Optimasi Penulisan Pengujian Perangkat Lunak dengan Large Language Model pada Metode Test-Driven Development

1st Muhammad Alif Rasyid Ramdhani
Fakultas Informatika
Universitas Telkom
Bandung, Indonesia
alifrasyid@students.telkomuniversity.ac.id

2nd Donni Richasdy
Fakultas Informatika
Universitas Telkom
Bandung, Indonesia
donnir@telkomuniversity.ac.id

3rd Ario Harry Prayogo
Fakultas Informatika
Universitas Telkom
Bandung, Indonesia
ariohp@telkomuniversity.ac.id

***Abstrak* — *Test-Driven Development (TDD)* adalah pendekatan dalam pengembangan perangkat lunak yang fokus pada peningkatan kualitas sejak awal, dengan cara membuat pengujian terlebih dahulu sebelum menulis kode. Meski demikian, ada tantangan yang sering terjadi, seperti kesulitan dalam menulis pengujian yang efektif, sehingga menghambat proses TDD. Penelitian ini bertujuan untuk memanfaatkan *Large Language Models (LLM)* sebagai bantuan dalam proses TDD, terutama dalam menghasilkan pengujian dan kode berdasarkan deskripsi dalam bahasa alami. Penelitian ini mengeksplorasi kemampuan LLM, khususnya model LLaMA 3, dalam menghasilkan *test case* secara otomatis dari deskripsi kebutuhan. Untuk mengevaluasi efektivitas model, dua skenario digunakan, yaitu deskripsi pendek dan panjang, serta dilakukan penilaian berdasarkan metrik otomatis. Penilaian dilakukan menggunakan metrik CodeBLEU, BLEU, dan chrF untuk mengukur tingkat kesamaan antara test case hasil dan referensi manual. Hasil penelitian menunjukkan bahwa skenario dengan deskripsi yang lebih lengkap memberikan skor CodeBLEU sebesar 32,20% dan chrF sebesar 44,34%. Selain itu, model juga menghasilkan *output* yang konsisten dalam setiap percobaan. Penelitian ini menunjukkan bahwa LLM dapat digunakan untuk mendukung proses TDD secara langsung tanpa dilakukan pelatihan tambahan, dan tetap mampu menghasilkan *test case* yang relevan.**

Kata kunci*— generasi tes, *large language model*, *test driven development.

I. PENDAHULUAN

Perkembangan perangkat lunak telah menjadi komponen utama dalam aspek kehidupan, mulai dari bisnis, pendidikan, kesehatan, hingga hiburan. Dengan meningkatnya kebutuhan pengguna, kualitas perangkat lunak sangat diperhatikan pada pengembangan perangkat lunak. Pengujian perangkat lunak merupakan cara penting untuk memastikan kualitas perangkat lunak [1]. Hal ini bertujuan untuk memaksimalkan fitur perangkat lunak seperti fungsionalitas dan kegunaan. *Test-Driven Development* merupakan salah satu pengembangan perangkat. *Test-Driven Development* di rancang untuk memastikan perangkat lunak berkualitas sejak tahap awal pengembangan.

Metode *Test-Driven Development* merupakan pendekatan dalam pengembangan perangkat lunak yang menekankan pembuatan pengujian, dengan pengembang menulis pengujian terlebih dahulu sebelum menulis kode yang dijalankan [2]. TDD mendorong pengembang untuk menulis kasus uji terlebih dahulu sebagai persyaratan kode, lalu mendefinisikan kode hingga semua kasus uji terpenuhi, sehingga mengurangi kesalahan dan memastikan penilaian kode berkelanjutan [3]. Tetapi TDD memiliki beberapa tantangan, salah satunya yaitu kesulitan dalam menulis tes yang efektif. Menulis tes yang efektif sangat di perlukan dalam pengujian perangkat lunak. Karena akan mempengaruhi kualitas perangkat lunak tersebut. Semakin baik hasil pengujiannya semakin baik juga kualitas perangkat lunaknya. Kesulitan dalam menulis pengujian yang efektif dan kurangnya keterampilan dalam mengembangkan pengujian yang baik menjadi hambatan bagi banyak pengembang [4].

