Implementasi Algoritma Breadth-First Search untuk Penelusuran Jalur *Fusion* Persona pada Game Persona 5 Royal

Muhammad Daffa Arrizki Yanma - 13524133
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: arrizkiyanma@gmail.com , 13524133@std.stei.itb.ac.id

***Abstrak*—Persona 5 Royal memiliki mekanik *fusion* yang memungkinkan pemain menggabungkan dua Persona untuk memperoleh Persona baru. Banyaknya kemungkinan kombinasi *fusion* dapat membuat pencarian jalur menuju Persona target menjadi kurang praktis jika dilakukan secara manual. Makalah ini membahas implementasi algoritma Breadth-First Search untuk mencari jalur normal *fusion* menuju satu Persona target. Program dikembangkan menggunakan TypeScript dan dijalankan melalui *command line interface*. Data Persona dan aturan *fusion* disimpan dalam format JSON yang diolah dari data Persona 5 Royal pada repository megaten-*fusion*-tool. Pencarian dilakukan dengan memodelkan Persona target sebagai state awal, kemudian menelusuri resep normal *fusion* yang dapat menghasilkan Persona tersebut. Ukuran optimum yang digunakan adalah jumlah langkah *fusion* paling sedikit, dengan pembanding tambahan berupa total level bahan dan urutan teks jalur *fusion*. Pengujian dilakukan pada lima Persona target valid, yaitu Raoul, Hanuman, Pale Rider, Garuda, dan Loa. Hasil pengujian menunjukkan bahwa seluruh target berhasil ditemukan dengan satu langkah *fusion*, expanded nodes bernilai 1, dan runtime berada pada rentang 2 sampai 4 ms.**

***Kata Kunci*—Breadth-First Search, fusion Persona, Persona 5 Royal, pencarian jalur, TypeScript.**

I. Pendahuluan

Persona 5 Royal merupakan permainan role-playing dari ATLUS yang memiliki mekanik Velvet Room, yaitu tempat pemain dapat melakukan *fusion* untuk membuat Persona baru [1]. Dalam permainan, hasil *fusion* bergantung pada Persona yang digunakan sebagai bahan. Semakin banyak Persona yang tersedia, semakin banyak kombinasi yang perlu ditelusuri ketika pemain ingin memperoleh Persona tertentu. Pencarian secara manual dapat memakan waktu karena satu target dapat dicapai melalui beberapa urutan *fusion* yang berbeda. Permasalahan tersebut dapat dinyatakan sebagai persoalan pencarian pada ruang status.

Breadth-First Search dipilih karena algoritma ini menelusuri simpul berdasarkan urutan pembangkitan dengan agenda FIFO [2]. Jika setiap aksi *fusion* diberi bobot yang sama, yaitu satu langkah, BFS memeriksa semua kemungkinan pada kedalaman tertentu sebelum melanjutkan ke kedalaman berikutnya. Karena setiap aksi *fusion* dianggap memiliki bobot satu langkah, BFS dapat menghasilkan jalur *fusion* dengan jumlah langkah minimum [3].

Makalah ini membahas implementasi BFS untuk menelusuri jalur *fusion* Persona pada Persona 5 Royal. Mekanik lanjutan seperti *skill* inheritance, biaya uang, *fusion* alarm, dan *fusion* accident berada di luar ruang implementasi agar pengujian tetap terpusat pada pencarian jalur *fusion* dengan jumlah langkah minimum.

II. Dasar Teori

A. Persona 5 Royal

Persona 5 Royal adalah permainan *role-playing* dari ATLUS yang menempatkan pemain sebagai seorang siswa pindahan yang menjalani kehidupan sekolah sekaligus berperan sebagai anggota Phantom Thieves [1]. Dalam permainan ini, pemain menjalani aktivitas harian, membangun hubungan sosial dengan karakter lain, serta memasuki dungeon untuk menghadapi musuh. Kombinasi antara aktivitas sosial dan eksplorasi dungeon membuat perkembangan karakter pemain berkaitan dengan strategi yang digunakan dalam pertarungan.

