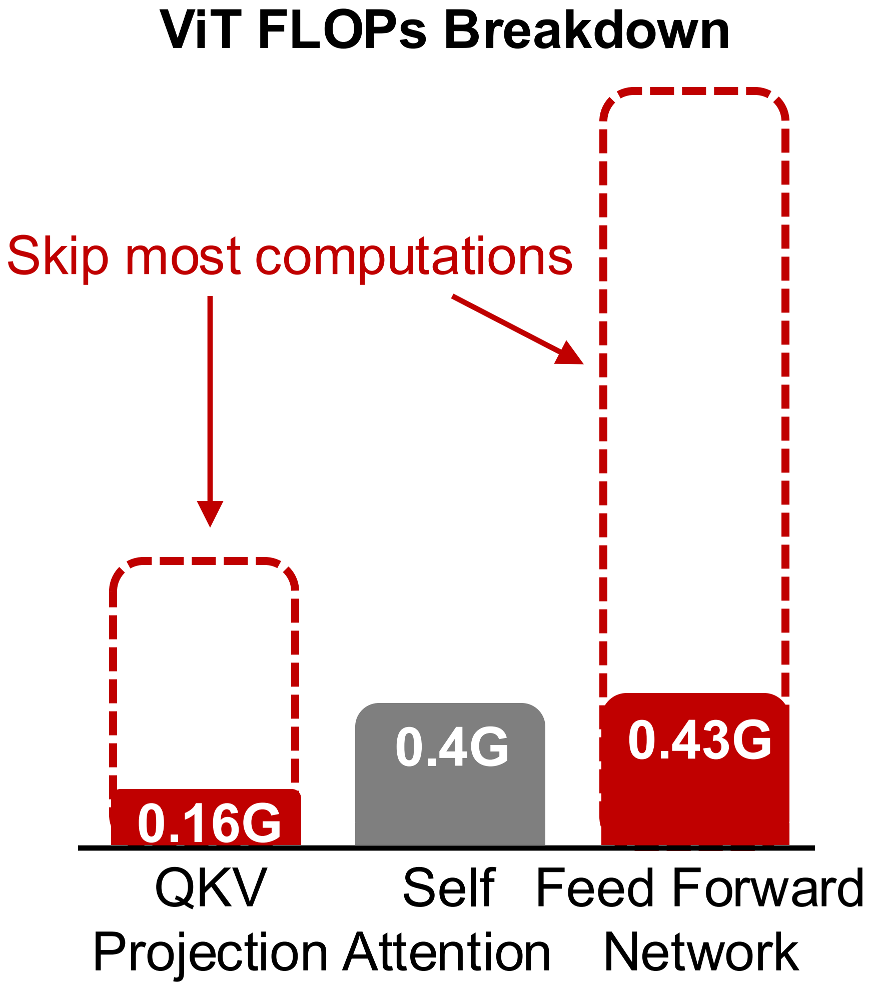




*ViT-large-patch14-336px, 80% reuse

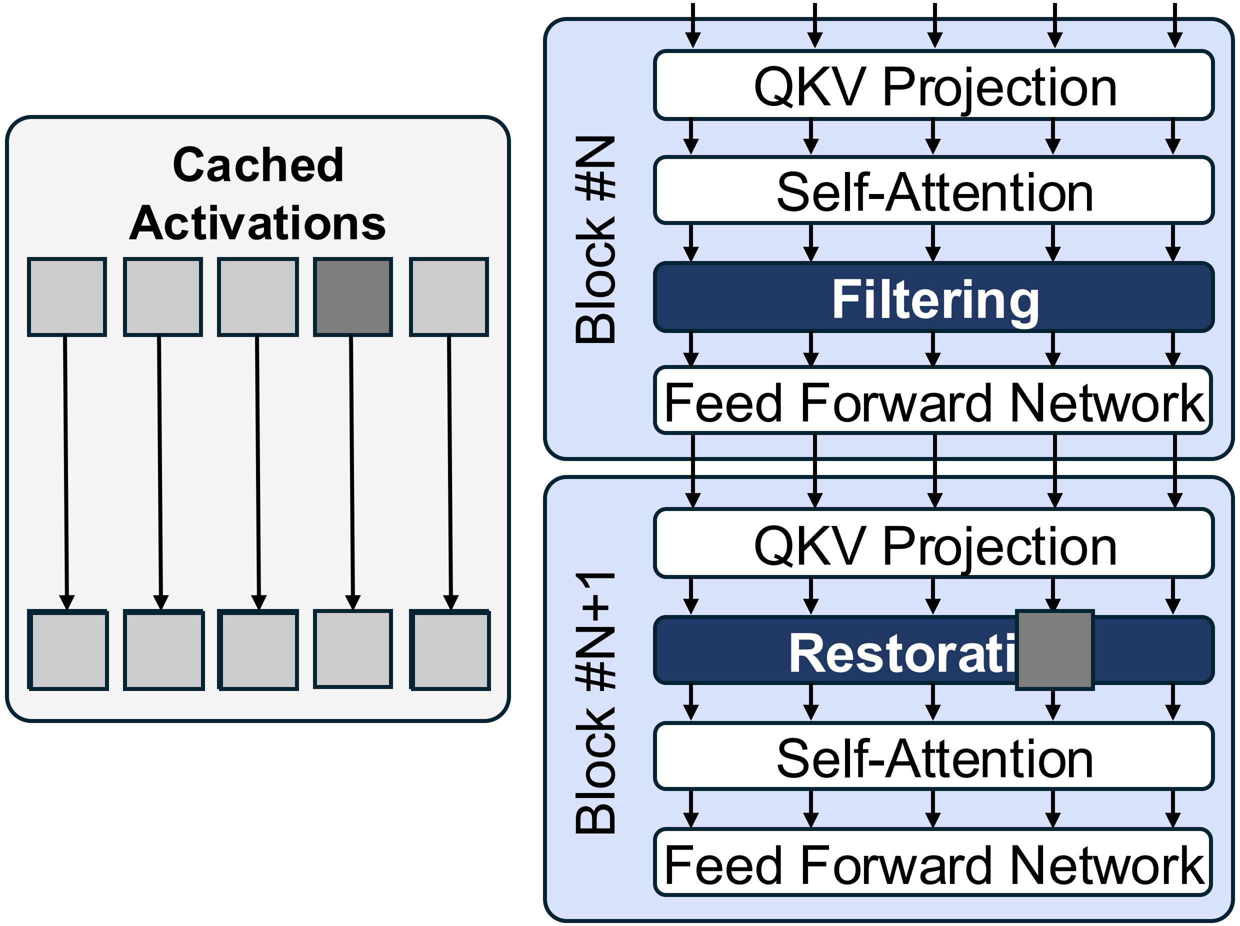
Example Flow: Other Frames with Reuse





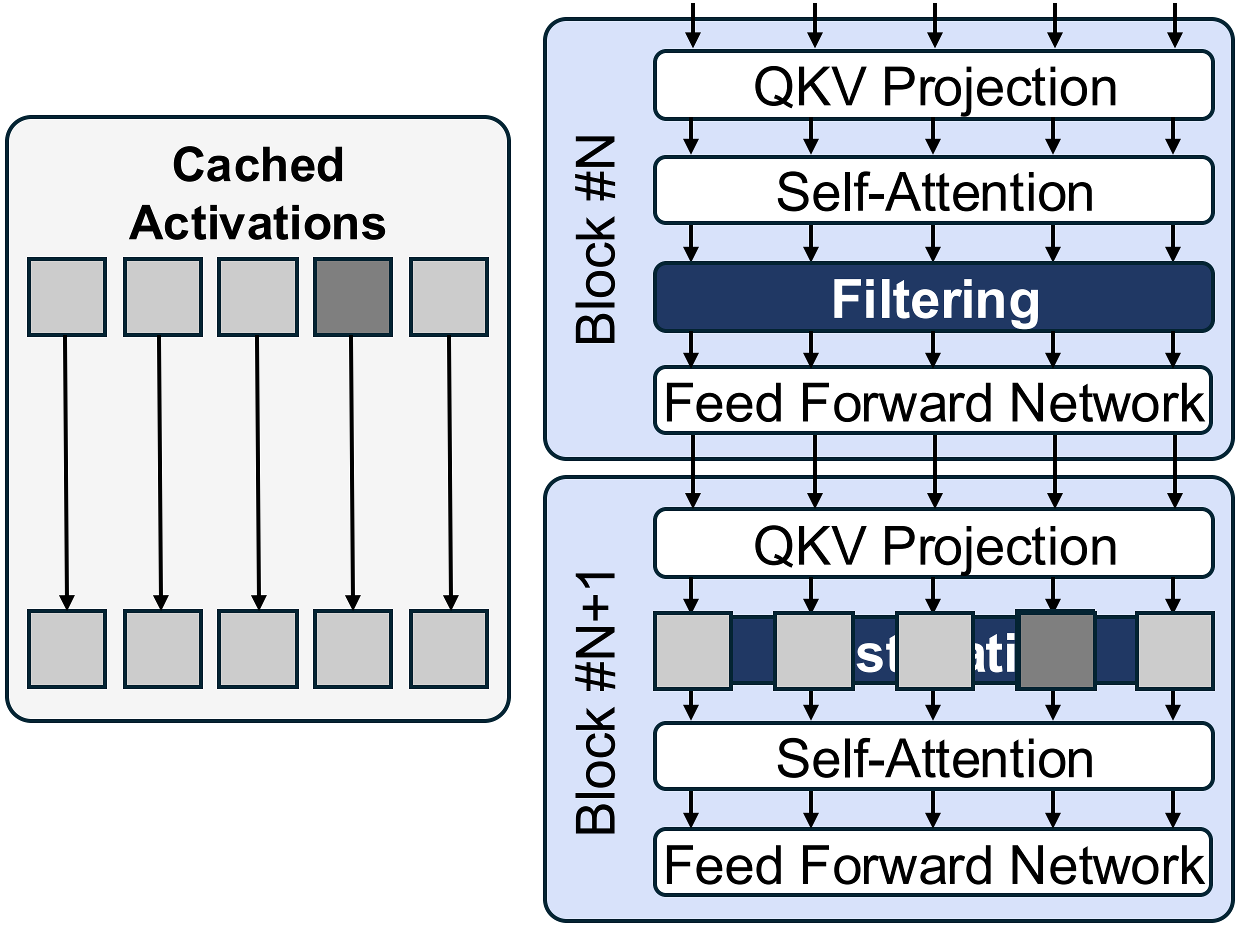
*ViT-large-patch14-336px, 80% reuse

Example Flow: Other Frames with Reuse



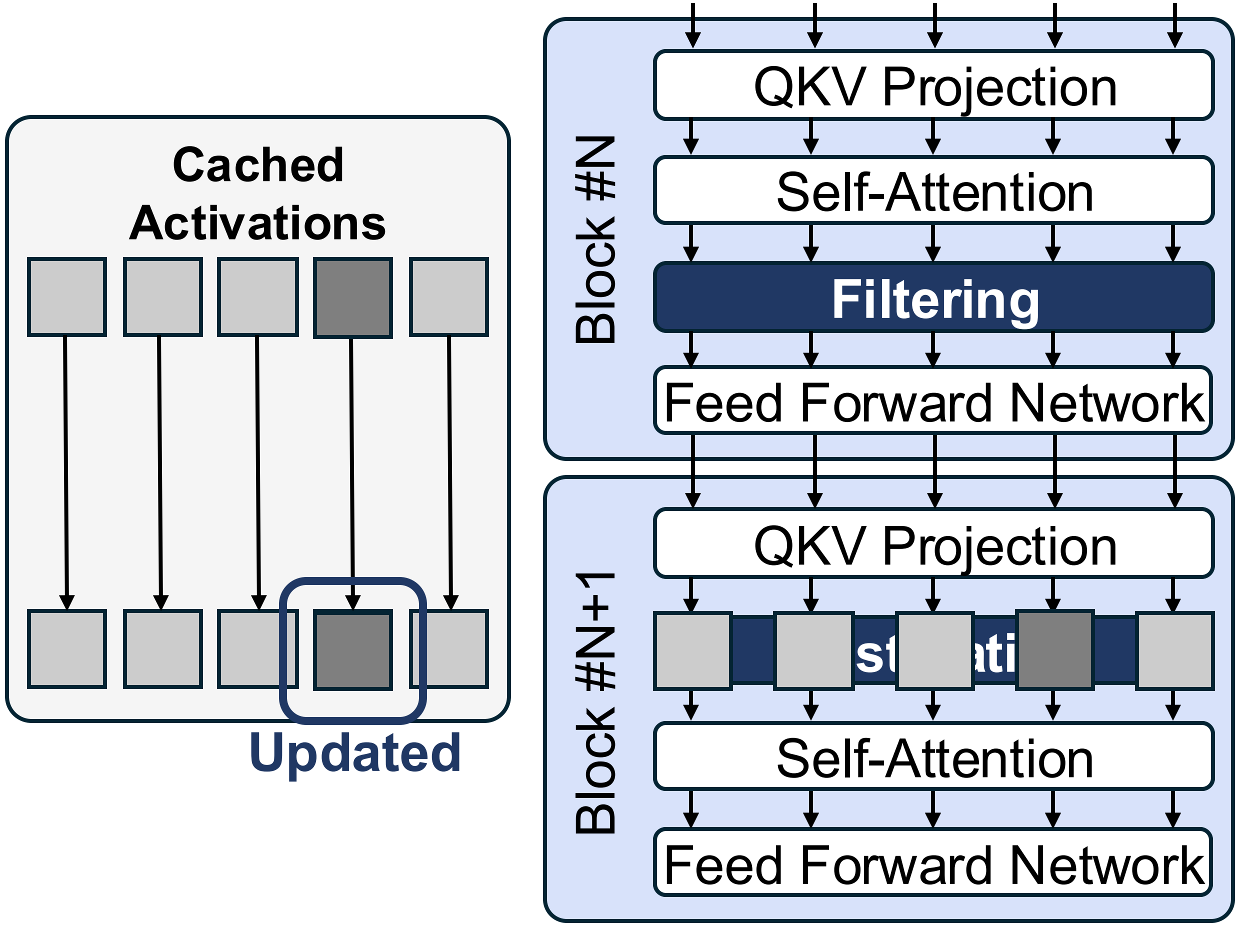
- For reused tokens, we fetch and restore cached outputs from the previous frame.

