TUGAS AKHIR

RANCANG BANGUN EXTENSION CHROME UNTUK GOLANG UNIT TEST GENERATOR DENGAN MODEL BAHASA GENERATIVE AI

Sebagai Memenuhi Persyaratan Mencapai Derajat Sarjana (S1)

DISUSUN OLEH:

RIZAL DARUSMAN

NIM.21106050080

PROGRAM STUDI INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS ISLAM NEGERI SUNAN KALIJAGA

YOGYAKARTA

2025

PENGESAHAN TUGAS AKHIR

KEMENTERIAN AGAMA
UNIVERSITAS ISLAM NEGERI SUNAN KALIJAGA
FAKULTAS SAINS DAN TEKNOLOGI
Jl. Marsda Adisucipto Telp. (0274) 540971 Fax. (0274) 519739 Yogyakarta 55281

PENGESAHAN TUGAS AKHIR

Nomor : B-2500/Un.02/DST/PP.00.9/12/2025

Tugas Akhir dengan judul : Rancang Bangun Extension Chrome untuk Golang Unit Test Generator dengan Model Bahasa Generative AI

yang dipersiapkan dan disusun oleh:

Nama : RIZAL DARUSMAN
Nomor Induk Mahasiswa : 21106050080
Telah diujikan pada : Selasa, 09 Desember 2025
Nilai ujian Tugas Akhir : A

dinyatakan telah diterima oleh Fakultas Sains dan Teknologi UIN Sunan Kalijaga Yogyakarta

TIM UJIAN TUGAS AKHIR

Ketua Sidang

Ir. Muhammad Didik Rohmad Wahyudi, S.T., MT.
SIGNED

Valid ID: 6943f00e9b58c

Penguji I

Ir. Maria Ulfah Siregar, S.Kom., MIT., Ph.D.
SIGNED

Valid ID: 6943fc9c9a6c2

Penguji II

Nurochman, S.Kom., M.Kom
SIGNED

Valid ID: 6942457dd2a6

Yogyakarta, 09 Desember 2025
UIN Sunan Kalijaga
Dekan Fakultas Sains dan Teknologi

Prof. Dr. Dra. Hj. Khurul Wardati, M.Si.
SIGNED

Valid ID: 6943afb9e7d06

SURAT PERSETUJUAN TUGAS AKHIR

Universitas Islam Negeri Sunan Kalijaga FM-UINSK-BM-05-03/R0

SURAT PERSETUJUAN SKRIPSI/TUGAS AKHIR

Hal : Persetujuan Skripsi / Tugas Akhir
Lamp : -

Kepada
Yth. Dekan Fakultas Sains dan Teknologi
UIN Sunan Kalijaga Yogyakarta
Di Yogyakarta

Assalammu'alaikum wr. wb.

Setelah membaca, meneliti, memberikan petunjuk dan mengoreksi serta mengadakan perbaikan seperlunya, maka saya selaku pembimbing berpendapat bahwa skripsi Saudara:

Nama	:	Rizal Darusman
NIM	:	21106050080
Judul Skripsi	:	Rancang Bangun Extension Chrome untuk Golang Unit Test Generator dengan Model Bahasa Generative AI

sudah dapat diajukan kembali kepada Program Studi Informatika Fakultas Sains dan Teknologi UIN Sunan Kalijaga Yogyakarta sebagai salah satu syarat untuk memperoleh gelar Sarjana Strata Satu dalam Program Studi Informatika.

Dengan ini kami mengharap agar skripsi/tugas akhir Saudari dapat segera dimunaqasyahkan. Atas perhatiannya saya ucapkan terima kasih.

Wassalammu'alaikum wr. wb.

Yogyakarta, 01 Desember 2025
Pembimbing

Muhammad Didik Rohmad Wahyudi, S.T., M.T.
NIP. 19841115 201903 1 003

SURAT PERNYATAAN KEASLIAN TUGAS AKHIR

SURAT PERNYATAAN KEASLIAN/BEBAS PLAGIASI

Yang bertanda tangan dibawah ini:

Nama	: Rizal Darusman
NIM	: 21106050080
Program Studi	: Informatika
Fakultas	: Sains dan Teknologi

Dengan ini menyatakan bahwa isi tugas akhir saya yang berjudul "Rancang Bangun Extension Chrome untuk Golang Unit Test Generator dengan Model Bahasa Generative AI" merupakan hasil pemikiran penulisan sendiri, tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan orang lain sepengetahuan penulis, kecuali yang tertulis dalam tugas akhir dan disebutkan dalam daftar Pustaka.

Yogyakarta, 02 Desember 2025

Rizal Darusman

NIM. 21106050080

ABSTRAK

Penulisan *unit test* secara manual dalam ekosistem Golang seringkali kompleks dan menyita waktu, yang menghambat efisiensi penerapan *Test-Driven Development*. Tantangan ini diperberat oleh keterbatasan sumber daya komputasi jika menggunakan model AI secara lokal. Penelitian ini bertujuan membangun otomatisasi *unit test* melalui ekstensi Chrome yang terintegrasi dengan layanan API GroqCloud untuk efisiensi komputasi. Metode pengembangan menggunakan *Extreme Programming* dengan arsitektur sistem berbasis Manifest V3 dan beberapa model Generative AI seperti contoh Mistral-saba-24b. Hasil evaluasi bahwa penggunaan *format prompting* mampu meningkatkan performa model secara signifikan dibandingkan tanpa pengondisian *prompt.* Model Mistral-saba-24b memperoleh nilai BLEU sebesar 0,6146, ROUGE-1 sebesar 0,7267, ROUGE-2 sebesar 0,6674, dan ROUGE-L sebesar 0,7014, yang mengindikasikan kemampuan model dalam menghasilkan unit test dengan tingkat kemiripan leksikal dan struktural yang tinggi terhadap unit test referensi. Meskipun nilai Exact Match masih berada pada 0%, hasil tersebut menunjukkan bahwa output yang dihasilkan telah mendekati struktur dan konteks unit test yang diharapkan. Evaluasi usabilitas menggunakan *System Usability Scale* (SUS) terhadap 45 responden menghasilkan skor 65,12 yang tergolong *Marginally Acceptable*. Hal ini mengindikasikan bahwa sistem secara fungsional dapat diterima pengguna, namun memerlukan peningkatan pengalaman pengguna untuk mencapai tingkat penerimaan yang optimal.

