Jurnal Teknologi Informasi dan Multimedia

p-ISSN : 2715-2529

e-ISSN : 2684-9151

https://journal.sekawan-org.id/index.php/jtim

Evaluasi Teknik *Prompting* pada *Large Language Model* untuk Otomatisasi Penyusunan Skenario *Unit Testing Smart Contract*

Salman El Farisi [1]*, Fahlia Athiyya Marva [2]

[1] Program Studi Teknik Informatika, Sekolah Tinggi Teknologi Terpadu Nurul Fikri, Indonesia
[2] Program Studi Sistem Informasi, Sekolah Tinggi Teknologi Terpadu Nurul Fikri, Indonesia

* Korespondensi: salman@nurulfikri.ac.id

Sitasi: S. E. Farisi and F. A. Marva, "Evaluasi Teknik Prompting pada Large Language Model untuk Otomatisasi Penyusunan Skenario Unit Testing Smart Contract", Jurnal Teknologi Informasi Dan Multimedia, vol. 8, no. 1, pp. 99-108, 2026. https://doi.org/10.35746/jtim.v8i1.912

Diterima: 03-12-2025
Direvisi: 09-01-2026
Disetujui: 20-01-2026

Abstract: Automated unit testing is essential for ensuring the security and reliability of smart contracts, particularly because their immutable nature prevents post-deployment modifications. However, manually creating test scenarios remains time-consuming, costly, and highly dependent on expert knowledge. A potential solution is to utilize AI technology, particularly Large Language Models (LLMs), to automatically generate test scenarios. This study fills the research gap in leveraging LLM technology in the software testing space by proposing a workflow for automatically generating unit test scenarios for blockchain smart contract code using Large Language Models (LLMs). The proposed workflow consists of two stages: converting Solidity smart contracts into structured Gherkin scenarios and translating those scenarios into executable Hardhat unit test scripts. This study proposes an automated workflow using Large Language Models (LLMs) to address these challenges. The workflow consists of two stages: con-verting Solidity smart contracts into structured Gherkin scenarios and trans-lating those scenarios into executable Hardhat unit test scripts. Using the Gemini 2.5 Pro model, the research evaluates three prompting techniques such as Chain-of-Thought, Few-Shot, and Role-Based through quantitative analysis based on code coverage metrics, including Statements, Branches, Functions, and Lines. The experimental results show that Role-Based Prompting achieves the highest average coverage (92.02%), followed by Few-Shot Prompting (89.52%), while Chain-of-Thought produces the lowest coverage (78.79%). Role-Based Prompting also attains the highest Branch coverage, demonstrating superior capability in capturing conditional logic within smart contracts.

Keywords: *Coverage Metrics; Large Language Models; Prompt Engineering; Smart Contract; Unit Testing*

Abstrak: Pengujian unit otomatis sangat penting untuk memastikan keamanan dan keandalan smart contract, terutama karena sifatnya yang tidak dapat diubah untuk mencegah modifikasi setelah *deployment*. Namun, pembuatan skenario pengujian secara manual memakan waktu, biaya, dan sangat bergantung pada pengetahuan dan pengalaman penguji. Salah satu pendekatan yang dapat dilakukan adalah dengan memanfaatkan teknologi AI terutama *Large Language Model* (LLM) untuk menghasilkan skenario pengujian secara otomatis. Studi ini mengisi celah penelitian dalam memanfaatkan teknologi LLM pada ruang lingkup pengujian perangkat lunak dengan mengusulkan alur kerja pembuatan skenario pengujian unit untuk kode smart contract blockchain secara otomatis menggunakan *Large Language Models* (LLMs). Alur kerja yang diusulkan dari dua tahap: mengonversi smart contract Solidity menjadi skenario Gherkin yang terstruktur dan menerjemahkan skenario tersebut menjadi skrip pengujian unit Hardhat yang dapat dieksekusi. Penelitian ini menggunakan model Gemini 2.5 Pro dalam mengevaluasi tiga teknik *prompting*, yaitu *Chain-of-Thought*, *Few-Shot*, dan *Role-Based*, melalui analisis kuantitatif berdasarkan metrik cakupan

pengujian (*test coverage*) dengan kriteria *Statements*, *Branch*, *Functions*, dan *Lines coverage*. Hasil eksperimen menunjukkan bahwa *Role-Based Prompting* mencapai cakupan rata-rata tertinggi (92,02%), diikuti oleh *Few-Shot Prompting* (89,52%), dan *Chain-of-Thought* dengan cakupan terendah (78,79%). *Role-Based Prompting* juga mencapai cakupan *Branch* tertinggi, menunjukkan kemampuan superior dalam menangkap logika kondisional dalam smart contract.

