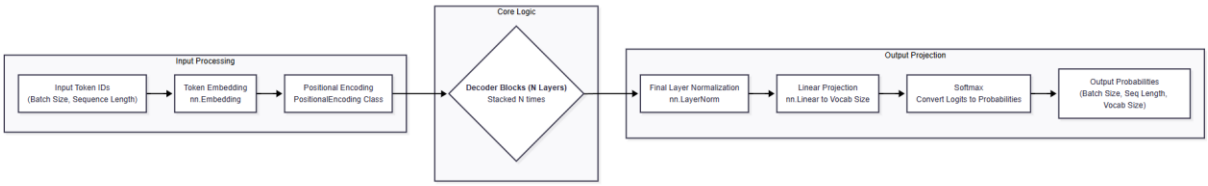
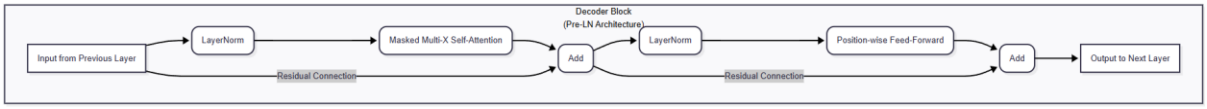


Gambar 1. Arsitektur Umum *Transformer* yang Diimplementasikan



Gambar 2. Detail dari Blok *Decoder* yang Diimplementasikan

A. Dataset

Dataset yang digunakan adalah MetaCDN Cache Dataset. *Dataset* ini merupakan kumpulan data yang berasal dari lingkungan *Content Delivery Network* (CDN) Meta, yang umumnya digunakan dalam penelitian *caching* untuk menganalisis pola akses dan strategi penyimpanan objek. *Dataset* ini berisi informasi penting seperti *timestamp* yang mencatat waktu akses suatu objek, *obj_id* sebagai identifikasi unik setiap objek dalam sistem *cache*, *obj_size* yang menunjukkan ukuran objek yang disimpan, serta *next_access_vtime* yang memperkirakan waktu akses berikutnya untuk objek tersebut. *Dataset* ini tersedia dalam format terkompresi OracleGeneral.zst, memungkinkan efisiensi penyimpanan dan pemrosesan data.

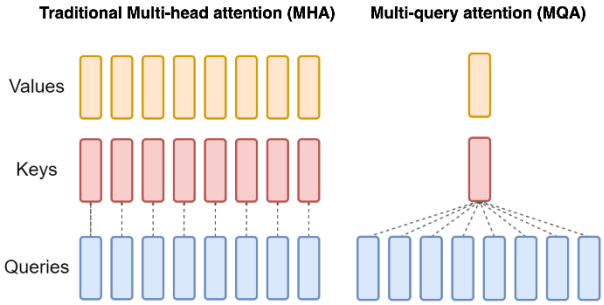
Berkas yang digunakan adalah meta_reag.oracleGeneral.zst yang berukuran 334 MB dan hanya diambil informasi mengenai *timestamp* dan *obj_id*. Selain itu, dilakukan *sampling* dengan hanya mengambil 15.000 akses I/O pertama untuk mengurangi beban pelatihan yang dilakukan. Dari 15.000 akses I/O pertama, terdapat 7.588 objek unik yang diakses sehingga masih akan terdapat potensi *caching* dilakukan.

Dataset ini dipilih karena ketertarikan terhadap topik pemanfaatan pembelajaran mesin dalam konteks strategi *caching* dan *prefetching*. Melihat *decoder-only transformer* yang memiliki potensi untuk tugas *sequence modelling*, terbayang potensi pemanfaatan model *transformer* untuk kepentingan *prefetching* dalam konteks *caching*. Pada hakikatnya, *prefetching* adalah mengambil objek yang kemungkinan akan diakses dalam waktu dekat berdasarkan objek yang sedang diakses sekarang, permasalahan ini mirip dengan menebak kata yang akan muncul selanjutnya berdasarkan kata yang sedang diproses saat ini.

B. Arsitektur Model

Model yang diimplementasikan pada tugas ini adalah *decoder-only transformer*. Disebut *"Decoder-Only"* karena *transformer* yang diimplementasikan hanya menggunakan blok *Decoder* dari arsitektur *transformer* asli yang memiliki *encoder* dan *decoder*. Arsitektur ini umumnya digunakan untuk tugas-tugas generatif seperti pemodelan bahasa, di mana tujuannya adalah memprediksi token berikutnya dalam sebuah urutan. Salah satu contoh paling terkenal dari model ini adalah GPT (*Generative Pre-trained Transformer*). Mengacu pada Gambar 1, secara umum alur pemrosesan pada model adalah *embedding*, *positional encoding*, *core logic* (*attention-head*), dan *output projection*.

