BAB 2
LANDASAN TEORI

2.1 Karya Terkait

Studi pada bidang *blockchain* dan kontrak cerdas terutama *benchmarking* telah muncul dalam beberapa tahun terakhir untuk mengeksplorasi berbagai aspek dari teknologi inovatif ini. Sebuah *framework* evaluasi diusulkan oleh A. Aldweesh [23] untuk memeriksa insentif yang selaras antara waktu eksekusi kontrak cerdas dan imbalan yang terkait untuk *miner*. Menjalankan klien *Python Ethereum* pada *Mac* khusus untuk *benchmarking* menghasilkan hasil yang menggembirakan. Analisis menunjukkan bahwa ada variasi yang signifikan, sekitar 50 kali lipat dalam jumlah kompensasi per detik CPU di antara berbagai fungsi yang ditemukan dalam kontrak *Ethereum* yang terkenal. Pengembangan kontrak cerdas menghasilkan keuntungan hingga enam kali lebih tinggi daripada operasi kontrak cerdas. Perbedaan ini dapat menyebabkan insentif yang tidak selaras yang dapat memengaruhi seberapa andal *blockchain* tersebut.

Alat *benchmark* untuk *Ethereum* yang disebut *OpBench* diusulkan oleh A. Aldweesh [24]. Alat ini menawarkan cara sederhana untuk memeriksa apakah imbalan *gas* untuk kode operasional (*opcode*) sesuai dengan jumlah waktu CPU yang digunakan. *OpBench*, yang dirancang untuk *Parity (Rust)*, *Go-Ethereum (GoLang)*, dan *PyEthApp (Python)*, memperlihatkan rasio *gas-to-CPU* yang tidak seragam. Beberapa *opcodes* menunjukkan perbedaan hingga 10 kali lipat. Menurut penelitian tersebut, Parity memiliki kinerja lebih baik daripada klien lain untuk sebagian besar *opcode*.

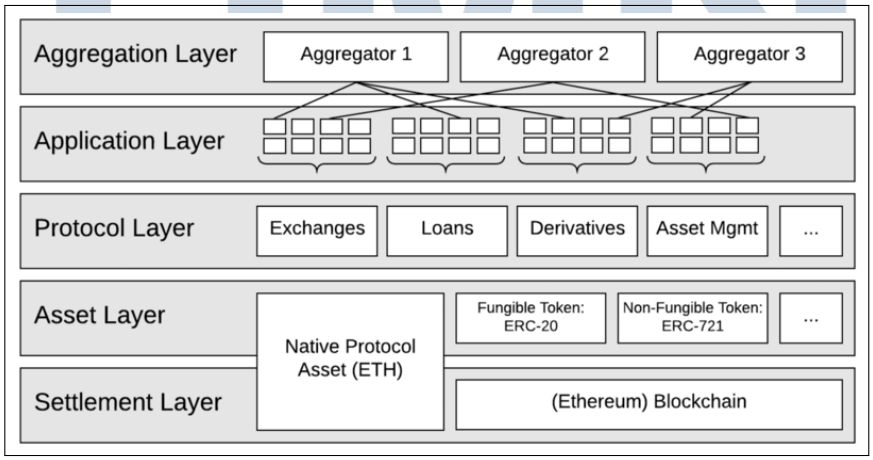
BCTMark adalah sebuah *framework* komprehensif yang diperkenalkan oleh D. Saingre [25] untuk memungkinkan perbandingan yang akurat terhadap teknologi *blockchain* dalam jaringan virtual. Dengan menggunakan *framework* baru ini, penelitian ini mengeksplorasi berapa banyak komputasi energi yang digunakan oleh eksekusi kontrak cerdas pada *blockchain* modern. Penelitian ini menggunakan pengukuran dan pemodelan konsumsi energi kontrak cerdas *Ethereum* untuk mengekstrak wawasan dari pengujian dan analisis transaksi *Ethereum* asli selama satu tahun. Penelitian ini juga mengklarifikasi bagaimana pemanggilan dan replikasi kontrak dapat mempengaruhi biaya energi.

