Proyek GenAI: *Chatbot* Berbasis *Open-Source* LLM *(Fine-Tuned)* Bahasa Indonesia

Last Uppdate: 19 Januari 2024

Pathway	-
Sub-Pathway	-
Competency	-
Key Behaviour	-
Level	*Intermediate*
Developer	Haidlir Naqvi (Telkom)
Created	19 Januari 2024
Version	1.1.1
Approval	1. - 2. -
Durasi Waktu Ajar	- menit

Modul Pembelajaran	Proyek GenAI: *Chatbot* Berbasis *Open-Source* LLM *(Fine-Tuned)* Bahasa Indonesia
Deskripsi Modul	Menjelaskan *Generative* AI (GenAI) dan *Large Language Model* (LLM), contoh beberapa *use case* GenAI khususnya di bidang Telekomunikasi, siklus proyek GenAI, serta mendemonstrasikan pelaksanaan proyek pengembangan GenAI untuk *chatbot*.
Enabling Learning Objective (ELO)	Mampu menjelaskan dan mendemonstrasikan proyek pengembangan GenAI untuk *chatbot*.
Daftar Topik	1. Pengantar *Generative* AI (GenAI) dan *Large Language Model* (LLM) 2. Siklus Proyek *Generative* AI 3. Pelaksanakan Proyek *Generative* AI (*Hands-on*)
Keywords	*1. Generative AI* *2. Large Language Model* 3. Siklus Proyek *Generative* AI *4. Chatbot* 5. Bahasa Indonesia

Topik 1

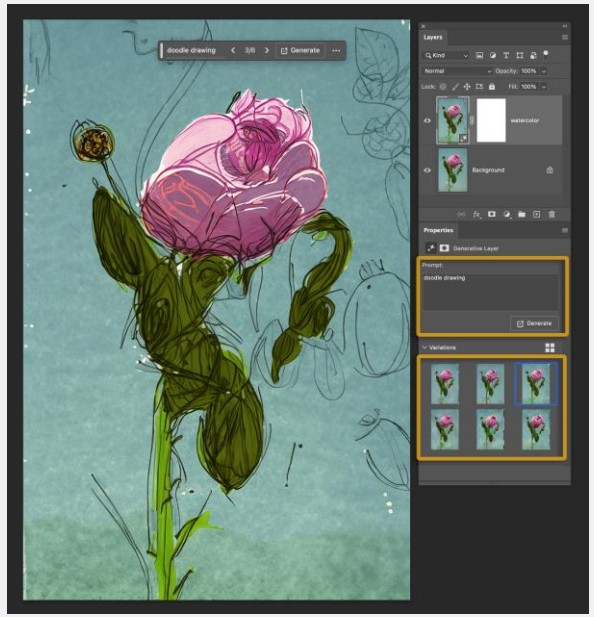
Pengantar *Generative AI* (GenAI) dan *Large Language Model* (LLM)

Generative Artificial Intelligence – Generative AI - GenAI

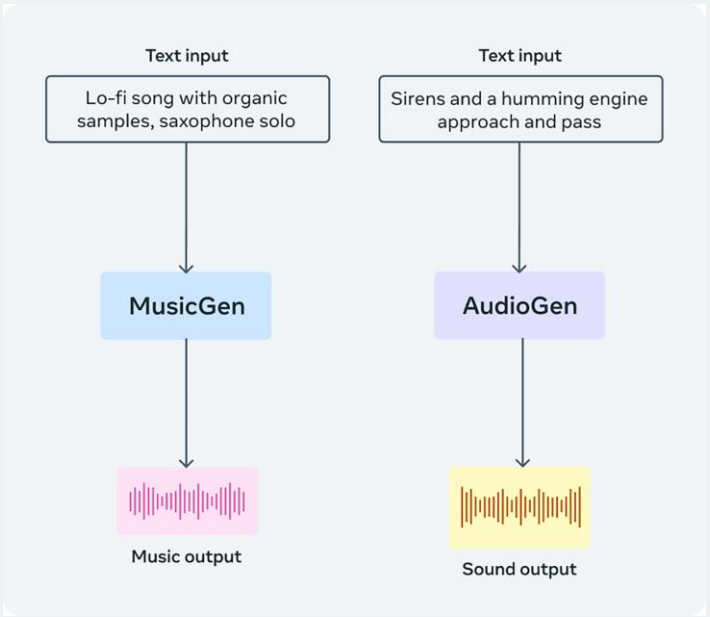
Newman (2023) pada laporan TMForum mendefinisikan Generative AI (GenAI) sebagai ke model *machine learning* yang memiliki kemampuan untuk menghasilkan output atau keluaran secara dinamis setelah model tersebut dilatih *(trained)*. Algoritma pada GenAI membuat konten baru berdasarkan pola yang dipelajari dari data yang telah ada atau *existing data*.

Salah satu bentuk interaksi GenAI yang paling sering digunakan adalah pengguna mengetikan teks yang disebut sebagai *prompt* dan selanjutnya model menghasilkan keluaran baik berbentuk teks juga, gambar, suara, dan bentuk media lainnya.

Implementasi GenAI yang luas membuat banyak perusahaan berlomba mengeluarkan produk GenAI seperti ChatGPT oleh OpenAI, Search Generative Experience oleh Google, dan Copilot oleh Github dan Microsoft



Photoshop Generative AI feature to edit photo according to the submitted prompt.

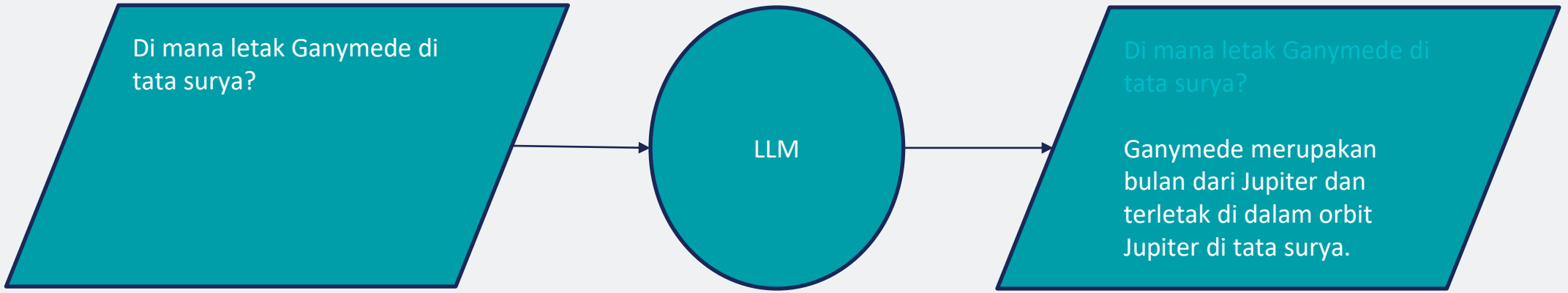


AudioCraft: A Generative AI Tool For Audio and Music

Large Language Model – LLM

Newman (2023) pada laporan TMForum mendefinisikan Large language models (LLMs) sebagai salah satu jenis GenAI yang dapat memahami dan menghasilkan bahasa seperti manusia.