Kata kunci: *Metrik Cakupan; Model Bahasa Besar; Rekayasa Prompt; Smart Contract; Pengujian Unit*

1. Pendahuluan

Perkembangan teknologi informasi di era Revolusi Industri 4.0 berlangsung sangat pesat dan memberikan dampak signifikan terhadap berbagai sektor, tidak hanya industri, tetapi juga bidang ekonomi, keuangan, dan layanan digital [1]. Transformasi ini didorong oleh berbagai teknologi baru, salah satunya adalah blockchain dan smart contract, yang telah membawa perubahan mendasar terutama dalam sektor Keuangan Terdesentralisasi (*Decentralized Finance*/DeFi). Nilai pasar DeFi bahkan pernah melampaui 180 miliar dolar AS, menunjukkan besarnya pengaruh teknologi ini dalam ekosistem finansial modern [2].

Blockchain merupakan sistem penyimpanan data terdistribusi berbentuk buku besar (*ledger*) yang direplikasi pada banyak node dalam jaringan tanpa adanya otoritas pusat. Setiap transaksi diverifikasi melalui mekanisme konsensus, sehingga data menjadi transparan, aman, dan tahan terhadap manipulasi [3]. Di atas teknologi blockchain, smart contract berperan sebagai program yang secara otomatis mengeksekusi logika tertentu ketika kondisi yang telah ditentukan terpenuhi, tanpa memerlukan pihak perantara.

Smart contract berfungsi sebagai fondasi utama berbagai aplikasi terdesentralisasi (*decentralized applications*/*dApps*) dan disimpan secara permanen di seluruh node jaringan blockchain [4]. Karakteristik utama yang diwarisinya adalah *immutability* [5] dan transparansi, yang memungkinkan siapa pun di jaringan melihat aturan dan riwayat transaksi kontrak [6]. Kedua sifat ini memungkinkan terciptanya sistem yang bersifat *self-executing* dan membangun kepercayaan antar pihak. Dalam pengujian, penggunaan Gherkin sebagai *domain-specific language* dalam pendekatan *Behavior-Driven Development* (BDD) sangat relevan karena mengadopsi struktur *Given–When–Then* yang mudah dipahami sekaligus dapat dieksekusi [7].

Salah satu karakteristik utama smart contract adalah *immutability*, yaitu sifat kode yang tidak dapat diubah setelah di-*deploy* ke jaringan blockchain [5]. Setelah kontrak dipublikasikan, seluruh logika di dalamnya menjadi permanen, yang pada satu sisi memberikan jaminan stabilitas, keandalan, dan keamanan tanpa ketergantungan pihak ketiga [8]. Namun di sisi lain, sifat ini menimbulkan risiko besar apabila terdapat kesalahan logika atau celah keamanan pada saat pengembangan. Beberapa kasus menunjukkan dampak fatal dari kurang optimalnya pengujian smart contract. Peretasan The DAO pada tahun 2016 menyebabkan kerugian sebesar 3,6 juta Ether akibat adanya kerentanan *reentrancy* yang seharusnya dapat dideteksi lebih awal melalui pengujian yang memadai [9]. Insiden serupa juga terjadi pada Wormhole Bridge pada Februari 2022, di mana lebih dari 120.000 wETH senilai lebih dari 320 juta dolar AS berhasil dicuri [10]. Peristiwa-peristiwa tersebut menegaskan bahwa pengujian smart contract merupakan aspek krusial dalam menjaga keamanan dan stabilitas sistem berbasis blockchain.