Large Language Models (LLM) menjadi salah satu solusi untuk tantangan pada TDD. Perkembangan LLM ini juga mengalami perkembangan yang cukup signifikan. LLM menunjukkan kemampuan luar biasa dalam berbagai tugas NLP, seperti penerjemahan mesin, menjawab pertanyaan, meringkas, pembuatan teks, pemeriksaan tata bahasa, dan sebagainya [5]. LLM ini mampu menghasilkan dan memahami bahasa setara manusia dengan konteks yang mendalam dan makna yang kaya [6]. *Large Language Models* (LLM) memungkinkan konversi dari bahasa alami ke bahasa pemrograman, menurunkan hambatan dalam penulisan kode [7].

Seiring dengan perkembangan LLM yang pesat, berbagai model tersedia dengan karakteristik dan keunggulannya masing-masing. Oleh karena itu, pemilihan model yang tepat menjadi penting untuk memastikan hasil yang optimal. Dari beberapa pilihan yang ada, penelitian ini memanfaatkan LLaMA *3* yang dapat diakses secara gratis melalui GroqCloud dengan kecepatan *inference* yang tinggi, sehingga mendukung replikasi penelitian tanpa biaya tinggi. Model ini dipilih karena memiliki performa yang kompetitif dalam berbagai *benchmark*, bahkan dalam beberapa kasus mengungguli model besar lainnya dalam kelas parameter

yang setara. Misalnya, pada *benchmark* HumanEval untuk kemampuan pemrograman, LLaMA 3 Instruct 70B mengungguli Mistral-7B dan Gemma-2 9B serta mendekati skor GPT-3.5 Turbo [8].

Penelitian ini membahas pemanfaatan *Large Language Models* (LLM) dalam mendukung proses *Test-Driven Development* (TDD), khususnya pada pembuatan test case otomatis berbasis requirement API. Ruang lingkup penelitian dibatasi pada proyek Pretix sebagai studi kasus, dengan menggunakan LLaMA 3 melalui layanan GroqCloud sebagai model utama. Fokus diarahkan pada evaluasi kemampuan LLM dalam menghasilkan test case tanpa mencakup implementasi logika backend, sehingga analisis lebih terkonsentrasi pada aspek pengujian. Kualitas test case kemudian diukur menggunakan tiga metrik otomatis, yaitu BLEU, CodeBLEU, dan chrF, yang bersama-sama memberikan penilaian komprehensif dari segi kesamaan teks, struktur kode, serta karakteristik penulisan.

Penelitian ini bertujuan untuk mengimplementasikan *Large Language Models* (LLM) dalam proses pengembangan perangkat lunak berbasis *Test-Driven Development* (TDD) serta mengevaluasi kinerjanya dalam membantu penulisan pengujian dan pembuatan kode secara otomatis. Penelitian ini diharapkan mampu memberikan manfaat secara praktis berupa referensi bagi peneliti maupun praktisi dalam mengintegrasikan LLM ke dalam proses TDD, sekaligus menyajikan informasi mengenai efektivitas penggunaannya dalam meningkatkan kualitas dan efisiensi pengembangan perangkat lunak.

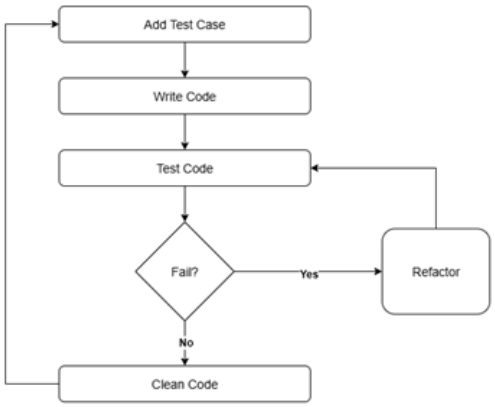
II. KAJIAN TEORI

A. Large Language Models

Large Language Model (LLM) adalah sistem AI yang dirancang untuk mengolah dan mengevaluasi data bahasa alami dalam jumlah besar, lalu menggunakan informasi tersebut untuk menghasilkan respons terhadap perintah pengguna [9]. Model ini dilatih pada kumpulan data teks besar, termasuk berbagai bahasa, sehingga mampu memahami konteks dan menghasilkan respons yang koheren [10]. LLM telah menjadi terobosan yang signifikan, mengubah cara kita memproses, menghasilkan, dan memahami bahasa manusia.