Kata Kunci: Generative AI, Golang, Unit Test Generator, Chrome Extension, GroqCloud

ABSTRACT

Manual unit test creation within the Golang ecosystem is often complex and time-consuming, hindering the efficiency of Test-Driven Development implementation. This challenge is further exacerbated by computational resource constraints when deploying AI models locally. This study aims to develop unit test automation through a Chrome extension integrated with the GroqCloud API to ensure computational efficiency. The development methodology employs Extreme Programming, utilizing a Manifest V3-based system architecture and several Generative AI models, such as Mistral-saba-24b. The evaluation results indicate that the use of structured prompting significantly improves model performance compared to approaches without prompt conditioning. The Mistral-saba-24b model achieved a BLEU score of 0.6146, ROUGE-1 of 0.7267, ROUGE-2 of 0.6674, and ROUGE-L of 0.7014, indicating its capability to generate unit tests with a high degree of lexical and structural similarity to referenceunit tests. Although the Exact Match score remains at 0%, the generated outputs closely approximate the expected structure and context of unit tests.Usability evaluation using the System Usability Scale (SUS) involving 45 respondents yielded a score of 65.12, classified as "Marginally Acceptable." This indicates that while the system is functionally acceptable to users, it requires improvements user experience aspects to achieve an optimal level of acceptance.

Keywords: Generative AI, Golang, Unit Test Generator, Chrome Extension, GroqCloud.

KATA PENGANTAR

Puja dan puji syukur kehadirat Allah SWT yang telah memberikan rahmat dan hidayah-Nya sehingga penulis dapat memulai mengerjakan sampai menyelesaikan tugas akhir yang berjudul "Rancang Bangun Extension Chrome untuk Golang Unit Test Generator dengan Model Bahasa Generative AI." Sholawat serta salam senantiasa tercurahkan kepada baginda Nabi Muhammad SAW, yang telah memberikan syafaat serta teladan bagi umat islam dalam menuntut ilmu dan mengamalkannya.

Tersusunnya tugas akhir ini tidak lepas dari dukungan berbagai pihak yang telah memberikan semangat, bimbingan, dan kontribusi selama proses penyusunan. Oleh karena itu, dengan kerendahan hati, penulis menyampaikan ucapan terima kasih yang sebesar-besarnya kepada:

1. Bapak Prof. Noorhaidi Hasan, S.Ag., M.A., M.Phil., Ph.D., selaku Rektor Universitas Islam Negeri Sunan Kalijaga Yogyakarta.
2. Ibu Prof. Dr. Dra. Hj. Khurul Wardati, M.Si., selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Sunan Kalijaga Yogyakarta.
3. Bapak Muhammad Mustakim, S.T., M.T., selaku Ketua Program Studi Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Sunan Kalijaga Yogyakarta.
4. Bapak Ir. Muhammad Didik Rohmad Wahyudi, S.T., M.T., selaku Dosen Pembimbing Tugas Akhir yang telah meluangkan waktu untuk membantu dan mengarahkan selama proses penulisan tugas akhir.
5. Bapak Eko Hadi Gunawan, M.ENG., selaku Dosen Pembimbing Akademik yang telah membantu penulis selama perkuliahan.
6. Seluruh Dosen dan Karyawan Program Studi Informatika UIN Sunan Kalijaga yang telah memberikan ilmu dan bantuan selama perkuliahan.
7. Kedua orang tua penulis Bapak Sudiro dan Ibu Sri Astuti yang telah memberikan dukungan materi dan moral kepada penulis hingga dapat

menempuh pendidikan tinggi agar menjadi pribadi yang berkontribusi pada negara.

8. Seluruh teman Informatika yang membantu penulis selama perkuliahan terutama teman Informatika 2021 yang telah menjadi saksi perjuangan penulis pada saat perkuliahan.

Tugas akhir ini tidak terlepas dari kesalahan dan kekurangan kualitas materi. Untuk itu, mohon kritik dan saran yang membangun agar dapat membawa manfaat bagi pembaca. Semoga tugas akhir ini dapat memberikan pengetahuan yang baru dan dapat dijadikan referensi pembaca untuk kedepannya.

Yogyakarta, 28 Mei 2025

Penulis

Rizal Darusman

NIM.21106050080

DAFTAR ISI

HALAMAN JUDUL i
PENGESAHAN TUGAS AKHIR ii
SURAT PERSETUJUAN TUGAS AKHIR iii
SURAT PERNYATAAN KEASLIAN TUGAS AKHIR iv
ABSTRAK v
ABSTRACT vi
KATA PENGANTAR vii
DAFTAR ISI ix
DAFTAR GAMBAR xi
DAFTAR TABEL xii
BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 5
1.3 Tujuan 5
1.4 Manfaat 5
1.5 Batasan Masalah 6
BAB II KAJIAN PUSTAKA 7
2.1 Bahasa Pemrograman Golang 7
2.2 Google Chrome Extension 9
2.3 Unit Test 11
2.4 Test-Driven Development 12
2.5 Generative Artificial Intelligence 15
2.6 Groq AI 15
2.7 Metode Extreme Programming 19
BAB III METODE PENGEMBANGAN SISTEM 21
3.1 Alat dan Bahan 21
3.2 Metode Pengembangan 22
3.3 Tahapan Pengembangan 24
BAB IV PERANCANGAN DAN IMPLEMENTASI SISTEM 26
4.1. Iterasi Pengembangan Pertama 26
4.1.1. Perencanaan (*Planning*) Iterasi Pertama 26

