

# MedAlpaca Technical Report

Nama : Jonathan Ibrahim

NPM : 2206103421

## Overview

MedAlpaca [9] pada dasarnya merupakan pretrained Alpaca yang di-fine tuning menggunakan koleksi dataset Medical Meadow dengan tujuan meningkatkan performa model di bidang medical NLP. Ada tiga basis model Alpaca yang digunakan oleh MedAlpaca. Pertama, variasi vanilla-nya yaitu Stanford Alpaca [1]. Kedua, ekstensi dari Stanford Alpaca dengan implementasi Low Rank Adaptation (LoRA) [2] dengan tujuan meningkatkan efisiensi memori saat training. Ketiga, ekstensi dari Alpaca-LoRA menggunakan 8-bit matrix multiplication [3] dan 8-bit optimizers [4] untuk lebih lanjut meningkatkan efisiensi memori saat training. Stanford Alpaca itu sendiri merupakan pretrained LLaMA [10] yang di-fine tuning menggunakan 52K data Instruction-Following yang dihasilkan melalui Text-davinci-003 milik OpenAI. Sementara itu, untuk LLaMA itu sendiri merupakan model arsitektur berbasis transformer dengan beberapa modifikasi seperti pre-normalization [7], SwiGLU activation function [6], dan rotary embeddings [5].

## Fundamental

Jika dilihat mulai dari akarnya, MedAlpaca merupakan keturunan dari LLaMA. LLaMA menggunakan basis arsitektur transformer, tapi ada beberapa modifikasi yang dilakukan. Pertama, menormalisasi input dari setiap sub-layer transformer menggunakan RMSNorm.

$$\bar{a}_i = \frac{a_i}{\text{RMS}(\mathbf{a})} g_i, \quad \text{where } \text{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n} \sum_{i=1}^n a_i^2}.$$

Kedua, mengganti ReLU activation function menjadi SwiGLU activation function.

$$\text{SwiGLU}(x, W, V, b, c, \beta) = \text{Swish}_{\beta}(xW + b) \otimes (xV + c)$$

Ketiga, menghapus absolute positional embeddings dan menggantinya dengan rotary positional embeddings.

$$f_{\{q,k\}}(\mathbf{x}_m, m) = \mathbf{R}_{\Theta, m}^d \mathbf{W}_{\{q,k\}} \mathbf{x}_m$$

where

$$\mathbf{R}_{\Theta, m}^d = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

Stanford Alpaca kemudian lahir dengan melakukan fine-tuning pada pretrained LLaMA menggunakan 52K data Instruction-Following yang dihasilkan melalui Text-davinci-003 milik OpenAI. Stanford Alpaca adalah model basis yang digunakan oleh MedAlpaca.

Dikarenakan jumlah trainable parameter yang sangat besar, MedAlpaca mengekstensi model basisnya menggunakan Low Rank Adaptation (LoRA) untuk meningkatkan efisiensi memori selama training. Anggap suatu LLM dilambangkan sebagai  $P_\phi(y|x)$  di mana  $\phi$  merupakan parameter weight-nya. Dalam kasus fine-tuning, model awalnya menggunakan pretrained weight  $\phi = \phi_0$ , lalu setelah di fine-tuning, weight barunya akan menjadi  $\phi = \phi_0 + \Delta\Phi$ . Parameter  $\Delta\Phi$  dicari melalui optimisasi gradient descent yang memaksimalkan fungsi objective  $L(\phi) = \sum_{(x,y) \in \mathcal{Z}} \sum_{t=1}^{|y|} \log(P_\phi(y_t|x, y_{<t}))$  di mana  $\mathcal{Z}$  merupakan training set berisi pasangan konteks-target yang biasa digunakan di NLP. Sampai di sini permasalahan optimisasi sudah berbentuk  $\max_{\phi} L(\phi)$ . Pada LLM,  $|\phi|$  dapat sangat besar yang membuat tingginya kebutuhan memory dan waktu training, serta membuat ukuran file model yang disimpan menjadi sangat besar. Untuk mengatasi masalah itu, masalah optimisasi perlu diubah dengan menggunakan parameter yang lebih kecil dari  $|\phi|$ . Anggap kita punya  $\Theta$  di mana  $|\Theta| \ll |\Phi|$ , lalu menggunakan  $\Theta$  kita bisa mengencode  $\Delta\Phi = \Delta\Phi(\Theta)$  sehingga didapatkan fungsi objective baru berupa  $L'(\Theta) = \sum_{(x,y) \in \mathcal{Z}} \sum_{t=1}^{|y|} \log(P_{\phi_0 + \Delta\Phi(\Theta)}(y_t|x, y_{<t}))$ , dan masalah optimisasi dapat menjadi  $\max_{\Theta} L'(\Theta)$ . Untuk mengencode  $\Delta\Phi$ , barulah digunakan metode Low Rank Adaptation di mana  $\Delta\Phi \in \mathbb{R}^{d \times k}$  dilakukan low-rank decomposition menjadi  $\Delta\Phi = \Delta\Phi(A, B) = AB$ ;  $A \in \mathbb{R}^{d \times r}$ ;  $B \in \mathbb{R}^{r \times k}$ ;  $r \ll \min(d, k)$ , sehingga  $|A| + |B| \ll |\Phi|$ .