B. Test-Driven Development

Test Driven Development (TDD) adalah teknik yang memungkinkan pengembang untuk memikirkan fungsionalitas baru sebelum menulis kode [3]. Tujuan sistem ini adalah memverifikasi kode secara akurat, memberikan kepercayaan kepada *programmer* dengan mengurangi kerumitan, dan memotivasi mereka untuk menghasilkan kode berkualitas lebih baik [11].



GAMBAR 1 *Workflow* TDD

Gambar 1 menggambarkan alur pengerjaan *Test-Driven Development (*TDD**)** yang terdiri dari beberapa tahapan berulang. Proses dimulai dengan penulisan kasus pengujian untuk fitur tertentu yang ingin dikembangkan. Setelah itu, dilakukan implementasi kode sesuai dengan kebutuhan fitur tersebut, kemudian pengujian dijalankan untuk memastikan kesesuaian antara kode dan spesifikasi yang telah ditentukan. Jika hasil pengujian belum sesuai atau terjadi kegagalan, langkah selanjutnya adalah melakukan refaktor pada kode hingga seluruh pengujian dapat dilalui dengan baik [1]

C. LLAMA3

LLaMA 3 merupakan generasi terbaru dari keluarga *Large Language Model* (LLM) yang dikembangkan oleh Meta AI. Model ini dirancang sebagai fondasi untuk berbagai tugas pemrosesan bahasa alami, dengan dukungan untuk *multilingualitas*, penalaran, pemrograman, serta penggunaan alat bantu [8]. Model ini digunakan karena memiliki performa yang kompetitif dalam berbagai *benchmark*, bahkan dalam beberapa kasus mengungguli model besar lainnya dalam kelas parameter yang setara. Misalnya, pada *benchmark* HumanEval untuk kemampuan pemrograman, LLaMA 3 Instruct 70B mengungguli Mistral-7B dan Gemma-2 9B serta mendekati skor GPT-3.5 Turbo [8].

Dalam penelitian ini, LLaMA 3 digunakan sebagai basis untuk menghasilkan *test case* secara otomatis dari *requirement* API, sebagai bagian dari pendekatan *Test-Driven Development* (TDD). Model digunakan secara langsung dalam kondisi tanpa *fine-tuning*, sehingga tetap mempertahankan kemampuan generalis dari LLM dalam memahami perintah. LLaMA 3 dipilih karena kemampuannya dalam memahami konteks teknis, khususnya pada perintah berbasis kode dan API testing, serta ketersediaannya melalui layanan *inference* seperti GroqCloud yang memungkinkan pengujian tanpa perlu infrastruktur pelatihan khusus.

D. BLEU

BLEU (*Bilingual Evaluation Understudy*) adalah pengukuran lengkap n-gram antara reaksi model dengan umpan balik referensi, memberikan gambaran umum dari akurasi bahasa dari reaksi model [12]. Skor BLEU keseluruhan dihitung sebagai kombinasi dari logaritma presisi n-gram yang dimodifikasi dan penalti singkat. Skor BLEU keseluruhan dirumuskan sebagai:

$$BLEU = BP \cdot exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \quad (1)$$

Di mana:

N = panjang n-gram maksimum (umumnya 4).

w_n = bobot untuk masing-masing n-gram.

p_n = Modified n-gram Precision.

BP = Brevity Penalty.

Persamaan 1 digunakan untuk menghitung satu skor yang menggambarkan seberapa baik hasil keluaran model. Skor ini didasarkan pada kecocokan susunan kata (n-gram) dan panjang kalimat, yang dalam praktiknya sering dianggap sejalan dengan penilaian manusia terhadap kualitas terjemahan. BLEU dipilih karena merupakan metrik umum dalam *Natural Language Processing* yang efektif mengukur kesamaan secara tekstual, sehingga berguna untuk membandingkan *test case* yang dihasilkan LLM dengan *test case* acuan.