Kesimpulan dari ketiga penelitian ini adalah A. Aldweesh [23] melakukan

studi pembandingan pertama untuk menentukan apakah jumlah *gas* yang digunakan untuk membuat dan menjalankan kontrak cerdas sebanding dengan jumlah pekerjaan komputasi. Pada penelitian tersebut juga merancang *framework benchmarking*, *OpBench* [24] untuk mengevaluasi penggunaan CPU dan *gas* yang digunakan untuk setiap opcode yang dieksekusi di dalam mesin virtual *Ethereum*. Di sisi lain, D. Saingre [25] membuat sebuah *framework benchmarking* untuk mengevaluasi konsumsi energi pada *blockchain* secara umum. Makalah penelitian tersebut membuat *benchmarking* terhadap kontrak cerdas dengan atau tanpa *framework* yang mengevaluasi konsumsi *gas* dan energi. Oleh karena itu, penelitian ini menambah kontribusi dengan menambahkan evaluasi dari kontrak cerdas DeFi dalam bentuk skor *testbench*.

2.2 Kontrak Cerdas DeFi

Dalam *blockchain*, kontrak cerdas merupakan sebuah teknologi revolusioner yang menawarkan sebuah metode fasilitasi transaksi yang aman dan otomatis. Kontrak yang dapat dijalankan secara otomatis memiliki aturan dan kondisi tertentu yang memastikan para pihak dalam transaksi memenuhi kewajiban tanpa keterlibatan suatu individu. Namun, DeFi dikembangkan untuk menggantikan sistem keuangan konvensional. Tidak ada perantara yang menyetujui transaksi untuk aplikasi DeFi karena tidak ada bank atau organisasi lain yang mengelola uang dan dapat juga untuk membuat protokol DeFi yang melarang keterlibatan dan manipulasi. Struktur DeFi *stack* dapat dilihat pada Gambar 2.1.



Gambar 2.1. DeFi *stack*

Sumber: [3]

Pelanggan pada masa lalu mengelola uang atau aset dengan memanfaatkan bank, sebuah sistem yang tersentralisasi. Kegiatan seperti menyetor atau menarik aset dari suatu pihak seperti bank adalah kegiatan yang mengandalkan pihak lain, dengan kemungkinan harus menghubungi pihak tersebut jika diperlukan. Kegiatan ini sebagian besar juga mengharuskan pelanggan untuk membayar sejumlah biaya layanan tersebut [1]. DeFi di sisi lain memiliki dompet pribadi berbentuk alamat digital yang dapat digunakan untuk melakukan kegiatan transaksi pada blockchain secara mandiri. Transaksi DeFi juga mengharuskan pengguna untuk membayar sejumlah biaya dalam bentuk *gas* agar pesanan atau pertukaran berhasil. Kontrak cerdas pada DeFi kemudian perlu memiliki fungsionalitas yang benar untuk aplikasi DeFi karena perubahan pada kontrak cerdas tidak dapat dilakukan setelah kontrak cerdas di-*deploy* pada jaringan *blockchain* [2].

DeFi diterapkan di dunia teknologi, salah satunya adalah mata uang kripto *blockchain Ethereum* [27][28]. Kapitalisasi pasar terbesar untuk *cryptotoken Ethereum* DeFi adalah *Chainlink* yang menggunakan *Solidity* [29], sebuah bahasa pemrograman berbasis kontrak cerdas yang memiliki peran besar dalam DeFi [30]. Terdapat beberapa kontrak cerdas yang sudah diterapkan pada DeFi dan aplikasi terdesentralisasi (*dapps*) dengan berbagai jenis kontrak cerdas untuk DeFi seperti NFT, *Decentralized Exchanges* (DEX), penagihan hutang, dan pinjaman kilat [28]. Kontrak cerdas DeFi sebagian besar menyertakan *token*-nya masing-masing yang dapat digunakan pengguna sebagai bukti kepemilikan seperti saham perusahaan, atau sebagai mata uang, mirip dengan koin sehari-hari dalam bentuk ERC (*Ethereum Request for Comments*) standar [31] seperti ERC-20, ERC-4626, dan ERC-3156. ERC ini adalah sekumpulan *interface*, kontrak, dan utilitas, yang merupakan kontrak inti yang mengimplementasikan perilaku yang ditentukan dalam setiap ERC.

2.2.1 ERC-20

Standar ERC-20 [32] adalah standar efisien yang paling terkenal dan banyak digunakan untuk mengelola *token*, menentukan bagaimana pertukaran *token* harus beroperasi, dan hanya digunakan untuk *token* yang dapat dipertukarkan. *Token* adalah aset bernilai dan kontrak cerdas memiliki kontrol bagaimana *token* tersebut dibuat, dihancurkan, dan dipertukarkan. *Token* dapat menggantikan sumber daya yang dapat ditukar seperti uang tunai, waktu, atau saham bisnis [33][34]. ERC-20 memiliki serangkaian standar *functions* dan *events* yang dapat dilihat pada Tabel 2.1

dan Tabel 2.2.