Pada *transformer* yang diimplementasikan, tidak terdapat tokenisasi sama sekali. Tokenisasi merupakan proses mengubah *raw text* menjadi representasi angka. Karena *vocabulary* pada tugas ini merupakan ID objek yang sudah merupakan *integer*, maka tidak dilakukan tokenisasi sama sekali dan hanya dilakukan *embedding* dan *positional encoding* pada pemrosesan masukan. *Embedding* yang dilakukan mengubah ID objek yang dimasukkan menjadi representasi dalam ruang vektor tertentu. Proses *positional encoding* dalam pemrosesan masukan digunakan untuk menambahkan konteks posisi token dalam sekuens masukan.



Gambar 3. Perbedaan *Multi-Head Self-Attention dan Multi-Query Self-Attention* (Sumber: https://towardsdatascience.com/demystifying-gqa-grouped-query-attention-3fb97b678e4a/)

Arsitektur *decoder blocks* yang diimplementasikan dalam tugas ini adalah *Pre-Layer Normalization Decoder Block*. Arsitektur ini disebut *"Pre-Layer Normalization*" karena dilakukan normalisasi masukan sebelum masukan tersebut diteruskan ke lapisan utama yaitu *self-attention* dan *feed-forward network* seperti yang diilustrasikan oleh Gambar 2. Dipilih arsitektur ini karena keunggulannya yang lebih stabil saat pelatihan dan mengurangi masalah *exploding* maupun *vanishing gradient* (Acharya, 2025).

Normalisasi pertama pada blok *decoder* digunakan untuk memastikan lapisan selanjutnya, yaitu *self-attention* selalu menerima masukan dengan distribusi yang konsisten (hal inilah yang mencegah gradien menjadi tidak stabil). Selanjutnya, masukan yang telah dinormalisasi diproses pada mekanisme atensi, baik *Multi-Head Self-Attention* (MHSA) maupun *Multi-Query Self-Attention* (MQSA) yang akan menghitung skor atensi atau skor relevansi antara token yang sedang diproses dengan token-token lainnya, tepatnya token-token sebelumnya (hal ini terjadi karena *causal masking*). Pada lapisan *self-attention* dihasilkan sebuah vektor *output* yang merupakan representasi setiap token yang telah diperkaya dengan informasi kontekstual dari token-token sebelumnya yang relevan menggunakan *weighted average* berbasis probabilitas *softmax*. Setelah itu, hasil akan dinormalisasi lagi dan diteruskan ke lapisan *Feed-Forward Network* (FFN). FFN diperlukan oleh model untuk menangkap hubungan yang lebih kompleks dan non-linear yang telah diperkaya oleh atensi dari lapisan sebelumnya. Terakhir, terdapat *residual connection* yang digunakan untuk mencegah masalah *vanishing gradients* yang dapat terjadi dari hasil pelatihan FFN (Dabbada, 2024).

MHSA adalah arsitektur atensi klasik pada *vanilla transformer*. Dalam pendekatan ini, masukan diproyeksikan menjadi beberapa set *Query (*Q*), Key (*K*),* dan *Value* (V) yang sepenuhnya independen, satu set untuk setiap *head*. Karena setiap *head* memiliki set Q, K, dan V yang independen, setiap *head* akan memiliki bobot projeksi Q, K, dan V yang dapat dilatih. Arsitektur ini memungkinkan setiap kepala untuk mempelajari jenis hubungan semantik yang berbeda, namun dengan konsekuensi jumlah parameter yang besar dan kebutuhan memori yang lebih signifikan untuk *KV Cache*, karena *Key* dan *Value* dari setiap *head* harus disimpan.

MQSA merupakan optimasi dari MHSA yang menggunakan satu set *Key* dan *Value* yang sama untuk setiap *head* dan tetap menggunakan set independen untuk *Query*. Matriks projeksi K dan V tunggal ini akan digunakan oleh semua *query head* saat melakukan perhitungan atensi. Arsitektur ini secara drastis mengurangi jumlah total parameter. Manfaat dari arsitektur ini adalah berkurangnya ukuran *KV Cache* yang berujung pada penghematan memori dan peningkatan kecepatan pemrosesan.