Example Flow: Other Frames with Reuse



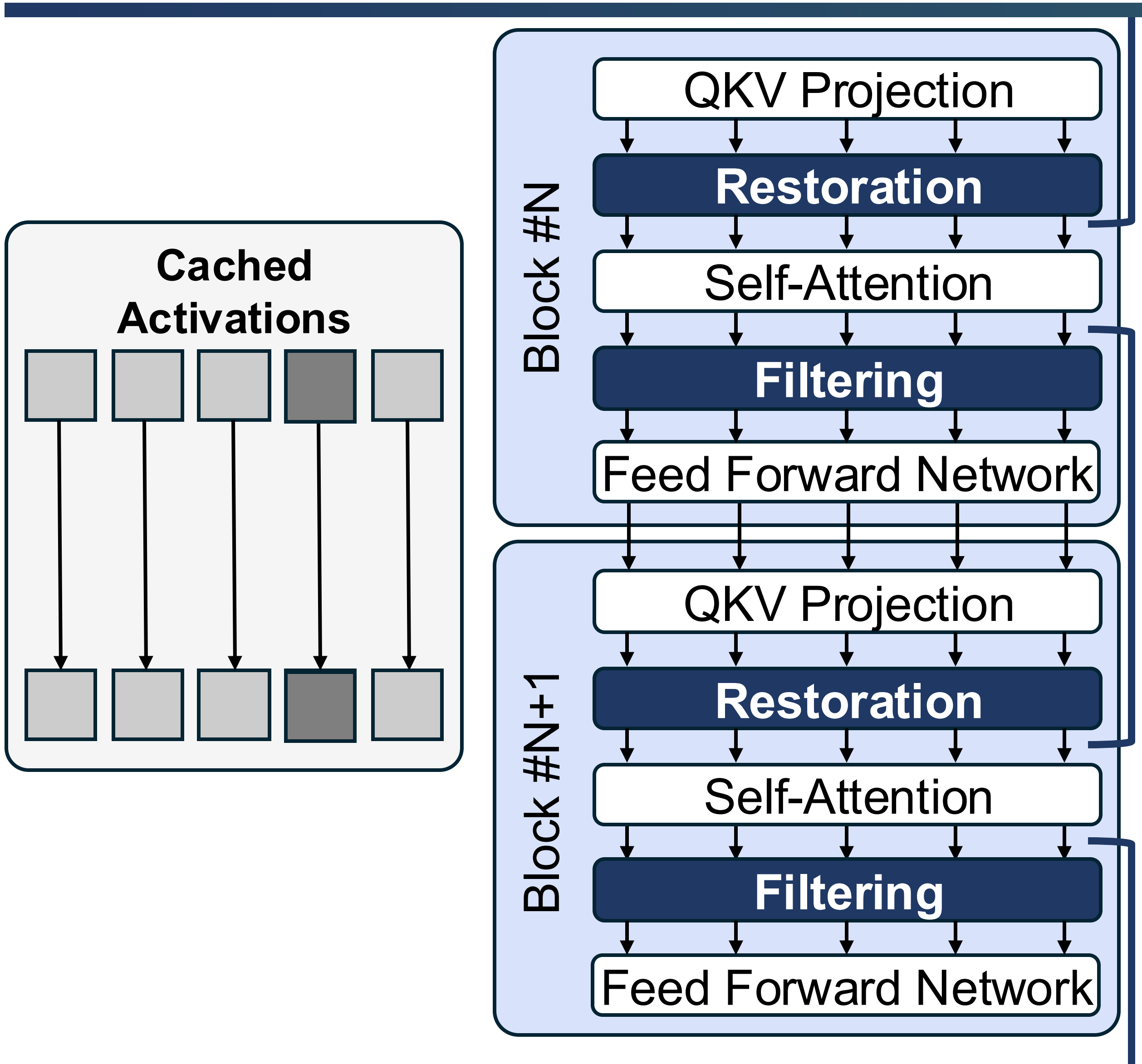
- For reused tokens, we fetch and restore cached outputs from the previous frame.

Example Flow: Other Frames with Reuse



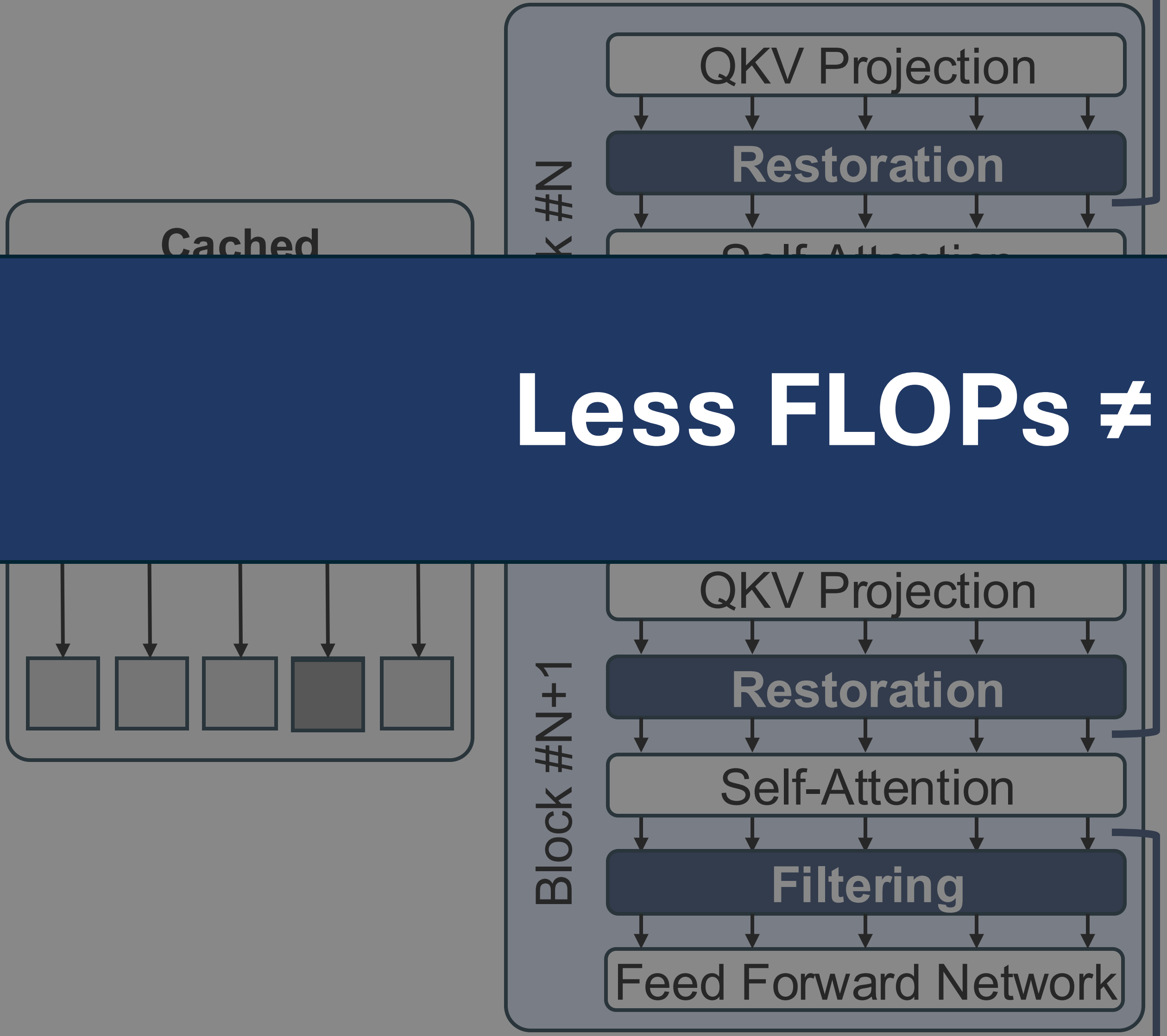
- For reused tokens, we fetch and restore cached outputs from the previous frame.
- Then, we update the cache for future reuse before moving to self-attention.

Example Flow: Other Frames with Reuse



- For reused tokens, we fetch and restore cached outputs from the previous frame.
- Then, we update the cache for future reuse before moving to self-attention.
- Each block repeats this process.

Example Flow: Other Frames with Reuse

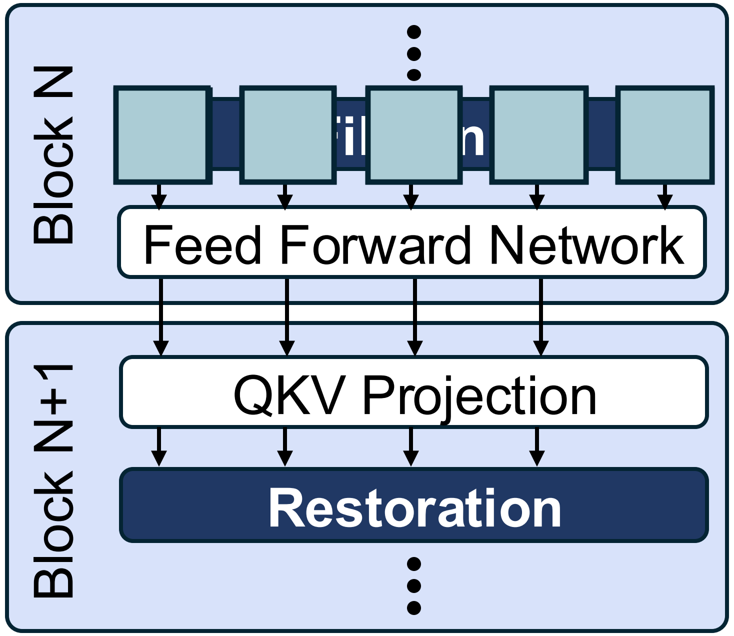


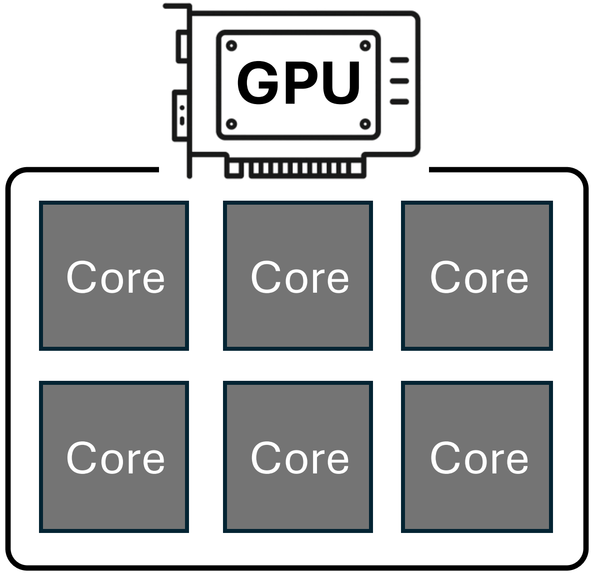
- For reused tokens, we fetch and restore cached outputs from the previous frame.

Less FLOPs ≠ Speedup

- Each block repeats this process.

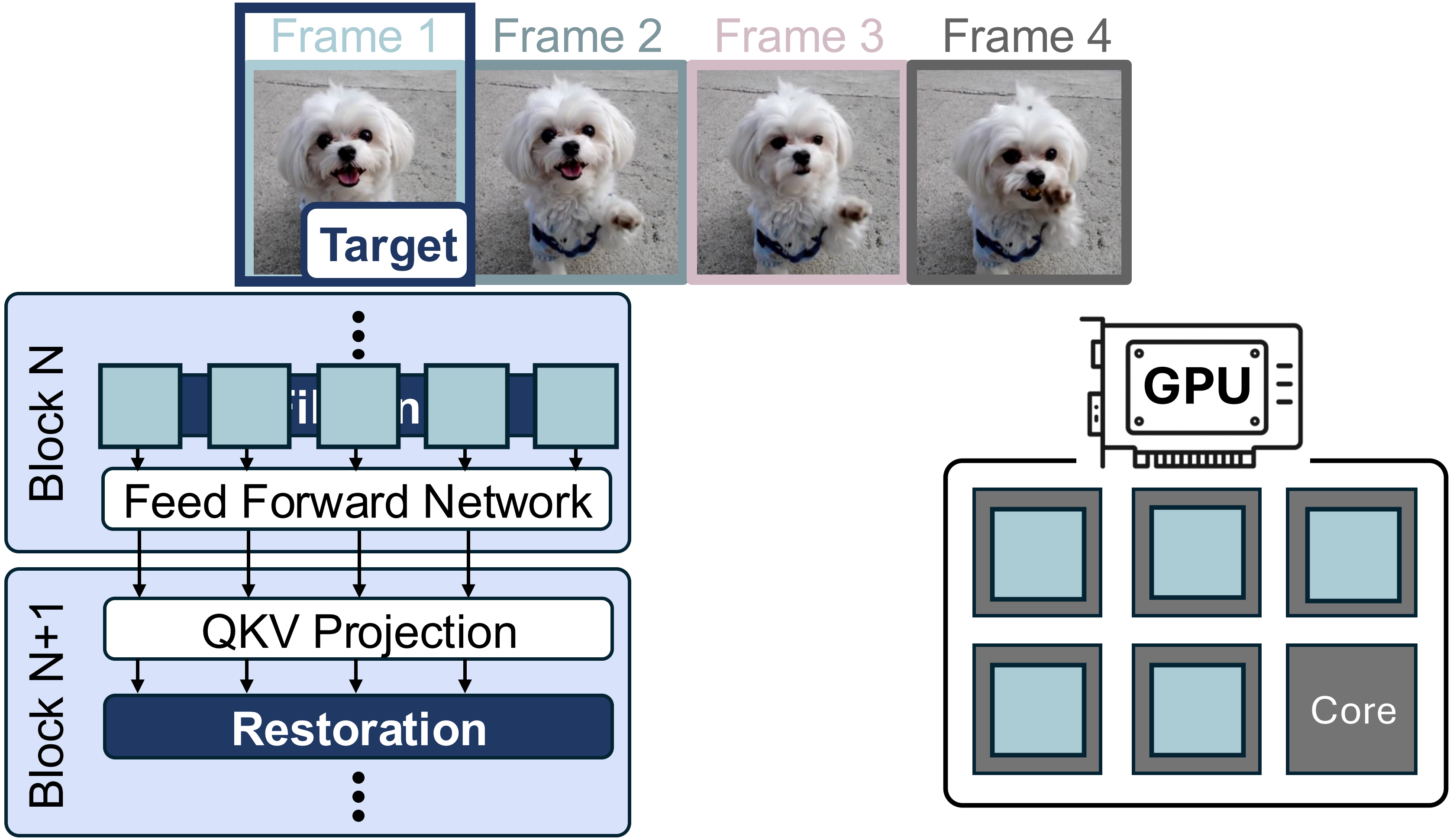
High GPU Utilization without Reuse





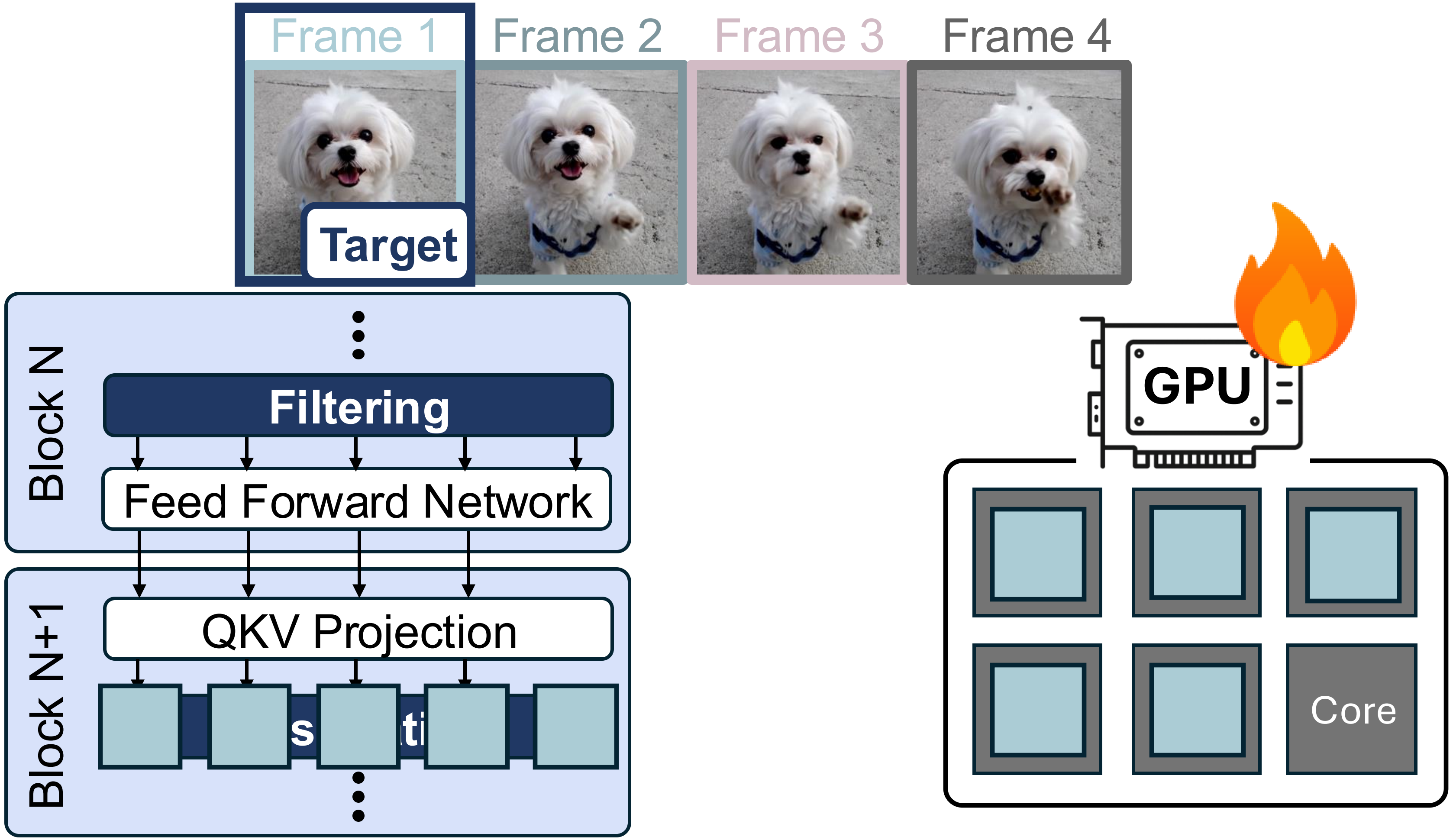
- GPUs thrive on dense, well-batched matrix multiplications.