Tabel 2.1. ERC-20 *Functions*

Functions
name()
symbol()
decimals()
totalSupply()
balanceOf(account)
transfer(to, amount)
allowance(owner, spender)
approve(spender, amount)
transferFrom(from, to, amount)

Tabel 2.2. ERC-20 *Events*

Events
Transfer(from, to, value)
Approval(owner, spender, value)

2.2.2 ERC-4626

Standar *interface* untuk brankas *token* adalah ERC-4626 [35], sebuah ekstensi dari ERC-20. Banyak kontrak yang berbeda seperti pasar peminjaman, agregator, dan *token* yang secara intrinsik berbunga dapat menggunakan *interface* standar ini [35]. Menerapkan brankas *token* yang sesuai dan modular membutuhkan navigasi untuk mengatasi masalah yang mungkin terjadi. ERC-4626 memiliki serangkaian standar *functions* dan *events* yang dapat dilihat pada Tabel 2.3 dan Tabel 2.4.

Tabel 2.3. ERC-4626 *Functions*

Functions
asset(assetTokenAddress)
totalAssets(totalManagedAssets)
convertToShares(assets, shares)
convertToAssets(shares, assets)
maxDeposit(receiver, maxAssets)
previewDeposit(assets, shares)
deposit(assets, receiver, shares)
maxMint(receiver, maxShares)
previewMint(shares, assets)
mint(shares, receiver, assets)
maxWithdraw(owner, maxAssets)
previewWithdraw(assets, shares)
withdraw(assets, receiver, owner, shares)
maxRedeem(owner, maxShares)
previewRedeem(shares, assets)
redeem(shares, receiver, owner, assets)

Tabel 2.4. ERC-4626 *Events*

Events
Deposit(sender, owner, assets, shares)
Withdraw(sender, receiver, owner, assets, shares)

2.2.3 ERC-3156

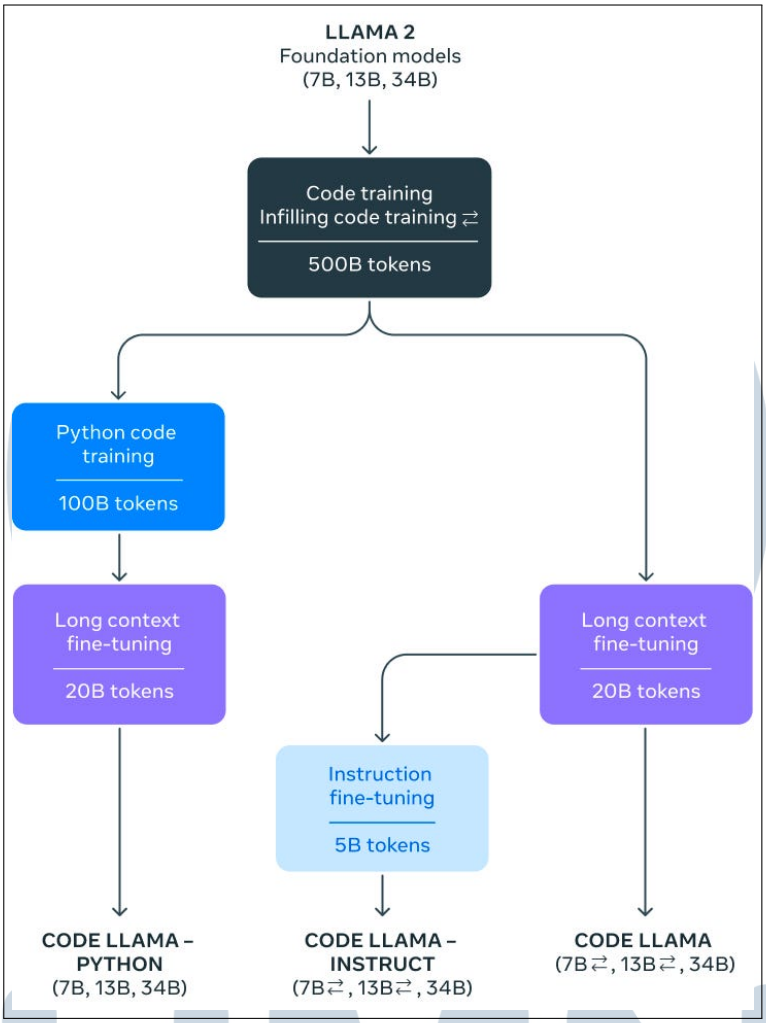
ERC-3156 [36] adalah kontrak cerdas *flash loan*, merupakan transaksi yang melibatkan peminjam menerima aset dari kontrak cerdas pemberi pinjaman dengan pemahaman bahwa aset akan dikembalikan sebelum transaksi berakhir bersama dengan biaya opsional. ERC ini memiliki *interface* untuk para pemberi pinjaman untuk menerima permintaan *flash loan* dan para peminjam untuk mengambil alih eksekusi transaksi dalam pemberi pinjaman sementara [36]. ERC-3156 hanya memiliki serangkaian standar *function* yang dapat dilihat pada Tabel 2.5.