Setelah melewati bagian blok *decoder* yang merupakan *core logic* dari *transformer*, hasilnya diteruskan ke bagian *output projection*. Pada bagian ini, wawasan akan digabungkan menjadi satu terlebih dahulu karena terdapat vektor sejumlah *attention-head* yang terpisah. Langkah pertama adalah normalisasi masukan sebelum diteruskan ke lapisan linear yang tujuannya sama dengan normalisasi pada blok *decoder*. Lapisan linear pada *output projection* digunakan untuk mengubahnya menjadi vektor yang merupakan

Tabel 1. Konfigurasi *Hyperparameter*

Konfigurasi	Parameter	Nilai Parameter
Baseline	Dimensi Model	128
	Jumlah *Attention-Head*	4
	Jumlah Lapisan	3
	Dimensi FFN	256
	Dropout	0.1
	Learning Rate	0.001
LargerModel LowerLR	Dimensi Model	256
	Jumlah *Attention-Head*	8
	Jumlah Lapisan	4
	Dimensi FFN	512
	Dropout	0.15
	Learning Rate	0.0005
SmallerModel HigherLR MoreDropout	Dimensi Model	64
	Jumlah *Attention-Head*	2
	Jumlah Lapisan	2
	Dimensi FFN	128
	Dropout	0.2
	Learning Rate	0.002
MediumModel VariedHeads	Dimensi Model	128
	Jumlah *Attention-Head*	8
	Jumlah Lapisan	3
	Dimensi FFN	256
	Dropout	0.1
	Learning Rate	0.001

projeksi keluaran dalam ruang *vocabulary* untuk klasifikasi. Terakhir, dilakukan *softmax* untuk menghitung probabilitas keluaran dari setiap token pada *vocabulary* (dalam implementasi hal ini ditangani oleh *CrossEntropyLoss*).

C. Pengaturan Pelatihan dan *Hyperparameter*

Pelatihan dilakukan selama lima *epoch* dan menggunakan *CrossEntropyLoss*. *CrossEntropyLoss* umumnya digunakan untuk persoalan klasifikasi multi-kelas (Pykes, 2024), yang salah satunya adalah prediksi token selanjutnya. Jumlah *epoch* lima dipilih karena keinginan untuk membuat proses pelatihan menjadi cepat (pada realita, pelatihan *transformer* dapat memakan waktu dalam skala jam hingga bulanan). Dengan jumlah *epoch* yang rendah, sumber daya komputasi yang dibutuhkan akan jauh lebih sedikit.

Konfigurasi *hyperparameter* baik untuk varian MHSA maupun MQSA dapat dilihat pada Tabel 1 (kedua *transformer* menggunakan konfigurasi *hyperparameter* yang sama). Dimensi model merujuk kepada dimensi vektor yang akan digunakan untuk merepresentasikan masukan pada tahap *embedding*, jumlah lapisan merujuk kepada jumlah lapisan blok *decoder,* dimensi FFN merujuk kepada dimensi vektor yang digunakan pada FFN dalam blok *decoder*, dan parameter lainnya dianggap sudah memiliki nama yang representatif sehingga tidak dijelaskan.

D. Hasil Evaluasi

Evaluasi performa model menggunakan simulasi *LRU cache* yang berukuran 0,1% dari total *vocabulary* dengan *top-1 prefetcher*, yang berarti model *prefetcher* hanya akan

Tabel 2. Hasil Evaluasi Model

Model	Jumlah Parameter	*Training Loss*	*Validation Loss*	*Miss Ratio Terbaik*
LRU tanpa *prefetching*				0,7043
MHSA-Baseline	2.347.812	0,3173	15,5823	0,6835
MHSA-LargerModel-LowerLR	6.001.572	0,3064	14,2879	0,6831
MHSA-SmallerModel-HigherLR-MoreDropout	1.045.924	0,6417	17,9205	0,6788
MHSA-MediumModel-VariedHeads	2.347.812	0,3104	15,0231	0,6886
MQSA-Baseline	2.273.508	0,3312	15,1268	0,6894
MQSA-LargerModel-LowerLR	5.541.028	0,3200	14,9909	0,6844
MQSA-SmallerModel-HigherLR-MoreDropout	1.037.604	0,6395	18,8699	0,6849
MQSA-MediumModel-VariedHeads	2.261.124	0,3375	14,9194	0,6869

melakukan *prefetching* terhadap satu objek dari objek yang sedang di-*request*. Hasil evaluasi dapat dilihat pada Tabel 2.

