GLU Variants Improve Transformer

arXiv:2002.05

202

Feliciann Elliot

MAI 5301 - Foundations Of Large Language Models

Background

In standard Transformers, attention mixes information, but feed-forward networks (FFN) perform the majority of nonlinear transformation. Since FFNs dominate parameter count, their internal structure strongly affects model capacity.

Improving the FFN design directly improves model capacity without modifying attention.

Role of Activation Functions in Neural Networks

Activation functions introduce non-linearity into neural networks

Non-linearity enables models to learn complex patterns beyond linear mappings

These are essential for tasks such as:

- Language understanding
- Image recognition
- Sequence modeling

Without activation functions, deep networks collapse into linear models

Standard Transformer Feed-Forward Network

Canonical Transformer feed-forward network $\mathrm{FFN}(x) = \phi(xW_1)W_2$

where the activation function ϕ is ReLU or GELU

- This represents a single nonlinear transformation
- No interaction occurs between hidden dimensions
- There is a uniform treatment of all features

Why ReLU and GELU Are Structurally Limited

ReLU characteristics

- Hard thresholding with binary feature suppression
- Sparse gradients during backpropagation

GELU characteristics

- Smooth, probabilistic activation
- Improved optimization behavior

Both are additive nonlinearities, feature-independent and incapable of modeling multiplicative interactions

Why ReLU and GELU Are Structurally Limited

While ReLU uses hard thresholding and GELU employs smooth probabilistic activation, both share a fundamental limitation.

They are additive, feature-independent nonlinearities incapable of modeling multiplicative interactions.

Why ReLU and GELU Are Structurally Limited

Feed-Forward Networks have a fundamental limitation where they lack a mechanism for conditional feature selection. As a result, their output depends solely on activation strength rather than feature relevance.

Gated Linear Units (GLU): Core Architecture

GLU introduces two parallel linear projections $\mathrm{GLU}(x) = \sigma(xW) \odot (xV)$

where one projection acts as a learned gate while the other carries content information.

The element-wise multiplication ($\odot$) between gate and content enables feature-wise modulation, creating conditional information flow based on input context.

Gated Linear Units (GLU): Core Architecture

The model learns when and where to activate features, rather than relying solely on activation strength.

GLU Variants

(Shazeer, 2020) introduced several GLU variants to isolate the source of performance improvements

This systematic variation enables controlled experimentation to determine whether performance gains originate from the gating structure itself, the choice of nonlinearity, or their interaction.

GLU Variants

The variants (GLU, ReGLU, GEGLU, SwiGLU, Bilinear) systematically vary the gate nonlinearity, from sigmoid to ReLU to GELU to Swish to none, isolating whether performance improvements stem from the gating structure, the specific activation function, or both.

Parameter and Compute Matching

Since GLU FFNs require three weight matrices versus two for standard FFNs, the hidden dimension is reduced by ~2/3 to maintain constant parameters and FLOPs. This ensures performance differences reflect architectural improvements, not simply larger models or increased computation.

Pre Training Configuration

All models use the T5 encoder-decoder architecture with span corruption on C4. Training conditions (optimizer Adafactor, learning rate schedule, and step count) are held constant across variants. Dropout is intentionally excluded during pre training. Performance is measured via log-perplexity on held-out C4 data, isolating architectural effects.

Pre Training Results

GEGLU and SwiGLU achieve the lowest perplexity scores, demonstrating consistent improvement over both ReLU FFN and GELU FFN baselines. These gains persist across training regimes, from short runs (65k steps) to full training (524k steps).

Gated feed-forward networks model token distributions more effectively than their non-gated counterparts.

Downstream Task Evaluation

Models were fine-tuned on GLUE, SuperGLUE, and SQuAD benchmarks. GLU variants improve average scores while simultaneously reducing performance variance across tasks.

- Stronger gains appear on reasoning and sentence-pair understanding tasks than on simpler classification problems.
- Gated architecture benefits generalize beyond the pre training objective, indicating improved representational capacity.

GEGLU and SwiGLU vs GELU: Architectural Distinction

GELU:

- Uses a single activation path
- No gating or feature modulation is involved

GEGLU / SwiGLU:

- Uses two linear projections
- Multiplicative gate × content interaction

SwiGLU combines smooth activation with explicit conditional feature selection, using Swish gating for smoother, gradient-controlled feature suppression.

Limitations

The paper (Shazeer, 2020) shows that gated feed-forward networks work better but does not explain why. Experiments use only medium-sized models and encoder-decoder architectures.

The study does not test how gating interacts with attention layers.

Impact on Modern LLM Architectures

- GEGLU adopted in PaLM (Chowdhery et al., 2022)
- SwiGLU adopted in LLaMA (Touvron et al., 2023)
- GLU-family FFNs now standard in large decoder-only models

Shazeer (2020) introduced the FFN design now used by most modern LLMs.

Conclusion

Improvements in FFN expressivity enhance model quality, but as models scale, the bottleneck shifts to inference costs.

Attention memory becomes the dominant runtime expense, motivating architectural innovations that reduce memory overhead while preserving performance.