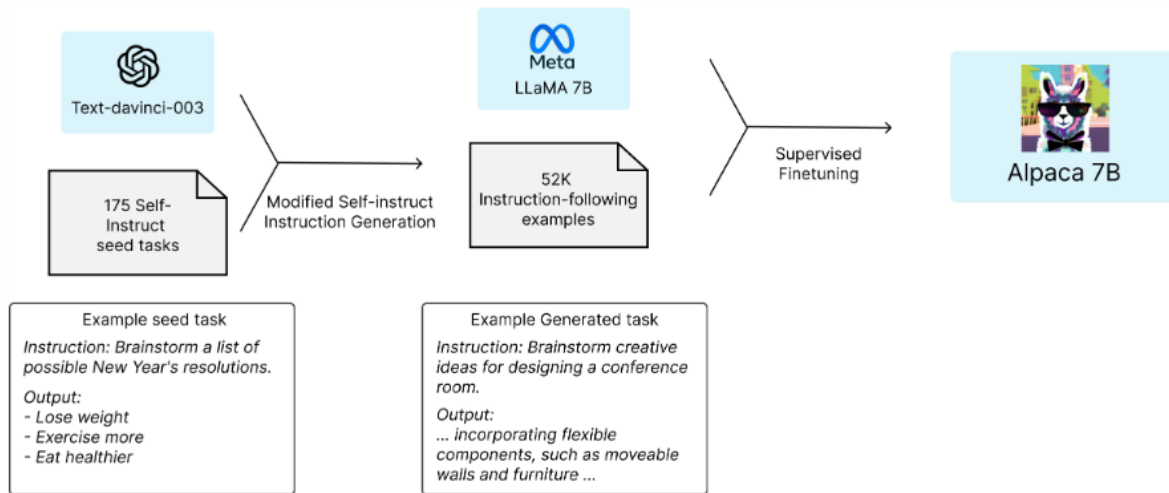
Untuk lebih meningkatkan efisiensi memori saat training, MedAlpaca mengekstensi model Alpaca-LoRA dengan metode 8-bit matrix multiplication. Secara umum, metode ini bekerja melalui kombinasi antara absmax vector-wise quantization dan mixed precision decomposition.