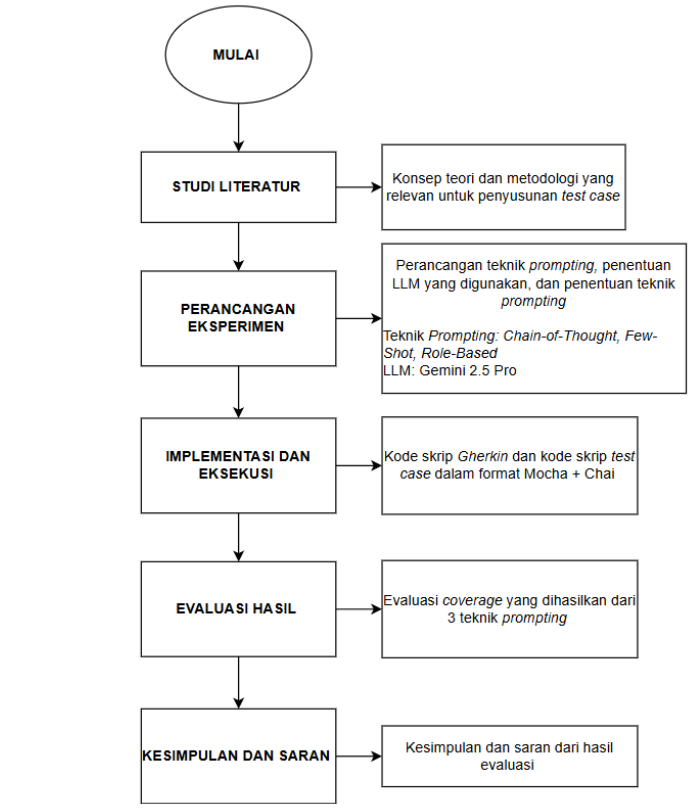
Dalam rekayasa perangkat lunak, pengujian, khususnya *unit testing*, merupakan tahapan penting untuk memastikan setiap komponen sistem berfungsi sesuai dengan spesifikasi yang ditentukan [11].

Salah satu pendekatan yang mulai dikembangkan untuk meningkatkan efisiensi pengujian adalah pemanfaatan kecerdasan buatan, khususnya *Large Language Model* (LLM). LLM merupakan model kecerdasan buatan berskala besar yang terbukti mampu mendukung pembuatan kode, deteksi *bug*, serta penyusunan skenario pengujian secara otomatis [12]. Untuk mengoptimalkan hasil yang dihasilkan LLM, digunakan teknik *prompt engineering*, yaitu perancangan instruksi yang strategis agar model menghasilkan respons yang relevan, akurat, dan kontekstual [13]. Kualitas *prompt* berpengaruh langsung terhadap kedalaman penalaran dan relevansi *output* yang dihasilkan [14]. Dalam penelitian ini, digunakan tiga metode utama, yaitu *Chain-of-Thought*, *Few-Shot*, dan *Role-Based prompting*. *Chain-of-Thought* mendorong model untuk berpikir secara bertahap sehingga meningkatkan performa dalam menyelesaikan masalah kompleks [15], [16]. *Few-Shot prompting* menyediakan beberapa contoh singkat untuk membantu model memahami pola dan konteks tugas [15], [16]. Sementara itu, *Role-Based prompting* menetapkan peran tertentu kepada model, seperti QA *engineer* atau blockchain *developer*, sehingga skenario pengujian yang dihasilkan menjadi lebih terarah, kontekstual, dan presisi [17], [18].

Sejumlah penelitian telah mengeksplorasi penggunaan LLM dalam pembuatan skenario uji otomatis. Lahbib et al. (2024) mengusulkan pendekatan berbasis LLM dan *prompt engineering* dengan memanfaatkan representasi *use case* dalam format XML untuk menghasilkan *test case* secara otomatis melalui model seperti ChatGPT dan Gemini [12]. Penelitian lain oleh Margarida et al. (2025) mengembangkan alur otomatisasi *acceptance testing* dengan mengonversi *user story* menjadi skenario Gherkin dan selanjutnya ke dalam skrip pengujian menggunakan Cypress [19]. Sementara itu, Zain et al. (2025) memperkenalkan framework E2E-TestGen yang memanfaatkan LLM untuk menghasilkan pengujian end-to-end adaptif dengan bantuan *object detection*, OCR, dan representasi *Hybrid DOM Dataset* (HDD) [20]. Meskipun hasilnya menjanjikan, penelitian-penelitian tersebut belum secara khusus mengkaji efektivitas teknik *prompting* pada konteks pengujian smart contract, terutama dari segi kualitas dan cakupan skenario uji yang dihasilkan.