LLM mengambil masukan *input* sebuah teks lalu menghasilkan keluaran *output* teks. Secara umum dari LLM tidak menghasilkan teks keluaran sekaligus, namun memproses urutan *token input* lalu memilih satu *token* dengan misal probabilitas terbesar. Token adalah representasi nilai dari sebuah teks. Sebagai *best practice*, 1 token adalah ¾ kata. Alasan mengapa teks diubah menjadi nilai token adalah karena model sebenarnya adalah persamaan matematis yang memetakan nilai masukan x ke nilai keluaran y atau f(x) = y.



Beberapa contoh LLM seperti GPT yang dikembangkan oleh OpenAI sebagai model dari aplikasi ChatGPT. Google juga mengembangkan PALM dan FLAN-T5 sebagai LLM. Bahkan perusahaan finansial seperti Bloomberg dan JP Morgan juga mengembangkan LLM yang khusus mengenali istilah dan format dokumen finansial seperti BloombergGPT dan DocLLM. Dalam konteks model bahasa, diksi *large* merujuk ke model *machine learning* yang memiliki jumlah parameter sangat banyak, biasanya dalam skala *billion* bahkan *trillion*.

Use Case LLM

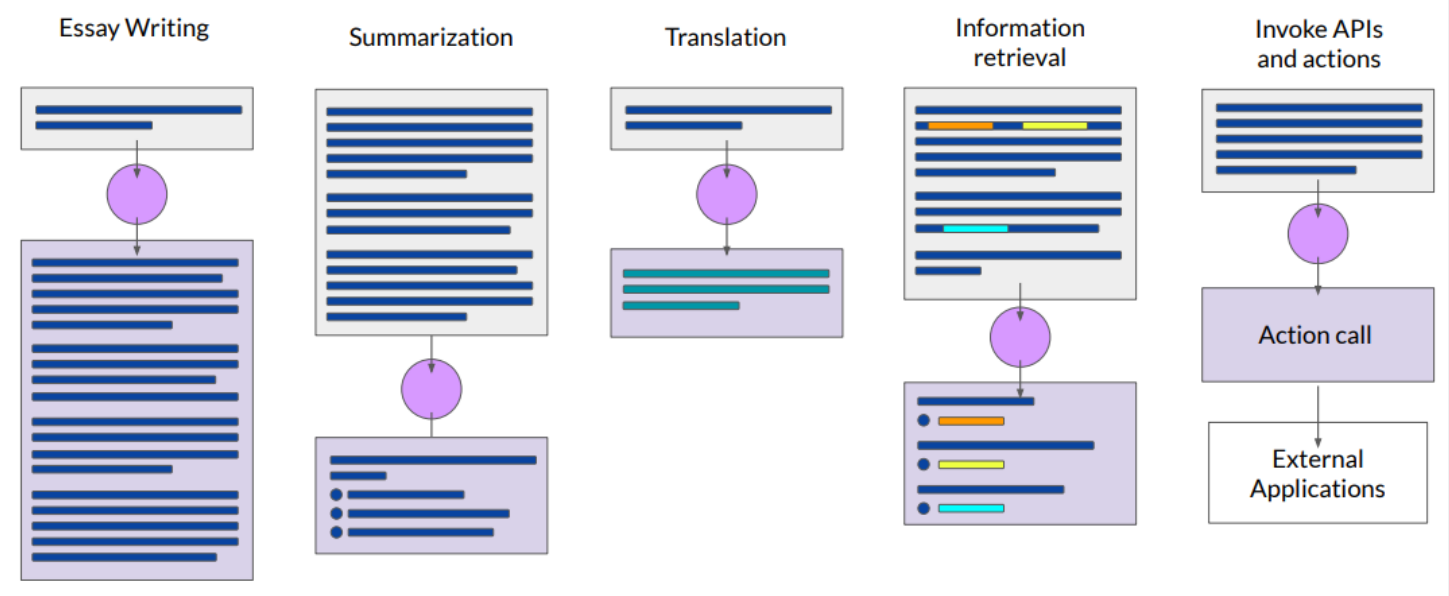
LLM sebagai salah satu jenis GenAI memiliki beberapa *use cases* atau *tasks* seperti: penulisan *essay,* merangkum artikel *(summarization),* alih bahasa *(translation),* mengekstrak dan mengambil informasi *(information retrieval),* dll.

Pada perkembangannya, LLM memiliki beberapa variasi arsitektur lebih cocok untuk mengerjakan *use case* tertentu. *Use cases* seperti Analisa sentimen, pengenalan entitas, dan klasifikasi kata lebih cocok menggunakan arsitektur *Autoencoding*.

Use case text generation lebih cocok menggunakan arsitektur *Autoregressive*.

Sementara itu, untuk *use cases* alih bahasa, merangkum artikel, dan tanya jawab lebih cocok menggunakan arsitektur *Sequence-to-Sequence*.

Pembahasan terkait detil setiap variasi arsitektur tidak dibahas pada modul ini.