E. CodeBLEU

CodeBLEU merupakan gabungan bobot dari BLEU asli, pencocokan n-gram terbobot, pencocokan AST sintaksis, dan pencocokan aliran data semantik [13]. Formula umum CodeBLEU diberikan sebagai rata-rata dari empat komponen pencocokan: n-gram, n-gram berbobot, pencocokan sintaksis, dan pencocokan aliran data:

$$CodeBLEU = \alpha \cdot BLEU + \beta \cdot BLEU_{weight} + \gamma \cdot Match_{ast} + \delta \cdot Match_{df} \quad (2)$$

Di mana:

$\alpha, \beta, \gamma, \delta$ = bobot untuk masing-masing komponen.

BLEU = komponen pencocokan n-gram token.

$BLEU_{weight}$ = komponen pencocokan n-gram berbobot IDF.

$Match_{ast}$ = komponen pencocokan struktur sintaksis berdasarkan *Abstract Syntax Tree*.

$Match_{df}$ = komponen pencocokan semantik berdasarkan aliran data

Persamaan 2 digunakan untuk menghitung satu nilai skor yang mewakili kualitas kode secara menyeluruh. Skor ini tidak hanya melihat kesamaan dari segi teks, tetapi juga memperhatikan apakah struktur dan fungsi dari kode yang dihasilkan sudah sesuai atau belum. Bobot $(\alpha, \beta, \gamma, \delta)$ biasanya ditentukan secara empiris berdasarkan penelitian awal atau mengikuti konfigurasi dari penulis asli metode CodeBLEU. Dalam publikasi aslinya [13], nilai yang direkomendasikan adalah:

$$\alpha = 0.25 \qquad \beta = 0.25 \qquad \gamma = 0.25 \qquad \delta = 0.25$$

Keempat komponen diberi bobot yang sama. Namun, dalam kasus tertentu bobot dapat disesuaikan, misalnya jika ingin menekankan aspek semantik (γ, δ lebih besar) atau aspek kemiripan teks (α, β lebih besar). Penyesuaian bobot dilakukan melalui eksperimen pada *validation set* untuk melihat kombinasi mana yang paling sesuai dengan tujuan evaluasi. Metrik ini dipilih karena dirancang khusus untuk evaluasi kode pemrograman, dengan mempertimbangkan tidak hanya kesamaan teks, tetapi juga kesesuaian sintak dan semantik kode.

F. chrF

chrF (character n-gram F-score) adalah metrik yang mengevaluasi kualitas teks yang dihasilkan berdasarkan kesamaan n-gram karakter dan n-gram kata [14]. chrF didasarkan pada perhitungan *precision* (P) dan *recall* (R) dari n-gram karakter yang dibagi antara teks yang dihasilkan dan teks referensi, kemudian digabungkan menjadi skor F-measure [14]. Formula F-score secara umum adalah:

$$F_\beta = (1 + \beta^2)\frac{P.R}{\beta^2.P + R} \quad (3)$$

Di mana:

P = proporsi n-gram karakter dalam keluaran model yang juga ditemukan dalam referensi (*Precision*).

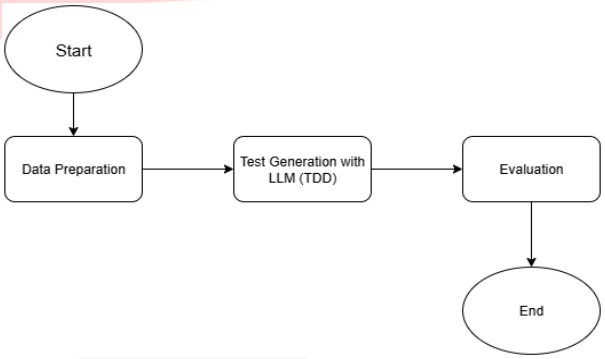
R = proporsi n-gram karakter dalam referensi yang juga ditemukan dalam keluaran model (*Recall*).

β = bobot relatif recall terhadap precision

Persamaan 3 bertujuan untuk mengukur keseimbangan antara *precision* dan *recall* pada tingkat n-gram karakter, sehingga lebih tangguh terhadap variasi tokenisasi dan perbedaan penulisan kecil, menjadikannya relevan untuk evaluasi kode. chrF dipilih untuk menangkap kesamaan pada tingkat penulisan, yang penting ketika struktur kata atau token berbeda tetapi maksud pengujian tetap sama.