4.1.2. Perancagan (*Design*) Iterasi Pertama 29
4.1.3. Implementasi Kode (*Coding*) Iterasi Pertama 38
4.1.4. Pengujian (*Testing*) Iterasi Pertama 45
4.1.5. Evaluasi *Acceptance Test Criteria* Iterasi Pertama 47
4.2. Iterasi Pengembangan Kedua 48
4.2.1. Perencanaan (*Planning*) Iterasi Kedua 48
4.2.2. Perancangan (*Design*) Iterasi Kedua 51
4.2.3. Implementasi Kode (*Coding*) Iterasi Kedua 55
4.2.4. Pengujian (*Testing*) Iterasi Kedua 60
4.2.5. Evaluasi *Acceptance Test Criteria* Iterasi Kedua 63
4.3. Iterasi Pengembangan Ketiga 64
4.3.1. Perencanaan (*Planning*) Iterasi Ketiga 64
4.3.2. Perancangan (*Design*) Iterasi Ketiga 67
4.3.3. Implementasi Kode (*Coding*) Iterasi Ketiga 72
4.3.4. Pengujian (*Testing*) Iterasi Ketiga 80
4.3.5. Evaluasi *Acceptance Test Criteria* Iterasi Kedua 81
4.4. Pengujian Praktik Test-Driven Development terhadap Output Model 83
4.5. Evaluasi 93
4.5.1. Evaluasi Model Bahasa berdasarkan Dataset 93
4.5.2. Evaluasi Hasil Kode Unit Test berdasarkan Dataset 99
4.5.3. Evaluasi Proyek Ekstensi Chrome Golang Unit Test Generator 102
BAB V KESIMPULAN DAN SARAN 112
5.1. Kesimpulan 112
5.2. Saran 112
DAFTAR PUSTAKA 114
LAMPIRAN 118
RIWAYAT HIDUP 119

DAFTAR GAMBAR

Gambar 2.1 Terminal Eksekusi TestHelloName.go 9
Gambar 2.2 Arsitektur Google Chrome Extension 10
Gambar 2.3 Tahapan Test-Driven Development 13
Gambar 2.4 Hasil Eksekusi unit test test.py 15
Gambar 2.5 Grafik Latency vs Throughput terhadap layanan inferensi model. 16
Gambar 2.6 Grafik Throughput terhadap layanan inferensi model 17
Gambar 2.7 Grafik Total Response Time terhadap layanan inferensi model 18
Gambar 2.8 Metode Extreme Programming 20
Gambar 4.1 Side Panel Feature 30
Gambar 4.2 Dashboard Settings 31
Gambar 4.3 Diagram Arsitektur Sistem 32
Gambar 4.4 Activity Diagram Upload File Go 34
Gambar 4.5 Activity Diagram Input Prompt 35
Gambar 4.6 Use Case Diagram Golang Unit Test Generator 37
Gambar 4.7 Manage Extension Chrome 42
Gambar 4.8 Halaman Chrome://extensions 42
Gambar 4.9 Deskripsi Golang Unit Test Generator Ekstensi Chrome 43
Gambar 4.10 UI Side Panel 43
Gambar 4.11 Refactoring UI Final 45
Gambar 4.12 Activity Diagram Output Prompt 52
Gambar 4.13 Activity Diagram Download Unit Test File 53
Gambar 4.14 Activity Diagram Selected Model 54
Gambar 4.15 UI untuk file-generator.js 55
Gambar 4.16 Light/Dark Mode 56
Gambar 4.17 Memilih Model Generative AI 57
Gambar 4. 18 Output model gemma-9b-it 58
Gambar 4.19 Output UI tag <think> 59
Gambar 4.20 Output UI Model Deepseek 59
Gambar 4.21 Activity Diagram Riwayat Chat 68
Gambar 4.22 Activity Diagram Update Profile User 69
Gambar 4.23 Activity Diagram Benchmark Unit Test Gen 70
Gambar 4.24 Activity Diagram Panduan Pengguna 71
Gambar 4.25 Activity Diagram Panduan TDD 72
Gambar 4.26 Dashboard Halaman Options 73
Gambar 4. 27 Chat History 74
Gambar 4. 28 Model Benchmark Unit Test 75
Gambar 4.29 Panduan Unit Testing 75
Gambar 4.30 Panduan TDD 76
Gambar 4.31 Panduan Pengguna 76
Gambar 4.32 Pengecekan Output Fungsi Mathbasic dengan Tipe Data Integer dan bukan Integer 86
Gambar 4.33 Pass by value dengan nilai dari tipe data selain integer 87
Gambar 4.34 Iterasi Pertama Unit Test Deepseek 89
Gambar 4.35 Iterasi Pertama Unit Test Mistral 90
Gambar 4.36 Iterasi Pertama Unit Test Llama 90
Gambar 4.37 Error di Iterasi Pertama pada Gemma 91

DAFTAR TABEL

Tabel 3.1 Spesifikasi Google Collab 21
Tabel 3.2 Tahapan Pengembangan Sistem 24
Tabel 4.1 Kebutuhan Fungsional Iterasi Pertama 27
Tabel 4.2 Kebutuhan Non Fungsional Iterasi Pertama 28
Tabel 4.3 Acceptance Test Criteria Iterasi Pertama 29
Tabel 4.4 Response Model Mixtral-8x-7b-32768 41
Tabel 4.5 Hasil Pengujian Black Box Iterasi Pertama 46
Tabel 4. 6 Kebutuhan Fungsional Iterasi Kedua 48
Tabel 4.7 Kebutuhan Non Fungsional Iterasi Kedua 49
Tabel 4.8 Acceptance Test Criteria Iterasi Kedua 50
Tabel 4.9 Hasil Pengujian Black Box Iterasi Kedua 61
Tabel 4.10 Kebutuhan Fungsional Iterasi Ketiga 65
Tabel 4.11 Kebutuhan Non Fungsional Iterasi Ketiga 66
Tabel 4.12 Acceptance Test Criteria Iterasi Ketiga 66
Tabel 4.13 Struktur file di folder options 77
Tabel 4. 14 Struktur Folder Options Hasil Refactoring 78
Tabel 4.15 Black Box Testing Iterasi Ketiga 81
Tabel 4.16 Format input untuk generate unit test 87
Tabel 4.17 Evaluasi Output Unit Test Terhadap Model dengan Praktik TDD 92
Tabel 4.18 Hasil Evaluasi Matrik Model Bahasa terhadap Dataset golang-test-from-stack 96
Tabel 4.19 Format Prompting 98
Tabel 4.20 Matriks Evaluasi dengan Format Prompting 98
Tabel 4.21 Evaluasi Hasil Unit Test 101
Tabel 4.22 Daftar Pernyataan System Usability Scale (SUS) 104
Tabel 4.23 Data Awal Hasil SUS dari Responden 105
Tabel 4.24 Perhitungan Skor System Usability Scale 107
Tabel 4.25 Skala Penilaian SUS Jeff Sauro 109