High GPU Utilization without Reuse



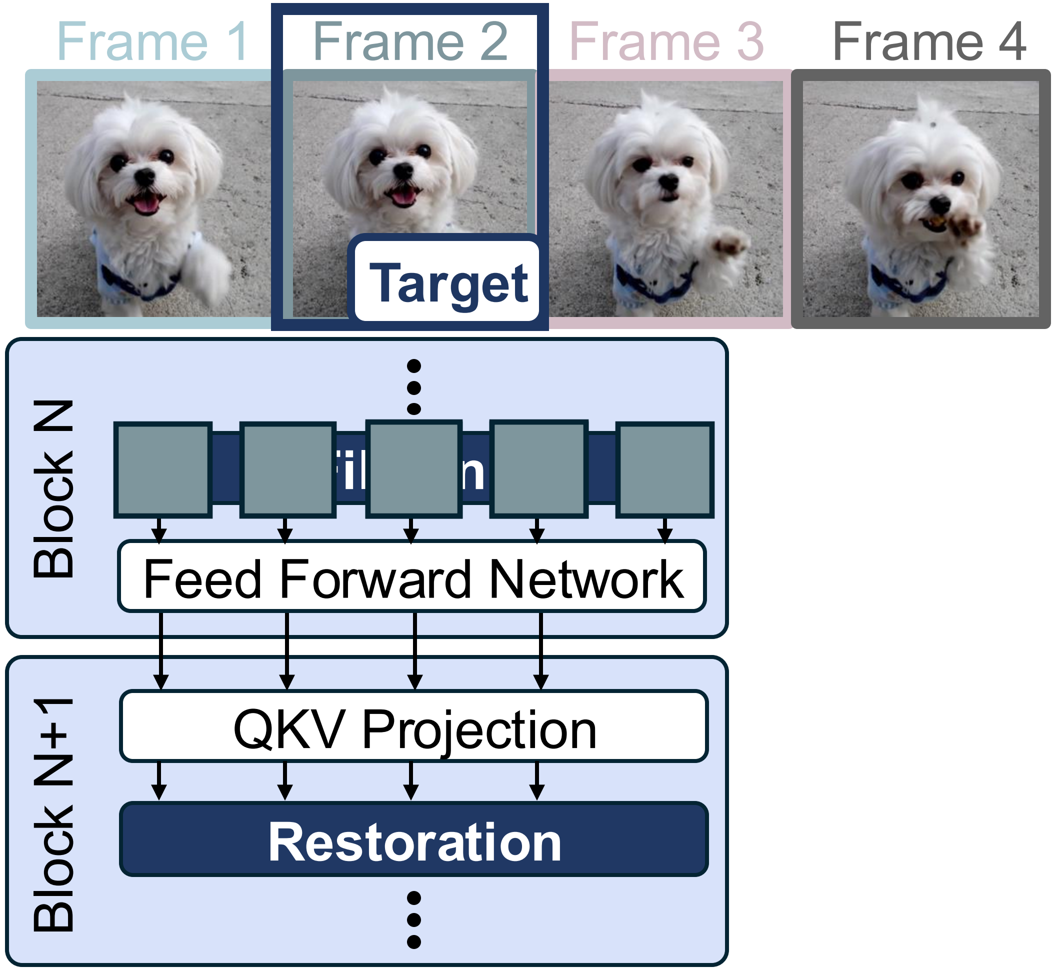
- GPUs thrive on dense, well-batched matrix multiplications.

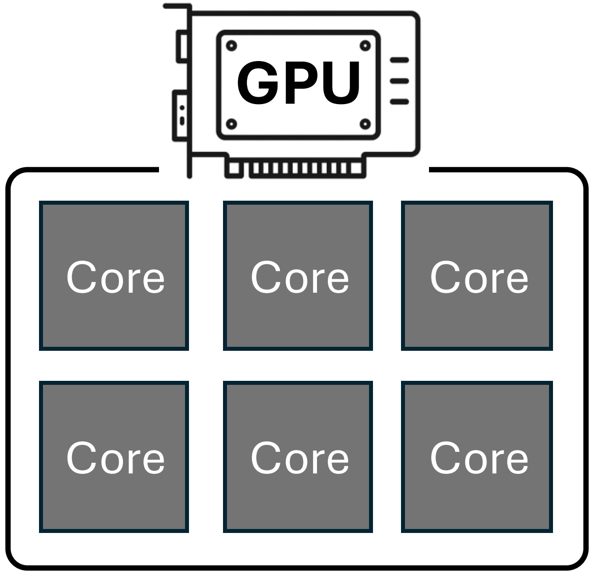
High GPU Utilization without Reuse



- GPUs thrive on dense, well-batched matrix multiplications.

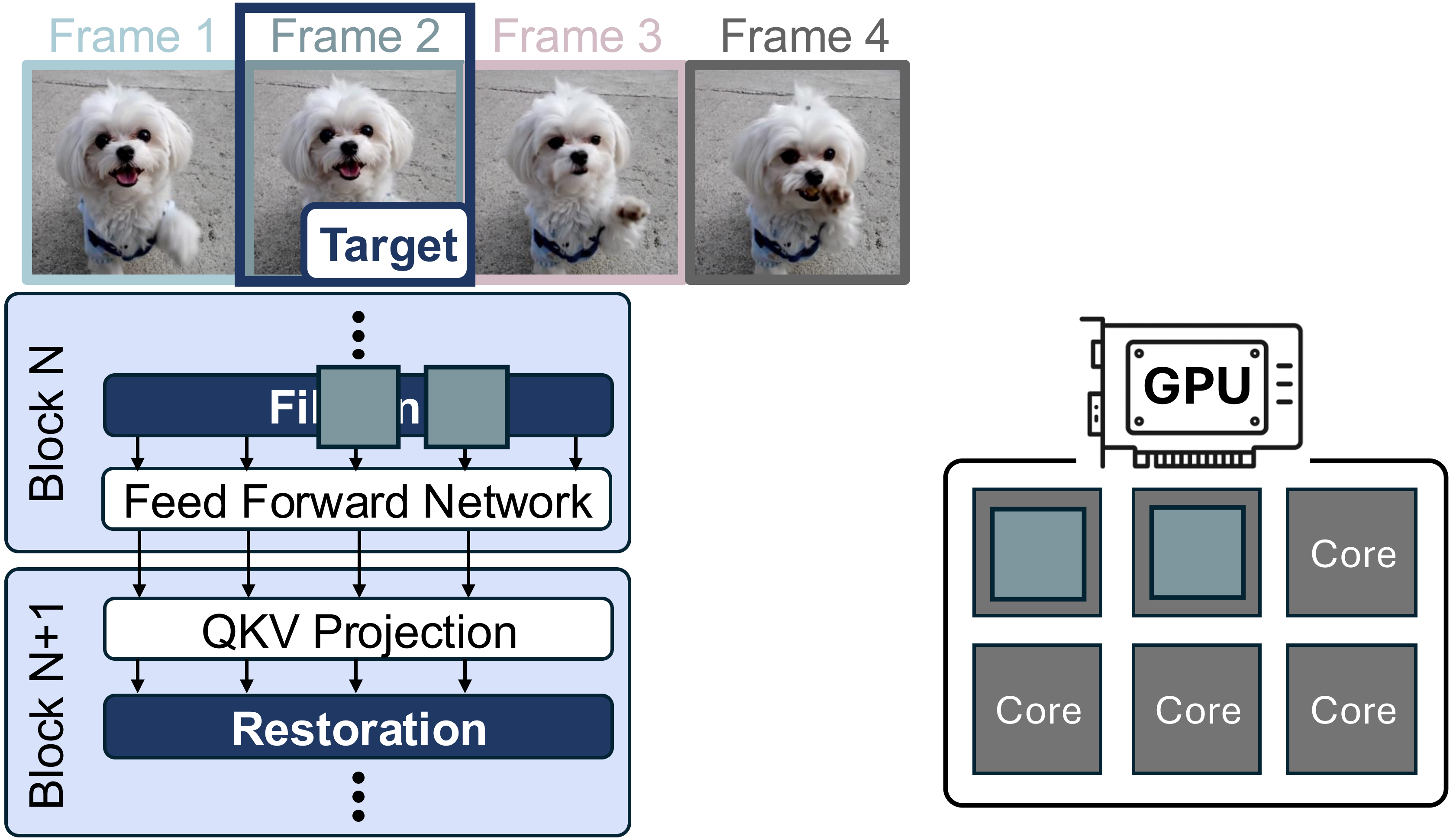
Low GPU Utilization Issue with Reuse





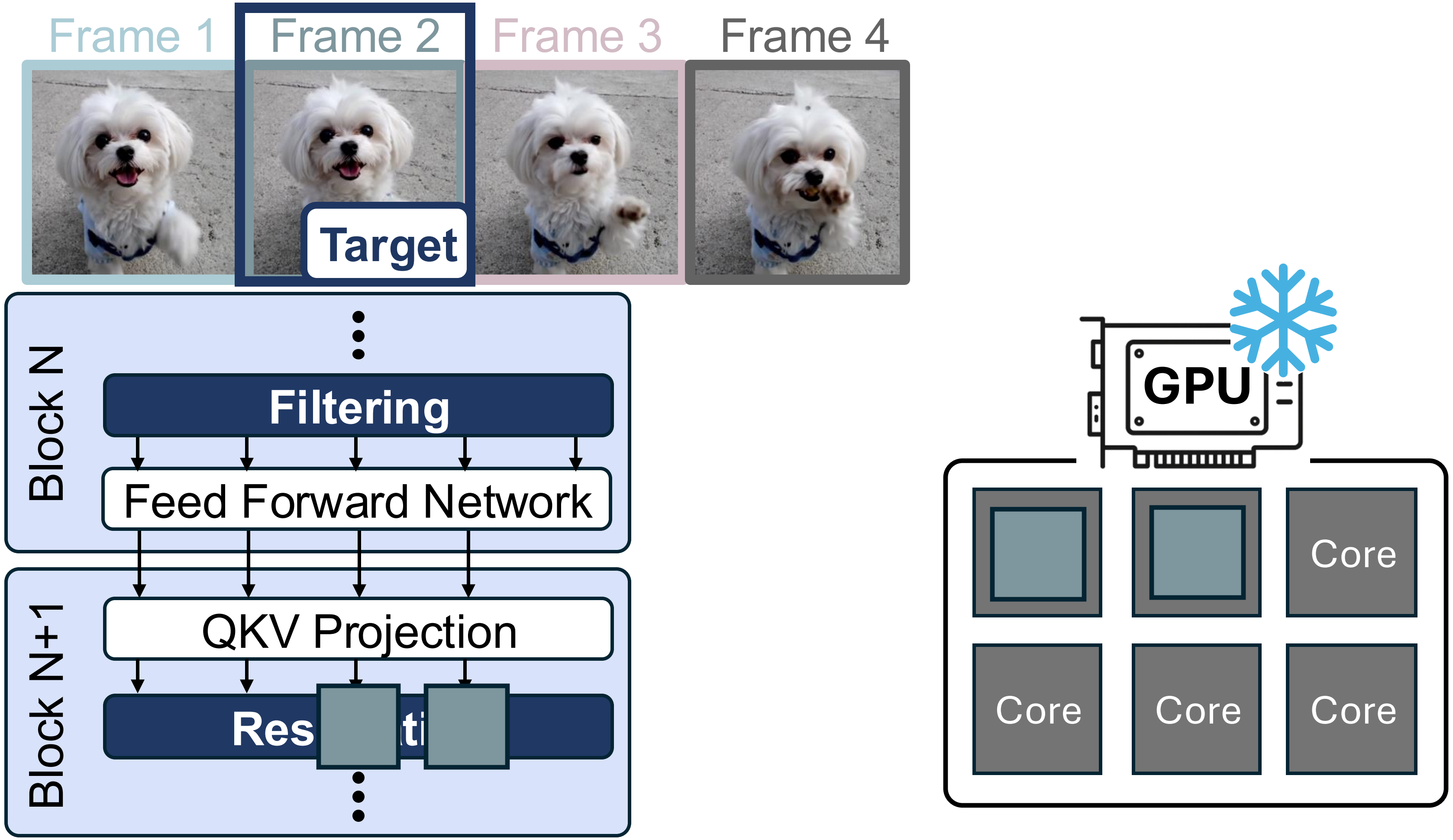
- High reuse makes the workload sparse and hurts utilization.

Low GPU Utilization Issue with Reuse



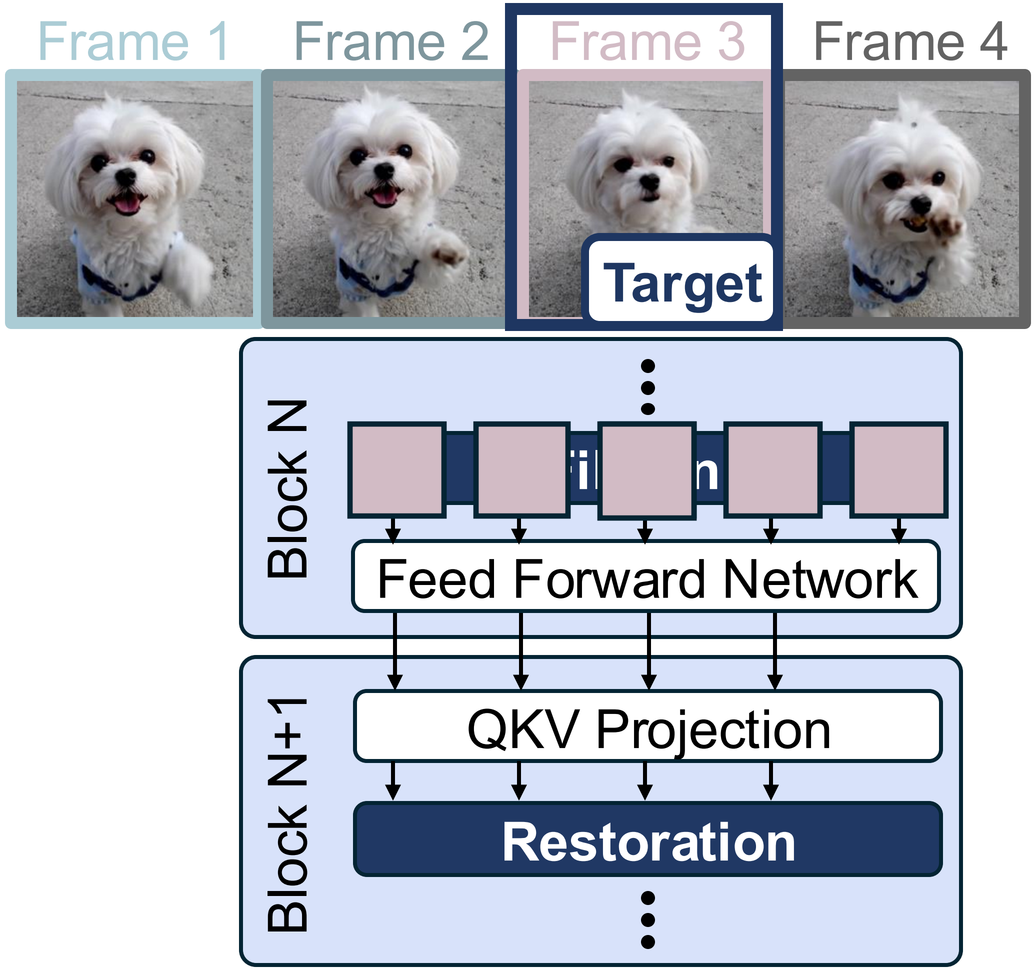
- High reuse makes the workload sparse and hurts utilization.