Diberikan matrix input  $X_{f16}$  dan weight  $W_{f16}$ , masing-masing memiliki tipe data floating point 16-bit.  $X_{f16}$  dan  $W_{f16}$  akan didekomposisi dengan melihat eksistensi outlier di dimensi fitur  $d$  dari  $X_{f16} \in \mathbb{R}^{n \times d}$ . Jika ditemukan setidaknya satu outlier dengan nilai melebihi threshold  $\alpha$  di dimensi  $d_i$  maka  $d_i$  akan dianggap sebagai fitur outlier. Nantinya  $X_{f16}$  akan terdekomposisi menjadi  $R \in \mathbb{R}^{n \times (d-o)}$  (submatrix berisi fitur non-outlier) dan  $O \in \mathbb{R}^{n \times o}$  (submatrix berisi fitur outlier). Sementara itu  $W_{f16}$  akan terdekomposisi menjadi  $W_R \in \mathbb{R}^{(d-o) \times k}$  dan  $W_O \in \mathbb{R}^{o \times k}$  mengikuti  $R$  dan  $O$ . Nantinya  $R$  dan  $W_R$  akan dimultiplikasi secara 8-bit dengan absmax vector-wise quantization, sementara  $O$  dan  $W_O$  akan dimultiplikasi normal secara 16-bit. Paper LLM-int8 [5] juga menyatakan bahwa  $|O| \leq 7$  untuk transformer dengan 13 miliar parameter, sehingga setidaknya operasi multiplikasi 16-bit tidak akan banyak mengonsumsi memori. Selanjutnya, untuk multiplikasi 8-bit dimulai dengan menentukan terlebih dahulu vector-wise constant  $C_{W_R}$  dan  $C_R$  dengan mencari row-wise dan column-wise absolute maximum dari  $R$  dan  $W_R$ . Setelah itu lakukan quantization  $R_{i8} = R * \left(\frac{127}{C_R}\right)$ ;  $W_{Ri8} = W_R * \left(\frac{127}{C_{W_R}}\right)$ , lalu multiplikasi matrix 8-bitnya  $V_{i32} = R_{i8}W_{Ri8}$ , kemudian di-dequantize untuk mengembalikan ke bentuk floating point 16-bitnya  $V_{f16} = (V_{i32} * (C_R \otimes C_{W_R})) / (127 * 127)$ . Hasil dari 8-bit multiplication setelah di-dequantize dan 16-bit multiplication akan diakumulasi menjadi output akhir. Selain 8-bit matrix multiplication, efisiensi memory saat training juga akan ditingkatkan dari sisi optimizer dengan 8-bit optimizer menggunakan block-wise quantization.

Pada akhirnya sebelum dilakukan fine-tuning, MedAlpaca telah menetapkan tiga basis model Alpaca yang akan digunakan, yaitu Alpaca (Vanilla), Alpaca-LoRA, dan Alpaca-LoRA 8-bit. Masing-masing model basis tersebut selanjutnya akan dilakukan fine-tuning dengan menggunakan data Medical Meadow sehingga lahirlah MedAlpaca, MedAlpaca-LoRA, dan MedAlpaca-LoRA 8-bit.

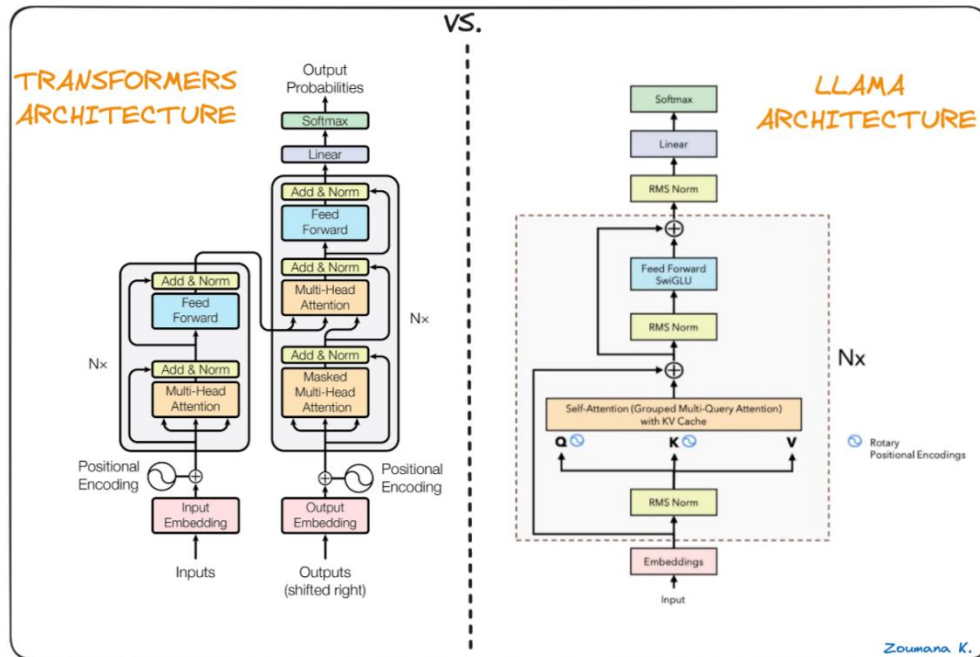
## Conceptual (Neural Architecture Model of Alpaca and MedAlpaca)