BAB I
PENDAHULUAN

1.1 Latar Belakang

Beberapa tahun terakhir, pendekatan dan metodologi dalam pengembangan perangkat lunak menghasilkan sistem yang andal, efisien, dan mudah dipelihara. Seiring meningkatnya kompleksitas sistem , kebutuhan pengguna dan tekanan waktu rilis fitur atau produk perangkat lunak, kualitas pengembangan perangkat lunak menjadi aspek yang krusial. Oleh karena itu, diperlukan praktik-praktik yang terbaik untuk menggunakan metode pengembangan perangkat lunak agar pengembang atau *developer* melakukan praktik secara konsisten untuk menjamin keandalan produk yang dihasilkan dari alur metode pengembangan perangkat lunak yang dipilih. Salah satu aspek penting dalam menjamin kualitas perangkat lunak adalah proses pengujian (*testing*), khususnya pada level *unit test*. *Unit test* tidak hanya membantu mendeteksi kesalahan sejak awal, tetapi juga memudahkan proses *refactoring* serta dokumentasi *code logic*.

Seiring waktu, muncul berbagai pendekatan pengujian otomatis yang terintegrasi langsung dalam alur pengembangan perangkat lunak. Dalam praktik pengembangan perangkat lunak modern, pendekatan *Test-Driven Development* menjadi metode yang efektif untuk meningkatkan kualitas kode yang dibuat dan meminimalisir kesalahan sejak tahap awal pengembangan. TDD mendorong pengembang untuk menulis *unit test* sebelum mengimplementasikan fungsi atau fitur yang akan dibuat pada logika kode program. *Unit test* berfungsi sebagai pedoman sekaligus jaminan bahwa setiap bagian dari kode berperilaku sesuai dengan harapan atau dokumentasi kode program atau fitur yang dibuat. Dengan mengimplementasikan praktik *Test-Driven Development* diharapkan sistem tidak memiliki *bug* atau *error* yang mempengaruhi kegagalan fungsi kode pemrograman dan kinerja perangkat lunak[1]. Selain itu, diharapkan proses melakukan *automating testing* dengan praktik tersebut berjalan dengan lancar sebelum melanjutkan pengujian integrasi komponen-komponen yang ada pada perangkat lunak[2].

Bahasa pemrograman Go atau Golang, yang dikembangkan oleh Google, telah menjadi salah satu bahasa pemrograman populer untuk pengembangan sistem *backend* berskala besar seperti membangun aplikasi *e-commerce*[3]. Tujuan dari proyek Golang yang dikembangkan oleh Google adalah untuk mengeliminasi dan rumitnya proses pengembangan perangkat lunak di Google, sehingga membuat proses tersebut menjadi lebih produktif dan *scalable.* Bahasa ini dirancang oleh dan untuk orang-orang yang menulis serta membaca, melakukan *debug* dan memelihara sistem perangkat lunak berskala besar. Oleh karena itu, tujuan utama Go bukanlah untuk melakukan penelitian dalam desain bahasa pemrograman, melainkan meningkatkan lingkungan kerja bagi para perancangnya dan rekan-rekan mereka.

Go lebih berfokus pada pemrograman untuk merekayasa perangkat lunak daripada bahasa untuk penelitian bahasa pemrograman [4]. Golang juga memiliki *library* atau pustaka bawaan untuk pengujian (*package testing*), sehingga mendukung praktik pengujian perangkat lunak secara native. Namun, dibalik kemudahannya, proses penulisan unit test secara manual tetap membutuhkan ketelitian tinggi dan seringkali menjadi beban tersendiri, terutama bagi pengembang yang masih mengenal ekosistem pemrograman Go atau sedang fokus pada pengembangan fitur yang harus diselesaikan juga. Apalagi dengan banyaknya kode pemrograman yang ditulis diikuti dengan kode *unit test* yang harus dibuat juga. Tantangan tersebut terjadi pada pendekatan *Test-Driven Development* yang mengharuskan pengujian dilakukan sebelum implementasi [5] dan semakin dirasakan dalam bahasa Go yang walaupun memiliki testing bawaan, namun menuntut eksplisitasi kode pengujian yang signifikan.

Dengan pesatnya teknologi kecerdasan buatan , khususnya dalam bidang *Natural Language Processing* (NLP)[6], munculah pendekatan yang dikenal sebagai *Generative Language Models* dan *Large Language Models* , seperti ChatGPT dengan berbagai LLM atau GLM yang dipakai seperti o1-mini, GPT-4, dan model sejenis lainnya. Model-model ini mampu memahami instruksi dari dalam bahasa alami atau input menghasilkan teks atau kode program yang sesuai dengan konteks. Salah satu yang paling menonjol dengan pemanfaatan model bahasa ini pada proses *code generation*[7], yaitu kemampuan menghasilkan kode

sumber dari input berbasis teks. Kemampuan ini tidak hanya pada penulisan kode program, tetapi juga dapat diterapkan dalam pembuatan *unit test* secara otomatis. Dengan memanfaatkan model bahasa *generative*, proses pembuatan *unit test* dapat diotomatisasi. Model seperti yang dimiliki oleh ChatGPT mampu membaca dan memahami logika fungsi dalam kode, kemudian menghasilkan kode dari skenario pengujian yang dibuat oleh *developer*. Hal ini membuka peluang besar dalam mempercepat siklus pengembangan perangkat lunak serta mengurangi waktu untuk melakukan pengujian manual.