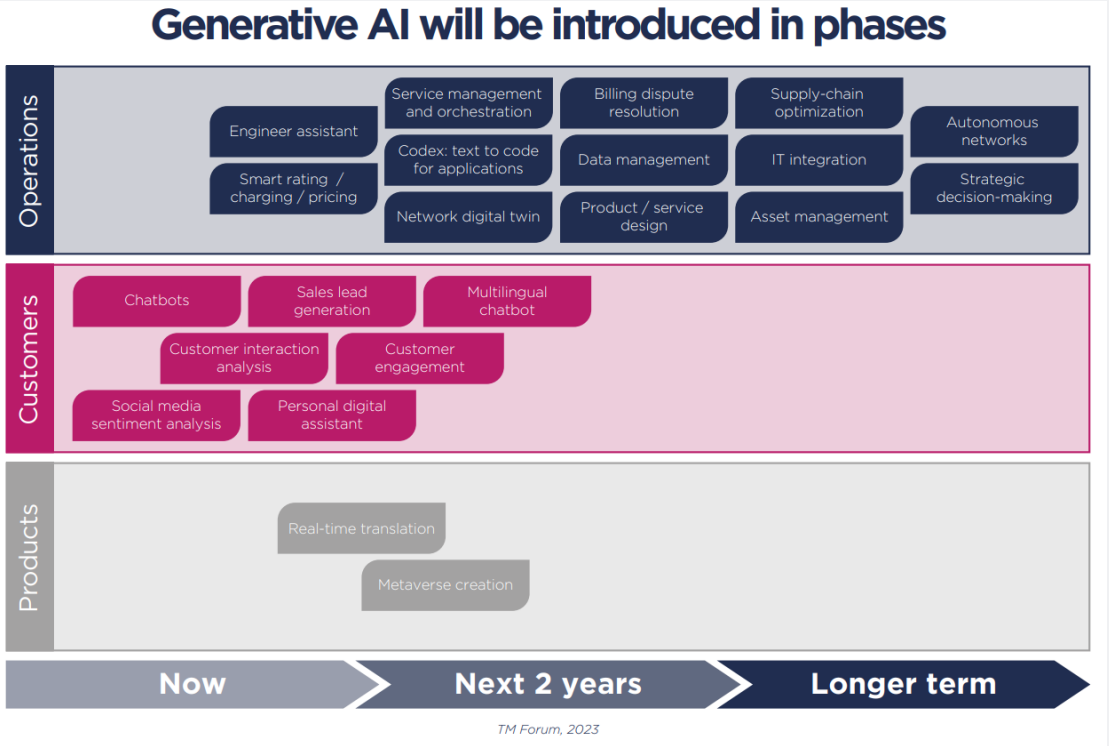


GenAI di Telekomunikasi

Newman (2023) pada laporan TMForum menampilkan beberapa *use case* GenAI untuk bidang telekomunikasi dan dikelompokkan berdasakarkan domain (produk, pelanggan, operasional) serta potensi tahun implementasinya (saat ini, 2 tahun ke depan, jangka panjang).

Salah satu *use case* GenAI di telekomunikasi yang dapat diimplementasikan saat ini salah satunya adalah *chatbot*. Operator dapat memasukkan *chatbot* sebagai salah satu interaksi kepada pelanggan.

GenAI memungkinkan operator mengubah pengalaman penggunaan *chatbot*. Saat ini *chatbot* menggunakan AI tradisional dan *rules-based systems*. Artinya pengembang harus menstrukturkan *query* apa yang berpotensi ditanyakan oleh pelanggan sebagai prakondisi. Di sisi lain GenAI *chatbots* menggunakan *unstructured data* dibandingkan *structured data* untuk memahami apa yang pelanggan inginkan dan bagaimana dapat dibantu.



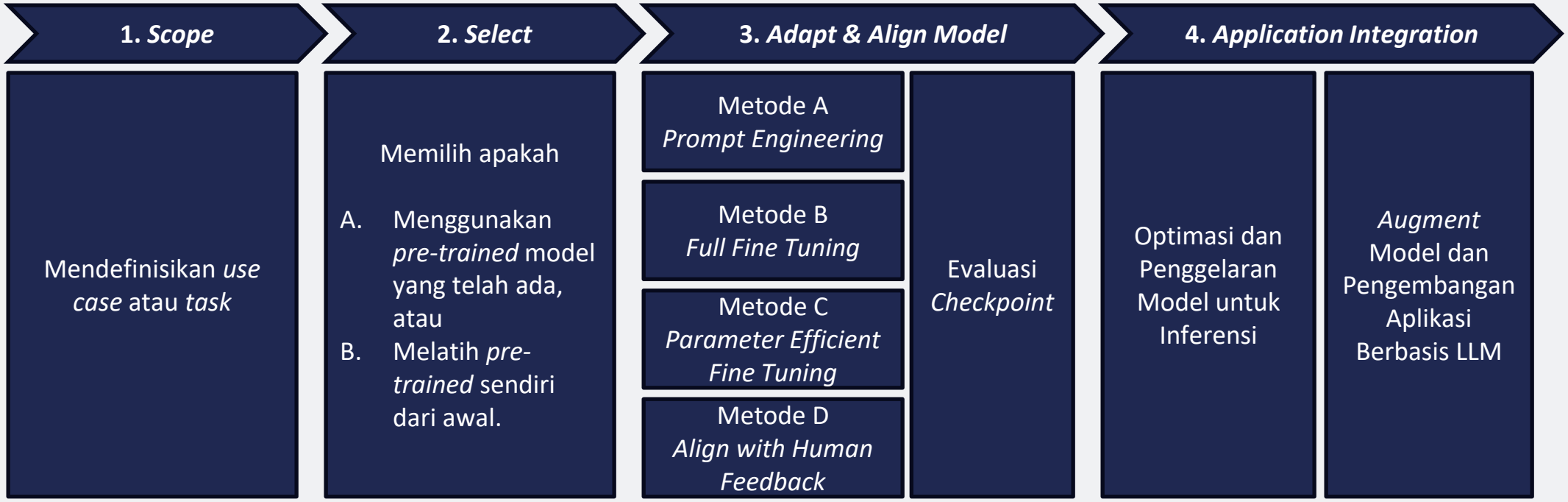
Pada modul ini, kita juga akan mempraktekan proses pengembangan *chatbot* yang memanfaatkan LLM sebagai model untuk melakukan *text generator*.

Topik 2

Siklus Proyek *Generative* AI

Proyek *Generative* AI (GenAI)

AWS dan DeepLearning.AI memperkenalkan *Generative* AI (GenAI) *project lifecycle* yang terdiri dari empat tahap mulai dari (1) *scope,* (2) *select model,* (3) *adapt and align model,* serta (4) *application integration*.



Proyek *Generative* AI (GenAI)

1. Scope → 2. Select → 3. Adapt & Align Model → 4. Application Integration

Tahap paling penting dalam setiap proyek adalah mendefinisikan *scope* atau ruang lingkup pekerjaan hingga seakurat dan sesempit mungkin yang dapat dilakukan.

