TUGAS AKHIR - EC234701

PENERAPAN MODEL MACHINE LEARNING UNTUK MENINGKATKAN EFISIENSI ALGORITMA CLOCK PADA CACHE REPLACEMENT

Muhammad Haekal Muhyidin Al-Araby
NRP 5024 22 1030

Dosen Pembimbing

Reza Fuad Rachmadi, S.T., M.T., Ph.D

NIP 19850403 201212 1 001

Dr. Arief Kurniawan, S.T., M.T

NIP 19740907 200212 1 001

Program Studi Strata 1 (S1) Teknik Komputer

Departemen Teknik Komputer

Fakultas Teknologi Elektro dan Informatika Cerdas

Institut Teknologi Sepuluh Nopember

Surabaya

2025

TUGAS AKHIR - EC234701

PENERAPAN MODEL MACHINE LEARNING UNTUK MENINGKATKAN EFISIENSI ALGORITMA CLOCK PADA CACHE REPLACEMENT

Muhammad Haekal Muhyidin Al-Araby
NRP 5024 22 1030

Dosen Pembimbing

Reza Fuad Rachmadi, S.T., M.T., Ph.D

NIP 19850403 201212 1 001

Dr. Arief Kurniawan, S.T., M.T
NIP 19740907 200212 1 001

Program Studi Strata 1 (S1) Teknik Komputer
Departemen Teknik Komputer
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya
2025

[Halaman ini sengaja dikosongkan]

FINAL PROJECT - EC234701

APPLICATION OF MACHINE LEARNING MODELS TO IMPROVE THE EFFICIENCY OF THE CLOCK ALGORITHM IN CACHE REPLACEMENT

Muhammad Haekal Muhyidin Al-Araby
NRP 5024 22 1030

Advisor

Reza Fuad Rachmadi, S.T., M.T., Ph.D

NIP 19850403 201212 1 001

Dr. Arief Kurniawan, S.T., M.T
NIP 19740907 200212 1 001

Undergraduate Study Program of Computer Engineering
Department of Computer Engineering
Faculty of Electrical Engineering and Intelligent Informatics
Sepuluh Nopember Institute of Technology
Surabaya
2025

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

PENERAPAN MODEL MACHINE LEARNING UNTUK MENINGKATKAN EFISIENSI ALGORITMA CLOCK PADA CACHE REPLACEMENT

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Teknik pada
Program Studi S-1 Teknik Komputer
Departemen Teknik Komputer
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh: **Muhammad Haekal Muhyidin Al-Araby**
NRP. 5024 22 1030

Disetujui oleh Tim Penguji Tugas Akhir:

Reza Fuad Rachmadi, S.T., M.T., Ph.D NIP: 19850403 201212 1 001	(Pembimbing I) .....
Dr. Arief Kurniawan, S.T., M.T NIP: 19740907 200212 1 001	(Pembimbing II) .....
Dr. Eko Mulyanto Yuniarno, S.T., M.T. NIP: 19680601 199512 1 009	(Penguji I) .....
Dr. Diah Puspito Wulandari, S.T., M.Sc. NIP: 19801219 200501 2 001	(Penguji II) .....
Arta Kusuma Hernanda, S.T., M.T. NIP: 1996202311024	(Penguji III) .....

Mengetahui,
Kepala Departemen Teknik Komputer FTEIC - ITS

Dr. Arief Kurniawan, S.T., M.T.
NIP. 19740907 200212 1 001

SURABAYA
Desember, 2025

[Halaman ini sengaja dikosongkan]

APPROVAL SHEET

APPLICATION OF MACHINE LEARNING MODELS TO IMPROVE THE EFFICIENCY OF THE CLOCK ALGORITHM IN CACHE REPLACEMENT

FINAL PROJECT
Submitted to fulfill one of the requirements
for obtaining a degree Bachelor of Engineering at
Undergraduate Study Program of Computer Engineering
Department of Computer Engineering
Faculty of Electrical Engineering and Intelligent Informatics
Sepuluh Nopember Institute of Technology

By: **Muhammad Haekal Muhyidin Al-Araby**
NRP. 5024 22 1030

Approved by Final Project Examiner Team:

Reza Fuad Rachmadi, S.T., M.T., Ph.D NIP: 19850403 201212 1 001	(Advisor I) .....
Dr. Arief Kurniawan, S.T., M.T NIP: 19740907 200212 1 001	(Co-Advisor II) .....
Dr. Eko Mulyanto Yuniarno, S.T., M.T. NIP: 19680601 199512 1 009	(Examiner I) .....
Dr. Diah Puspito Wulandari, S.T., M.Sc. NIP: 19801219 200501 2 001	(Examiner II) .....
Arta Kusuma Hernanda, S.T., M.T. NIP: 1996202311024	(Examiner III) .....