### *Alpaca (Stanford Alpaca)*



**Gambar 1** Skema Pembuatan Model Alpaca [1]

Alpaca sebenarnya merupakan LLaMA yang di-fine tuning menggunakan 52K data instruction-following yang digenerate dengan bantuan Text-davinci-003. Dimulai dengan mempersiapkan data berupa 175 pasangan instruction-output yang ditulis secara manual, lalu data tersebut diberikan ke Text-davinci-003 milik OpenAI sebagai context untuk mengenerate lebih banyak data berupa 52K data instruction-following. Selanjutnya, varian model pretrained LLaMA digunakan dan dilakukan fine-tuning menggunakan generated data tersebut. Hasil fine-tuning tersebut kemudian menghasilkan model baru bernama Alpaca. Untuk arsitektur LLaMA itu sendiri berbasis arsitektur transformer yang dilakukan beberapa modifikasi (seperti yang sudah dijelaskan di Fundamental). Bentuk arsitektur secara visual dapat dilihat pada gambar berikut.



**Gambar 2** Arsitektur Transformer vs LLaMA [8]

Terdapat beberapa varian dari LLaMA, seperti 7B, 13B, 33B, dan 65B, yang masing-masing melambangkan jumlah parameter pada modelnya. Karena Alpaca merupakan LLaMA yang di-fine tuning, maka otomatis varian Alpaca akan mirip dengan LLaMA.

### *MedAlpaca*

MedAlpaca merupakan Alpaca yang di fine-tuning menggunakan data koleksi dari Medical Meadow. Berikut adalah deskripsi dari data-data yang digunakan.

Dataset	Source	Description	n
<b>Finetuning</b>			
Medical Flash Cards	Anki Flashcards	Rephrased Q&A pairs derived from the front and back sides of medical flashcards	33,955
Stack Exchange	Academia	Q&A pairs generated from questions and their top-rated answers	39,633
	Biology		7,482
	Fitness		3,026
	Health		1,428
	Bioinformatics		906
Wikidoc	Living Textbook	Q&A pairs generated from paragraphs, where questions were formulated from rephrased paragraph titles, and answers were extracted from paragraph text	67,704
	Patient Information	Q&A pairs generated from paragraph headings and associated text content	5,942
<b>Evaluation</b>			
USMLE	Step 1	Multiple choice questions from the USMLE self-assessment with image-based questions excluded	119
	Step 2		120
	Step 3		135

**Gambar 3** Data untuk Fine Tuning dan Evaluasi MedAlpaca [9]

MedAlpaca menggunakan basis Alpaca-7B dan Alpaca-13B. Selain itu, MedAlpaca juga melakukan ekstensi untuk model basisnya berupa Alpaca-LoRA dan Alpaca-LoRA 8bit di mana modelnya dinilai lebih efisien secara memori saat training tapi mengorbankan akurasi (penjelasan mengenai LoRA dan 8bit bisa dilihat pada bagian Fundamental). Dengan demikian, terdapat 6 model MedAlpaca yang menjadi percobaan, yaitu MedAlpaca-7B, MedAlpaca-7B LoRA, MedAlpaca-7B LoRA 8bit, MedAlpaca-13B, MedAlpaca-13B LoRA, MedAlpaca-13B LoRA 8bit. Akan tetapi, hasil evaluasi menunjukkan bahwa LoRA dan LoRA 8bit menurunkan zero shot performance cukup banyak untuk varian MedAlpaca-13B.

Table 2: Zero shot performance on the USMLE self assessment

Model	Step1	Step2	Step3
LLaMA 7b [15]	0.198	0.202	0.203
Alpaca 7b naive [11]	0.275	0.266	0.293
Alpaca 7b LoRA	0.220	0.138	0.252
MedAlpaca 7b	0.297	0.312	0.398
MedAlpaca 7b LoRA	0.231	0.202	0.179
MedAlpaca 7b LoRA 8bit	0.231	0.241	0.211
ChatDoctor (7b) [10]	0.187	0.185	0.148
LLaMA 13b [15]	0.222	0.248	0.276
Alpaca 13b naive	0.319	0.312	0.301
MedAlpaca 13b	<b>0.473</b>	<b>0.477</b>	<b>0.602</b>
MedAlpaca 13b LoRA	0.250	0.255	0.255
MedAlpaca 13b LoRA 8bit	0.189	0.303	0.289