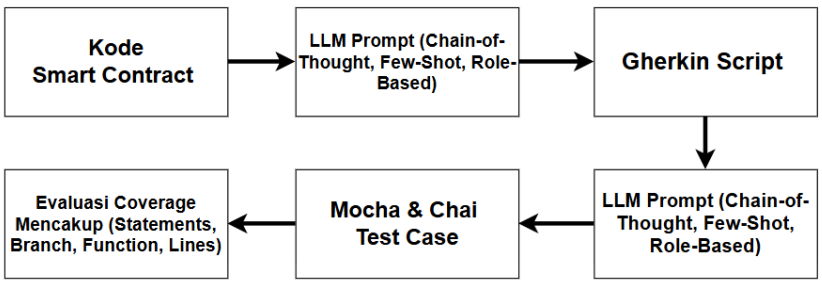
Berdasarkan celah penelitian yang telah diidentifikasi, studi ini berfokus pada pengembangan metode otomatisasi pembuatan *test scenario* berbasis *Large Language Model* (LLM) untuk pengujian smart contract, serta melakukan analisis komparatif terhadap efektivitas berbagai teknik *prompting* yang digunakan. Permasalahan utama yang dirumuskan dalam penelitian ini meliputi: (1) bagaimana merancang metode otomatisasi pembuatan *test scenario* berbasis LLM yang mampu menghasilkan skenario uji yang relevan dan komprehensif untuk pengujian smart contract; serta (2) sejauh mana perbedaan teknik *prompting* memengaruhi kualitas dan efektivitas skenario pengujian yang dihasilkan. Adapun tujuan utama penelitian ini adalah menghasilkan metode pengujian yang lebih efisien, adaptif, dan sesuai dengan karakteristik sistem blockchain yang bersifat terdesentralisasi dan tidak dapat diubah (*immutable*). Penelitian ini memberikan kontribusi metodologis berupa kerangka baru dalam pengujian smart contract berbasis LLM, sekaligus menyajikan evaluasi empiris mengenai pengaruh teknik *prompting* terhadap kualitas *test scenario*. Hasil studi ini diharapkan dapat menjadi landasan bagi pengembangan metode pengujian otomatis yang lebih tangguh di lingkungan blockchain pada masa mendatang.

2. Bahan dan Metode



Gambar 1. Diagram Alir Penelitian

Metodologi penelitian ini terdiri atas lima tahapan utama. Tahap pertama adalah studi literatur, yang dilakukan untuk menghimpun dan menelaah konsep-konsep terkait smart contract, *unit testing*, serta pendekatan otomatisasi yang telah dikembangkan sebelumnya guna mengidentifikasi celah penelitian. Tahap kedua adalah perancangan eksperimen, yang mencakup penyusunan kerangka kerja eksperimental, penetapan objek uji, pemilihan teknik prompting dan LLM, serta perumusan alur konversi dan variabel penelitian. Tahap ketiga merupakan implementasi dan eksekusi, yaitu penerapan LLM untuk menghasilkan artefak pengujian secara otomatis dalam bentuk skrip Mocha dan Chai. Tahap keempat adalah evaluasi hasil, yang dilakukan dengan mengeksekusi seluruh skrip pengujian dan menganalisis nilai cakupan kode guna membandingkan efektivitas masing-masing teknik prompting. Tahap terakhir adalah penarikan kesimpulan dan penyusunan saran berdasarkan temuan penelitian untuk menjawab rumusan masalah dan memberikan arah pengembangan selanjutnya.



Gambar 2. Diagram Alir Eksperimen

Alur penelitian ini dirancang melalui tiga tahapan inti yang berorientasi pada transformasi bertingkat dan evaluasi kuantitatif. Tahap pertama diawali dengan pemanfaatan kode smart contract sebagai objek kajian utama yang dianalisis oleh *Large Language Model Gemini* 2.5 Pro. Pada tahap ini, terdapat tiga teknik prompting yang digunakan, yaitu *Chain-of-Thought, Few-Shot*, dan *Role-Based* yang diterapkan secara terpisah untuk mengekstraksi struktur logika dan fitur fungsional smart contract. Hasil dari proses tersebut berupa Gherkin *script*, yakni representasi formal yang mengadopsi pendekatan *Behavior-Driven Development* dan berfungsi sebagai perantara yang memperjelas perilaku sistem melalui skenario yang terstandarisasi dan dapat ditelusuri.

Tahap kedua melanjutkan proses konversi dengan menjadikan Gherkin Script sebagai masukan yang kembali diproses menggunakan Gemini 2.5 Pro dengan ketiga teknik *prompting* yang sama. Tujuan tahap ini adalah mentransformasikan skenario berbasis bahasa natural tersebut menjadi Mocha & Chai test case yang kompatibel dengan mekanisme eksekusi pada framework Hardhat. Tahap konversi ini tidak hanya memetakan spesifikasi perilaku menjadi skrip pengujian teknis yang operasional, tetapi juga memungkinkan penilaian komparatif terhadap konsistensi dan ketelitian masing-masing teknik prompting dalam memahami konteks pengujian pada dua tingkat abstraksi yang berbeda.

