Richard Rafer Guy – 222117056

# Task Multi Hop Reasoning

Multihop reasoning adalah proses penalaran yang melibatkan beberapa langkah logis dan beberapa informasi tertentu untuk menjawab sebuah pertanyaan.

## I. Penjelasan Dataset yang Digunakan

Pada percobaan yang dilakukan, terdapat 2 dataset yang digunakan sebagai task Multi Hop Reasoning. Kedua dataset yang digunakan, bisa didapatkan melalui referensi paper utama dari teknik Decomposed Prompting. Berikut adalah dataset yang digunakan :

A. **MuSiQue :** merupakan dataset untuk multi-hop question answering berbasis paragraf. Setiap pertanyaan membutuhkan beberapa gabungan informasi untuk menjawab. Pada percobaan yang dilakukan, terdapat 15 data saja dari dataset tersebut untuk dieksekusi oleh program. Pada dataset tersebut telah terdapat pertanyaan, jawaban, dan paragraph (yang dapat digunakan sebagai context pertanyaan), dan beberapa informasi lainnya.

B. **WikiMultiHopQA :** merupakan dataset yang dapat digunakan untuk multi-hop reasoning juga. Pada percobaan yang dilakukan, terdapat 15 data saja yang digunakan untuk dieksekusi oleh program. Pada dataset tersebut telah terdapat pertanyaan, jawaban, context (yang dapat digunakan), dan beberapa informasi lainnya.

Seluruh dataset yang digunakan disimpan dalam sebuah folder, terdapat 2 file dataset yang menyimpan masing masing dataset, dan setiap data / record disimpan dalam format JSON.

## II. Penjelasan Library yang Digunakan

Berikut adalah daftar library yang digunakan dalam kode program, beserta penjelasan fungsi/kegunaannya masing-masing:

A. **os** : digunakan untuk berinteraksi dengan sistem operasi (misalnya mendapatkan value dari env variable).
B. **google.generativeai** : digunakan untuk mengatur API key, inisialisasi model, dan mengirim prompt serta menerima hasil dari model.
C. **python-dotenv** : merupakan library untuk membaca dan memuat isi dari file .env.
D. **json** : digunakan untuk membaca dan menyimpan data dalam format JSON.
E. **argparse** : digunakan untuk membaca argumen yang diberikan melalui command-line saat mengeksekusi program.
F. **re** : library ini digunakan untuk pemrosesan regex, untuk ekstraksi atau pembersihan teks.
G. **time** : library ini membantu memberikan delay proses, untuk menghindari rate limit API.

H. **ast** : library ini digunakan untuk mengevaluasi ekspresi Python dengan aman. Dapat untuk mengubah string jadi list/dict Python secara aman.

## III. Penjelasan Kode Program

Bagian ini akan membahas program yang digunakan, cara kerja, hingga peran masing-masing komponen yang mereferensi ke dokumentasi kode yang telah dilakukan :

A. File **utils/gemini_client.py,** digunakan untuk menyiapkan konfigurasi gemini client dengan API Key yang terdapat pada .env, dan juga menyiapkan function untuk mengirim promp dan mendapatkan hasil response dari LLM. Selain itu **utils/io_utils.py**, sebagai penyedia function untuk load & save data dalam format JSON.

B. File **main.py** berfungsi sebagai titik awal program. Program ini menerima argumen --task (commaqa, reverse, lettercat) dan `--output` lewat command line menggunakan argparse, lalu menentukan controller sesuai task melalui fungsi `get_controller()`. Fungsi `main()` memanggil controller tersebut untuk memproses dataset dan menyimpan hasilnya ke file JSON. Di bagian atas, file ini mengimpor berbagai library dan fungsi pendukung seperti load_json, save_json, GeminiClient, serta re, os, dan time untuk pemrosesan teks dan delay.