Conclusion

These shifts motivate architectural innovations that trade excess attention capacity for reduced memory bandwidth while preserving downstream performance.

References

Shazeer, N. (2020). GLU variants improve transformer. arXiv:2002.05202

Chowdhery, A., et al. (2022). PaLM: Scaling language modeling with pathways. arXiv:2204.02311

Touvron, H., et al. (2023). LLaMA: Open and efficient foundation language models. arXiv:2302.13971

Questions?

GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

arXiv:2305.13245

Feliciann Elliot

MAI 5301 - Foundations Of Large Language Models

What Is Grouped-Query Attention (GQA)?

GQA is a Transformer attention mechanism that groups query heads

Each group shares a single key and value head

Number of key–value heads G satisfies: $1 < G < H$ where H is the total number of query heads.

What Is Grouped-Query Attention (GQA)?

GQA unifies Multi-Head Attention (MHA) and Multi-Query Attention (MQA) as special cases within a single framework, enabling flexible trade-offs between model quality and inference efficiency.

What Is the Key–Value (KV) Cache?

During autoregressive inference, Transformers store keys (K) and values (V) from all previous tokens. This stored state is called the KV cache.

At each new token generation step, queries attend to all cached keys and values to maintain context across the sequence.

What Is the Key–Value (KV) Cache?

KV cache size grows with both sequence length and the number of attention heads, creating substantial memory overhead for long contexts or models with many heads.

Autoregressive Inference: What Happens at Each Token

Tokens are generated sequentially, one step at a time

At each decoding step:

- The current token produces a query (Q)
- The model loads all stored keys (K) and values (V) from previous tokens
- Attention is computed between the new query and the entire KV cache

This process repeats for every generated token

Why the KV Cache Dominates Inference Cost

KV cache size scales with both sequence length (L) and number of attention heads (H).

The KV cache must be read from memory at every token generation step. Memory bandwidth, limits throughput during autoregressive decoding.

As model size increases, head count grows proportionally, expanding KV cache size and memory traffic per decoding step. This makes the bandwidth bottleneck increasingly severe for larger models.

Why GQA Was Created

Large language models suffer from slow autoregressive inference. Latency is dominated by loading the key-value (KV) cache from memory, making memory bandwidth the primary bottleneck rather than computation.

Prior architectures force a binary choice between quality (MHA) and speed (MQA), with no intermediate options for balancing these competing objectives.

The Core Problem Addressed by GQA

The KV cache size scales with: $O(L \times H \times d)$ where

L = sequence length, H = attention heads.

The MHA limitation here is that the KV storage scales linearly with head count, creating memory bottlenecks at scale, while the MQA limitation is that, reducing KV storage to a single head collapses attention diversity, degrading model quality and causing training instability.

Related Works Landscape

Method	Core Idea	Limitation
Multi-Head Attention (MHA)	Uses independent key–value heads per query head to maximize attention diversity	KV cache scales linearly with head count and sequence length, leading to high inference latency
Multi-Query Attention (MQA)	Shares a single key–value head across all query heads to reduce memory usage	Collapses attention diversity, causing quality degradation and training instability
FlashAttention	Reorders attention computation to reduce memory access during attention score calculation	Optimizes computation but does not reduce KV cache size or memory footprint
Quantization / Distillation	Compresses weights or trains smaller proxy models to reduce model size	Model-agnostic compression that does not address attention-specific redundancy
Sparse / Speculative Decoding	Reduces computation via token or layer sparsity or auxiliary draft models	Adds system complexity and does not directly reduce KV cache storage

Grouped-Query Attention (GQA)'s Solution

Grouped-Query Attention (GQA) reduces KV cache size through partial key–value sharing, preserving attention diversity while directly addressing inference-time memory bandwidth constraints.

What Makes GQA Stand Out?

- It does not eliminate KV head diversity entirely
- GQA introduces a continuous design parameter G
- It allows explicit tuning of Attention capacity and Inference efficiency

GQA can also be applied post hoc to pretrained models through uptraining, allowing existing checkpoints to be adapted for improved inference without full retraining.

Multi Head Attention vs Multi Quality Attention vs Grouped Query Attention

MHA - H query heads, H key heads, H value heads (full diversity)

MQA - H query heads, 1 key head, 1 value head (no KV diversity)

GQA - H query heads, G key heads, G value heads (partial KV diversity, where $1 < G < H$)

GQA maintains partial head diversity by grouping multiple query heads to share each key-value pair, balancing memory efficiency with representational capacity.

Query Grouping

Query heads are partitioned into G disjoint groups. Each group attends to a shared key-value pair, while attention diversity is maintained within each group through independent query projections.

This prevents complete attention subspace collapse as multiple KV heads preserve distinct representational subspaces, enabling different groups to capture varied input aspects while reducing memory overhead compared to MHA.

Reduced KV Cache Size

KV cache size is reduced by a factor of $\frac{H}{G}$ meaning less data must be loaded per decoding step and memory bandwidth pressure decreases proportionally.