E. Temuan Penting

Hasil dari eksperimen ini menunjukkan potensi pemanfaatan *transformer* untuk *prefetching*. Hasil evaluasi menunjukkan peningkatan yang konsisten untuk setiap model *prefetcher* (peningkatan sebesar 2% hingga 3% dari *LRU* tanpa *prefetching*) meskipun hanya melakukan *top-1 prefetcher*. Pada realitanya, *prefetcher* umumnya tidak hanya melakukan *top-1 prefetcher* karena objek-objek tidak selalu berasosiasi kuat. Meskipun tidak berasosiasi kuat, pengaksesan suatu objek tertentu masih dapat mengindikasikan bahwa objek lainnya akan diakses dalam waktu dekat seperti yang ditunjukkan oleh Yang et al. (2017) melalui publikasi Mithril. Peningkatan kinerja *cache* dapat disebabkan oleh *prefetching* objek yang meski tidak langsung diakses, tetap diakses sebelum objek tersebut dikeluarkan dari *cache* karena ukuran *cache* yang memadai untuk jarak pengaksesan tersebut.

Selain temuan sentimen positif, terdapat permasalahan *overfitting* pada model. Hal ini diindikasikan dengan kesenjangan besar antara *Training Loss* yang sangat rendah dengan *Validation Loss* yang sangat tinggi. *Overfitting* ini merupakan hal yang diekspektasikan terjadi karena terdapat asumsi bahwa ketika objek A diakses maka objek yang diakses setelahnya merupakan objek yang pasti (yang mana tidak terjadi pada kondisi nyata).

Temuan lain adalah tidak adanya model yang benar-benar unggul. Dari Tabel 2, terlihat bahwa setiap model memiliki kinerja yang sangat mirip satu sama lain. Konfigurasi *"LargerModel-LowerLR"* mencapai *validation loss* yang lebih rendah dibandingkan konfigurasi lain, hal ini disebabkan karena jumlah *parameter* yang lebih tinggi memungkinnya menangkap pola yang lebih kompleks. Konfigurasi "*SmallerModel-HigherLR-MoreDropout*" memiliki *validation loss* paling tinggi, namun model ini juga memiliki kinerja *caching* yang paling baik yang mungkin disebabkan karena model pada awalnya masih cukup *general* sebelum *overfitting* dari data pelatihan. Konfigurasi *"Baseline"* dan *"MediumModel-VariedHeads*" menunjukkan bahwa peningkatan jumlah *attention-head* tidak selalu meningkatkan kinerja model.

Temuan terakhir adalah perbandingan MHSA dan MQSA dari segi parameter dan penggunaan memori. Terlihat tren yang jelas bahwa model-model MQSA memiliki jumlah parameter yang lebih sedikit sesuai dengan teorinya. Penghematan memori juga didukung melalui inspeksi penggunaan VRAM selama pelatihan menggunakan kakas NVIDIA-SMI yang menunjukkan pelatihan model-model MHSA menggunakan 600 MiB VRAM, sementara MQSA hanya menggunakan 478 MiB VRAM.

F. References

Acharya, R. (2025, March 8). *What changed in the Transformer architecture*. Dipetik May 26, 2025, dari HuggingFace: https://huggingface.co/blog/rishiraj/what-changed-in-the-transformer-architecture

Dabbada, S. S. (2024). *What is the purpose of using skip connections in neural networks and how does they work*. Dipetik May 26, 2025, dari Kaggle: https://www.kaggle.com/discussions/general/543200

Pykes, K. (2024, August 10). *Cross-Entropy Loss Function in Machine Learning: Enhancing Model Accuracy*. Dipetik May 26, 2025, dari DataCamp: https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning

Shazeer, N. (2019). Fast Transformer Decoding: One Write-Head is All You Need. *Computing Research Repository.*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention is All you Need. *Advances in Neural Information Processing Systems.* Curran Associates, Inc.

Yang, J., Karimi, R., Sæmundsson, T., Wildani, A., & Vigfusson, Y. (2017). Mithril: Mining Sporadic Associations for Cache Prefetching. *Symposium on Cloud Computing*, (hal. 66-79).