LLM memiliki kemampuan untuk melakukan beberapa *tasks*, tetapi kemampuan tersebut bergantung pada jenis arsitektur dan ukuran parameter dari model. Oleh karena itu, harus dipertimbangkan fungsi spesifik LLM pada aplikasinya. Beberapa pertanyaan yang dapat diajukan untuk memastikan *scope* dari proyek GenAI, antara lain.

1. Apa saja *use case* atau *task* yang akan dilakukan oleh LLM? Apakah LLM akan focus ke satu *task* spesifik saja atau LLM memiliki kemampuan lebih dari satu *task*?
2. Apa saja target kualitatif *(expected output)* dan/atau kuantitatif *(success metric)* yang digunakan untuk menguji model yang dikembangkan?
3. Apa *interface* yang digunakan untuk berinteraksi dengan model? Apa format *input* dan *output* dari model? Apa format *prompt* yang digunakan?

Dengan mendefinisikan *scope* serta kemampuan model secara spesifik dapat menghemat waktu dan yang lebih penting adalah biaya komputasi yang diperlukan.

Proyek *Generative* AI (GenAI)

1. Scope | **2. Select** | 3. Adapt & Align Model | 4. Application Integration

Aktivitas selanjutnya setelah mendefinisikan *scope* adalah memilih apakah menggunakan *pre-trained* model untuk dikembangkan lebih lanjut atau melatih model dari *scratch*. Selain itu perlu ditentukan apakah pengembangan akan dilakukan secara internal atau menggunakan penyedia LLM eksternal.

	Opsi A: *Train LLM From Scratch (Internal Build)*	**Opsi B: *Finetune Pre-trained Opensource LLM (Internal Build)***	**Opsi C: *Finetune Pre-trained Commercial LLM (Buy/Borrow)***
PROS	• Proteksi data sensitif dan hak atas kekayaan intelektual (HaKI) • Transparansi dan control penuh • Transparansi biaya • Tidak terkunci pada vendor	• Proteksi data sensitif dan hak atas kekayaan intelektual (HaKI) • Transparansi dan control penuh • Transparansi biaya • Tidak terkunci pada vendor	• *Time to market* paling cepat • Butuh sedikit bahkan tidak butuh infrastruktur komputasi dan data • Butuh kompetensi AI dengan level yang lebih rendah dibading Opsi B • Kualitas *commercial* LLM lebih baik *(generalized)* dibanding *open source* LLM saat ini.
CONS	• Biaya pengembangan sangat tinggi • *Time to market* yang panjang • Butuh kompetensi AI internal • Butuh infrastruktur komputasi • Butuh data *training* yang besar	• *Time to market* cukup panjang namun yang lebih cepat dibanding Opsi A • Masih butuh data *training* walaupun lebih sedikit dibanding Opsi A • Butuh infrastruktur komputasi • Butuh kompetensi AI tingkat menengah.	• Transparansi dan kendali yang rendah • Kustomisasi tingkat lanjut harus dilakukan vendor • Kebutuhan untuk membagi data sensitif • Terkunci pada vendor • Resiko perubahan harga oleh vendor

Sumber: https://www.appliedai.de/assets/files/LLM-Whitepaper-final_Digital03.pdf

Proyek *Generative* AI (GenAI)

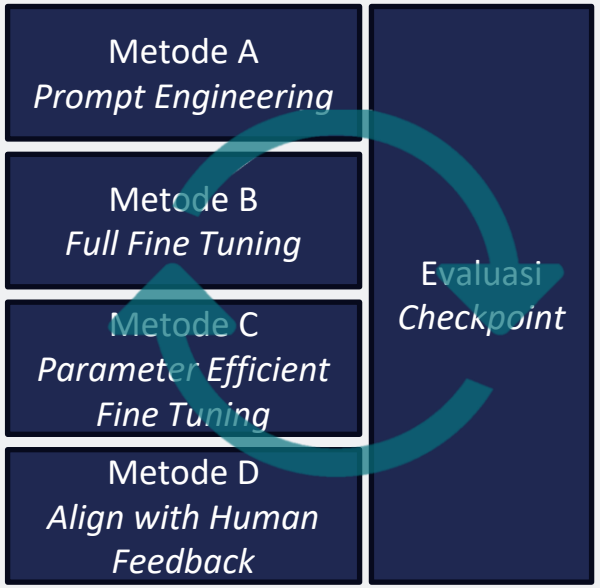
1. Scope | 2. Select | **3. Adapt & Align Model** | 4. Application Integration

Setelah memilih model pondasi pada langkah sebelumnya, selanjutnya dimulai proses mengadaptasikan model agar dapat melakukan *task* sesuai yang didefinisikan pada *scope*. Terdapat beberapa opsi metode baik yang melibatkan proses *training* ataupun cukup berkreasi dengan *prompt*. Proses adaptasi model dilakukan secara iteratif di mana dilakukan evaluasi di setiap iterasinya.

A ***Prompt Engineering***
Kreasi *prompt* dapat dilakukan dengan mengubah diksi kata maupun melakukan *in-context learning* baik *zero-shot*, *one-shot*, maupun *few-shot*.

B ***Full Fine Tuning***
Proses memodifikasi nilai parameter/*weight* model agar untuk mempelajari dataset pada *task* yang telah didefinisikan pada *scope*.

C ***Parameter Efficient Fine Tuning***
Serupa namun berbeda dengan ***Full Fine Tuning***. Pada ***Parameter Efficient Fine Tuning*** modifikasi hanya dilakukan pada beberapa lapisan parameter ataupun dengan menambah lapisan parameter baru.
Beberapa metodenya seperti: Low Rank Adaption (LoRA), Quantized LoRA (QLoRA), Soft Prompt (Prompt Tuning), dsb.



D ***Align with Human Feedback***
Metode ini mengkombinasikan metode *fune tuning* namun menggunakan pendekatan *reinforcement learning* untuk memberikan umpan balik dalam proces *training*-nya.
Beberapa contoh metodenya seperti: Reinforcement learning from human feedback (RLHF) dan Direct Preference Optimization (DPO)

Evaluasi *Checkpoint* dapat dilakukan baik secara kualitatif maupun kuantitatif sesuai dengan jenis *task*-nya, seperti *metric Rouge* untuk merangkum teks, dll.