Penerapan model bahasa ini semakin potensial apabila diintegrasikan langsung dengan perangkat lunak yang membutuhkan model bahasa sebagai output dari permintaan input teks dari aplikasi khususnya pada lingkungan aplikasi peramban internet seperti Chrome. Dengan memanfaatkan Chrome untuk membuat aplikasi berupa Ekstensi bisa menerapkan model bahasa sebagai kekuatan utama untuk mengintegrasikan *Generative AI* atau *Large Language Models* yang mana membuat aplikasi untuk proses pembuatan *unit test* secara otomatis melalui Ekstensi Chrome. Dengan pendekatan ini, proses pembuatan *unit test* secara otomatis langsung dari browser, tanpa perlu berpindah alat atau lingkungan pengembangan sehingga *developer* tidak perlu membuka website ChatGPT dengan berbagai tab baru yang mana bisa berfokus pada dokumen skenario pengujian dengan memanfaatkan fitur-fitur dari Ekstensi Chrome seperti *side panel* dengan membagi layar menjadi dua bagian[8].

Pada penelitian yang membahas berbagai integrasi LLM ke perangkat lunak disebutkan di *paper* "Unifying the Perspective of NLP and Software Engineering: A Survey on Language Models for Code"[6] di *section* 2.1.3 *Testing and Analysis* pada bagian *Unit Test Generations* terdapat beberapa penelitian yang menerapkan integrasi AI dengan metode *Non-neural, Non-Transformer neural,* dan *Transformer-based*. Dalam kategori pendekatan atau algoritma berbasis Transformer, terdapat setidaknya 13 proyek penelitian yang mengimplementasikan *Large Language Models* (LLM) untuk tugas *unit test generation.* Beberapa diantaranya adalah: AthenaTest[9][10], TestPilot[11], A3Test[12], TeCo[13], CodaMosa[14], ChatTester[15], ChatUniTest[16] [17], PBT-GPT[18],

MuTAP[19], RLSQM[20], CoverUp[21], dan TELPA[22]. Proyek-proyek tersebut memanfaatkan model bahasa yang telah dilatih sebelumnya (*pre-trained language models*), seperti BART Transformer, untuk menghasilkan *unit test* secara otomatis. Selain itu, masing-masing proyek menggunakan dataset tertentu sebagai bagian dari proses pelatihan dan evaluasi model. Sebagai contoh, proyek AthenaTest menggunakan dataset Method2Test[23], yang dibuat dalam bahasa pemrograman Java, untuk melakukan *fine tuning* dan evaluasi terhadap model BART dalam rangka meningkatkan pemahaman dan relevansi data *unit test* yang dihasilkan.

Namun, pendekatan integrasi LLM secara langsung ke dalam proyek perangkat lunak, khususnya pada ekstensi browser seperti Chrome, dapat menimbulkan tantangan signifikan terhadap kinerja sistem. Jika model LLM yang telah melalui proses *fine tuning* dioperasikan secara lokal (*on-device*), proses *inferensi* dari model tersebut akan mengkonsumsi sumber daya komputer pengguna dalam jumlah besar, seperti RAM dan CPU. Hal ini dapat berdampak pada performa keseluruhan perangkat, menjadikan pengalaman pengguna tidak efektif, terutama bagi pengguna dengan spesifikasi perangkat terbatas.

Sebagai solusi terhadap permasalahan ini, pendekatan yang lebih efisien dan ringan adalah dengan menggunakan model LLM berbasis *cloud* melalui API. Dalam pengembangan aplikasi ekstensi Chrome yang dilakukan pada penelitian ini, digunakan layanan API gratis dari GroqAI[24], yang memungkinkan integrasi LLM melalui kontrak endpoint yang telah ditentukan. Dengan pendekatan ini, proses pemanggilan dan *generation respons* dilakukan di server Groq, sehingga tidak membebani perangkat pengguna. Selain itu, arsitektur ini juga memudahkan pengelolaan sumber daya dan memungkinkan skalabilitas yang lebih baik pada aplikasi berbasis ekstensi browser. Pada model yang dipilih, penulis menggunakan varian Deepseek-r1-distill-llama-70b, Mistral-saba-24b, Gemma2-9b-it, dan LLaMA-3.3-70b-versatile. Model-model dengan parameter tinggi ini dipilih karena memiliki kemampuan penalaran (*reasoning*) dan pemahaman konteks kode yang mendalam, yang sangat krusial untuk menghasilkan *unit test* Golang yang valid dan kompleks. Dukungan infrastruktur LPU dari Groq memastikan model besar ini

tetap dapat memberikan respons cepat (*real-time*) tanpa membebani kinerja perangkat pengguna.

1.2 Rumusan Masalah

Dari permasalahan yang diuraikan dalam latar belakang, penulis merumuskan masalah: Bagaimana penerapan Generative AI atau *Large Language Models* dari provider GroqAI melalui ekstensi Chrome agar dapat mengotomatisasi pembuatan *unit test* Golang untuk meningkatkan efisiensi dalam *Test Driven Development*?"

1.3 Tujuan

Penelitian ini bertujuan untuk membangun sebuah *unit test generator* berbasis ekstensi Chrome yang mengintegrasikan *Large Language Models* (LLM). Sistem ini berfungsi menerima input kode program Go dan menghasilkan *unit test* secara otomatis, sehingga dapat meningkatkan efisiensi waktu, menjaga standarisasi kode, serta mendukung penerapan metodologi *Test-Driven Development* (TDD) tanpa perlu penulisan manual.