Tabel 2.5. ERC-3156 *Functions*

Functions
maxFlashLoan(token)
flashFee(token, amount)
flashLoan(receiver, token, amount, data)
onFlashLoan(initiator, token, amount, fee, data)

2.3 Code-LLaMa

Code-Llama merupakan Llama 2 dengan fokus pada pengkodean, dimodifikasi melalui pelatihan pada dataset yang hanya digunakan untuk memuat konten yang berhubungan dengan kode. Versi ini mencakup periode pengambilan sampel yang lebih lama dari kumpulan data terkait kode yang mengarah pada peningkatan kinerja pengkodean. Code-Llama menunjukkan kompetensi dalam menulis kode dan menjelaskannya dalam bentuk tertulis. Model ini dapat merespon perintah dalam bentuk tertulis dan kode. Contohnya, pengguna memiliki kemampuan untuk mengusulkan pengembangan fungsi yang menghasilkan deret Fibonacci. Program ini berguna untuk *debugging* dan penyelesaian kode, dan mendukung sejumlah bahasa pemrograman populer, termasuk Python, C ++, Java, PHP, Typescript (JavaScript), C#, dan Bash [15]. Struktur Code-LLaMa dapat dilihat pada Gambar 2.2.



Gambar 2.2. Struktur Model Code-LLaMa

Sumber: [37]

Dengan 500 miliar *token* kode dan data terkait kode, Code-Llama dilatih dengan tiga ukuran yang berbeda: 7B, 13B, dan 34B. Model dasar dan instruksional 7B dan 13B dilengkapi dengan kemampuan untuk mengisi bagian yang hilang (*Fill in missing parts* atau FIM), memungkinkan model tersebut untuk menambah kode yang ada dengan mulus, memfasilitasi tu*gas*-tu*gas* seperti penyelesaian kode tanpa pengaturan tambahan. Versi yang dirancang khusus untuk Python, yang disebut Code Llama - Python, dikustomisasi menggunakan kumpulan data besar yang berisi 100 miliar *token Python*. Hal ini menggambarkan pentingnya Python dalam komunitas AI dan penggunaannya yang umum dalam pembuatan kode [15].

Dataset	Sampling prop.	Epochs	Disk size
Code Llama (500B tokens)			
Code	85%	2.03	859 GB
Natural language related to code	8%	1.39	78 GB
Natural language	7%	0.01	3.5 TB
Code Llama - Python (additional 100B tokens)			
Python	75%	3.69	79 GB
Code	10%	0.05	859 GB
Natural language related to code	10%	0.35	78 GB
Natural language	5%	0.00	3.5 TB

Gambar 2.3. *Training* pada Code-LLaMa dan Code-LLaMa Python

Sumber: [37]

Setiap model memenuhi kebutuhan yang berbeda dalam hal latensi dan penyajian. Sebagai contoh, model 7B berjalan dengan baik hanya dengan satu GPU. Sebagai perbandingan, model 34B memberikan hasil yang lebih baik dan menawarkan dukungan pengkodean yang lebih baik. Model 7B dan 13B yang lebih kecil mengutamakan kecepatan dan ideal untuk pekerjaan yang membutuhkan waktu penyelesaian yang cepat seperti penyelesaian kode secara instan [15].

2.4 Benchmarking Kontrak Cerdas

Benchmarking adalah teknik untuk memeriksa dan membandingkan norma dan pengalaman di berbagai operasi bisnis. Perusahaan Xerox pertama kali menggunakan strategi ini pada tahun 1980-an untuk memperkuat posisinya sebagai pemimpin pasar [38] yang kemudian berhasil dan juga memotivasi perusahaan lain untuk mengadopsi metode *benchmarking*. Penelitian ini [20] yang juga mempelajari kasus Xerox menyatakan *performance benchmarking*, salah satu pendekatan *benchmarking* untuk membandingkan subjek dengan yang lain untuk menilai kinerjanya. Kontrak cerdas merupakan aset yang layak untuk dijadikan subjek *benchmark* karena berfungsi sebagai tulang punggung aplikasi terdesentralisasi dan sistem *blockchain*. Hal itu membutuhkan evaluasi menyeluruh untuk memastikan kinerja yang optimal karena setelah kontrak di *deploy* pada sistem *blockchain*, kontrak tersebut tidak dapat diperbarui [2].