Aspek lain yang juga perlu dilakukan di dalam tahapan ini adalah menyiapkan *dataset* dan memastikan bahwa model tidak memberikan *output* yang tidak berbahaya *(harmless)*.

Proyek *Generative* AI (GenAI)

1. Scope | 2. Select | 3. Adapt & Align Model | **4. Application Integration**

Tahap terakhir setelah model telah diadaptasikan sesuai dengan *task* yang didefinisikan di *scope*, maka model tersebut direkomentasikan untuk dioptimasi sebelum nantinya diintegrasikan dengan aplikasi.

Optimasi LLM

Optimasi dilakukan agar model ketika dijalankan di infrastruktur komputasi tidak terlalu butuh banyak tenaga komputasi dan memori. Terdapat beberapa teknik optimasi

1. ***Distillation*,** upaya menularkan *behavior* ke model baru dengan ukuran yang lebih kecil.
2. ***Quantization*,** upaya mengurangi *bit floating point* dengan *tradeoff* menurunnya presisi nilai parameter.
3. ***Pruning*,** upaya menghapus parameter model yang memiliki nilai dekat atau sama dengan nol.

Integrasi LLM ke Aplikasi

Model dapat diintegrasikan ke dalam aplikasi agar dapat dimanfaatkan oleh pengguna akhir *(end user).* Aplikasi harus mengelola masukan dari pengguna *(user input)* dan meneruskannya ke LLM yang akan mengembalikan keluaran seperti teks hasil rangkuman, respon pertanyaan, dll. Hal ini dilakukan dengan memanfaatkan *library* orkestrasi seperti *LangChain*.

Library tersebut dapat juga menyediakan akses ke sumber data eksternal atau API di aplikasi lain. *Library* ini memungkinkan LLM mendapatkan konteks yang lebih luas, faktual, dan terkini. Mengingat salah satu keterbatasan LLM adalah model hanya memiliki pengetahuan implisit sesuai dengan *dataset* yang digunakan pada proses *training*.

Topik 3

Pelaksanakan Proyek *Generative* AI (Hands-on)

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

Topik terakhir pada modul ini akan menunjukkan bagaimana proyek GenAI dilaksanakan dengan studi kasus membuat *chatbot* yang dapat berinteraksi dalam bahasa Indonesia.

Desclaimer

1. Studi kasus ini hanya melakukan satu iterasi saja. Pada kenyataannya pengembangan model butuh beberapa kali iterasi hingga *deliverable* sesuai dengan *scope*.
2. Oleh karena itu, baris kode yang digunakan dibuat sesingkat mungkin agar mudah dipahami. Pada kenyataannya pengembangan aplikasi berbasis LLM dapat memiliki baris kode yang lebih banyak untuk memastikan aspek *non-functional requirements* seperti *reliability, availability, performance, scalability, security, reusability, portability*, *maintainability,* dsb.
3. *Slide* hanya mencuplik beberapa potongan baris kode. Sementara itu, baris kode versi penuh dapat diakses melalui tautan yang dilampirkan.

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | 2. Select | 3. Adapt & Align Model | 4. Application Integration

Product manager dari sebuah produk digital merencanakan fitur baru yaitu *chatbot* yang dapat menjawab pertanyaan pengguna terkait produk tersebut. Beberapa *requirements* yang dikumpulkan dapat dilihat pada *board* berikut.

1. Interaksi *chatbot* menggunakan **Bahasa Indonesia**
2. *Chatbot* dapat menyapa pengguna dan menawarkan bantuan.
3. *Chatbot* dapat menjawab pertanyaan terkait produk.
4. *Deliverable* proyek adalah LLM yang dapat digunakan oleh aplikasi *chatbot*.
5. *Task* LLM adalah *text* generation dengan format ***prompt*** menggunakan ***Chat Markup Language***
6. Pengujian dilakukan secara kualitatif dengan memperhatikan kohesivitas teks.
7. Pondasi LLM tidak menggunakan layanan komersial agar tidak diperlukan *sharing* data ke vendor.
8. Ukuran LLM sebaiknya seminimal mungkin agar tidak membutuhkan komputasi tinggi pada penggunaannya.

Dari *requirements* yang terkumpul terlihat poin 1, 2, 3, 5 menetukan fungsi dari model yang akan dikembangkan. Sedangkan poin 7 dan 8 mensyaratkan aspek non-fungsional. Poin 6 menentukan cara pengujiannya. Poin 4 menyebutkan *deliverable* proyeknya yaitu LLM yang selanjutnya akan digunakan oleh pengembang aplikasi.

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | **2. Select** | 3. Adapt & Align Model | 4. Application Integration

Tahapan selanjutnya adalah memilih opsi model pondasi berdasarkan *scope* yang telah ditentukan. *Finetune Pre-trained Opensource LLM* menjadi opsi untuk memotong waktu pengembangan. Selanjutnya, pilih satu dari ribuan model yang tersedia di repositori model publik.

Mengingat salah satu *scope* adalah LLM harus dapat merespon masukan dengan menggunakan Bahasa Indonesia, maka ambil beberapa *pre-trained* model yang telah dilatih secara *unsupervised* menggunakan bahasa tersebut. Sebagai contoh adalah:

1. Facebook XGLM | Tautan
2. NVIDIA Nemotron | Tautan
3. Meta LLAMA | Tautan
4. HuggingFace Zephyr | Tautan
5. Bigscience Bloom | Tautan

Seusai *scope*, *task* dari model ada *chat*. Pada repositori publik terdapat veberapa varian *chat* untuk beberapa model *open source* tersebut.

Walaupun demikian, bilamana ukuran model dipertimbangkan, mayoritas model dengan *task chat* yang telah tersedia memiliki *weight* lebih dari sama dengan 7 milyar parameter. Selain itu beberapa model dengan *task chat* tersebut belum sesuai dengan format *Chat Markup Language*.

Aspek lain yang juga perlu dipertimbangkan ketika menggunakan sumber daya *open source* adalah jenis *open source license* yang disematkan.

Menimbang hal-hal tersebut maka pada proyek ini dipilih sebagai *pre-trained* model adalah Bloom1B1.

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | 2. Select | **3. Adapt & Align Model** | 4. Application Integration

Pada langkah ini *pre-trained* model dilatih untuk beradaptasi dengan *task* yang dipilih yaitu *chat* dengan Bahasa Indonesia dan menggunakan format *prompt Chat Markup Language*.