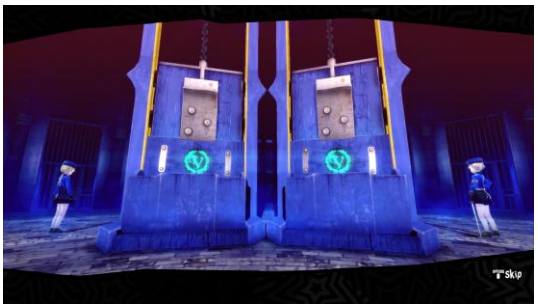
Salah satu unsur utama dalam Persona 5 Royal adalah penggunaan Persona sebagai sumber kemampuan karakter [1]. Setiap Persona memiliki atribut dan *skill* yang berbeda, sehingga pemilihan Persona berpengaruh terhadap cara pemain menghadapi musuh.

Gambar 2.1. Persona 5 Royal

Sumber: https://www.sega.com/persona/persona-5-royal

B. *Fusion Persona*

Fusion Persona adalah mekanik pada Persona 5 Royal yang dilakukan melalui Velvet Room. Velvet Room memungkinkan pemain melakukan *fusion* untuk membuat Persona baru [1]. Dalam permainan, mekanik ini digunakan untuk memperoleh Persona yang dapat membantu pemain menghadapi musuh yang lebih kuat. Proses *fusion* memperluas pilihan Persona yang dapat digunakan pemain, karena Persona hasil *fusion* dapat memiliki atribut dan kemampuan yang berbeda dari Persona bahan.



Gambar 2.2. *Fusion* 2 Persona

Sumber: https://www.rpgsite.net/feature/5486-persona-5-royal-strength-confidant-*fusion*-solutions-guide

C. *Breadth-First Search*

Breadth-First Search atau BFS termasuk ke dalam kelompok uninformed search atau blind search [2]. BFS menggunakan queue dengan prinsip First-In, First-Out atau FIFO [2]. Pada BFS, simpul berikutnya yang diekspansi dipilih berdasarkan urutan pembangkitannya [3]. Dengan aturan FIFO, simpul yang lebih dahulu dimasukkan ke dalam queue akan diperiksa lebih dahulu [2], [3].

Pohon pencarian BFS pada Route Planning menunjukkan bahwa simpul-simpul pada satu tingkat diperiksa sebelum pencarian dilanjutkan ke tingkat berikutnya [2]. Oleh karena itu, BFS menghasilkan lintasan dengan jumlah langkah paling sedikit atau minimum [2].

Jumlah langkah perlu dibedakan dari total biaya lintasan. Dalam Route Planning menjelaskan bahwa lintasan dengan langkah paling sedikit tidak selalu menjadi lintasan dengan biaya paling rendah apabila biaya setiap langkah berbeda [2]. BFS juga memiliki kebutuhan ruang yang dapat bersifat eksponensial [3]. Pada proses pencarian, simpul-simpul hidup disimpan di dalam queue sampai memperoleh giliran untuk diekspansi [2].

III. Ruang Lingkup Implementasi

Bagian ini menjelaskan ruang lingkup implementasi program pencarian kombinasi *fusion* Persona pada Persona 5 Royal. Implementasi difokuskan pada pencarian jalur *fusion* optimum berdasarkan jumlah langkah *fusion* dengan menggunakan algoritma Breadth-First Search.

1. Spesifikasi Data: Data yang digunakan merupakan data Persona 5 yang terdiri dari daftar Persona, level Persona, arcana, dan aturan normal *fusion*. Data disimpan dalam format JSON dan diambil dari repository megaten-fusion-tool pada bagian data Persona 5 [4].
2. Masukan Program: Program menerima satu masukan utama, yaitu nama Persona target yang ingin dicari kombinasi *fusion*-nya.
3. Asumsi Ketersediaan Persona: Program diperlakukan sebagai kalkulator *fusion*. Seluruh Persona yang terdapat pada data dianggap tersedia sebagai kandidat bahan *fusion*.
4. Batasan Mekanik *Fusion*: Implementasi hanya mencakup normal *fusion* yang menggunakan dua Persona sebagai bahan. Mekanik lain seperti special *fusion*, biaya uang, *skill* inheritance, *fusion* alarm, *fusion* accident, dan batasan level pemain tidak dimasukkan ke dalam program.