III. METODE

A. Desain Perancangan Sistem



GAMBAR 2 *Flowchart*

Desain Sistem penulisan pengujian perangkat lunak dengan LLM terdiri dari beberapa tahapan yang digambarkan pada 2 Proses dimulai dengan persiapan data, *test generation* dengan LLM, hingga evaluasi *test* yang dihasilkan.

B. Data Preparation

Penelitian ini menggunakan data requirement API dari proyek open-source *Pretix*, sebuah sistem manajemen penjualan tiket daring yang memiliki dokumentasi API lengkap, mencakup penjelasan parameter, struktur respons, dan contoh penggunaan. *Pretix* dipilih karena bersifat open-source sehingga dapat diakses bebas, memiliki kompleksitas dan keragaman endpoint yang mencerminkan kondisi nyata dalam pengembangan perangkat lunak, serta menyediakan dokumentasi yang terstruktur dan jelas. Dengan karakteristik tersebut, *Pretix* dinilai relevan sebagai studi kasus untuk mengevaluasi efektivitas pemanfaatan *Large Language Models (LLM)* dalam mendukung penerapan *Test-Driven Development (TDD)*.

Data requirement yang dikumpulkan berjumlah 192 entri dan dikategorikan menjadi dua jenis. Pertama, deskripsi pendek yang disajikan dalam format *Excel* berisi ringkasan requirement API yang diorganisir per baris untuk mewakili satu kasus uji. Kedua, deskripsi panjang dalam format *JSON* yang memuat detail lebih lengkap, seperti *path parameter*, *query parameter*, dan *status code* pada setiap endpoint. Kedua jenis data ini digunakan sebagai masukan bagi model LLM untuk menghasilkan kode unit test secara otomatis,

sekaligus memungkinkan analisis efektivitas model pada berbagai tingkat kompleksitas requirement.

C. Test Generation with LLM

Proses pembuatan *test case* secara otomatis dalam penelitian ini dilakukan dengan memanfaatkan *Large Language Model (LLM)* melalui layanan *Groq Cloud API*. Tahapan ini mencakup konfigurasi model, perancangan strategi *prompting*, serta proses generasi *test case* berdasarkan requirement yang telah disiapkan sebelumnya. Interaksi dengan LLM dilakukan melalui permintaan *HTTP POST* menggunakan *payload JSON* yang memuat parameter konfigurasi tertentu, sehingga respons yang dihasilkan dapat sesuai dengan kebutuhan pengujian.

Peran utama *prompt engineering* adalah mengarahkan LLM untuk berperilaku sebagai penguji perangkat lunak profesional dan menghasilkan kode *unit test* menggunakan *framework Pytest*. Mekanisme generasi dilakukan secara otomatis dengan dua pendekatan, yaitu dari requirement berbentuk deskripsi pendek yang diolah dari file *Excel*, serta dari requirement detail dalam format *JSON*. Pada pendekatan pertama, sistem mengekstrak informasi penting dari setiap baris data *Excel* untuk membangun *prompt*, sedangkan pada pendekatan kedua, sistem membaca struktur *JSON* yang lebih kompleks seperti deskripsi entitas, detail *endpoint*, dan parameter untuk menghasilkan *prompt* yang lebih komprehensif. Kedua mekanisme ini memungkinkan pembuatan *unit test* yang konsisten dengan variasi tingkat detail requirement yang tersedia.

D. Rancangan Pengujian dan Evaluasi

Tahap rancangan pengujian dan evaluasi pada penelitian ini bertujuan untuk menilai performa *Large Language Model (LLM)* dalam menghasilkan kode *unit test*. Pengujian dilakukan dengan membandingkan *test case* hasil generasi model (*generated test*) dengan *test case* acuan yang ditulis secara manual (*reference test*). Kedua jenis *test case* disimpan dalam sebuah file *Excel*, di mana setiap baris berisi pasangan kode yang kemudian diproses secara berurutan. Proses pengujian meliputi pembacaan data, validasi kelengkapan *test case*, perhitungan metrik evaluasi pada setiap pasangan kode, serta agregasi hasil berupa nilai rata-rata untuk memperoleh gambaran performa keseluruhan model.