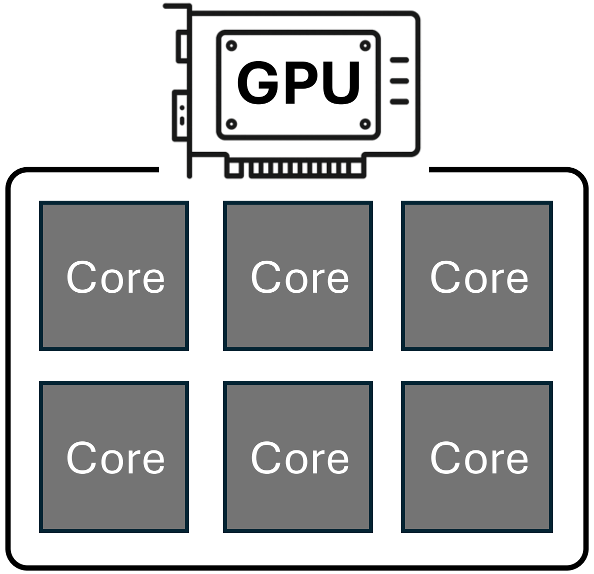
Low GPU Utilization Issue with Reuse



- High reuse makes the workload sparse and hurts utilization.

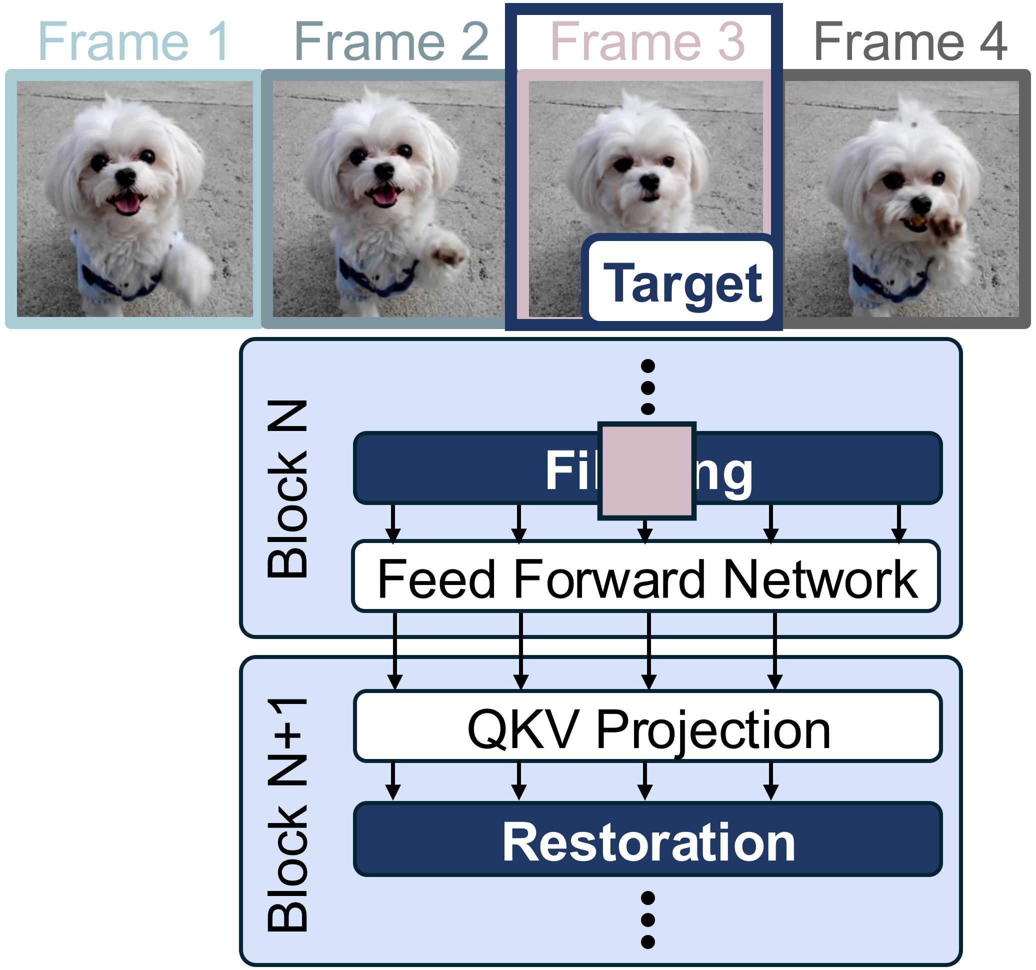
Low GPU Utilization Issue with Reuse

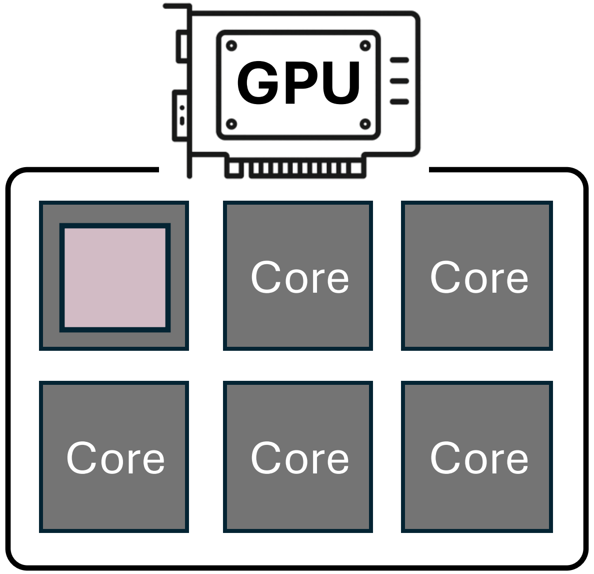




- High reuse makes the workload sparse and hurts utilization.

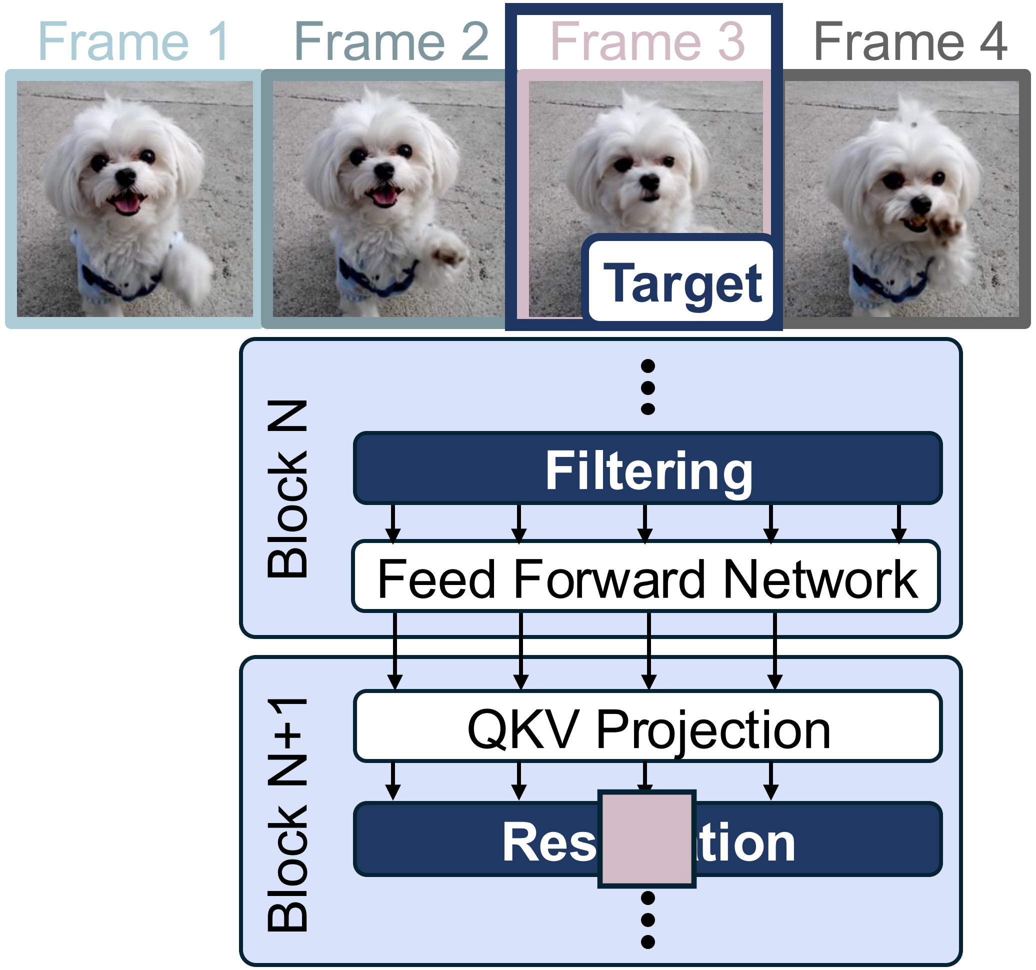
Low GPU Utilization Issue with Reuse

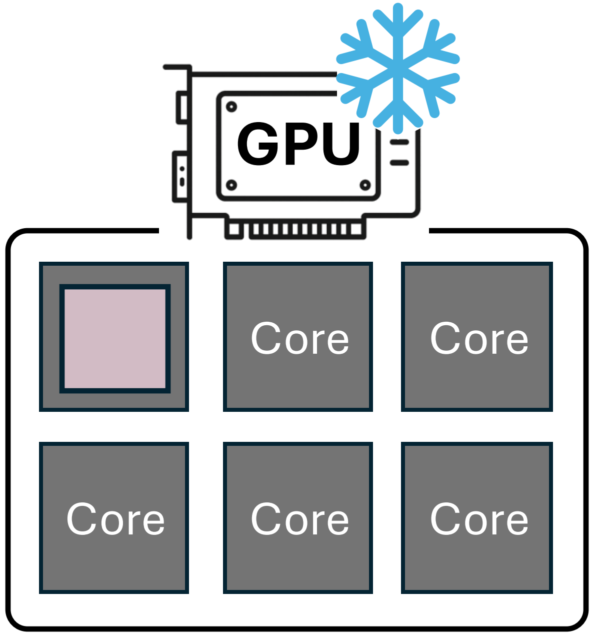




- High reuse makes the workload sparse and hurts utilization.

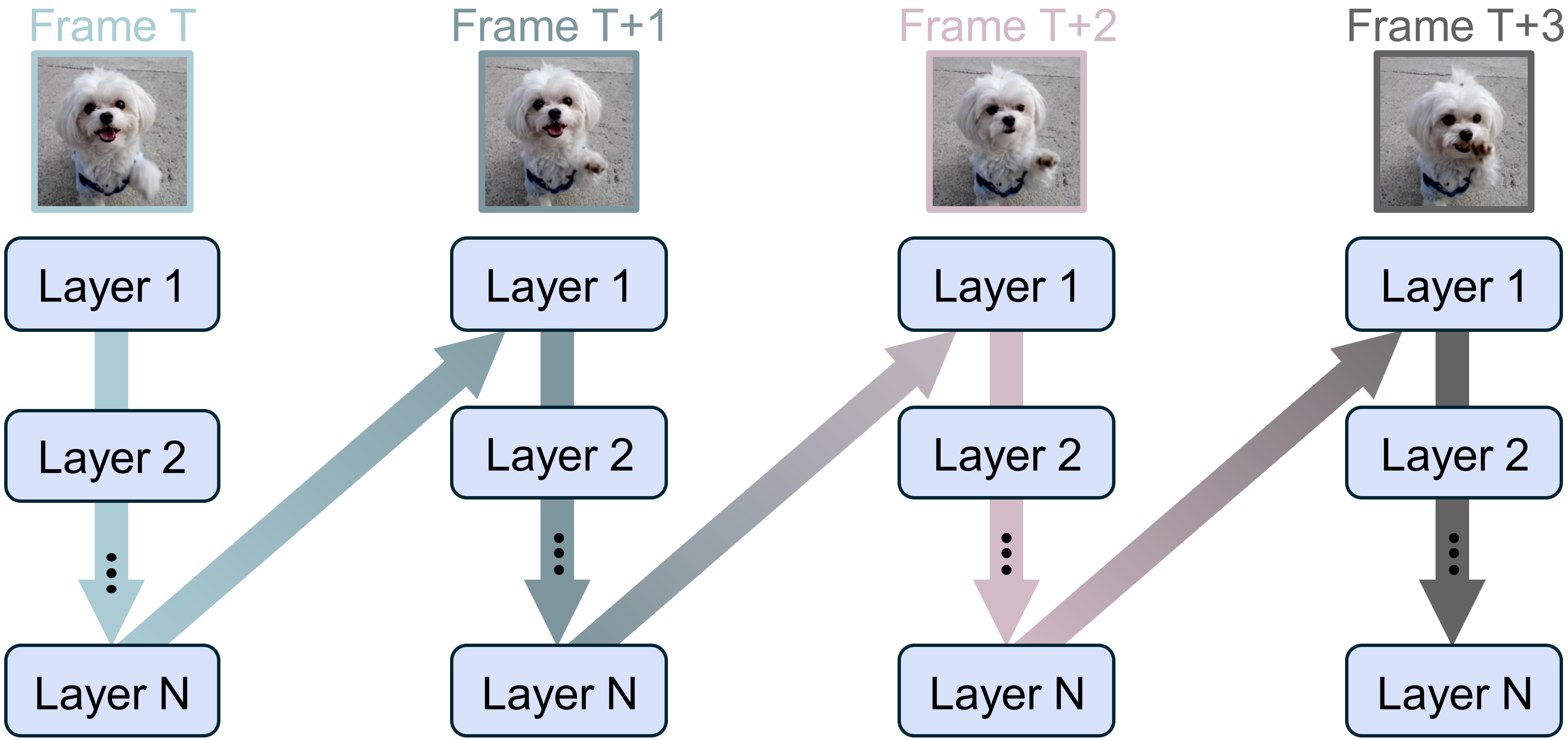
Low GPU Utilization Issue with Reuse





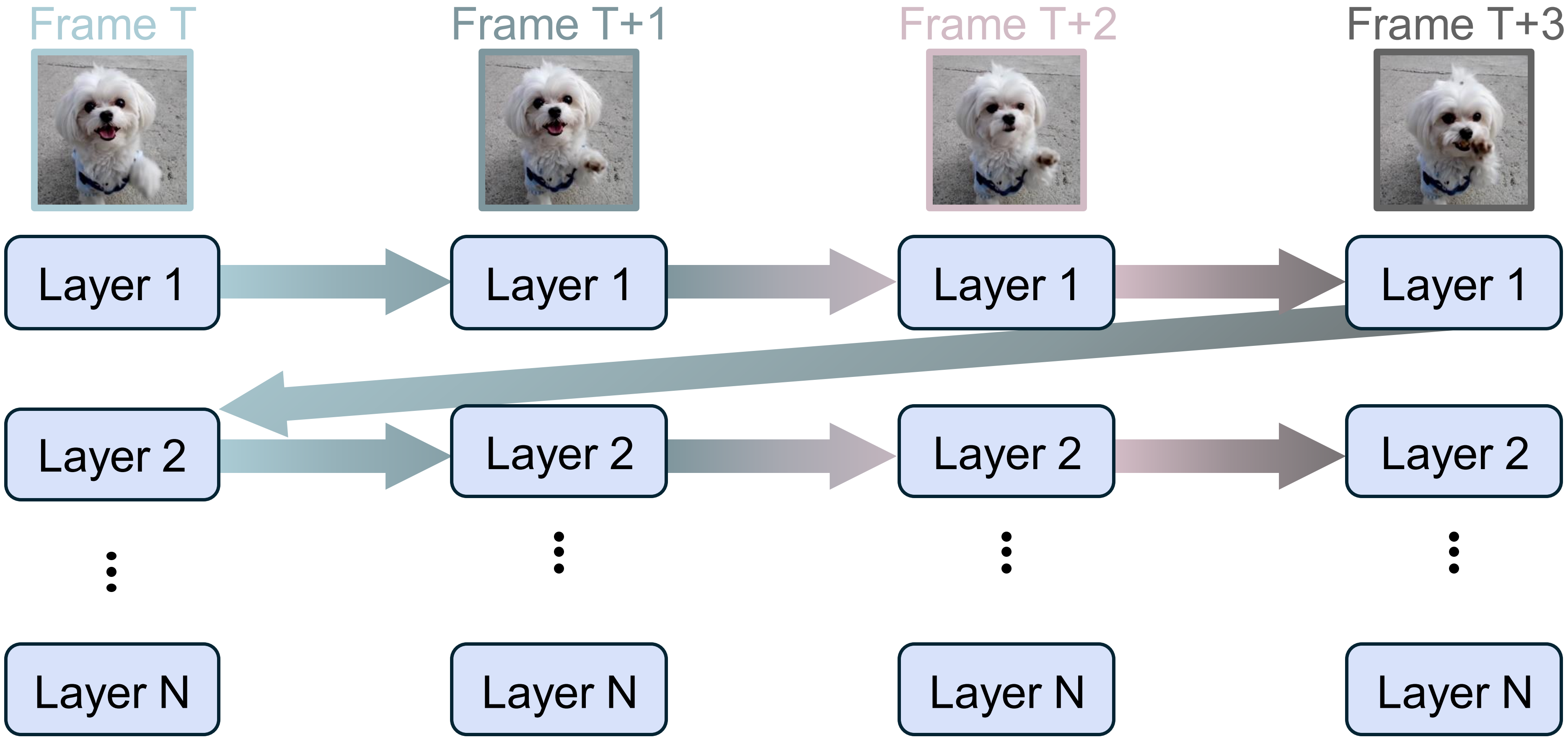
- High reuse makes the workload sparse and hurts utilization.