Load & Preprocess Dataset

Dataset yang digunakan adalah:

1. *alpaca-gpt4-indonesian*
2. *sharegpt-indonesian*
3. *evol-instruct-indonesian*
4. *Indoqa*

Dataset tersebut akan di-*load* dan di-*preprocess* unntuk menghasilkan format yang sesuai dengan *prompt*.

Format *Prompt*

Format *prompt* yang digunakan merujuk ke *Chat Markup Language*.

Walaupun demikian, agar *bloom* lebih mudah beradaptasi dengen format tersebut separator *chat* dimodifikasi menggunakan *bos_token* dan *eos_token* yang digunakan oleh Bloom.

Fine Tuning

Metode *fine tuning* yang digunakan adalah **Full Fine Tuning.**

Pada modul ini akan digunakan sebagian dari seluruh gabungan dataset untuk menyingkat waktu *training*. Para realitanya, *training* dapat memakan waktu minggu-bulanan.

Evaluasi

Evaluasi dilakukan secara kualitatif dengan mencoba *chat* secara langsung dan menilainya secara subjektif.

Sumber baris kode penuh dalam bentuk *python notebook* dapat diakses di sini.

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | 2. Select | **3. Adapt & Align Model** | 4. Application Integration

Load & Preprocess Dataset + Format *Prompt* → *Fine Tuning* → Evaluasi

```
1. alpaca_ds = load_dataset("FreedomIntelligence/alpaca-gpt4-indonesian")
2. def preprocess_alpaca(ds):
3.     chatML = []
4.     for conversation in ds['conversations']:
5.         chatML.append(
6.             {
7.                 'role': 'user' if conversation['from'] == 'human' else 'assistant',
8.                 'content': conversation['value'].strip(),
9.             }
10.         )
11.     return {
12.             'input_ids': chat_tokenizer.apply_chat_template(
13.                                 chatML,
14.                                 tokenize=True,
15.                                 add_generation_prompt=False,
16.                                 return_tensors="pt",
17.                                 truncation=True,
18.                                 padding=True,
19.                         )
20.     }
```

Baris kode di samping adalah protongan kode untuk *preprocess dataset alpaca-gpt4-Indonesian.*

Line 1 me-*load* dataset yang dipilih.

Line 4 s.d. *10* mengiterasi semua baris *dataset* dan di-*format* sesuai interaksi *chat*.

Line 12 s.d. *19* mengonversi data *string* interaksi *chat* yang telah digabung menjadi nilai angka atau *token* sesuai format *prompt*.

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | 2. Select | **3. Adapt & Align Model** | 4. Application Integration

Load & Preprocess Dataset + Format *Prompt* → *Fine Tuning* → Evaluasi

```
1. checkpoint="bigscience/bloom-1b1"
2. chat_tokenizer=AutoTokenizer.from_pretrained(checkpoint)
3. chat_tokenizer.chat_template="{% if not add_generation_prompt is defined %}{% set
   add_generation_prompt = false %}{% endif %}{% for message in messages %}{{ bos_token
   }}{{message['role'] + '\n' + message['content'] + '\n'}}{{ eos_token }}{% if not
   loop.last %}{{ '\n' }}{% endif %}{% endfor %}{% if add_generation_prompt %}{{
   '\n'}}{{ bos_token }}{{'assistant' + '\n' }}{% endif %}“


4. def group_texts(tokenized_ds):
5.     tokenized_ds = {k: sum(tokenized_ds[k], []) for k in tokenized_ds.keys()}
6.     new_input_ids = []
7.     for row in tokenized_ds['input_ids']:
8.         total_length = len(row)
9.         new_input_ids += [row[i : i + max_length] for i in range(0, total_length,
   max_length)]
10.    return {
11.        'input_ids': new_input_ids,
12.        'labels': new_input_ids,
13.    }
```

Baris kode di samping adalah protongan kode untuk menyiapkan *tokenizer* yang bertugas mengubah *string* ke token.

Seperti telah dijelaskan sebelumnya, pada proses tokenisasi terjadi juga modifikasi format ke *Chat Markup Language*. Hal ini dikonfigurasi pada *Line 3*.

Selain itu, pada *line* 4 adalah potongan fungsi kode untuk membagi deretan token ke ukuran yang identic. Penyamaan Panjang token ini diperlukan untuk mempercepat proses *training* secara paralel.

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | 2. Select | **3. Adapt & Align Model** | 4. Application Integration

Load & Preprocess Dataset + Format *Prompt* → *Fine Tuning* → Evaluasi

```
1. lm_dataset = ds_chatML.map(group_texts, batched=True, num_proc=4)
2. lm_dataset.set_format("torch")


3. data_collator = DataCollatorForLanguageModeling(tokenizer=chat_tokenizer, mlm=False)
4. lm_dataset = lm_dataset.train_test_split(test_size=0.2)
5. train_dataloader = DataLoader(lm_dataset["train"], shuffle=True, batch_size=8,
   collate_fn=data_collator)
6. eval_dataloader = DataLoader(lm_dataset["test"], batch_size=8,
   collate_fn=data_collator)
```

Baris kode di samping adalah protongan kode untuk menyiapkan data sebelum proses *training*.

Line 1 mengeksekusi fungsi pada *slide* sebelumnya dan *line* 2 deretan token dikonversi menjadi *tensor* pada *pytorch*.

Selanjutnya pada *line 3, data collator* melakukan proses *masking* pelatihan model bahasa secara otomatis. Sehingga pengembang model tidak perlu melakukan *coding* sendiri.

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | 2. Select | **3. Adapt & Align Model** | 4. Application Integration

Load & Preprocess Dataset + Format *Prompt* → ***Fine Tuning*** → Evaluasi

```
1. model = AutoModelForCausalLM.from_pretrained(
2.     checkpoint,
3.     )

4. optimizer = AdamW(model.parameters(), lr=2e-5)

5. num_epochs = 3
6. num_training_steps = num_epochs * len(train_dataloader)
7. lr_scheduler = get_scheduler(
8.     name="linear", optimizer=optimizer, num_warmup_steps=0,
   num_training_steps=num_training_steps
9. )

10.device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
11.model.to(device)
```

Baris kode di samping adalah protongan kode untuk menyiapkan *pre-trained* model dan *hyperparamters*.