Evaluasi kualitas dilakukan menggunakan tiga metrik utama, yaitu *CodeBLEU*, *BLEU*, dan *chrF*. *CodeBLEU* digunakan untuk menilai kesamaan struktur, sintaks, dan aliran data kode, sehingga lebih sesuai dengan konteks pemrograman. *BLEU* dipakai untuk mengukur kesamaan pada tingkat token, meskipun awalnya dirancang untuk teks umum. Sementara itu, *chrF* berfokus pada kesamaan di tingkat karakter dan *n-gram* karakter, sehingga lebih toleran terhadap variasi penulisan. Skor dari masing-masing metrik kemudian dihitung rata-rata dan disajikan dalam bentuk persentase agar lebih mudah diinterpretasikan dalam menilai efektivitas LLM pada skenario *Test-Driven Development (TDD)*.

IV. HASIL DAN PEMBAHASAN

A. Hasil Pengujian Skenario 1 (Requirement Pendek)

Pada skenario pertama, digunakan *requirement* dengan deskripsi pendek yang bersifat ringkas dan langsung, diambil dari dokumentasi resmi Pretix. Setiap *requirement* diuji sebanyak 5 kali percobaan untuk menghasilkan *test case* secara otomatis menggunakan LLM.

TABEL 1 Skor Evaluasi Test Case Skenario 1

Percobaan	CodeBLEU (%)	BLEU (%)	chrF (%)
1	28.92	18.52	41.27
2	28.88	18.79	41.65
3	29.10	18.67	41.69
4	28.94	18.60	41.66
5	28.98	18.78	41.38
Rata-rata	**28.96**	**18.67**	**41.53**

B. Hasil Pengujian Skenario 2 (Requirement Panjang)

Pada skenario kedua, digunakan *requirement* dengan deskripsi panjang dan kompleks, diambil dari dokumentasi resmi Pretix. Setiap *requirement* diuji sebanyak 5 kali percobaan untuk menghasilkan *test case* secara otomatis menggunakan LLM.

TABEL 2 Skor Evaluasi Test Case Skenario 2

Percobaan	CodeBLEU (%)	BLEU (%)	chrF (%)
1	32.77	18.74	44.38
2	32.33	18.69	44.43
3	31.64	18.67	44.53
4	31.83	18.34	44.06
5	32.52	18.45	44.29
Rata-rata	**32.20**	**18.58**	**44.34**

C. Analisis Hasil Pengujian Skenario 1

Hasil percobaan pada Skenario 1 menunjukkan bahwa sistem menghasilkan output yang konsisten. Hal ini terlihat dari perbedaan nilai minimum dan maksimum pada lima kali percobaan yang sangat kecil di semua metrik, yakni CodeBLEU, BLEU, dan chrF. Konsistensi ini disebabkan oleh penggunaan input, parameter, serta konfigurasi yang sama pada setiap pengujian sehingga tidak menimbulkan variasi berarti dalam hasil.

Dari sisi struktur kode, skor CodeBLEU sekitar 28,96% menunjukkan adanya perbedaan yang cukup signifikan antara *generated test case* dan *reference test case*. Perbedaan ini terutama muncul pada penggunaan fungsi *assert*, susunan logika, dan penamaan variabel. Meskipun test case yang dihasilkan model tetap valid secara fungsional, struktur internalnya cenderung lebih generik dan tidak mengikuti gaya atau pola penulisan test case referensi, sehingga kesamaan sintaksis dan alur data menjadi rendah.

Sementara itu, skor BLEU sebesar 18,67% menegaskan bahwa urutan token antara hasil generasi dan test case

referensi berbeda cukup jauh. Model sering menghasilkan variasi penyusunan pernyataan atau bahkan menghilangkan detail tertentu, yang menyebabkan rendahnya kesesuaian tekstual. Namun, skor chrF yang lebih tinggi, yaitu 41,53%, menunjukkan bahwa secara karakteristik penulisan masih terdapat banyak kemiripan. Kesamaan istilah seperti *assert*, *status_code*, dan *response* tetap terjaga, meskipun terdapat variasi dalam struktur kalimat atau penamaan variabel. Hal ini menandakan bahwa meski berbeda dalam detail implementasi, test case yang dihasilkan masih relevan dalam konteks pengujian.