1.4 Manfaat

Implementasi ekstensi Chrome untuk otomatisasi pembuatan *unit test* Golang berbasis Generative AI diharapkan memberikan beberapa manfaat berikut:

1. Meningkatkan efisiensi proses *Test-Driven Development* dengan mengurangi waktu penulisan *unit test* secara manual dan memudahkan pengembang pemula untuk mempraktikkan *Test-Driven Development* tanpa memahami secara mendalam struktur penulisan *unit test* di Golang.
2. Menjamin konsistensi dan kualitas skenario pengujian karena *test case* dihasilkan dari model bahasa yang sudah terlatih untuk *code generation* dengan milyaran parameter.
3. Mengurangi beban kerja pengembang, sehingga mereka dapat lebih fokus pada desain dan implementasi fitur utama.

4. Menyediakan *benchmark* dan data empiris bagi peneliti lain untuk mengevaluasi kinerja model bahasa pada tugas *unit test generation*.
5. Jika pengguna membuka dokumen *source code* atau skenario pengujian di browser (misalnya di GitHub, GitLab, atau platform dokumentasi internal), ekstensi ini dapat membuka *side panel* dengan mengaktifkan ekstensi chrome yang dibuat.

1.5 Batasan Masalah

Agar penelitian ini berjalan secara terfokus dan terarah, penulis menetapkan beberapa batasan sebagai ruang lingkup dalam pengembangan sistem, yaitu sebagai berikut:

1. Penelitian ini hanya difokuskan pada pembuatan *unit test* untuk kode yang ditulis menggunakan bahasa pemrograman Go (Golang). Bahasa pemrograman lain tidak menjadi bagian dari ruang lingkup penelitian ini.

2. Model bahasa generatif yang digunakan dalam penelitian ini terbatas pada model yang tersedia melalui layanan API GroqAI, dengan fokus spesifik pada varian: Deepseek-r1-distill-llama-70b, Mistral-saba-24b, Gemma2-9b-it, dan LLaMA-3.3-70b-versatile. Pengujian dan perbandingan output *unit test* dari model-model tersebut dilakukan dengan mengikuti batasan *rate limit* (batas laju permintaan) dan kuota token yang berlaku pada layanan tingkat gratis (*free tier*). Fitur tambahan seperti *debugging, refactoring,* atau manajemen proyek tidak termasuk dalam cakupan pengembangan.

3. *Unit test* yang dihasilkan mungkin atau akan mengikuti standar umum pengujian di Golang, termasuk penggunaan pustaka bawaan (testing) dan pustaka pihak ketiga yang populer. Namun demikian, output dari masing-masing model bahasa dapat bervariasi, tergantung pada arsitektur model, *prompt*, dan respons yang dihasilkan dari masing-masing API.

4. Penelitian ini berfokus pada lingkungan browser Google Chrome, untuk lingkungan browser lainnya seperti Microsoft Edge, Mozilla Firefox, atau lainnya tidak menjadi bagian dari ruang lingkup penelitian dan tidak dievaluasi dalam studi ini.

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan hasil penelitian yang telah dilakukan, disimpulkan bahwa proyek Rancang Bangun Ekstensi Chrome untuk Golang Unit Test Generator dengan Model Bahasa Generative AI berhasil dikembangkan dan diimplementasikan dengan baik. Sistem ekstensi ini mampu mengotomatisasi pembuatan kode unit test pada bahasa pemrograman Golang melalui integrasi dengan layanan Generative AI GroqCloud. Berdasarkan hasil pengujian *black-box,* seluruh fungsi utama seperti unggah file, pemilihan model, penyimpanan riwayat, dan antarmuka pengguna berjalan sesuai dengan kebutuhan yang telah dirancang.

Dari hasil evaluasi model bahasa terhadap dataset *golang-test-from-the-stack* , diperoleh bahwa model Deepseek-R1-distill-LLaMA menunjukan performa yang stabil dan akurat dalam menghasilkan *unit test* yang sesuai dengan referensi, dengan tingkat keberhasilan kompilasi mencapai lebih dari 90 persen serta *line coverage* dan *branch coverage* diatas 95 persen. Selain itu, hasil *System Usability Scale* (SUS) dari 45 responden memperoleh nilai rata-rata 65,12 yang termasuk diterima namun terdapat perbaikan, yang mana menunjukan ekstensi ini cukup digunakan dan memiliki tingkat kegunaan yang sedang.

5.2. Saran

Penelitian ini masih memiliki ruang untuk pengembangan lebih lanjut. Berdasarkan evaluasi dan tingkat kegunaan sistem yang berada pada kategori *Marginally*, disarankan agar penelitian berikutnya dilakukan pada peningkatan pada aspek kemudahan penggunaan. Pengembangan dapat difokuskan pada penyederhanaan alur interaksi pengguna, seperti navigasi yang lebih intuitif, pengaturan otomatis untuk pemilihan model dari kesukaan pengguna, serta penambahan panduan interaktif (tutorial *on-screen*) agar pengguna baru lebih mudah memahami fungsi setiap fitur tanpa harus membaca dokumentasi secara

terpisah. Selanjutnya mengonsumsi respons dari model bahasa dengan UI yang lebih intuitif.

Selain fokus pada peningkatan *usability*, sistem mungkin dapat dikembangkan dengan fitur integrasi eksternal, misalnya koneksi langsung ke github atau VS Code, dimana pengguna tidak perlu mengimpor kode program hanya menjalankan ekstensi saja sudah mengenerate file kode unit test.

DAFTAR PUSTAKA

[1] A. Bandi, P. V. S. R. Adapa, and Y. E. V. P. K. Kuchi, "The Power of Generative AI: A Review of Requirements, Models, Input–Output Formats, Evaluation Metrics, and Challenges," Future Internet, vol. 15, no. 8, p. 260, Jul. 2023, doi: 10.3390/fi15080260.

[2] A. Bulajic, S. Sambasivam, and R. Stojic, "Overview of the Test Driven Development Research Projects and Experiments," presented at the InSITE 2012: Informing Science + IT Education Conference, 2012, pp. 165–187. doi: 10.28945/1647.

[3] A. M. Dakhel, A. Nikanjam, V. Majdinasab, F. Khomh, and M. C. Desmarais, "Effective Test Generation Using Pre-trained Large Language Models and Mutation Testing," Aug. 31, 2023, arXiv: arXiv:2308.16557. doi: 10.48550/arXiv.2308.16557.