Line 1 mengunduh parameter *pre-trained* model dari repositori publik.

Line 4 s.d. 9 mengonfigurasi *hyperparameters*.

Line 10 memindahkan lokasi memori model dan menentukan perangkat komputasi apakah pada CPU atau GPU.

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | 2. Select | **3. Adapt & Align Model** | 4. Application Integration

Load & Preprocess Dataset + Format *Prompt* → ***Fine Tuning*** → Evaluasi

```
1. progress_bar = tqdm(range(num_training_steps))

2. model.train() # swtich to training mode
3. for epoch in range(num_epochs):
4.     for batch in train_dataloader:
5.         batch = {k: v.to(device) for k, v in batch.items()}
6.         outputs = model(**batch)
7.         loss = outputs.loss
8.         loss.backward()
9.
           optimizer.step()
10.        lr_scheduler.step()
11.        optimizer.zero_grad()
12.        progress_bar.update(1)
```

Baris kode di samping adalah protongan kode untuk mengeksekusi proses *training*.

Line 1 menyiapkan *progress bar* agar terpantau prosesnya.

Line 2 s.d. *12* melakukan *forward* dan *backward propagation* pada setiap *epoch*.

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | 2. Select | **3. Adapt & Align Model** | 4. Application Integration

```
1. model.eval() # swtich to inferencing mode

2. messages = [
3.     {
4.         "role": "system",
5.         "content": "Anda adalah BaGoEs, Chatbot yang dikembangkan oleh Group of Expert.
   Jawab pertanyaan dengan maksimal dua kalimat.",
6.     },
7.     {
8.         "role": "user",
9.         "content": "Perkenalkan diri Anda!",
10.    },
11. ]

12.input_ids = chat_tokenizer.apply_chat_template(
13.                     messages,
14.                     tokenize=True,
15.                     add_generation_prompt=True,
16.                     return_tensors="pt",
17.             )
18.generated_text = model.generate(input_ids=input_ids.to(device),
19.                                generation_config=GenerationConfig(max_new_tokens=200),
20.                 )
21.print(chat_tokenizer.decode(generated_text[0], skip_special_tokens=False))
```

Evaluasi

Baris kode di samping adalah protongan kode untuk menyiapkan prompt untuk proses inferensi.

Line 1 memindahkan model untuk evaluasi.

Line 2 s.d. *11* menyiapkan interaksi *chat*.

Line 12 s.d. *17* mengonversi interaksi *chat pada* format *string* menjadi token.

Line 18 menginputkan token ke model untuk membangkitkan keluaran token.

Line 21 mengonversi token keluaran ke format *string* Kembali dan menampilkan ke layer.

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | 2. Select | **3. Adapt & Align Model** | 4. Application Integration

Load & Preprocess Dataset + Format *Prompt* → *Fine Tuning* → Evaluasi

```
1. <s>system
2. Anda adalah BaGoEs, Chatbot yang dikembangkan oleh Group of
   Expert. Jawab pertanyaan dengan maksimal dua kalimat.
3. </s>
4. <s>user
5. Perkenalkan diri Anda!
6. </s>
7. <s>assistant
8. Halo! Nama saya BaGoEs, dan saya adalah Chatbot yang
   dikembangkan oleh Group of Expert. Saya di sini untuk membantu
   menjawab pertanyaan Anda sebaik mungkin. Mari kita mulai?
9. </s>
```

Baris kode di samping adalah hasil potongan baris kode pada *slide* sebelumnya.

Line 1 s.d. *7* adalah *prompt* sebagai masukan model.

Line 8 dan seterusnya adalah keluaran dari model.

Diksi `<s>` dan `</s>` adalah token separator dari *Chat Markup Language* yang dimodifikasi sesuai konfigurasi Bloom.

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | 2. Select | 3. Adapt & Align Model | **4. Application Integration**

Pada tahap terakhir studi kasus modul ini tidak melakukan optimasi dan mengintegrasikan LLM ke sebuah aplikasi *chat* berbasis *web*.

Studi kasus ini tidak melakukan optimasi LLM

Optimasi tidak dilakukan karena ukuran model yang digunakan sebagai *pre-trained* cukup kecil.

Proses opsional antara paska lulus evaluasi dan sebelum integrasi aplikasi.

Pada studi kasus ini digunakan repositori model untuk memudahkan kolaborasi antara pengembang model dan pengembang aplikasi. Repositori model yang digunakan adalah Huggingface. Huggingface adalah Github untuk model *machine learning*.

Paska lulus evaluasi, model diunggah ke Huggingface. Hasil pengunggahan *(push)* dapat diakses di sini.

Integrasi LLM ke Aplikasi

Aplikasi dikembangkan dengan spesifikasi sebagai berikut.

- Antarmuka: *Webchat* dengan *input form* berformat teks.
- Infrastruktur Komputasi dan *Hosting*: *Public Cloud* menggunakan Huggingface Spaces.
- Bahasa Pemrograman: *Python*
- *Library* orkestrasi: Gradio

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | 2. Select | 3. Adapt & Align Model | **4. Application Integration**

```
1.  import gradio as gr
2.  from transformers import AutoTokenizer, AutoModelForCausalLM, GenerationConfig

3.  tokenizer = AutoTokenizer.from_pretrained("haidlir/bloom-chatml-id")
4.  model = AutoModelForCausalLM.from_pretrained("haidlir/bloom-chatml-id")

5.  def predict(message, history):
6.      history_chatml_format = []
7.      for human, assistant in history:
8.          history_chatml_format.append({"role": "user", "content": human })
9.          history_chatml_format.append({"role": "assistant", "content":assistant})
10.     prefix = "Kamu adalah BaGoEs, sebuah chatbot. Beri jawaban pendek dan singkat."
11.     history_chatml_format.append({"role": "system", "content": prefix})
12.     history_chatml_format.append({"role": "user", "content": message})
13.
        model_inputs = tokenizer.apply_chat_template(
14.                     history_chatml_format,
15.                     tokenize=True,
16.                     add_generation_prompt=True,
17.                     return_tensors="pt",
18.                 )
19.
        generated_text = model.generate(input_ids=model_inputs,
20.                                     generation_config=GenerationConfig(max_new_tokens=512),
21.                                    )
22.     len_input = len(model_inputs[0])
23.     return tokenizer.decode(generated_text[0][len_input:], skip_special_tokens=True).strip()

24. gr.ChatInterface(predict).launch()
```

Baris kode di samping adalah isi file `app.py` untuk menjalankan *webchat* berbasis *chatbot* yang telah dilatih.