D. Analisis Hasil Pengujian Skenario 2

Hasil percobaan pada Skenario 2 menunjukkan tingkat konsistensi yang tinggi di seluruh metrik evaluasi. Selisih antara nilai minimum dan maksimum sangat kecil pada *CodeBLEU*, *BLEU*, maupun *chrF*, sehingga dapat disimpulkan bahwa model menghasilkan keluaran yang stabil ketika diuji berulang dengan data dan konfigurasi yang sama. Kondisi ini menguatkan validitas hasil evaluasi karena setiap percobaan memberikan gambaran yang konsisten terhadap performa model dalam skenario tersebut.

Dari sisi struktur kode, skor *CodeBLEU* meningkat dari 28,96% pada Skenario 1 menjadi 32,20% pada Skenario 2. Peningkatan ini mengindikasikan bahwa *test case* yang dihasilkan model semakin menyerupai struktur referensi, misalnya dalam penggunaan *client.get()* atau *client.post()* di awal, diikuti dengan *assert* terhadap status respons, serta validasi data tambahan. Model juga lebih sering menggunakan ekspresi yang sesuai konteks seperti *assertEqual(response.status_code, 200)* dibandingkan *assertTrue(response.ok)*. Namun demikian, perbedaan masih terjadi pada aspek penamaan variabel, susunan blok uji, serta penghilangan bagian *setup*, sehingga skor keseluruhan belum optimal.

Pada sisi kesesuaian tekstual, nilai *BLEU* justru menurun tipis dari 18,67% menjadi 18,58%. Hal ini terjadi karena model menghasilkan variasi penulisan yang lebih bebas, seperti memecah pernyataan, menambahkan komentar, atau menggunakan parameter berbeda, misalnya *response.json()['id']* dibandingkan *data['id']*. Selain itu, gaya penulisan yang lebih deskriptif juga memunculkan n-gram baru yang tidak terdapat dalam referensi. Kondisi ini menunjukkan kelemahan BLEU yang cenderung memberi penalti pada perbedaan urutan token meskipun makna pengujian tetap sama [12][15]. Sebaliknya, skor *chrF* meningkat signifikan dari 41,53% menjadi 44,34%, menandakan bahwa secara karakteristik penulisan, hasil model semakin mirip dengan referensi. Konsistensi penggunaan istilah teknis seperti *status_code*, *assert*, *response*, dan *client*, serta elemen khas Python seperti *def*, *return*, dan *json()* berkontribusi besar terhadap peningkatan ini. Perbedaan minor seperti tanda kurung atau urutan argumen tidak terlalu memengaruhi skor *chrF*, sehingga metrik ini lebih mampu menangkap kemiripan gaya penulisan kode dibandingkan BLEU.

V. KESIMPULAN

Penelitian ini bertujuan untuk mengimplementasikan *Large Language Model* (LLM) sebagai alat bantu dalam proses *Test-Driven Development* (TDD), khususnya dalam menghasilkan *test case* otomatis berdasarkan *requirement* API. Melalui dua skenario, yaitu dengan deskripsi pendek (Skenario 1) dan deskripsi panjang (Skenario 2), diperoleh hasil bahwa model mampu menghasilkan keluaran dengan konsistensi tinggi antar percobaan. Skenario 2 menunjukkan performa lebih baik pada metrik *CodeBLEU* (32,20%) dan *chrF* (44,34%) dibandingkan Skenario 1, menandakan bahwa semakin detail *requirement*, semakin baik pula struktur *test case* yang dibentuk oleh model. Namun, nilai *BLEU* sedikit menurun pada Skenario 2 (18,58%), yang menandakan bahwa urutan token masih berbeda dari referensi meskipun strukturnya lebih sesuai. Secara keseluruhan, LLM terbukti stabil dan berpotensi mendukung TDD sebagai alat bantu generatif, meskipun hasilnya belum sepenuhnya menyerupai *test case* manual dan membutuhkan penyempurnaan lebih lanjut.