C. File **wiki_controller.py,** digunakan untuk memproses task dengan dataset WikiMultiHopQA. Ketika awal dibuat, dataset akan dimuat dan disimpan. Setelah itu terdapat beberapa function seperti :

1. **print_question (Line 16-19)** : untuk menampilkan dataset yang sudah dimuat.
2. **replace_references (Line 21-27)** : untuk mengganti placeholder #n dengan jawaban ke-n yang telah dihasilkan sebelumnya.
3. **get_generated_answer (Line 32-37)**: untuk mengambil jawaban akhir dari model LLM dengan format “`Answer: <ANS>`”.
4. **clean_text (Line 39-40)**: untuk menghapus teks dalam tanda kurung dan kurawal dari string (karena teks tersebut sebagai penanda subtask yang digunakan pada sebuah permintaan dan tidak perlu diberikan dalam prompt kepada LLM).
5. **get_references (Line 42-44)**: untuk mengambil semua angka referensi #n dari string pertanyaan yang diberikan.
6. **get_references_array (Line 46-49)**: digunakan untuk mendapatkan jawaban ke-n yang memiliki format list tapi disimpan pada string dari referensi #n pada sebuah permintaan. Contoh : `"[\"Ab\",\"Cd\"]"` maka akan menghasilkan `[“Ab”, “Cd”]` dengan tipe list.
7. **get_formatted_context (Line 124-136)**: digunakan untuk mengonversi konteks menjadi format string secara lebih terstruktur dan siap digunakan dalam prompt
8. **implic_RAG (Line 51-63)**: untuk mendapatkan konteks yang tepat dan bermanfaat berdasarkan pertanyaan tertentu dengan memanfaatkan template prompt dan LLM Gemini.
9. **generate_chain (Line 138-145)**: untuk mendekomposisi pertanyaan menjadi beberapa langkah soal (QS) dengan bantuan LLM dan template prompt tertentu.
10. **chain_processing (Line 66-121)**: memproses setiap langkah hasil dari function generate_chain dengan menggunakan subtask yang sesuai, dengan bantuan LLM dan

template prompt tertentu. Setelah itu menyimpan hasil secara berurutan. Terdapat 3 subtask yang dapat digunakan :

a. **get_ent_qa** : untuk mengidentifikasi entitas utama dalam pertanyaan, seperti nama sekolah, tempat, atau objek yang sedang dibahas.
b. **get_atr_qa** : untuk mengambil atribut / informasi yang diperlukan dari masing-masing entitas, seperti lokasi, waktu, atau deksripsi lain. Pada bagian ini akan terdapat konteks hasil implicit RAG untuk membantu memberikan jawaban yang tepat.
c. **comp_qa** : membandingkan atribut dari kedua entitas untuk mendapatkan hasil akhir dari pertanyaan utama.

Sehingga untuk menghasilkan output akhir sebuah pertanyaan, akan dilakukan pemrosesan secara **iterative** dengan menjalankan subtask tertentu (contoh langkah proses dan output terdapat pada poin 12). Akhir iterasi pemrosesan adalah ditandai dengan perintan "[EOQ]" sebagai perintah penutup pada hasil dekomposisi pertanyaan.

11. **Solve (Line 147-174)**: untuk memproses seluruh pertanyaan atau data pada dataset, dari dekomposisi, proses pengeksekusian, evaluasi dan penyimpanan hasil akhir.
12. **Contoh Output Pemrosesan Data:**

```
- Question: "Are Sam Earle and Felix Luckeneder from the same country?"
- QS: ["[get_ent_qa] Who are the people being compared in the question?",
-       "(foreach)[get_atr_qa] What is the nationality of #1?",
-       "[comp_qa] Do they have the same nationality?",
-        "[EOQ]"]
- Output : [
-     "[\"Sam Earle\", \"Felix Luckeneder\"]",
-     [ "Sam Earle => Canadian", "Felix Luckeneder => Austrian" ],
-     "No"
- ]
```

D. File **musique_controller.py :** digunakan untuk memproses task dengan dataset MuSiQue. Ketika awal dibuat, dataset akan dimuat dan disimpan. Setelah itu terdapat beberapa function seperti :