Tahap terakhir merupakan proses evaluasi kuantitatif yang ditujukan untuk menilai efektivitas test case yang dihasilkan oleh setiap teknik *prompting*. Seluruh skrip pengujian dijalankan terhadap smart contract yang sama guna menjaga kesetaraan kondisi pengujian dan meminimalkan bias. Analisis kemudian dilakukan menggunakan empat metrik *coverage*, yaitu *Statements, Branches, Functions*, dan *Lines* yang secara kolektif memberikan gambaran komprehensif mengenai tingkat kecakupan eksekusi kode. Perbandingan hasil pada setiap metrik tersebut selanjutnya menjadi dasar penilaian terhadap kemampuan masing-masing teknik *prompting* dalam menghasilkan skenario pengujian yang representatif dan berkualitas tinggi.

3. Hasil

Penelitian ini menggunakan metode eksperimen kuantitatif untuk mengevaluasi efektivitas komparatif tiga teknik prompt engineering, yaitu *Chain-of-Thought, Few-Shot*, dan *Role-Based*. Proses penelitian menerapkan alur kerja otomasi pembuatan *unit testing* dalam dua tahap, yang terdiri atas konversi kode smart contract menjadi skrip Gherkin, kemudian dilanjutkan dengan konversi skrip tersebut menjadi kasus uji (*test case*) Mocha & Chai.

Tolak ukur keberhasilan dievaluasi secara kuantitatif menggunakan metrik *coverage*, yang mengukur persentase cakupan pada empat kriteria utama: *Statements, Branches, Functions,* dan *Lines*. LLM Gemini 2.5 Pro digunakan untuk mengkonversi Kode Smart Contract menjadi Gherkin Script, yang kemudian dikonversi lagi menjadi Mocha & Chai *test case*.

3.1. Few-Shot Prompting

Teknik *Few-Shot Prompting* dalam penelitian ini dirancang untuk memandu LLM menghasilkan *unit test* yang fungsional melalui proses konversi dua tahap yang masing-masing disertai contoh (*few-shot examples*). Pada tahap pertama, model diberikan contoh konversi fungsi Solidity menjadi skenario Gherkin berformat *Given-When-Then*, sehingga LLM dapat memahami pola pemetaan logika kontrak ke dalam skenario pengujian yang terstandar. Pada tahap kedua, model memperoleh contoh kode *unit test* Hardhat yang memperlihatkan penggunaan struktur *describe* dan *it*, mekanisme *async–await*, serta *assertion* Mocha & Chai. Kedua contoh ini berfungsi sebagai pola dasar yang menuntun model

dalam menghasilkan artefak pengujian dengan struktur, sintaks, dan gaya yang konsisten. Dengan acuan eksplisit pada setiap tahap, rancangan *few-shot prompt* ini memastikan bahwa keluaran LLM tidak hanya selaras dengan format yang diharapkan, tetapi juga siap diimplementasikan dalam proses pengujian otomatis.

Prompt 1. Konversi Smart Contract menjadi Gherkin

```
Berikut contoh cara mengonversi fungsi Solidity menjadi skenario
Gherkin:

[TEMPEL POTONGAN KODE SKENARIO GHERKIN DISINI]

Sekarang ubah kode smart contract berikut menjadi skrip Gherkin
dengan gaya serupa untuk kontrak ini:
[TEMPEL KODE SMART CONTRACT DISINI]
```

Prompt 2. Konversi Gherkin menjadi *Test Case* Hardhat

```
Berikut adalah contoh potongan kode program hardhat
[TEMPEL POTONGAN KODE HARDHAT DISINI]

Sekarang ubah skrip Gherkin berikut menjadi kode test Hardhat
dengan gaya yang sama untuk Gherkin ini:

[TEMPEL KODE SKENARIO GHERKIN DISINI]
```

3.2. Role-Based Prompting

Teknik *Role-Based Prompting* pada penelitian ini dirancang dengan menetapkan LLM sebagai *QA Engineer* Blockchain profesional untuk memastikan keluaran selaras dengan praktik pengujian yang sesuai standar industri. Pada tahap pertama, penetapan peran ini memandu model dalam mengonversi smart contract menjadi skenario Gherkin yang sistematis melalui struktur *Given When Then*. Pada tahap berikutnya, peran yang sama digunakan untuk menerjemahkan skenario Gherkin tersebut menjadi unit test Hardhat dengan penggunaan struktur describe, it, mekanisme *async–await*, dan *assertion* Mocha & Chai. Penetapan peran yang eksplisit ini membantu meningkatkan konsistensi, koherensi, dan ketepatan hasil generasi karena LLM beroperasi dalam kerangka identitas profesional yang jelas.