[4] A. Rokhman, "Dicoding TDD," Mengenal Testing dalam Dunia Pengembangan Software. Accessed: Jun. 13, 2025. [Online]. Available: https://www.dicoding.com/blog/mengenal-testing-dalam-dunia-pengembangan-software/

[5] A. S. Gills, "Definition Go Programming Language," What is the Go or Golang programming language? Accessed: Jun. 11, 2025. [Online]. Available: https://www.techtarget.com/searchitoperations/definition/Go-programming-language

[6] B. Maureen, Worldwide Developer Population Grows to 27 Million. Accessed: Oct. 27, 2025. [Online]. Available: https://evansdata.com/press/viewRelease.php?pressID=365

[7] B. Steenhoek, M. Tufano, N. Sundaresan, and A. Svyatkovskiy, "Reinforcement Learning from Automatic Feedback for High-Quality Unit Test Generation," Jan. 06, 2025, arXiv: arXiv:2310.02368. doi: 10.48550/arXiv.2310.02368.

[8] C. Informatika, Yusuf Muharam, and Taufik Hidayat, "PENGEMBANGAN APLIKASI BACK-END E-COMMERCE MENGGUNAKAN REST API GOLANG UNTUK OPTIMALISASI KINERJA SERVER," Comput. J. Inform., vol. 11, no. 01, pp. 7–13, Jun. 2024, doi: 10.55222/computing.v11i01.1479.

[9] C. Yang, J. Chen, B. Lin, J. Zhou, and Z. Wang, "Enhancing LLM-based Test Generation for Hard-to-Cover Branches via Program Analysis," Apr. 07, 2024, arXiv: arXiv:2404.04966. doi: 10.48550/arXiv.2404.04966.

[10] "Chrome Extension Stat," Counting Chrome Extensions – Chrome Web Store Statistics. Accessed: Jun. 12, 2025. [Online]. Available: https://www.debugbear.com/blog/counting-chrome-extensions

[11] “chrome for developers.” [Online]. Available: https://developer.chrome.com/docs/extensions/reference/api/sidePanel

[12] “CodaMOSA,” CodaMOSA. [Online]. Available: https://github.com/microsoft/codamosa

[13] F. Jonathan and M. A. I. Pakereng, “Test-Driven Development pada Pengembangan Aplikasi Android untuk Memantau COVID-19,” IJCIT Indones. J. Comput. Inf. Technol., vol. 6, no. 1, May 2021, doi: 10.31294/ijcit.v6i1.9502.

[14] “Go,” Go Programming Language (Introduction). Accessed: Jun. 11, 2025. [Online]. Available: https://www.geeksforgeeks.org/go-programming-language-introduction/

[15] “Golang Testing Documentation,” Add a Test. Accessed: Jun. 11, 2025. [Online]. Available: https://go.dev/doc/tutorial/add-a-test

[16] “Golang Testing Documentation,” testing. Accessed: Jun. 11, 2025. [Online]. Available: https://pkg.go.dev/testing

[17] “Groq AI,” Limits Models. Accessed: Jun. 21, 2025. [Online]. Available: https://console.groq.com/dashboard/limits

[18] “Groq Inc.,” Groq – Accelerating AI Inference. Accessed: Jun. 10, 2025. [Online]. Available: https://groq.com/

[19] Groq Shows Promising Results in New LLM Benchmark, Surpassing Industry Averages. Accessed: Jun. 20, 2025. [Online]. Available: https://www.hpcwire.com/off-the-wire/groq-shows-promising-results-in-new-llm-benchmark-surpassing-industry-averages/

[20] Groq, ArtificialAnalysis.ai LLM Benchmark Doubles Axis To Fit New Groq LPUTM Inference Engine Performance Results. Accessed: Jun. 20, 2025. [Online]. Available: https://groq.com/artificialanalysis-ai-llm-benchmark-doubles-axis-to-fit-new-groq-lpu-inference-engine-performance-results/

[21] H. Zhu, P. A. V. Hall, and J. H. R. May, “Software unit test coverage and adequacy,” ACM Comput. Surv., vol. 29, no. 4, pp. 366–427, Dec. 1997, doi: 10.1145/267580.267590.

[22] J. A. Pizzorno and E. D. Berger, “CoverUp: Effective High Coverage Test Generation for Python,” May 09, 2025. doi: 10.1145/3729398.

[23] J. Jiang, F. Wang, J. Shen, S. Kim, and S. Kim, “A Survey on Large Language Models for Code Generation,” Nov. 10, 2024, arXiv: arXiv:2406.00515. doi: 10.48550/arXiv.2406.00515.

[24] J. Lennon, “Reason Learn Golang,” Top 5 Reasons Why You Should Learn Golang.

Accessed: Jun. 11, 2025. [Online]. Available: https://zerotomastery.io/blog/why-you-should-learn-golang/

[25] K. Ardonov and R. Lavine, "Groq Funding Series," Analysis: Groq computes a $300m series C. Accessed: Jun. 20, 2025. [Online]. Available: https://globalventuring.com/analysis-groq-computes-a-300m-series-c/

[26] K. Beck, Test-driven development: by example, 20. printing. in The Addison-Wesley signature series. Boston: Addison-Wesley, 2015.

[27] L. Yang et al., "On the Evaluation of Large Language Models in Unit Test Generation," Sep. 25, 2024, arXiv: arXiv:2406.18181. doi: 10.48550/arXiv.2406.18181.

[28] M. Scapicchio and C. Stryker, "IBM Generative AI." Accessed: Jun. 23, 2025. [Online]. Available: https://www.ibm.com/think/topics/generative-ai

[29] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, "An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation," Dec. 11, 2023, arXiv: arXiv:2302.06527. doi: 10.48550/arXiv.2302.06527.

[30] M. Tufano, D. Drain, A. Svyatkovskiy, S. K. Deng, and N. Sundaresan, "Unit Test Case Generation with Transformers and Focal Context," May 20, 2021, arXiv: arXiv:2009.05617. doi: 10.48550/arXiv.2009.05617.