Berdasarkan temuan penelitian ini, dapat diajukan sejumlah saran untuk pengembangan di masa mendatang. Pertama, penggunaan teknik *fine-tuning* pada data domain spesifik agar model mampu menghasilkan *test case* dengan gaya dan struktur lebih sesuai. Kedua, perluasan dataset *requirement* dari berbagai proyek *open-source* dengan variasi dokumentasi dan struktur API untuk menguji kemampuan generalisasi model. Ketiga, penggunaan metrik evaluasi tambahan agar efektivitas keluaran dapat dinilai lebih menyeluruh. Terakhir, diperlukan evaluasi oleh pengembang atau ahli pengujian secara langsung guna menilai kualitas dari aspek kelayakan, kemudahan pemahaman, serta kesesuaian dengan praktik pengujian manual, mengingat metrik otomatis belum sepenuhnya mampu menggambarkan kualitas yang dirasakan manusia.

REFERENSI

[1] H. M. Abushama, H. A. Alassam, and F. A. Elhaj, “The effect of Test-Driven Development and Behavior-Driven Development on Project Success Factors: A Systemantic Literature Review Based Study,” 2020.

[2] K. Karlsson and J. Gunnarsson, “Test-Driven Development Using LLM,” 2024.

[3] N. Pombo and C. Martins, “Test driven development in action: Case study of a cross-platform web application,” *EUROCON 2021 - 19th IEEE Int. Conf. Smart Technol. Proc.*, no. July, pp. 352–356, 2021, doi: 10.1109/EUROCON52738.2021.9535554.

[4] D. Staegemann *et al.*, “A Literature Review on the Challenges of Applying Test-Driven Development in Software Engineering,” *Complex Syst. Informatics Model. Q.*, vol. 2022, no. 31, pp. 18–28, 2022, doi: 10.7250/csimq.2022-31.02.

[5] Z. Liu, Y. Tang, X. Luo, Y. Zhou, and L. F. Zhang, “No Need to Lift a Finger Anymore? Assessing the Quality of Code Generation by ChatGPT,” *IEEE Trans. Softw. Eng.*, vol. 50, no. 6, pp. 1548–1584, 2024, doi: 10.1109/TSE.2024.3392499.

[6] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, “A Survey on Large Language Model (LLM) Security and Privacy: The Good, The Bad, and The Ugly,” *High-Confidence Comput.*, vol. 4, no. 2, p. 100211, 2024, doi: 10.1016/j.hcc.2024.100211.

[7] L. Chen, “Research on Code Generation Technology based on LLM Pre - training,” vol. 10, no. 1, 2024.

[8] A. Grattafiori *et al.*, “The Llama 3 Herd of Models,” pp. 1–92, 2024, [Online]. Available: http://arxiv.org/abs/2407.21783.

[9] A. M. Suruj, “A Compact Guide to Large Language Models,” no. November, 2023.

[10] J. Yang *et al.*, “Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond,” *ACM Trans. Knowl. Discov. Data*, vol. 18, no. 6, pp. 1–24, 2024, doi: 10.1145/3649506.

[11] M. M. Moe and K. K. Oo, “Evaluation of Quality, Productivity, and Defect by applying Test-Driven Development to perform Unit Tests,” *2020 IEEE 9th Glob. Conf. Consum. Electron. GCCE 2020*, pp. 435–436, 2020, doi: 10.1109/GCCE50665.2020.9291950.

[12] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “BLEU: a Method for Automatic Evaluation of Machine Translation,” *Proc. 40th Annu. Meet. Assoc. Comput. Linguist. (ACL),* pp. 311–318, 2002.

[13] S. Ren *et al.*, “CodeBLEU: a Method for Automatic Evaluation of Code Synthesis,” vol. 1949, no. Weaver 1955, 2020, [Online]. Available: http://arxiv.org/abs/2009.10297.

[14] M. Popović, “Chrf: Character n-gram f-score for automatic mt evaluation,” *10th Work. Stat. Mach. Transl. WMT 2015 2015 Conf. Empir. Methods Nat. Lang. Process. EMNLP 2015 - Proc.*, no. September, pp. 392–395, 2015, doi: 10.18653/v1/w15-3049.

[15] S. Lu *et al.*, “CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation,” *Adv. Neural Inf. Process. Syst.*, 2021.