IV. Metodologi

Metodologi implementasi pencarian kombinasi *fusion* Persona dilakukan melalui beberapa tahapan, mulai dari pembacaan data *fusion* hingga penampilan jalur optimum menuju Persona target. Program menerima satu nama Persona sebagai masukan, kemudian menelusuri data normal *fusion* untuk menemukan kombinasi dengan jumlah langkah *fusion* paling sedikit menggunakan algoritma Breadth-First Search. Berikut alur implementasi yang digunakan:

1. Persiapan Data Persona: Data Persona dan aturan normal *fusion* dibaca dari file JSON yang sudah disiapkan dalam program. Data tersebut mencakup nama Persona, level, arcana, serta relasi *fusion* antar-Persona yang diambil dari repository megaten-*fusion*-tool [4]. Pada tahap ini, data dimuat ke dalam struktur data TypeScript agar dapat digunakan dalam proses pencarian.
2. Normalisasi Masukan Target: Masukan pengguna berupa nama Persona target diproses agar tidak bergantung pada perbedaan huruf besar dan huruf kecil. Nama target dibandingkan dengan daftar Persona yang tersedia pada data program. Jika target tidak ditemukan dalam data, program menghentikan proses pencarian dan menampilkan pesan bahwa Persona target tidak valid
3. Pembentukan State Pencarian: Setiap state dalam pencarian menyimpan informasi kombinasi *fusion* yang sudah terbentuk serta jumlah langkah yang sudah ditempuh. State awal dibentuk dari Persona target yang dimasukkan pengguna. Program kemudian mencari aturan normal *fusion* yang dapat menghasilkan target tersebut. Setiap aturan yang sesuai akan menjadi kandidat jalur *fusion* yang dapat diperluas pada proses pencarian.
4. Penerapan Breadth-First Search: Algoritma BFS digunakan untuk menelusuri kandidat kombinasi *fusion* berdasarkan kedalaman pencarian. Queue digunakan untuk menyimpan state yang akan diperiksa,

sedangkan struktur visited digunakan untuk mencegah pemeriksaan berulang terhadap kombinasi yang sama. BFS dipilih karena pencarian dilakukan berdasarkan jumlah langkah *fusion*, dan BFS dapat menghasilkan solusi minimum step pada kasus dengan bobot aksi yang seragam [3].

V. IMPLEMENTASI

A. Typescript

Program dikembangkan menggunakan TypeScript dalam lingkungan Node.js dan dijalankan melalui *command line interface*. Data yang digunakan disimpan dalam format JSON. Berkas personas.json memuat daftar Persona yang digunakan program, dan *fusion*s.json memuat aturan normal *fusion* yang dapat diproses oleh algoritma. Data tersebut berasal dari data Persona 5 Royal pada repository megaten-*fusion*-tool [4], kemudian disesuaikan ke bentuk yang lebih langsung digunakan oleh program. Data asli tetap disimpan pada folder p5, sedangkan folder data berisi data hasil penyesuaian yang dipakai saat program dijalankan.

Struktur program dibagi menjadi beberapa modul. index.ts berperan sebagai titik masuk program untuk membaca masukan pengguna, memuat data, menjalankan pencarian, dan menampilkan hasil. loadData.ts digunakan untuk membaca data JSON. normalizeName.ts digunakan agar pencarian nama Persona tidak dipengaruhi perbedaan huruf besar dan huruf kecil. buildRecipes.ts membentuk pemetaan aturan *fusion*, sedangkan bfs*Fusion*.ts memuat proses pencarian menggunakan BFS. Hasil pencarian kemudian disusun ke bentuk keluaran terminal melalui formatResult.ts.

B. Alur Program

Alur program dimulai dari masukan pengguna berupa nama Persona target melalui terminal. Setelah itu, program memeriksa apakah target terdapat dalam daftar Persona yang tersedia pada data. Jika nama target tidak ditemukan, program menghentikan pencarian dan menampilkan pesan bahwa Persona target tidak valid.

Setelah target dinyatakan valid, program memuat daftar Persona dan aturan normal *fusion* dari berkas JSON. Aturan *fusion* disusun menjadi relasi antara dua Persona bahan dan satu Persona hasil. Pada tahap ini, program membentuk pemetaan berdasarkan Persona hasil agar daftar kombinasi yang dapat menghasilkan target dapat diperoleh dengan lebih cepat. Karena ruang lingkup implementasi hanya mencakup normal *fusion*, setiap aturan *fusion* terdiri dari dua Persona bahan.