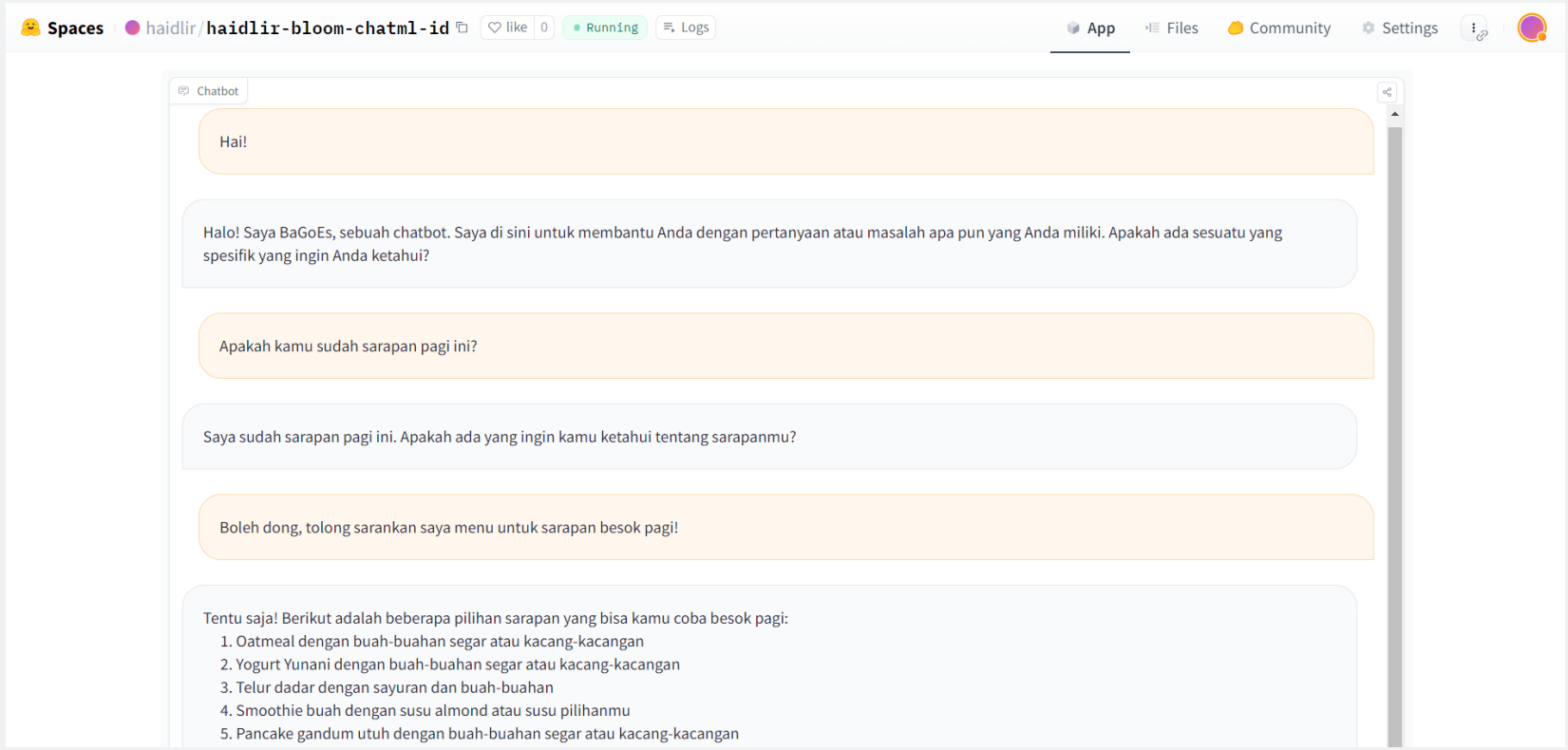
Line 3 dan *4* menginisiasi *tokenizer* dan model sesuai dengan nama identitas repositori lokasi pengunggahan model yang telah dilatih.

Line 5 s.d. *23* berisi proses ketika sebuah pesan di-*submit*.

Line 24 menjalankan *webchat* yang akan memanggil proses pada *line 5* s.d. 23.

Proyek *Generative* AI: Membuat *Chatbot* Berbahasa Indonesia

1. Scope | 2. Select | 3. Adapt & Align Model | **4. Application Integration**



Tangkapan gambar di samping adalah contoh tampilan aplikasi *webchat* yang terintegrasi dengan LLM yang telah dilatih pada tahapan sebelumnya.

Kesimpulan

1. *Generative* AI (GenAI) merupakan model *machine learning* yang memiliki kemampuan untuk menghasilkan *output* atau keluaran secara dinamis setelah model tersebut dilatih (*trained)* di mana algoritma pada GenAI membuat konten baru berdasarkan pola yang dipelajari dari data yang telah ada atau *existing* data.
2. *Large language models* (LLMs) merupakan salah satu jenis GenAI yang dapat memahami dan menghasilkan bahasa seperti manusia.
3. AWS dan DeepLearning.AI memperkenalkan Generative AI (GenAI) *project lifecycle* yang terdiri dari empat tahap mulai dari (1) mendifinisikan *scope*, (2) memilih model pondasi, (3) mengadaptasi model, serta (4) mengintegrasikan model ke aplikasi.
4. Salah satu bentuk Proyek GenAI yang dapat dilakukan adalah pengembangan *chatbot* berbasis LLM yang dapat berinteraksi dalam bahasa Indonesia.

Daftar Pustaka

1. Barth, A, Eigenbrode, S, Chambers, M & Fregly, C 2023, *Generative AI with Large Language Models*, DeepLearning.AI & Amazon Web Services.
2. Chang, DrPY-C & Pflugfelder, B 2023, *A Guide for Large Language Model Make-or-Buy Strategies: Business and Technical Insights*, retrieved <https://www.appliedai.de/assets/files/LLM-Whitepaper-final_Digital03.pdf>.
3. Newman, M 2023, I Kemp (ed), *Generative AI: Operators take their first steps*, retrieved 19 January 2024, <https://inform.tmforum.org/research-and-analysis/reports/generative-ai-operators-take-their-first-steps>.

Terima Kasih