Conventional Scheduling



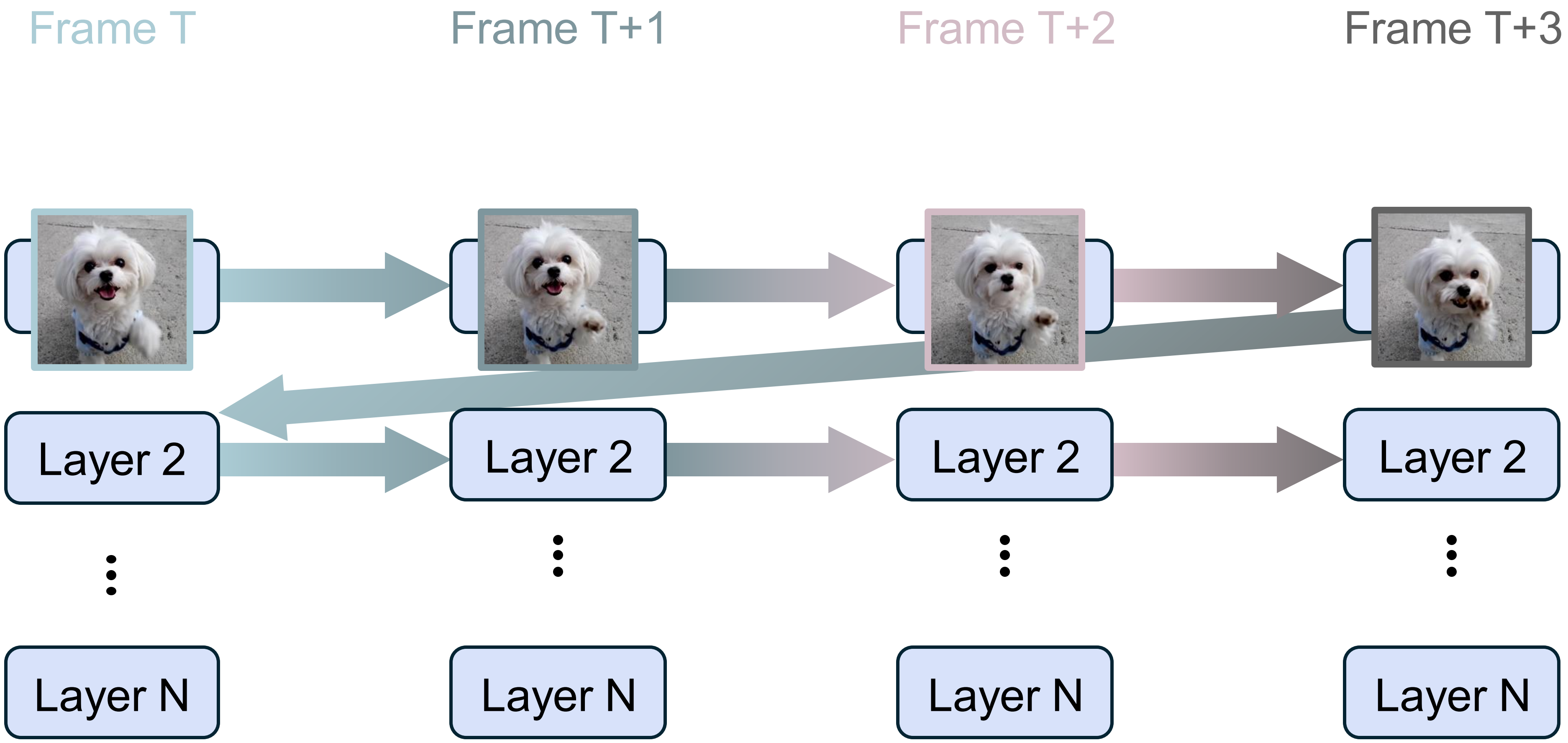
- Process each from through all layers before starting the next frame.

Layer-wise Scheduling



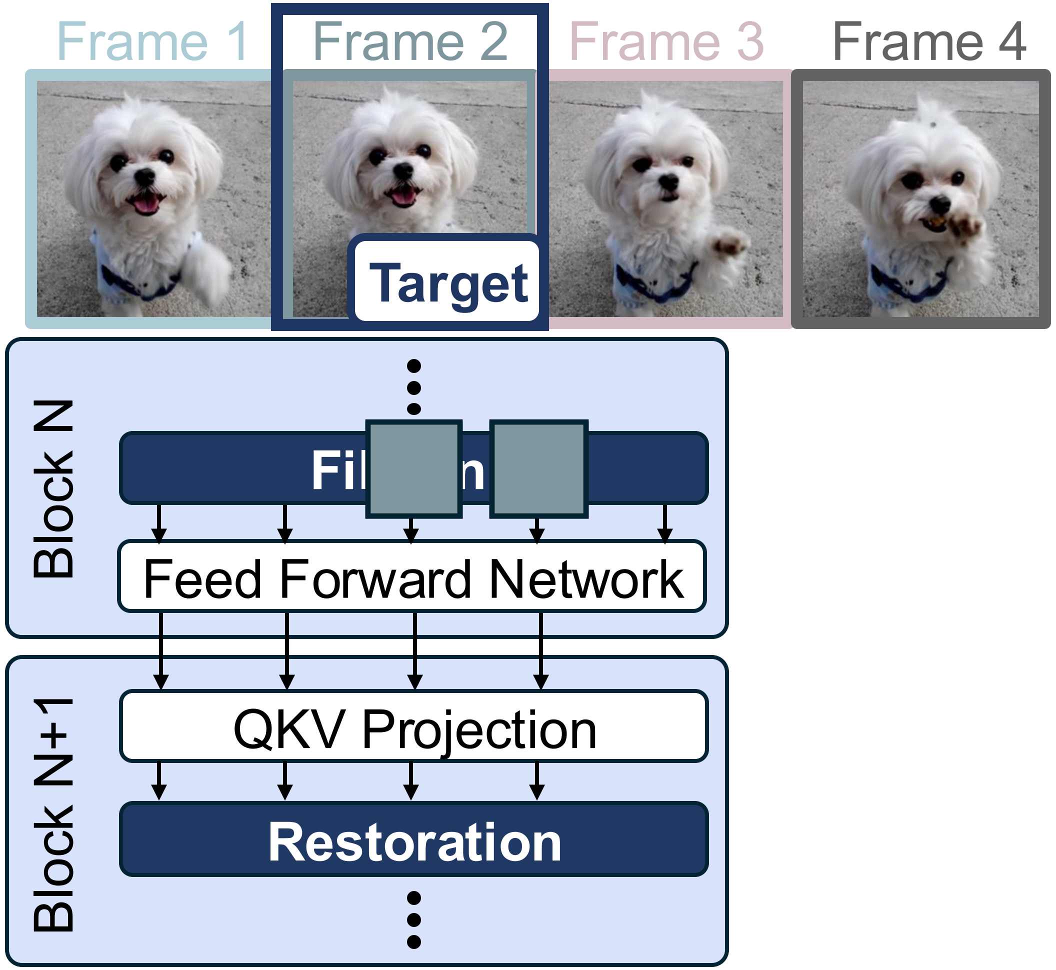
- Staggering frames across layers to improve computational efficiency.

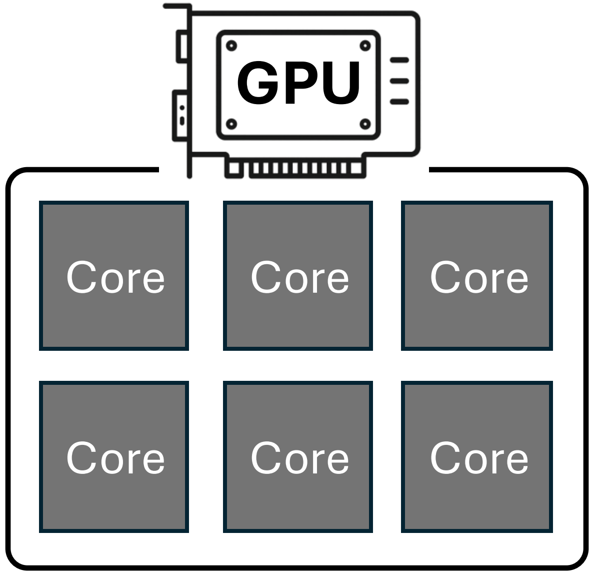
Layer-wise Scheduling



- Staggering frames across layers to improve computational efficiency.

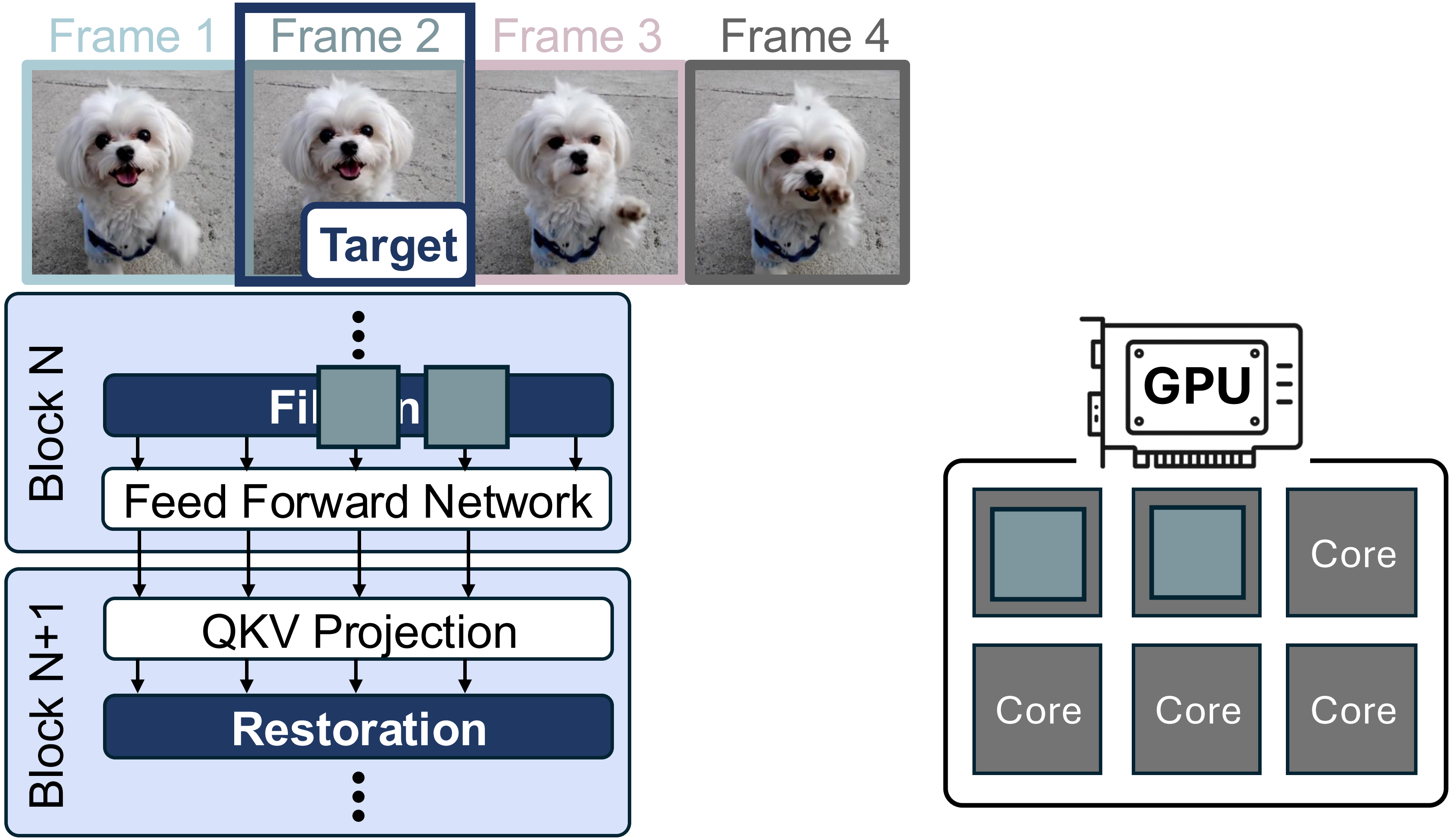
Sparse Computation Compaction





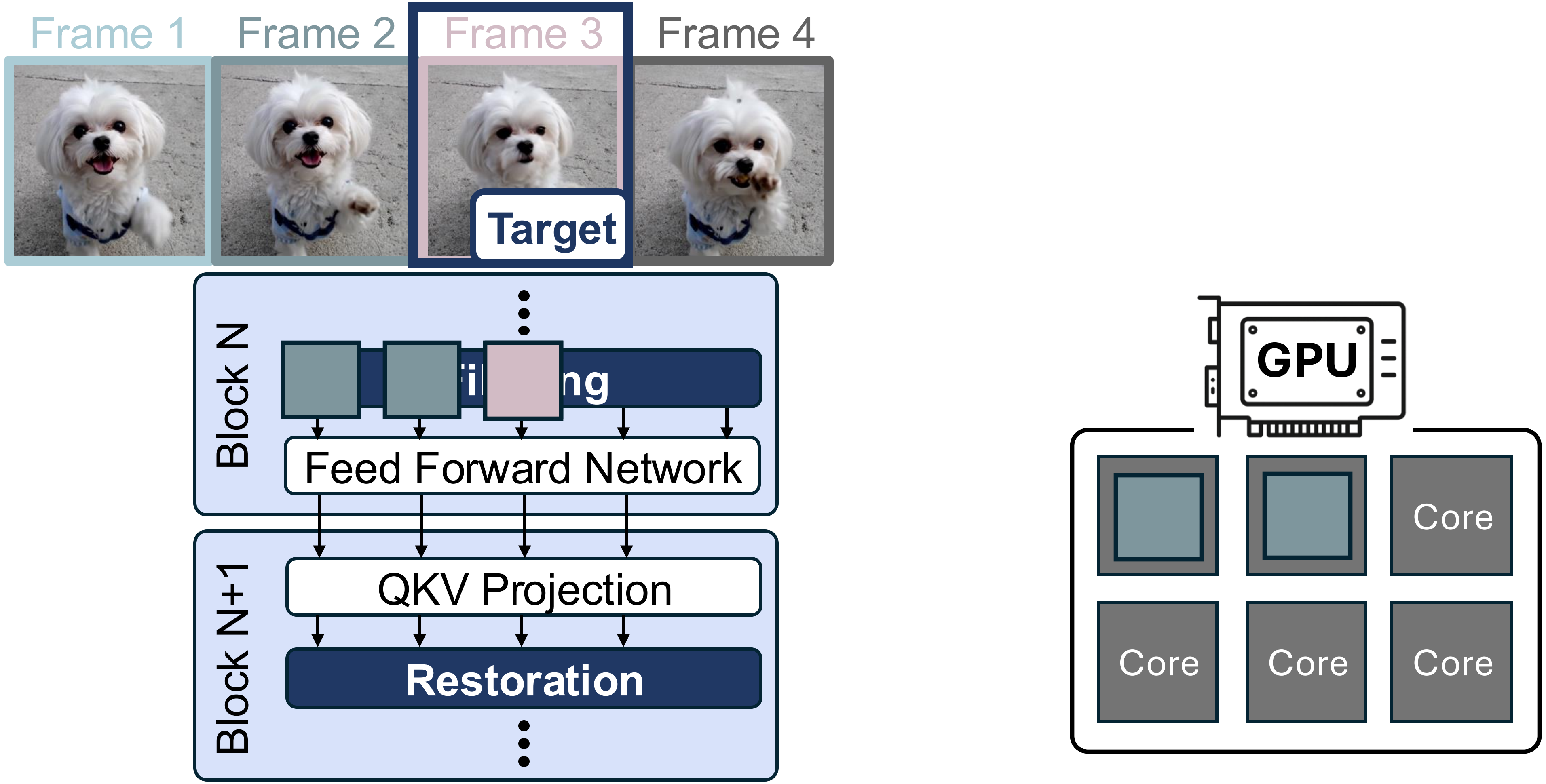
- Staggering frames across layers to improve computational efficiency