Pencarian kombinasi dilakukan menggunakan algoritma Breadth-First Search. Queue digunakan untuk menyimpan kandidat jalur *fusion* yang akan diperiksa, struktur visited digunakan untuk mencegah pemeriksaan berulang terhadap kombinasi yang sama. Setiap kandidat yang diperiksa memuat informasi Persona yang sedang ditelusuri, jalur *fusion* sementara, dan jumlah langkah yang sudah ditempuh. Ketika program menemukan jalur yang dapat menghasilkan Persona target, proses pencarian dihentikan dan jalur tersebut dikembalikan sebagai hasil.

Optimum pada implementasi ini adalah jumlah langkah *fusion* paling sedikit. Setiap proses normal *fusion* dianggap memiliki bobot satu langkah. Dengan BFS yang memeriksa kandidat berdasarkan kedalaman pencarian, jalur pertama yang memenuhi kondisi pencarian diperlakukan sebagai jalur optimum berdasarkan jumlah langkah *fusion*. Program tidak memperhitungkan biaya uang, *skill* inheritance, special *fusion*, *fusion* alarm, *fusion* accident, atau batasan level pemain.

Keluaran program berisi nama Persona target, jumlah langkah *fusion*, jumlah simpul yang diperiksa, waktu eksekusi, dan urutan *fusion* yang ditemukan. Jika kombinasi normal *fusion* menuju target tidak tersedia pada data, program menampilkan pesan bahwa kombinasi *fusion* tidak ditemukan.

C. Implementasi program

a. Pemeriksaan State dari Queue
Bagian utama BFS berjalan selama queue masih berisi state. Pada setiap iterasi, program mengambil state paling depan dari queue. Jika state tersebut sudah pernah diperiksa, program melewatinya.

```
while (queue.length > 0) {
  const node = queue.shift();
  if (!node) {
    continue;
  }

  if (bestStepCount !== undefined && node.steps.length > bestStepCount) {
    break;
  }

  const visitedKey = createVisitedKey(node);
  if (visited.has(visitedKey)) {
    continue;
  }
  visited.add(visitedKey);
```

Pengambilan elemen menggunakan queue.shift() menunjukkan bahwa program memakai prinsip FIFO. Dengan prinsip ini, state yang masuk lebih awal akan diperiksa lebih dahulu.

b. Pemeriksaan kandidat hasil
Jika daftar pending kosong, berarti jalur *fusion* pada state tersebut sudah lengkap. State tersebut kemudian disimpan sebagai kandidat hasil. Program juga membatasi pencarian menggunakan maxDepth agar pencarian bertingkat tidak berjalan terlalu jauh.

```
  if (node.pending.length === 0) {
    const candidate = createCandidate(node.steps, maps);
    candidates.push(candidate);
    bestStepCount = candidate.steps.length;
    continue;
  }

  if (bestStepCount !== undefined && node.steps.length >= bestStepCount) {
    continue;
  }

  if (node.depth >= maxDepth) {
```

```
    continue;
  }
```

Kandidat hasil tidak langsung dikembalikan karena masih mungkin terdapat kandidat lain dengan jumlah langkah yang sama. Kandidat tersebut nantinya dibandingkan berdasarkan jumlah langkah, total level bahan, dan urutan teks jalur *fusion*.

c. Penulusuran bahan *fusion*
Jika kedalaman pencarian masih berada di bawah maxDepth, bahan *fusion* juga dapat dimasukkan ke queue untuk ditelusuri kembali. Bagian ini memungkinkan program membentuk jalur *fusion* bertingkat.

```
    if (nextSteps.length < maxDepth) {
      queueNestedRecipe(recipe, 'ingredientA', remainingPending, nextSteps, queue);
      queueNestedRecipe(recipe, 'ingredientB', remainingPending, nextSteps, queue);
      queue.push({
        pending: [...remainingPending, recipe.ingredientA, recipe.ingredientB],
        steps: nextSteps,
        depth: nextSteps.length
      });
    }
  }
```

Pada bagian ini, program dapat menelusuri bahan pertama, bahan kedua, atau kedua bahan sekaligus. Semua kemungkinan tersebut dimasukkan ke queue agar tetap mengikuti urutan pemeriksaan BFS.