Prompt 3. Konversi Smart Contract menjadi Gherkin

```
Anda adalah seorang QA Engineer Blockchain profesional.
Tugas anda adalah membuat skenario pengujian dalam format Gherkin
dari smart contract berikut.

Smart Contract:
[TEMPEL KODE SMART CONTRACT DI SINI]
```

Prompt 4. Konversi Gherkin menjadi *Test Case* Hardhat

```
Anda adalah QA Engineer Blockchain profesional yang akan mener-
jemahkan skenario Gherkin ke dalam kode pengujian Hardhat.
Berikut ini adalah skenario gherkin yang harus kamu ubah menjadi
test case hardhat :
Gherkin
[TEMPEL KODE GHERKIN DI SINI]
```

3.3. Chain-of-Thought Prompting

Rancangan *Chain-of-Thought* (CoT) *Prompting* pada penelitian ini disusun untuk memandu LLM melakukan penalaran bertahap dalam menghasilkan unit test yang dapat diimplementasikan. Pada tahap awal, *prompt* mengarahkan model untuk mengidentifikasi fungsi dan event penting dalam smart contract, kemudian menyusunnya menjadi skenario Gherkin yang merepresentasikan alur *Given When Then*. Selanjutnya, *prompt* memberikan panduan langkah demi langkah mengenai cara menerjemahkan setiap bagian Gherkin menjadi kode pengujian Hardhat, yaitu *Given* sebagai tahap inisialisasi, *When* sebagai aksi transaksi, dan *Then* sebagai verifikasi menggunakan *assertion*. Dengan struktur instruksi yang berurutan, teknik CoT ini membantu LLM menghasilkan *test case* yang lebih logis, konsisten, dan selaras dengan praktik pengujian perangkat lunak, sekaligus meminimalkan hilangnya detail penting dalam proses konversi otomatis.

Prompt 5. Konversi Smart Contract menjadi Gherkin

```
Analisis smart contract berikut secara bertahap:
1  Identifikasi fungsi utama dan event penting.
2  Tentukan precondition (Diberikan), action (Ketika), dan hasil
   (Maka)
3  Buat skenario Gherkin untuk setiap fungsi yang ada di kode
   program smart contract

Smart Contract:
[TEMPEL KODE SMART CONTRACT DI SINI]
```

Prompt 6. Konversi Gherkin menjadi *Test Case* Hardhat

```
Pikirkan langkah demi langkah bagaimana setiap bagian dari
Gherkin diterjemahkan ke dalam kode test Hardhat:
- Given → setup awal
- When → aksi transaksi
- Then → verifikasi hasil dengan expect()

Baca dan analisa kode gherkin, untuk setiap skenario buatkan test case
hardhatnya. Sekarang ubah Gherkin berikut menjadi test Hardhat yang
lengkap.
Skenario Gherkin:
```

```
[TEMPEL SKENARIO GHERKIN DI SINI]
```

3.4. Pengujian

Pengujian dilakukan dengan menjalankan seluruh skrip pengujian yang dihasilkan dari tiga teknik prompting pada tahapan sebelumnya. Proses eksekusi pengujian dilakukan menggunakan perintah **"hardhat coverage --testfiles 'test/fewshot.js'"** untuk memperoleh data cakupan pengujian. Melalui perintah tersebut, sistem akan menghasilkan analisis terhadap tingkat *statement coverage*, *branch coverage*, *function coverage*, serta *line coverage*. Hasil analisis cakupan ini kemudian digunakan untuk menilai sejauh mana skrip pengujian telah mampu mengevaluasi fungsionalitas kontrak pintar secara menyeluruh.