[31] M. Tufano, D. Drain, A. Svyatkovskiy, and N. Sundaresan, "Generating Accurate Assert Statements for Unit Test Cases using Pretrained Transformers," in Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test, May 2022, pp. 54–64. doi: 10.1145/3524481.3527220.

[32] "Manivest v2," Manivest V2 Get Started. Accessed: Jun. 12, 2025. [Online]. Available: https://developer.chrome.com/docs/extensions/mv2/getstarted

[33] Microsoft, "methods2test." doi: https://doi.org/10.1145/3524842.3528009.

[34] N. Pantelaios and A. Kapravelos, "Manifest V3 Unveiled: Navigating the New Era of Browser Extensions," Apr. 12, 2024. doi: 10.14722/madweb.2024.23080.

[35] P. Nie, R. Banerjee, J. J. Li, R. J. Mooney, and M. Gligoric, "Learning Deep Semantics for Test Completion," Mar. 07, 2023, arXiv: arXiv:2302.10166. doi: 10.48550/arXiv.2302.10166.

[36] P. Runeson, "A survey of unit testing practices," IEEE Softw., vol. 23, no. 4, pp. 22–29, Jul. 2006, doi: 10.1109/MS.2006.91.

[37] Quang, "golang-test-from-the-stack." Hugging Face Datasets, 2025. doi: https://huggingface.co/datasets/lqdunxgx2005/golang-test-from-the-stack.

[38] R. Pike, "Go at Google," Go at Google: Language Design in the Service of Software Engineering. Accessed: Jun. 02, 2025. [Online]. Available: https://go.dev/talks/2012/splash.article

[39] "Reason Golang," Top 8 Reasons to learn Go Language in 2022. Accessed: Jun. 11, 2025. [Online]. Available: https://www.studytonight.com/post/top-reasons-to-learn-go-language

[40] S. Alagarsamy, C. Tantithamthavorn, and A. Aleti, "A3Test: Assertion-Augmented Automated Test Case Generation," Feb. 20, 2023, arXiv: arXiv:2302.10352. doi: 10.48550/arXiv.2302.10352.

[41] T. is DASC, "medium," Seven Golang Features you must know about. Accessed: Jun. 11, 2025. [Online]. Available: https://medium.com/@thisisdasc/seven-golang-features-you-must-know-about-944485d413fe

[42] V. Garousi, M. Felderer, and F. N. Kılıçaslan, "A survey on software testability," Inf. Softw. Technol., vol. 108, pp. 35–64, Apr. 2019, doi: 10.1016/j.infsof.2018.12.003.

[43] V. Vikram, C. Lemieux, J. Sunshine, and R. Padhye, "Can Large Language Models Write Good Property-Based Tests?," Jul. 22, 2024, arXiv: arXiv:2307.04346. doi: 10.48550/arXiv.2307.04346.

[44] Y. Chen, Z. Hu, C. Zhi, J. Han, S. Deng, and J. Yin, "ChatUniTest: A Framework for LLM-Based Test Generation," May 07, 2024, arXiv: arXiv:2305.04764. doi: 10.48550/arXiv.2305.04764.

[45] Y. Tang, Z. Liu, Z. Zhou, and X. Luo, "ChatGPT vs SBST: A Comparative Assessment of Unit Test Suite Generation," Jul. 02, 2023, arXiv: arXiv:2307.00588. doi: 10.48550/arXiv.2307.00588.

[46] Yoshi, "Architecture Chrome Extension," Architecture of Chrome Extension. Accessed: Jun. 12, 2025. [Online]. Available: https://medium.com/@yoshi2586/architecture-of-chrome-extension-9188c026c069

[47] Z. Yuan et al., "No More Manual Tests? Evaluating and Improving ChatGPT for Unit Test Generation," May 19, 2024, arXiv: arXiv:2305.04207. doi: 10.48550/arXiv.2305.04207.

[48] Z. Zhang et al., "Unifying the Perspectives of NLP and Software Engineering: A Survey on Language Models for Code," Jun. 26, 2024, arXiv: arXiv:2311.07989. doi: 10.48550/arXiv.2311.07989.

LAMPIRAN

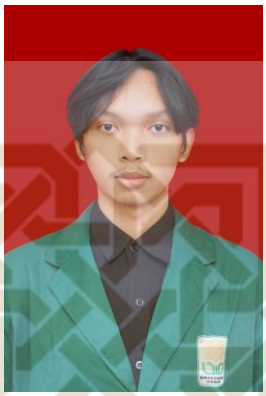
Repository Golang Unit Test Generator: https://github.com/Rizald95/golang0-unit-test

Repository Evaluasi: https://github.com/Rizald95/Golang0-unit-test-evaluation

Link Google Colab Evaluasi Bleu, Rouge, dan Exatch Match Part 1: https://colab.research.google.com/drive/17nCGwniqyMNTacrZCQ0MoLqUPUse_U7e?usp=sharing

Link Evaluasi BLEU, ROUGE, dan Exatch Match Part 2 : https://colab.research.google.com/drive/1laj9Hl9_TedojkA5DsZ0o0fQDmLDyzGC?usp=sharing

RIWAYAT HIDUP



Nama	: Rizal Darusman
Tempat, Tanggal Lahir	: Wonosobo, 9 Mei 2002
No. Telephone	: 089687973839
Email	: rizalbario3@gmail.com
Alamat	: Jalan Mandalika, Dusun Mandalika, Desa Tanjunganom Kecamatan Kaliwiro, Kabupaten Wonosobo, Kode Pos 56364
Riwayat Pendidikan	: SDN 1 TANJUNGANOM SMPN 1 KALIWIRO SMAN 1 WONOSOBO UIN SUNAN KALIJAGA YOGYAKARTA
Hobi	: Makan, Tidur, Suka Suasana Tenang
Motto Hidup	: Selama Manusia sudah cukup dan hidup tenang artinya menang dalam menguasai kehidupan.