d. Pemilihan hasil akhir
Setelah proses pencarian selesai, program memeriksa daftar kandidat yang ditemukan. Jika tidak ada kandidat, fungsi mengembalikan undefined. Jika terdapat kandidat, program memilih kandidat terbaik.

```
 if (candidates.length === 0) {
  return undefined;
 }

 candidates.sort(compareCandidate);
 const best = candidates[0];

 return {
  target,
  steps: best.steps,
  totalSteps: best.steps.length,
  totalIngredientLevel: best.totalIngredientLevel,
  expandedNodes,
  runtimeMs: Math.max(0, Math.round(performance.now() - startedAt))
 };
}
```

Pemilihan kandidat dilakukan dengan fungsi compareCandidate. Urutan pemilihannya adalah jumlah langkah *fusion* paling sedikit, total level bahan paling rendah, lalu urutan teks jalur *fusion*.

Hasil akhir yang dikembalikan berisi Persona target, daftar langkah *fusion*, jumlah langkah, total level bahan, jumlah simpul yang diekspansi, dan waktu eksekusi.

VI. PENGUJIAN DAN HASIL

Bagian ini menjelaskan hasil pengujian program pencarian jalur *fusion* Persona menggunakan Breadth-First Search. Pengujian dilakukan dengan memberikan beberapa nama Persona target melalui *command line interface*. Hasil yang diamati meliputi keberhasilan pencarian, jumlah langkah *fusion*, total level bahan, jumlah simpul yang diekspansi, waktu eksekusi, dan jalur *fusion* yang dihasilkan.

A. Skenario Pengujian

Pengujian dilakukan terhadap lima Persona target yang valid, yaitu Raoul, Hanuman, Pale Rider, Garuda, dan Loa. Target tersebut dipilih untuk melihat apakah program dapat menemukan jalur normal *fusion* dari data yang tersedia. Parameter yang diamati adalah jumlah langkah optimum, jumlah simpul yang diekspansi, waktu eksekusi, dan jalur *fusion* yang dihasilkan.

B. Skenario Pengujian

Pengujian Tabel berikut menunjukkan hasil pengujian program terhadap lima persona target.

No	Target	Total Combination	Optimal *Fusion*	Expanded Nodes	Runtime
1	Raoul	599	1	1	3
2	Hanuman	439	1	1	4
3	Pale Rider	525	1	1	3
4	Garuda	382	1	1	3
5	Loa	366	1	1	2

C. Ouput program

Salah satu contoh hasil pengujian diperoleh dari target Raoul. Program dijalankan dengan perintah berikut.

```
npm run dev -- "Raoul"
```

Keluaran program adalah sebagai berikut.

```
Target: Raoul
Total normal fusion combinations: 599
Optimal fusion steps: 1
Expanded nodes: 1
```

Runtime: 3 ms

Fusion path:

1. Alilat + Mandrake = Raoul

Output tersebut menunjukkan bahwa program menemukan jalur *fusion* langsung menuju Raoul melalui kombinasi Alilat dan Mandrake. Karena jumlah langkahnya satu, jalur ini menjadi jalur optimum berdasarkan ukuran yang digunakan dalam implementasi, yaitu jumlah langkah *fusion* paling sedikit.

D. *Hasil Jalur Fusion*

Berikut jalur *fusion* yang dihasilkan dari setiap target pengujian.

No.	Target	Jalur *Fusion*
1	Raoul	Alilat + Mandrake = Raoul
2	Hanuman	Ardha + Bicorn = Hanuman
3	Pale Rider	Crystal Skull + Kelpie = Pale Rider
4	Garuda	Ariadne + Kumbhanda = Garuda
5	Loa	Ariadne Picaro + Titania = Loa

Hasil pada tabel menunjukkan bahwa program mengembalikan satu jalur *fusion* untuk setiap target. Jika terdapat lebih dari satu kandidat dengan jumlah langkah yang sama, program memilih kandidat berdasarkan total level bahan yang lebih rendah, lalu urutan teks jalur *fusion*.

VII. Analisis

Hasil pengujian menunjukkan bahwa BFS dapat menemukan jalur normal *fusion* untuk seluruh Persona target yang diuji. Pada lima pengujian, seluruh target menghasilkan Optimal *Fusion* Steps sebesar 1. Hasil ini sesuai dengan asumsi implementasi bahwa seluruh Persona pada data dianggap tersedia sebagai bahan *fusion*. Dengan asumsi tersebut, ketika target memiliki resep normal *fusion* langsung, program tidak perlu membangun rantai *fusion* yang lebih panjang.