Sparse Computation Compaction



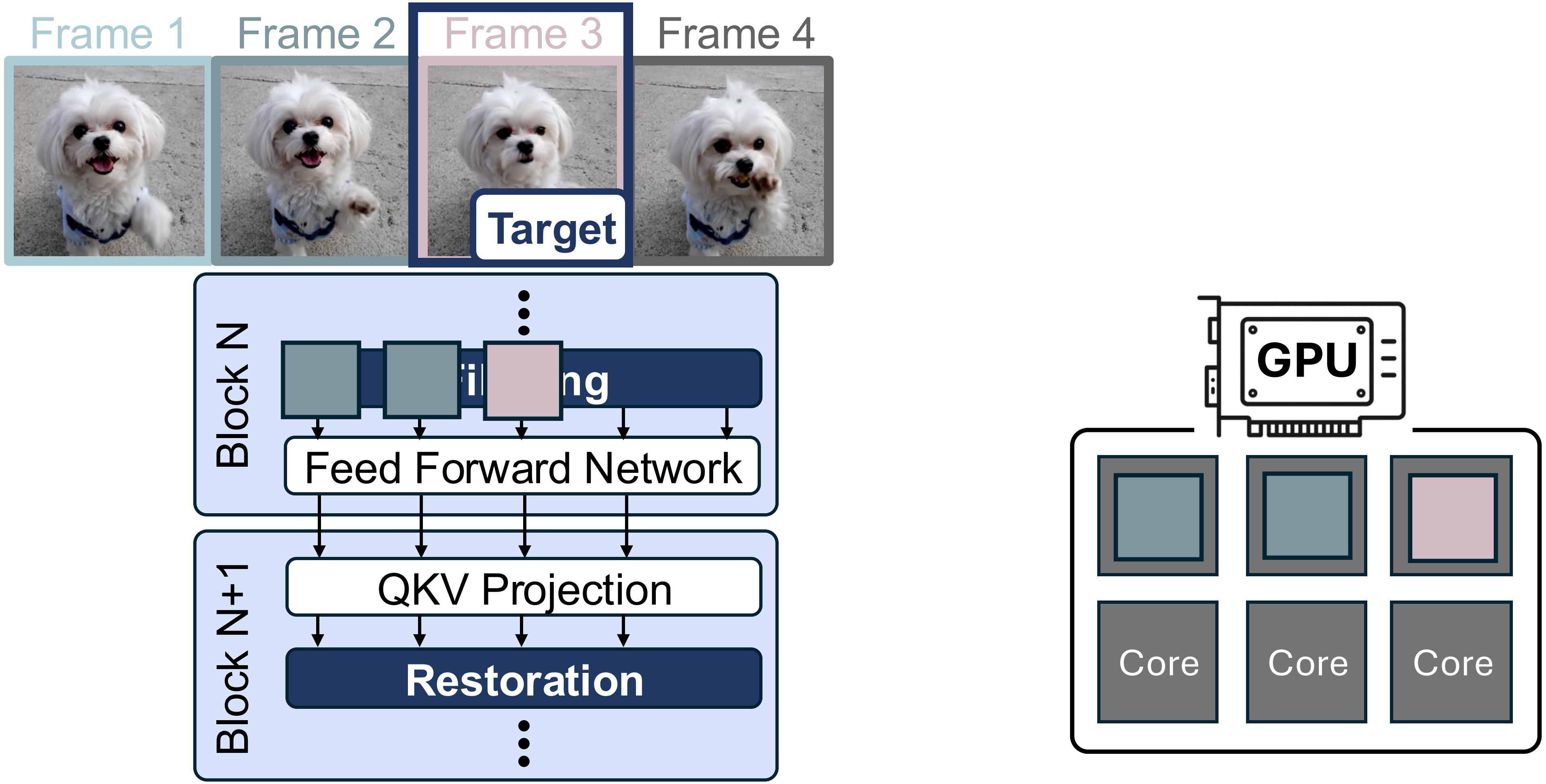
- Staggering frames across layers to improve computational efficiency

Sparse Computation Compaction



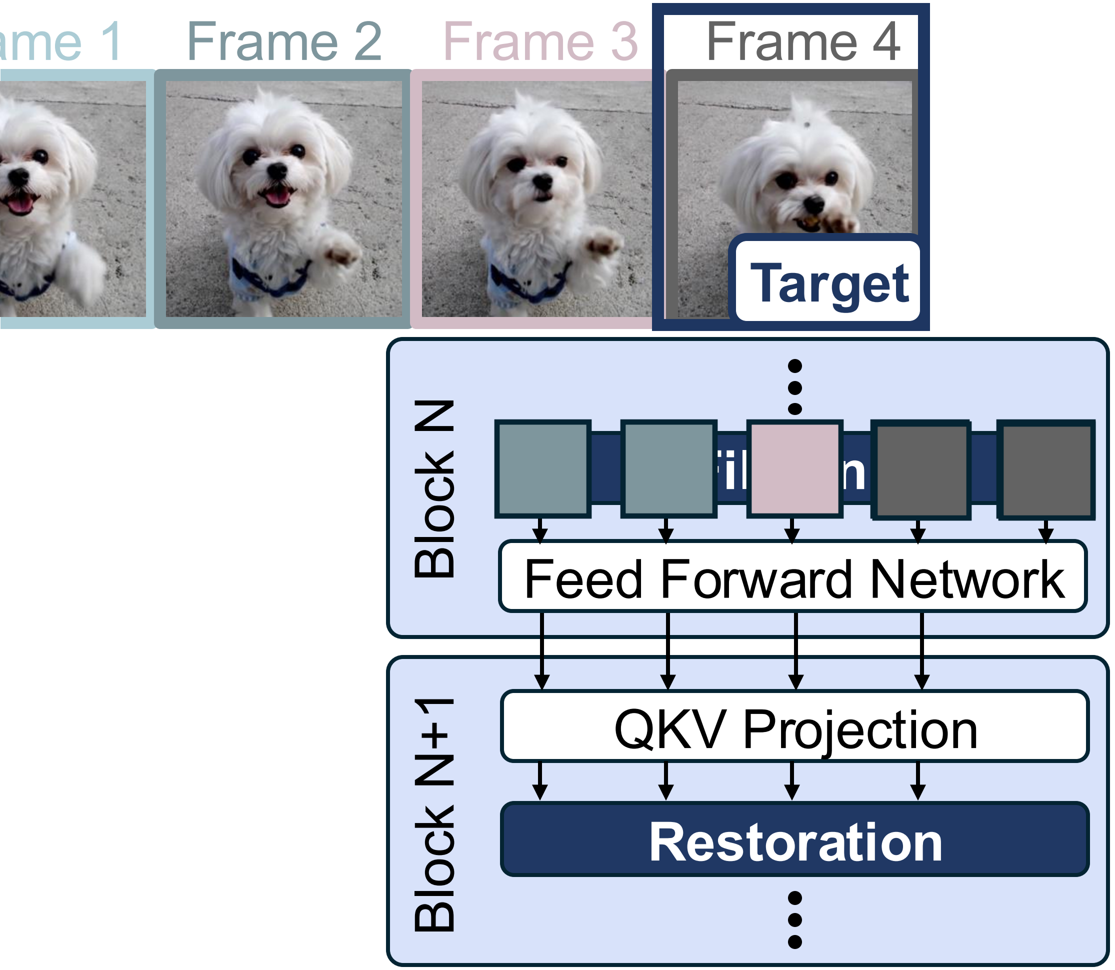
- Staggering frames across layers to improve computational efficiency

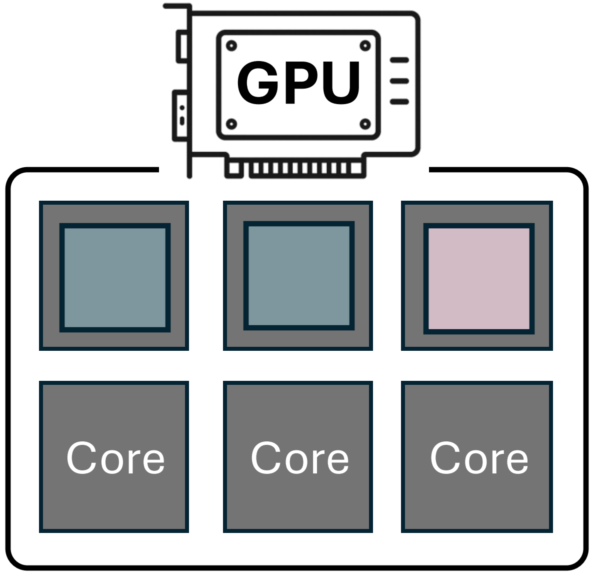
Sparse Computation Compaction



- Staggering frames across layers to improve computational efficiency

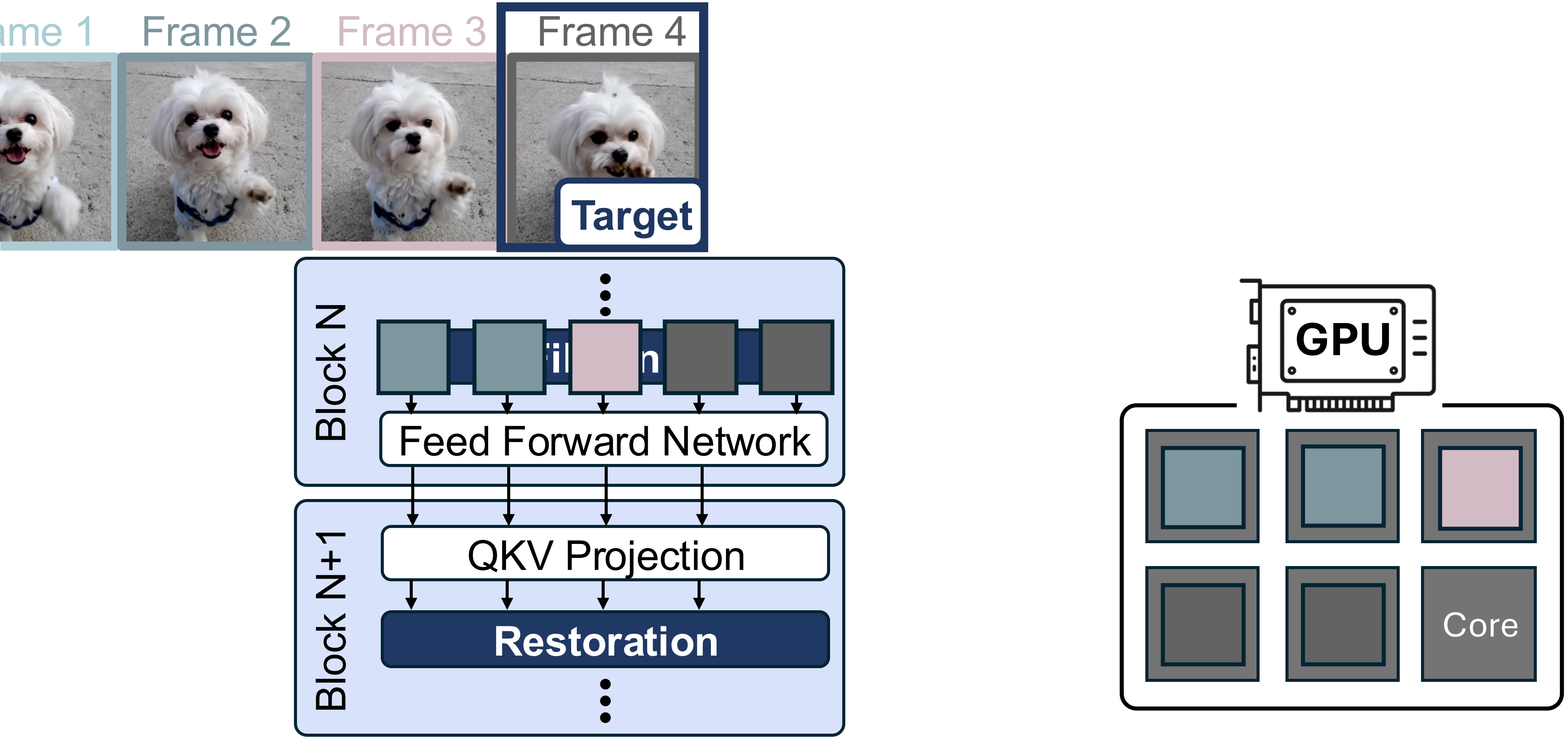
Sparse Computation Compaction





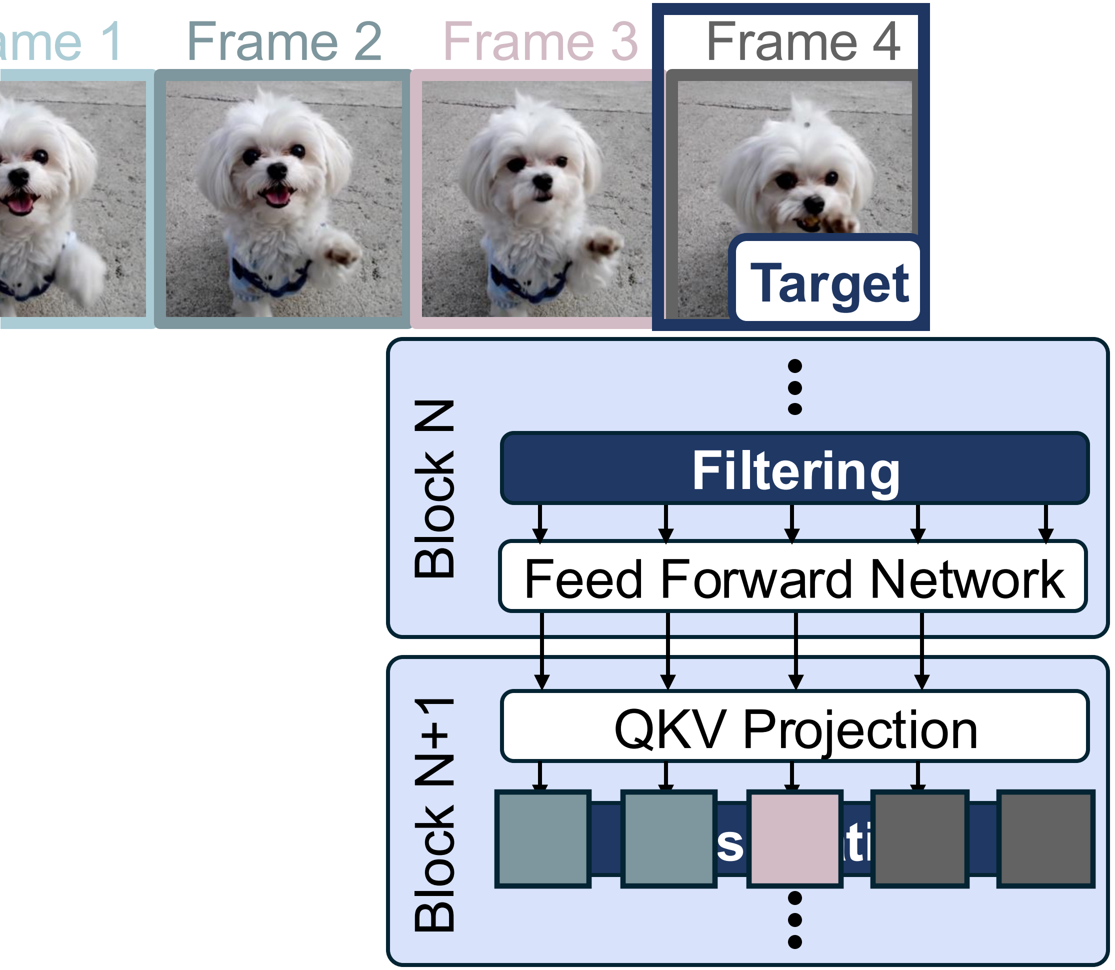
- Staggering frames across layers to improve computational efficiency