**Gambar 4** Hasil Evaluasi MedAlpaca [9]

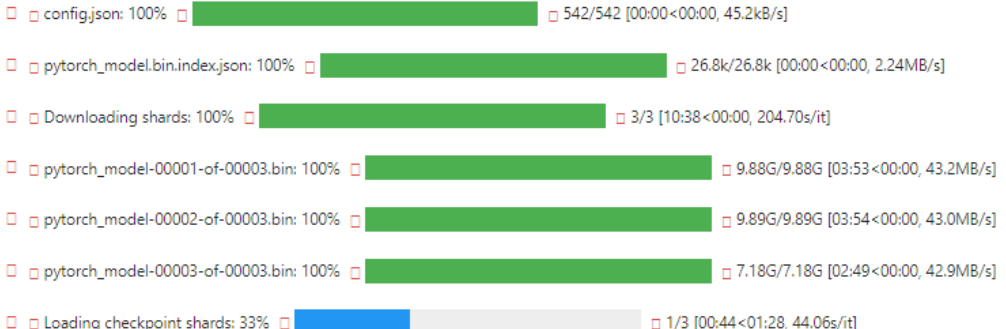
## Practical

Karena membutuhkan RAM ~ 30 GB untuk menjalankan model, maka hasil percobaan tidak dapat dikeluarkan baik di Google Colab / Kaggle Kernel.

```
❗ Your notebook tried to allocate more memory than is available. It has restarted.
```

```
answer = pl(f"Context: {context}\n\nQuestion: {question}\n\nAnswer: ")
print(answer)
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required
for this version of SciPy (detected version 1.24.3
warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}")
```



The image shows a series of progress bars for downloading and loading model components. The components and their progress are:

- config.json: 100% (green bar)
- pytorch\_model.bin.index.json: 100% (green bar)
- Downloading shards: 100% (green bar)
- pytorch\_model-00001-of-00003.bin: 100% (green bar)
- pytorch\_model-00002-of-00003.bin: 100% (green bar)
- pytorch\_model-00003-of-00003.bin: 100% (green bar)
- Loading checkpoint shards: 33% (blue bar)

Salah satu jalannya adalah dengan menggunakan Azure ML yang memiliki RAM ~ 32 GB.

Berikut adalah hasil percobaan MedAlpaca-7B setelah menggunakan Azure ML.

Query 1:

```
'Context: Diabetes is a metabolic disease that causes high blood sugar. The
symptoms include increased thirst, frequent urination, and unexplained weight
loss.\n\nQuestion: What are the symptoms of diabetes?\n\nAnswer: '
```

Answer 1:

```
'The symptoms of diabetes include increased thirst, frequent urination,
unexplained weight loss, increased appetite, blurred vision, and fatigue.'
```

Query 2:

```
'How to cure COVID-19?'
```

Answer 2:

```
'The first step is to stop the spread of the virus. This is the responsibility
of everyone.\n
The second step is to find a cure.\n
The third step is to find a vaccine.\n
The fourth step is to find a way to prevent the virus from coming back.\n
The fifth step is to find a way to prevent other viruses from becoming
```

[illegible]



## Reference

- [1] Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., ... & Hashimoto, T. B. (2023). Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models*. <https://crfm.stanford.edu/2023/03/13/alpaca.html>, 3(6), 7.
- [2] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2021). Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- [3] Dettmers, T., Lewis, M., Belkada, Y., & Zettlemoyer, L. (2022). Llm.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*.
- [4] Dettmers, T., Lewis, M., Shleifer, S., & Zettlemoyer, L. (2021). 8-bit optimizers via block-wise quantization. *arXiv preprint arXiv:2110.02861*.
- [5] Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., & Liu, Y. (2024). Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568, 127063.
- [6] Shazeer, N. (2020). Glue variants improve transformer. *arXiv preprint arXiv:2002.05202*.
- [7] Zhang, B., & Sennrich, R. (2019). Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32.
- [8] Keita, Z. (2023, November 14). Llama.cpp Tutorial: A Complete Guide to Efficient LLM Inference and Implementation. <https://www.datacamp.com/tutorial/llama-cpp-tutorial>
- [9] Han, T., Adams, L. C., Papaioannou, J. M., Grundmann, P., Oberhauser, T., Löser, A., ... & Bressen, K. K. (2023). MedAlpaca--An Open-Source Collection of Medical Conversational AI Models and Training Data. *arXiv preprint arXiv:2304.08247*.
- [10] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... & Lample, G. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.