1. **print_questions (Line 16-19)**: untuk menampilkan dataset yang sudah dimuat.
2. **get_generated_answer (Line 21-26)**: untuk mengambil jawaban akhir dari model LLM dengan format "Answer: <ANS>".
3. **get_fullcontext (Line 28-32)**: digunakan untuk mengonversi konteks menjadi format string secara lebih terstruktur dan siap digunakan dalam prompt
4. **implic_RAG (Line 34-50)**: untuk mendapatkan konteks yang tepat dan bermanfaat berdasarkan pertanyaan tertentu dengan memanfaatkan template prompt dan LLM.
5. **chain_processing (Line 52-76)**: memproses setiap langkah hasil dengan menggunakan subtask yang sesuai, dengan bantuan LLM dan template prompt tertentu. Setelah itu menyimpan hasil secara berurutan. Terdapat 2 subtask yang dapat digunakan :
   a. **Subtask pertama** berkaitan dengan entity identification untuk mengidentifikasi relasi dari entitas yang disebutkan
   b. **Subtask kedua** berkaitan dengan relational chaining, dengan tujuan untuk menemukan informasi turunan dari entitas tersebut (yang didapat dari hasil subtask pertama).

Sehingga untuk menghasilkan output akhir sebuah pertanyaan, akan dilakukan pemrosesan secara **iterative** dengan menjalankan subtask tertentu (contoh langkah proses dan output terdapat pada poin 7). Akhir iterasi pemrosesan adalah ditandai dengan perintan "[EOQ]" sebagai perintah penutup pada hasil dekomposisi pertanyaan.

6. **solve (Line 78-100)**: untuk memproses seluruh pertanyaan atau data pada dataset, dari dekomposisi, proses pengeksekusian, evaluasi dan penyimpanan hasil akhir.
7. **Contoh Output Pemrosesan Data**:

```
- Question: "Which company owns the manufacturer of Learjet 60?"
- QS: ["Learjet 60 >> manufacturer",
-      "#1 >> owned by",
-      "[EOQ]"]
- Output: ["Bombardier Aerospace",
-          "Bombardier Inc"]
```

E. File **prompts/wiki/template.py,** untuk menyiapkan semua template prompt yang akan digunakan oleh wiki_controller pada proses eksekusi task dengan dataset WikiMultiHopQA.
   1. **decomp_template** : untuk menghasilkan langkah dekomposisi dari pertanyaan utama.
   2. **get_ent_qa_template**: untuk mendapatkan entitas utama dari pertanyaan utama.
   3. **get_atr_qa_template** : untuk mendapatkan attribute / informasi dari sebuah entitas dengan bantuan context yang telah disusun.
   4. **comp_qa_template** : untuk mendapatkan hasil komparasi antara atribut dan informasi pada beberapa entitas.
   5. **Implic_rag_template** : untuk mendapatkan context yang sesuai dari pertanyaan tertentu.

F. File **prompts/musique/template.py**, menyiapkan template template prompt yang akan digunakan oleh musique_controller pada proses eksekusi task pada dataset MuSiQue.
   1. **starter_qa** : untuk mendapatkan hasil dari subtask pertama.
   2. **finisher_qa** : untuk mendapatkan hasil dari subtask kedua, hasil dari subtask pertama juga akan dimasukan pada template prompt ini, untuk menghasilkan jawaban dari subtask kedua.
   3. **impli_rag_template** : untuk mendapatkan context yang sesuai dari pertanyaan tertentu.

## IV. Mekanisme Perhitungan Performansi & Hasil Eksekusi

Perhitungan performansi dilakukan dengan menghitung persentase jawaban benar yang dihasilkan melalui proses yang telah dirancang dengan teknik Decomposed Prompting untuk kasus MultiHopReasoning. Jawaban yang dihasilkan LLM akan dibandingkan dengan jawaban benar yang telah tersedia pada dataset. Berikut adalah hasil performansi dari penggunaan Decomposed Prompting dengan LLM Gemini 2.0 Flash :

A. **Pada Dataset MuSiQue**: Perhitungan score dilakukan dengan jumlah jawaban benar dibandingkan dengan jumlah total soal yang ada pada dataset.
   1. Benar : 6
   2. Salah : 9
   3. Score : 40.00%

B. **Pada Dataset WikiMultiHopQA** : Perhitungan score dilakukan dengan jumlah jawaban benar dibandingkan dengan jumlah total soal yang ada pada dataset.
   1. Benar : 15
   2. Salah : 0
   3. Score : 100%