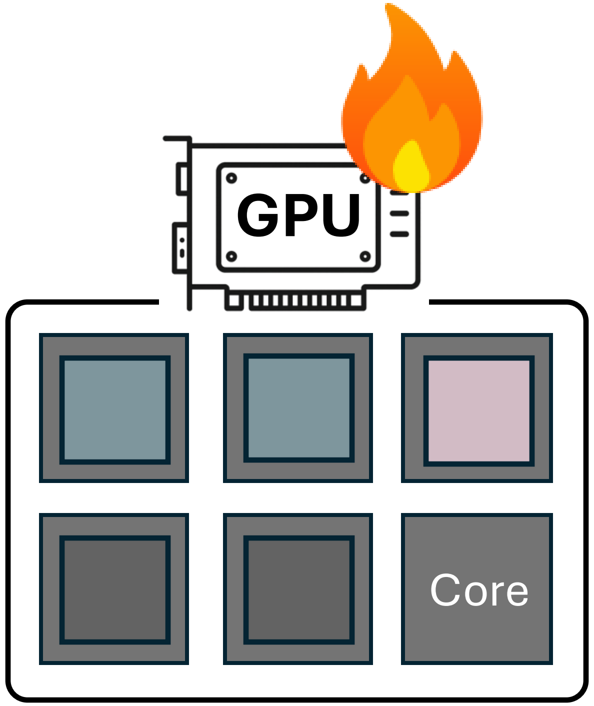
Sparse Computation Compaction



- Staggering frames across layers to improve computational efficiency

Sparse Computation Compaction





- Staggering frames across layers to improve computational efficiency

More Details in the Paper!

ReuseViT Architecture

- Frame Reordering
- Dataflow
- Decision Layer
- Restoration Layer

Learning Objectives

- Gumbel Softmax Reparameterization
- Dual Loss Term
- Handling Error Accumulation

Inference Optimization

- Layer-wise Scheduling
- Cached Memory Compaction
- Sparse Computation Compaction

Covered in today's talk

Evaluation Methodology

End Models

- Retrieval: CLIP4Clip
- Question answering: FrozenBiLM
- Question grounding: TempCLIP

Baselines

- Original ViT
- DiffRate[1]
- CMC[2]
- Eventful[3]

Datasets

- Retrieval: MSR-VTT
- Question answering: How2QA
- Question grounding: NExT-GQA

Environments

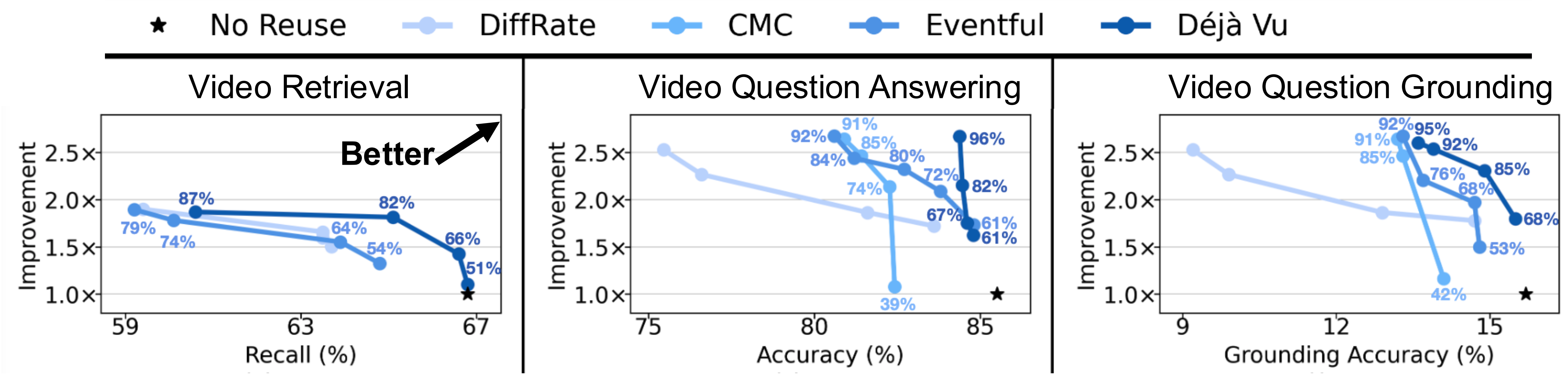
- Two Intel Xeon Gold 6226R
- 192GB DRAM
- Nvidia RTX 3090 GPU
- Ubuntu 24.04 / CUDA 12.1 / PyTorch 2.1

[1] Chen et al., "DiffRate: Differentiable Compression Rate for Efficient Vision Transformers," ICCV 2023.

[2] Song et al., "CMC: Video Transformer Acceleration via CODEC Assisted Matrix Condensing," ASPLOS 2024.

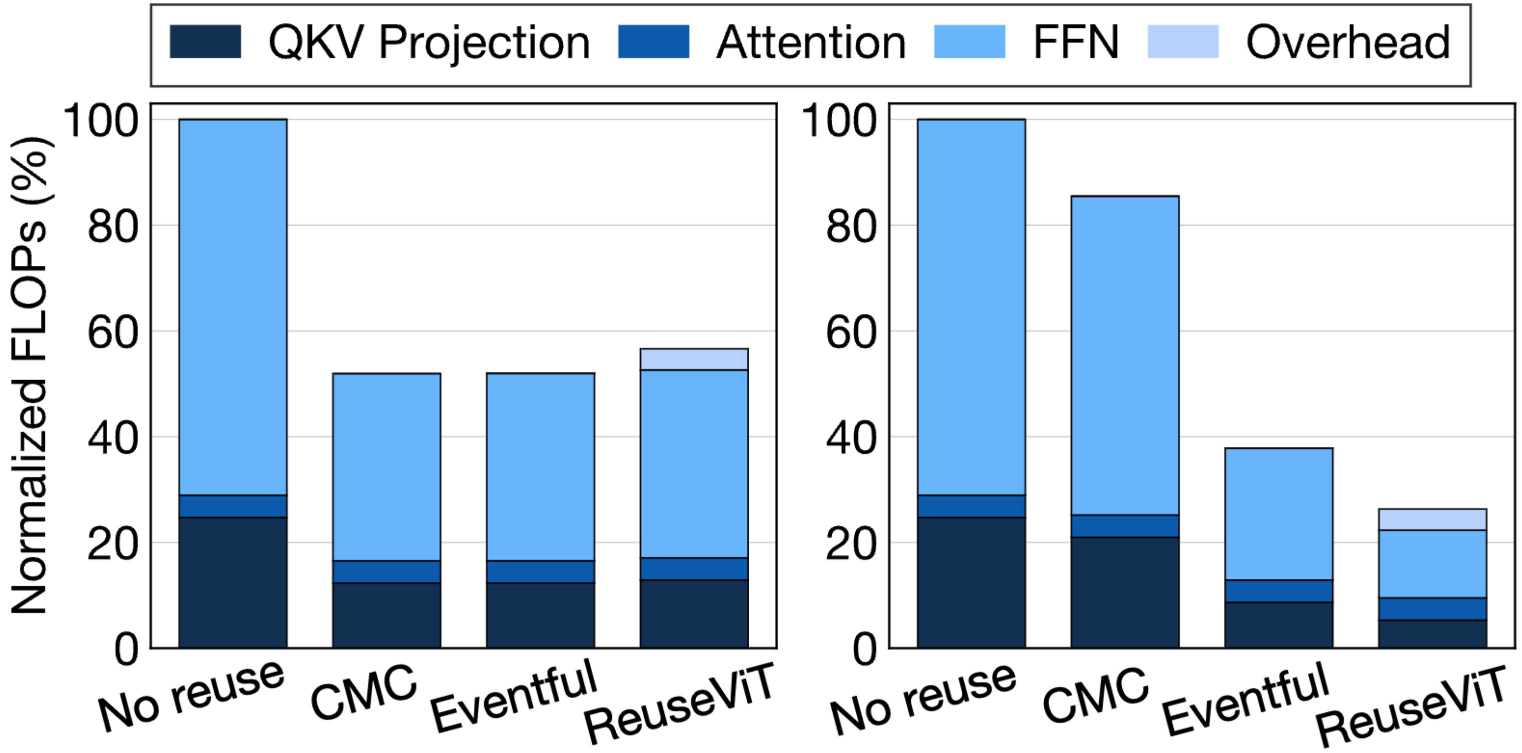
[3] Dutson et al., "Eventful transformers: leveraging temporal redundancy in vision transformers," ICCV 2023.

Trade-off Between Accuracy & Throughput



- **Best accuracy-throughput tradeoff** across all three tasks
- **Up to 2.64× faster** within ~2% task error

Deeper FLOPs Breakdown



Fixed reuse rate to 50% **Fixed accuracy** to 84.5%

- ReuseViT experience small overhead (~4%) at same reuse rate.
- Overhead is compensated by achieving higher reuse rate.

Additional Results

- FLOPs-accuracy tradeoff
- Memory overhead analysis
- Ablation study for design and training
- Ablation study for inference optimization

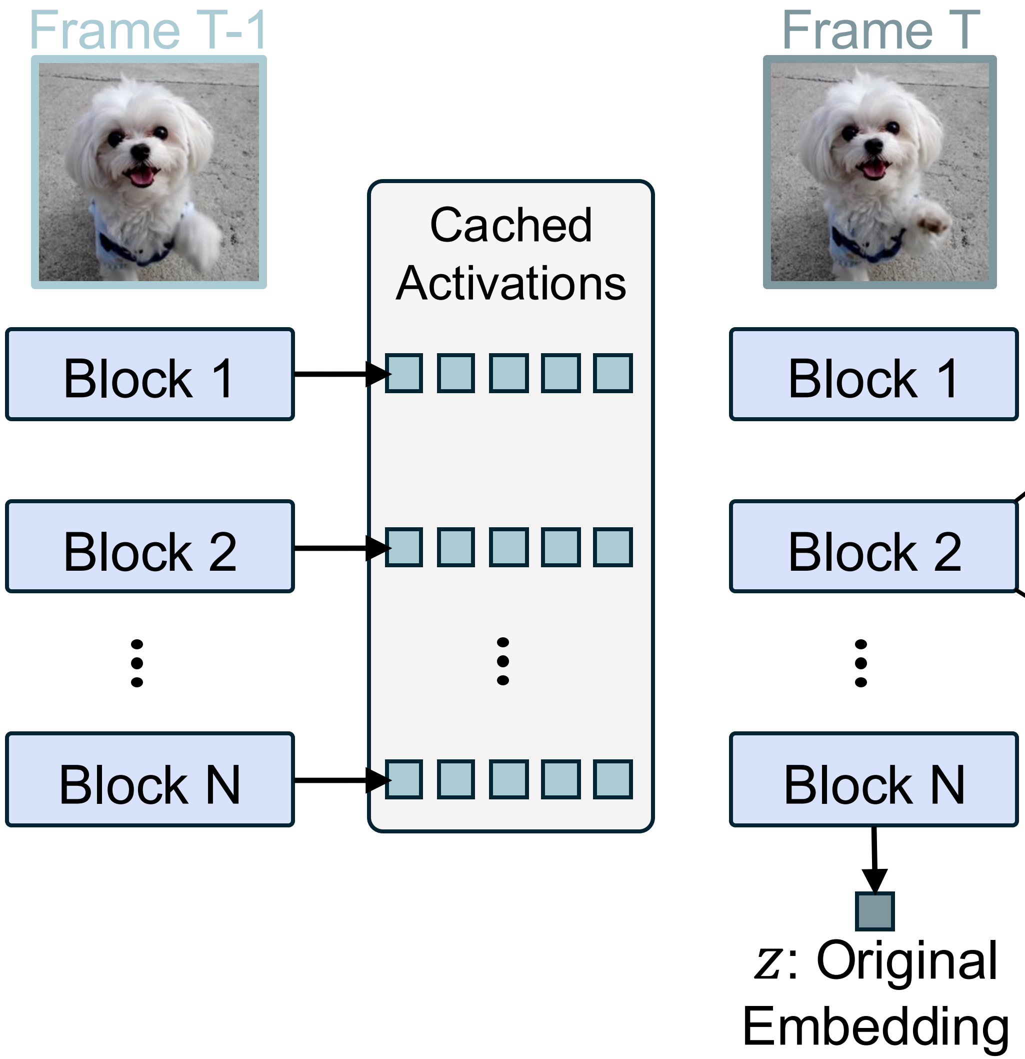
Conclusion

- Déjà Vu
  - Algorithm-system co-designed solution to reuse computation with learning-based approach
- Contributions
  - Learns when to reuse FFN/QKV per token across frames
  - Trained to balance reuse rate and task accuracy
  - Efficient runtime via layer-wise scheduling and compaction
- Results
  - Outperforms every other prior baselines
  - Up to 2.64× faster with ~2% accuracy drop