4. Pembahasan

Tabel 1. Hasil Pengujian

Skor Coverage	Teknik Prompting		
	Role-Based	***Few-Shot***	***Chain-of-Thought***
Statement	95.83	95.83	87.5
Branch	90	80	70
Functions	85.71	85.71	71.43
Lines	96.55	96.55	86.21
Rata-rata	**92.02**	**89.52**	**78.79**

Berdasarkan data hasil pengujian pada Tabel 1 di atas, dapat disimpulkan bahwa *Role-Based prompting* memberikan performa terbaik dengan rata-rata *coverage* mencapai 92.02%, diikuti oleh *Few-Shot Prompting* dengan nilai 89.52%, sedangkan Chain-of-Thought memperoleh rata-rata paling rendah yaitu 78.79%. Temuan ini menegaskan bahwa penelitian berhasil mengidentifikasi teknik prompting yang paling efektif dalam menghasilkan skrip pengujian otomatis berbasis LLM. Secara lebih rinci, *Role-Based Prompting* terbukti menunjukkan kemampuannya dalam menangkap struktur logika secara lebih komprehensif.

Sementara itu, *Few-Shot Prompting* memberikan kinerja yang baik pada kategori *Statement, Functions*, dan *Lines*, tetapi masih menunjukkan keterbatasan pada *Branch coverage* yang hanya mencapai 80%. Adapun *Chain-of-Thought* konsisten menghasilkan skor paling rendah pada seluruh metrik, sehingga kurang optimal untuk menghasilkan unit test yang lengkap dan implementatif. Secara keseluruhan, temuan ini menegaskan bahwa pendekatan berbasis peran lebih efektif dalam memandu LLM menghasilkan artefak pengujian yang berkualitas dan berorientasi implementasi.

5. Kesimpulan

Berdasarkan hasil eksperimen yang telah dilaksanakan, penelitian ini menyimpulkan bahwa teknik *prompt engineering* memberikan pengaruh yang signifikan terhadap kualitas *unit testing* smart contract yang dihasilkan secara otomatis. Dari tiga teknik yang diuji, *Role-Based Prompting* menunjukkan performa terbaik dengan rata-rata *coverage* sebesar 92.02%, diikuti oleh *Few-Shot Prompting* dengan 89.52%, sementara *Chain-of-Thought* menghasilkan rata-rata terendah yaitu 78.79%. Keunggulan *Role-Based Prompting* terutama terlihat pada capaian *Branch coverage* sebesar 90%, yang merupakan metrik penting dalam memverifikasi logika kondisional serta mengidentifikasi potensi *edge case* pada smart contract. Sementara itu, *Few-Shot Prompting* memberikan performa baik pada kategori *Statement, Functions,* dan *Lines*, tetapi tetap menunjukkan kelemahan pada *Branch coverage* yang hanya mencapai 80%.

Temuan ini mengindikasikan bahwa pemberian konteks peran atau persona kepada LLM lebih efektif dibandingkan hanya memberikan contoh (*Few-Shot*) dalam menghasilkan pengujian yang komprehensif, stabil, dan mampu menangkap potensi *bug* dengan lebih andal. Penelitian ini memiliki keterbatasan pada generalisasi hasil, karena seluruh eksperimen dilakukan menggunakan satu model LLM yaitu Gemini 2.5 Pro dan evaluasi hanya mengandalkan metrik *coverage* yang tidak secara langsung mengukur kemampuan deteksi *bug*.

Penelitian lanjutan disarankan untuk menguji berbagai model LLM lainnya, menerapkan metrik evaluasi yang lebih kuat seperti *Mutation Testing*, serta mengeksplorasi teknik *prompting* hibrida yang menggabungkan keunggulan *Role-Based* dan *Few-Shot* guna mencapai hasil yang lebih optimal.

Referensi

[1] L. Hertati and O. Safkaur, "Dampak Revolusi Industri 4.0 Era Covid-19 pada Sistem Informasi Akuntansi Terhadap Struktur Modal Perusahaan," *J. Ris. Akunt. dan Keuang.*, vol. 8, no. 3, pp. 503–518, 2020, https://doi.org/10.17509/jrak.v8i3.23557.

[2] DeFiLlama, "DefiLlama - DeFi Dashboard," 2024. [Online]. Available: https://defillama.com/. [Accessed: Oct. 2, 2025]

[3] E. Barceló, K. Dimić-Mišić, M. Imani, V. Spasojević Brkić, M. Hummel, and P. Gane, "Regulatory Paradigm and Challenge for Blockchain Integration of Decentralized Systems: Example—Renewable Energy Grids," *Sustain.*, vol. 15, no. 3, 2023, https://doi.org/10.3390/su15032571

[4] A. Mattew and M. Anno Suwarno, "Rancang Bangun Aplikasi Donasi Terdesentralisasi Berbasis Blockchain," *Ikraith-Informatika*, vol. 7, no. 2, pp. 23–32, 2022, https://doi.org/10.37817/ikraith-informatika.v7i2.2247.