Nilai Expanded Nodes pada seluruh pengujian bernilai 1. Artinya, BFS hanya perlu mengekspansi state awal yang berisi Persona target. Setelah target diperiksa, program langsung menemukan resep normal *fusion* yang dapat menghasilkan target tersebut. Waktu eksekusi program berada pada rentang 2 sampai 4 ms. Nilai ini menunjukkan bahwa pencarian pada target yang diuji dapat diselesaikan dengan cepat. Runtime yang kecil dipengaruhi oleh jumlah simpul yang diekspansi yang juga kecil. Selain itu, data *fusion* sudah disusun ke dalam peta pencarian, sehingga program tidak perlu memindai seluruh data *fusion* secara berulang pada setiap iterasi BFS.

Perbedaan nilai Total Normal *Fusion* Combinations menunjukkan bahwa setiap target memiliki jumlah kombinasi normal *fusion* yang berbeda pada data program. Walaupun jumlah kombinasi yang tercatat berbeda, seluruh target tetap menghasilkan langkah optimum 1 karena ditemukan kombinasi langsung menuju target. Perbedaan jumlah kombinasi tersebut lebih berpengaruh terhadap banyaknya alternatif yang tersedia, sedangkan ukuran optimum tetap ditentukan oleh jumlah langkah *fusion*.

Secara umum, hasil pengujian menunjukkan bahwa program sudah memenuhi ruang lingkup implementasi, yaitu menerima satu Persona target dan mengembalikan satu jalur normal *fusion* optimum berdasarkan jumlah langkah.

VIII. Kesimpulan

Program pencarian *fusion* Persona pada Persona 5 Royal berhasil diimplementasikan menggunakan algoritma Breadth-First Search. Program menerima satu masukan berupa nama Persona target, memuat data Persona dan aturan normal *fusion* dari berkas JSON, lalu mencari satu jalur *fusion* optimum berdasarkan jumlah langkah *fusion* paling sedikit. Implementasi dibuat menggunakan TypeScript dan dijalankan melalui *command line interface*.

Berdasarkan hasil pengujian terhadap lima Persona target valid, program dapat menemukan jalur normal *fusion* untuk seluruh target yang diuji. Setiap target menghasilkan jalur optimum dengan satu langkah *fusion*, karena pada model implementasi ini seluruh Persona pada data dianggap tersedia sebagai bahan *fusion*. Nilai expanded nodes yang kecil dan runtime dalam rentang 2 sampai 4 ms menunjukkan bahwa pencarian berjalan cepat pada kasus yang diuji.

Penerapan BFS sesuai dengan ruang lingkup masalah karena setiap normal *fusion* dianggap memiliki bobot yang sama, yaitu satu langkah. Dengan kondisi tersebut, BFS dapat digunakan untuk mencari jalur dengan jumlah langkah minimum. Program belum mempertimbangkan special *fusion*, biaya uang, *skill* inheritance, *fusion* alarm, *fusion* accident, dan batasan level pemain.

IX. Lampiran

Repository GitHub program:

- https://github.com/daypft/makalah-stima

X. Ucapan Terimakasih

Penulis mengucapkan terima kasih kepada:

1. Tuhan Yang Maha Esa
2. Orang Tua yang selalu mendukung penulis.

Yang telah membantu penulis dalam menyelesaikan makalah ini.

References

[1] ATLUS, "Persona 5 Royal Official Website," ATLUS. [Accessed: Jun. 19, 2026].

[2] N. U. Maulidevi, "Penentuan Rute (Route/Path Planning), Bagian 1: BFS, DFS, UCS, Greedy Best First Search," IF2211 Strategi Algoritma, Institut Teknologi Bandung, 2026. [Accessed: Jun. 19, 2026].

[3] Rinaldi, N. U. Maulidevi, and M. L. Khodra, "Algoritma Branch & Bound, Bagian 1," IF2211 Strategi Algoritma, Institut Teknologi Bandung, 2026. [Accessed: Jun. 19, 2026].

[4] aqiu384, "megaten-*fusion*-tool," GitHub repository. [Online]. Available: https://github.com/aqiu384/megaten-*fusion*-tool. [Accessed: Jun. 19, 2026].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Muhammad Daffa Arrizki Yanma dan 13524133