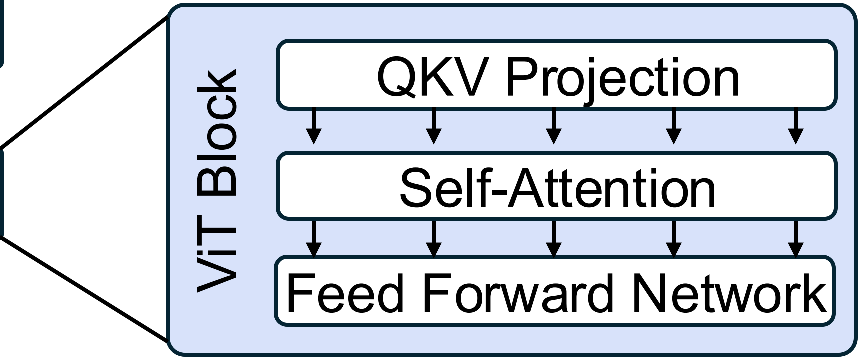
Artifacts Available
acm
V1.1

Backup Slides

Training Inputs Setup

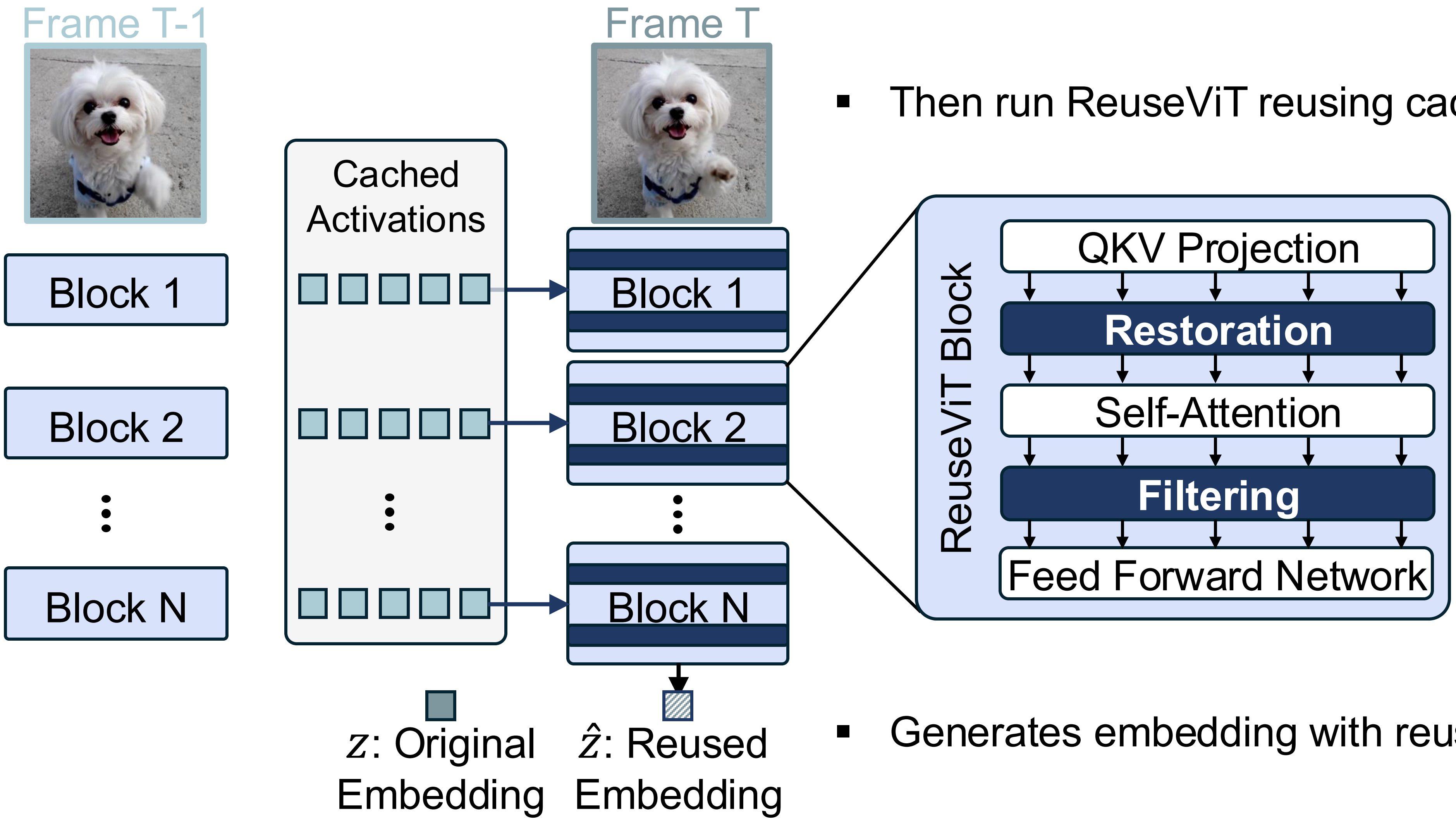


- We begin with a pair of consecutive frames.
- We first run the original ViT, not ReuseViT



- Previous frame generates cached activations
- Current frame generates original embedding.

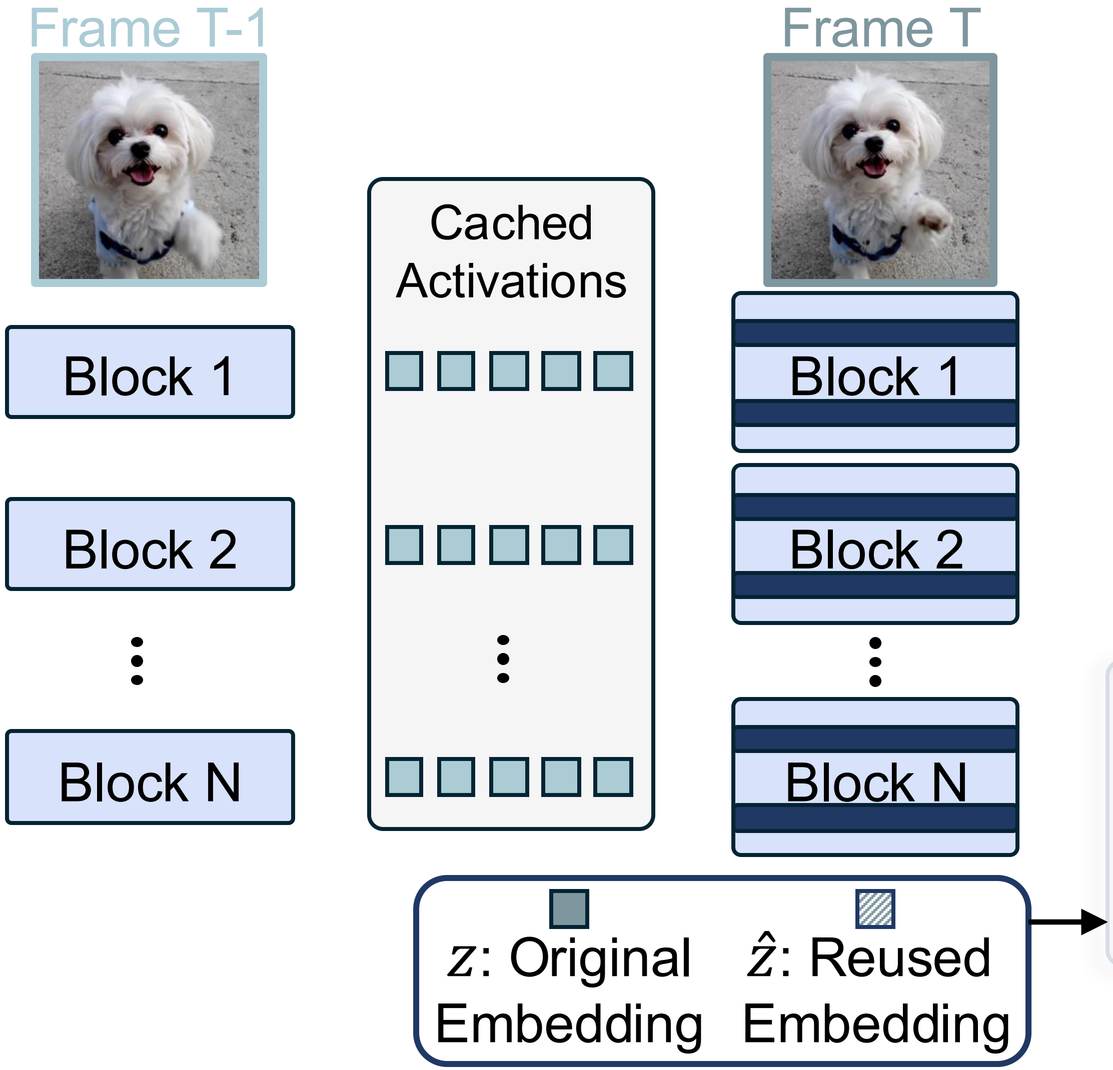
ReuseViT Pass



- Then run ReuseViT reusing cached activations.

- Generates embedding with reuse involved.

Accuracy Loss

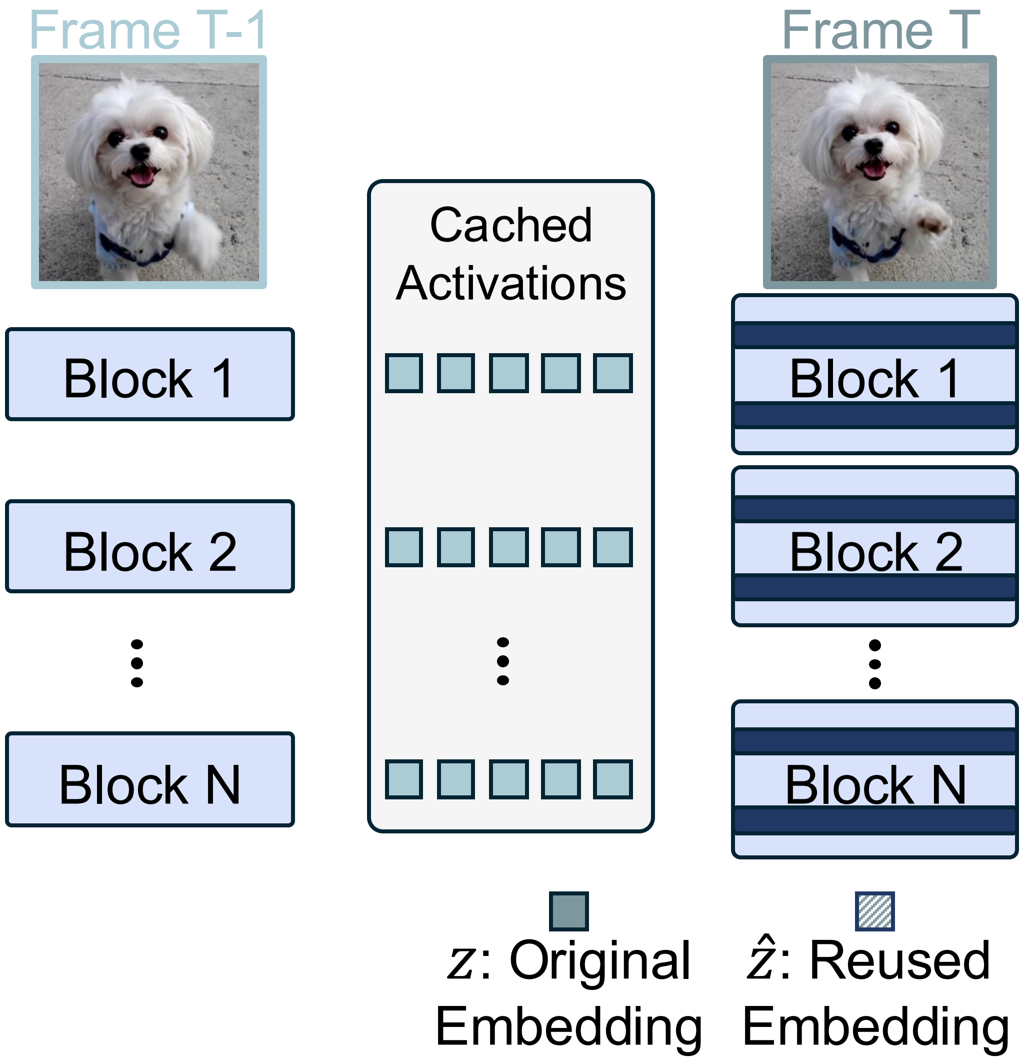


$$\mathcal{L} = 1 - \cos(z, \hat{z})$$

Accuracy Loss

Compare embeddings with cosine similarity.

Efficiency Loss



Efficiency Loss

Encourage model to meet target reuse rate.

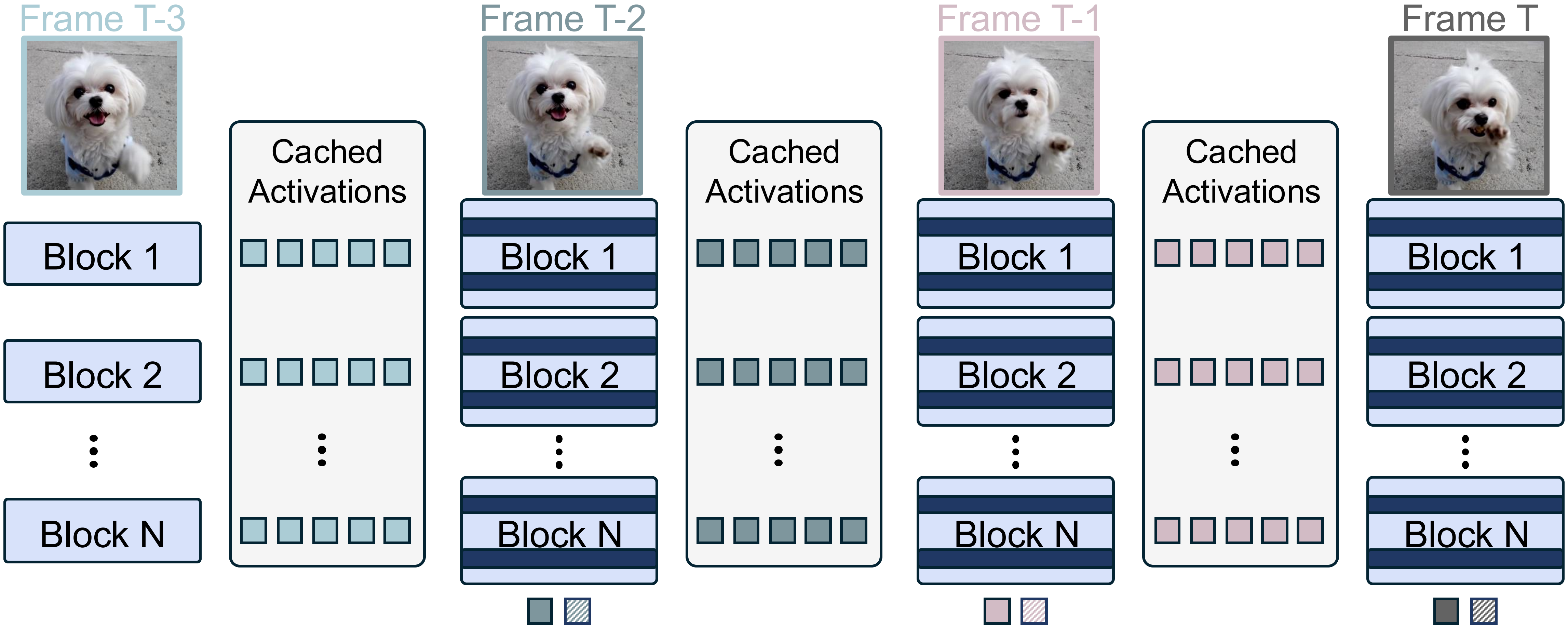
$$\mathcal{L} = \underline{1 - \cos(z, \hat{z})} + \overline{\alpha \cdot \max(0, R_{target} - \text{avg}(M))}$$

Accuracy Loss

-

Compare embeddings with cosine similarity.

Grouped Frame Training



- Training on multiple frames improves efficiency and error modeling.

ReuseViT Architecture | Dual Learning Objective | Layer-wise Scheduling

ReuseViT Architecture | Dual Learning Objective | Layer-wise Scheduling

ReuseViT Architecture | Dual Learning Objective | Layer-wise Scheduling

Déjà Vu Overview

How do we incorporate a self-decision mechanism into the ViT?

01 **ReuseViT Architecture**

What should the model optimize during training?

02 **Dual Learning Objective**

How do we ensure it runs efficiently on commodity GPUs?

03 **Layer-wise Scheduling**

Déjà Vu Overview

How do we incorporate a self-decision mechanism into the ViT?

01 **ReuseViT Architecture**

What should the model optimize during training?

02 **Dual Learning Objective**

How do we ensure it runs efficiently on commodity GPUs?

03 **Layer-wise Scheduling**

Déjà Vu Overview

How do we incorporate a self-decision mechanism into the ViT?

01 ReuseViT Architecture

What should the model optimize during training?

02 Dual Learning Objective

How do we ensure it runs efficiently on commodity GPUs?

03 **Layer-wise Scheduling**