Hal yang harus ditentukan terkait pada evaluasi untuk *benchmark* adalah, pertama kontrak cerdas harus berfungsi. Dalam kontrak cerdas Solidity, ketika sebuah kontrak dikompilasi, kontrak tersebut akan diubah menjadi serangkaian akronim dalam bentuk kode operasi (*opcodes*) [27]. Opcodes ini bertindak sebagai

pemicu komputasi bagi pengirim transaksi untuk berinteraksi, yang mengharuskan pengirim untuk membayar biaya tambahan *gas*. Biaya *gas* ditentukan oleh opcodes dalam kontrak cerdas, yang bervariasi dalam jumlah dan jenis operasi [22]. Opcodes tersebut memiliki biaya *gas* yang tetap pada berbagai jenis operasi. Oleh karena itu, semakin rumit kontrak cerdas, semakin tinggi biaya *gas*nya. Beberapa biaya *gas* dari *yellow paper Ethereum* dapat dilihat pada Tabel 2.6.

Tabel 2.6. Biaya *gas Ethereum*

Nama Operasi	***gas***	**Deskripsi**
ADDRESS	2	Get address
CREATE/CALL	32,000/21,000	Membuat akun baru
ADD/SUB	3	Operasi aritmatika
BALANCE	400	Menunjukan saldo akun
KECCAK256	3	Operasi hash Keccak256
TRANSACTION	21,000	Operasi transaction
SSTORE	20,000	Sistem penampungan (*storage*)

Sumber: [27]

Benchmarking biaya *gas* dari kontrak cerdas yang dibuat oleh manusia berbeda dengan kontrak cerdas yang dibuat secara otomatis oleh LLM karena biaya *gas* dapat langsung ditentukan saat mengkode opcode yang diinginkan untuk kontrak cerdas. Penulis [22] telah mengidentifikasi desain pola penghematan *gas* yang berbeda, dibagi dengan beberapa kategori:

1. *External transactions*: Pola yang terkait dengan penulisan kontrak dan pengiriman transaksi dari alamat eksternal, seperti program JavaScript yang menggunakan *library* standar *Web3.js*. Desain yang disebutkan dalam kategori ini terkait dengan *proxy*, kontrak data, dan log peristiwa.
2. *Storage*: Pola yang terkait dengan penggunaan penyimpanan untuk penyimpanan data jangka panjang. Desain yang disebutkan dalam kategori ini terkait dengan pembatasan penyimpanan, variabel pengepakan, dan boolean.
3. *Saving space*: Pola yang terkait dengan memori dan konservasi ruang penyimpanan. Desain yang disebutkan terkait dengan tipe variabel, tipe data, ukuran variabel, nilai *default*, meminimalkan data *on-chain*, dan fungsi eksternal.

4. *Operations*: Pola yang terkait dengan *gas* yang digunakan dalam tu*gas* yang dilakukan oleh fungsi kontrak cerdas. Desain yang disebutkan terkait dengan membatasi panggilan eksternal, menggunakan panggilan fungsi internal, menggunakan lebih sedikit fungsi, *libraries*, *short circuit*, *constant short circuit*, *modifier*, operasi yang berlebihan, penukaran baris, dan penulisan nilai.

5. *Miscellaneous*: Pola yang tidak dapat dimasukkan ke dalam pola-pola sebelumnya. Desain yang disebutkan dalam kategori ini terkait dengan membebaskan penyimpanan dan mengoptimalkan kode

Desain pola penghematan *gas* tersebut dapat digunakan untuk mengevaluasi biaya *gas* pada kontrak cerdas yang dihasilkan secara otomatis. Metode *benchmarking* kontrak cerdas oleh penelitian [23] adalah dengan menggunakan jaringan *blockchain* lokal, *Ethereum Virtual Machine* (EVM) atau testnet *blockchain* publik secara lokal untuk menyebarkan kontrak cerdas agar dapat dieksekusi. Akan tetapi, penelitian ini menyatakan bahwa beberapa elemen, seperti *overhead* validasi transaksi, *overhead* validasi tanda tangan, dan bukti atau komputasi kerja, dapat mengganggu jika benchmark dilakukan dengan menggunakan *testnet*.