[5] I. Qasse, I. M. Ali, N. Ahmed, M. Hamdaqa, and B. Þ. Jónsson, "The Myth of Immutability: A Multivocal Review on Smart Contract Upgradeability," 2025, https://doi.org/10.48550/arXiv.2504.02719

[6] S. A. Latifa Albshaier and M. M. H. Rahman, "A_Review_of_Blockchains_Role_in_E-Commerce_Transa.pdf," *Mdpi*, 2024, https://doi.org/10.3390/computers13010027..

[7] "Reference | Cucumber." [Online]. Available: https://cucumber.io/docs/gherkin/reference/. [Accessed: Oct. 25, 2025]

[8] C. Zhang, Q. Wei, and X. Li, "Security Analysis of Ponzi Schemes in Ethereum Smart Contracts," pp. 1–31, 2025, https://doi.org/10.48550/arXiv.2510.03819

[9] D. Siegel, "Understanding The DAO Attack," 2016. http://coindesk.com/learn/understanding-the-dao-attack [Accessed: Oct. 10, 2025].

[10] Merkle Science, "Hack Track: Analysis of Wormhole Token Bridge Exploit," Merkle Science, Feb. 4, 2022. https://www.merklescience.com/blog/hack-track-analysis-of-wormhole token-bridge-exploit. [Accessed: Oct. 25, 2025]

[11] M. G. Alkhairi, S. P. A. Alkadri, and P. Y. Utami, "Implementasi Unit Testing Dan End-To-End Testing Pada Sistem Informasi Akademik Teknik Informatika," *JIPI (Jurnal Ilm. Penelit. dan Pembelajaran Inform.*, vol. 9, no. 4, pp. 2208–2219, 2024, https://doi.org/10.29100/jipi.v9i4.5626.

[12] L. Naimi, E. M. Bouziane, M. Manaouch, and A. Jakimi, "A new approach for automatic test case generation from use case diagram using LLMs and prompt engineering," *2024 Int. Conf. Circuit, Syst. Commun. ICCSC 2024*, 2024, https://doi.org/10.1109/ICCSC62074.2024.10616548.

[13] A. M. Rincon, A. M. R. Vincenzi, and J. P. Faria, "LLM Prompt Engineering for Automated White-Box Integration Test Generation in REST APIs," *2025 IEEE Int. Conf. Softw. Testing, Verif. Valid. Work. ICSTW 2025*, pp. 21–28, 2025, https://doi.org/10.1109/ICSTW64639.2025.10962507.

[14] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J. & Schmidt, D. C., "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT," arXiv preprint arXiv:2302.11382, Feb. 2023. https://doi.org/10.48550/arXiv.2302.11382.

[15] A. Aqsa, A. Aslam, and M. Saeed, "Efficient Prompt Engineering: Techniques and Trends for Maximizing LLM Output," no. May, 2025, https://doi.org/10.5281/zenodo.15186123.

[16] J. Almeida, "Prompt Engineering: A Comparative Study of Prompting Techniques in AI Language Models," *2025 15th IEEE Integr. STEM Educ. Conf. ISEC 2025*, pp. 1–4, 2025, https://doi.org/10.1109/ISEC64801.2025.11147384.

[17] A. Njifenjou, V. Sucal, B. Jabaian, and F. Lefèvre, "Role-Play Zero-Shot Prompting with Large Language Models for Open-Domain Human-Machine Conversation," arXiv preprint arXiv:2406.18460, 2024, https://doi.org/10.48550/arXiv.2406.18460.

[18] L. Mitchell, "Prompt engineering for LLMs," Gravitee, Jun. 16, 2023. https://www.gravitee.io/blog/prompt-engineering-for llms. [Accessed: Nov. 4, 2025]

[19] M. Ferreira, L. Viegas, J. P. Faria, and B. Lima, "Acceptance Test Generation with Large Language Models: An Industrial Case Study," pp. 1–11, 2025, https://doi.org/10.1109/ast66626.2025.00007.

[20] Z. Ul Abideen and G. Junxia, "Intent Based E2E Automated Test Case Generation for Web Applications Using LLM," *Proc. - 2025 IEEE 49th Annu. Comput. Software, Appl. Conf. COMPSAC 2025*, vol. 2, pp. 1281–1290, 2025, https://doi.org/10.1109/COMPSAC65507.2025.00161.