Inference becomes faster without retraining the model. The memory footprint shrinks while query-side computation remains unchanged, directly addressing the bandwidth bottleneck that dominates autoregressive generation costs.

Mean-Pooled Checkpoint Conversion

Existing MHA checkpoints are converted to GQA by grouping original KV heads and mean pooling their projection matrices.

This minimizes representational loss during architectural change. Instead of randomly selecting or reinitializing KV heads, mean pooling aggregates learned representations, retaining the knowledge captured across all original attention heads.

Uptraining Strategy

The converted model is further pre trained for a small fraction α of the original training duration (typically $\alpha = 0.05$) using the same dataset and objective as the original training run.

This achieves adaptation at a fraction of original compute cost.

Evaluation Setup

Base architecture: T5.1.1 encoder–decoder Transformer

Model scales evaluated: T5-Large and T5-XXL

Tasks selected to stress autoregressive decoding:

- Summarization, translation, and question answering were chosen to stress autoregressive decoding and highlight inference bottlenecks.

Metrics capture both task quality (ROUGE, BLEU, F1) and inference efficiency (time per sample).

Inference Efficiency vs Task Performance

Reducing KV cache size lowers memory reads per decoding step

GQA decreases KV storage by a factor of H/G

This directly reduces inference latency while retaining attention diversity

The result is a higher performance at lower inference cost than MHA-Large

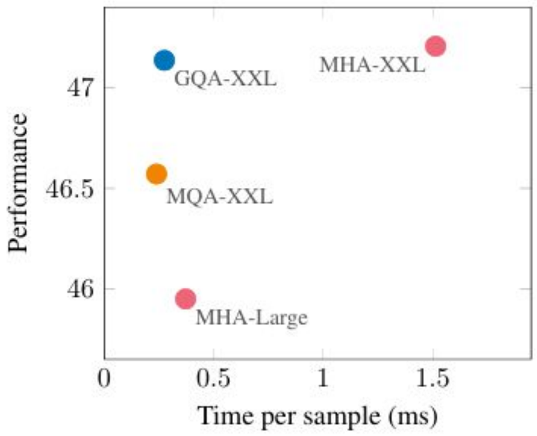

Figure 3: **Uptrained MQA yields a favorable tradeoff compared to MHA with higher quality and faster speed than MHA-Large, and GQA achieves even better performance with similar speed gains and comparable quality to MHA-XXL.** Average performance on all tasks as a function of average inference time per sample for T5-Large and T5-XXL with multi-head attention, and 5% uptrained T5-XXL with MQA and GQA-8 attention.

Inference Time and Task Performance

MQA achieves fastest inference but lower average quality.

GQA-8 recovers most MHA performance at near-MQA speed.

Model	T_{infer}	Average	CNN	arXiv	PubMed	MediaSum	MultiNews	WMT	TriviaQA
	s		R_1	R_1	R_1	R_1	R_1	BLEU	F1
MHA-Large	0.37	46.0	42.9	44.6	46.2	35.5	46.6	27.7	78.2
MHA-XXL	1.51	47.2	43.8	45.6	47.5	36.4	46.9	28.4	81.9
MQA-XXL	0.24	46.6	43.0	45.0	46.9	36.1	46.5	28.5	81.3
GQA-8-XXL	0.28	47.1	43.5	45.4	47.7	36.3	47.2	28.4	81.6

Table 1: Inference time and average dev set performance comparison of T5 Large and XXL models with multi-head attention, and 5% uptrained T5-XXL models with multi-query and grouped-query attention on summarization datasets CNN/Daily Mail, arXiv, PubMed, MediaSum, and MultiNews, translation dataset WMT, and question-answering dataset TriviaQA.

Effect of Attention Architecture on Inference Efficiency

Architectural change: Partial KV sharing through grouped query attention (GQA)

Measured effect:

- Smaller KV cache size and reduced memory bandwidth usage during inference

Observed outcome:

- Near-MHA task performance
- Near-MQA inference speed

Task Performance Results

- GQA-8 nearly matches MHA-XXL performance
- Consistently outperforms MQA
- Performance degradation relative to MHA is minimal

Limitations

The study does not compare uptraining results with GQA models trained from scratch, leaving open questions about optimal training strategies.

Evaluation relies heavily on ROUGE-based metrics, which may not capture all aspects of generation quality. Testing is limited to encoder-decoder architectures, with no evaluation on decoder-only models.

Summary of GQA's Contribution

- Introduces tunable attention efficiency
- Enables low-cost migration from MHA to efficient attention
- Addresses the dominant inference bottleneck in modern LLMs
- Complements, rather than replaces, other efficiency methods

References

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., & Sanghai, S. (2023).

Training generalized multi-query transformer models from multi-head checkpoints. arXiv preprint arXiv:2305.13245.

Shazeer, N. (2019). Fast transformer decoding: One write-head is all you need. arXiv preprint arXiv:1911.02150.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., & Ré, C. (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. arXiv preprint arXiv:2205.14135.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Levskaya, A., & Dean, J. (2022).

Efficiently scaling transformer inference. arXiv preprint arXiv:2211.05102.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020).

Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research.

Questions?