Acknowledged,
Head of Computer Engineering Department ELECTICS - ITS

Dr. Arief Kurniawan, S.T., M.T.
NIP. 19740907 200212 1 001

SURABAYA
December, 2025

[Halaman ini sengaja dikosongkan]

PERNYATAAN ORISINALITAS

Yang bertanda tangan dibawah ini:

Nama Mahasiswa / NRP	:	Muhammad Haekal Muhyidin Al-Araby / 5024 22 1030
Departemen	:	Teknik Komputer
Dosen Pembimbing / NIP	:	Reza Fuad Rachmadi, S.T., M.T., Ph.D / 19850403 201212 1 001

Dengan ini menyatakan bahwa Tugas Akhir dengan judul ”PENERAPAN MODEL MACHINE LEARNING UNTUK MENINGKATKAN EFISIENSI ALGORITMA CLOCK PADA CACHE REPLACEMENT” adalah hasil karya sendiri, berfsifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari ditemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima sanksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, December 2025

Mengetahui Dosen Pembimbing	Mahasiswa
Reza Fuad Rachmadi, S.T., M.T., Ph.D NIP. 19850403 201212 1 001	Muhammad Haekal Muhyidin Al-Araby NRP. 5024 22 1030

[Halaman ini sengaja dikosongkan]

STATEMENT OF ORIGINALITY

The undersigned below:

Name of student / NRP	:	Muhammad Haekal Muhyidin Al-Araby / 5024 22 1030
Department	:	Computer Engineering
Advisor / NIP	:	Reza Fuad Rachmadi, S.T., M.T., Ph.D / 19850403 201212 1 001

Hereby declared that the Final Project with the title of "*APPLICATION OF MACHINE LEARNING MODELS TO IMPROVE THE EFFICIENCY OF THE CLOCK ALGORITHM IN CACHE REPLACEMENT*" is the result of my own work, is original, and is written by following the rules of scientific writing.

If in future there is a discrepancy with this statement, then I am willing to accept sanctions in accordance with provisions that apply at Sepuluh Nopember Institute of Technology.

Surabaya, December 2025

Acknowledged Advisor	Student
Reza Fuad Rachmadi, S.T., M.T., Ph.D NIP. 19850403 201212 1 001	Muhammad Haekal Muhyidin Al-Araby NRP. 5024 22 1030

[Halaman ini sengaja dikosongkan]

ABSTRAK

Nama Mahasiswa : Muhammad Haekal Muhyidin Al-Araby

Judul Tugas Akhir : PENERAPAN MODEL MACHINE LEARNING UNTUK MENINGKATKAN EFISIENSI ALGORITMA CLOCK PADA CACHE REPLACEMENT

Pembimbing : 1. Reza Fuad Rachmadi, S.T., M.T., Ph.D
2. Dr. Arief Kurniawan, S.T., M.T

Penelitian ini berfokus pada pengembangan dan analisis algoritma *cache eviction* Clock berbasis *machine learning*. Studi ini bertujuan untuk meningkatkan kinerja algoritma Clock dengan memanfaatkan model *supervised learning* dalam proses pengambilan keputusan untuk *cache eviction*. Simulasi dilakukan menggunakan perangkat lunak kustom berbasis LibCacheSim dengan skenario akses data *trace replay*. Kinerja algoritma yang dikembangkan dianalisis menggunakan metrik seperti *hit ratio*, *miss ratio*, dan jumlah promosi. Hasil penelitian menunjukkan bahwa mengintegrasikan *machine learning* ke dalam algoritma Clock dapat meningkatkan kinerja dengan mengurangi jumlah promosi *cache*, menjadikannya lebih optimal dibandingkan algoritma Clock konvensional.

Kata Kunci: *Cache, Clock, LRU, FIFO, Sistem, Machine Learning, Supervised Learning*

[Halaman ini sengaja dikosongkan]

ABSTRACT

Name : Muhammad Haekal Muhyidin Al-Araby
Title : *APPLICATION OF MACHINE LEARNING MODELS TO IMPROVE THE EFFICIENCY OF THE CLOCK ALGORITHM IN CACHE REPLACEMENT*
Advisors : 1. Reza Fuad Rachmadi, S.T., M.T., Ph.D
2. Dr. Arief Kurniawan, S.T., M.T

This research focuses on the development and analysis of a machine learning-based Clock cache eviction algorithm. The study aims to improve the performance of the Clock algorithm by utilizing a supervised learning model in the decision-making process for cache eviction. Simulations were conducted using custom software based on LibCacheSim with a trace replay data access scenario. The performance of the developed algorithm was analyzed using metrics such as hit ratio, miss ratio, and the number of promotions. The results indicate that integrating machine learning into the Clock algorithm can enhance performance by reducing the number of cache promotions, making it more optimal than the conventional Clock algorithm.

Keywords: *Cache, Clock, LRU, FIFO, System, Machine Learning, Supervised Learning*

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Tuhan Yang Maha Esa, yang atas rahmat dan karunia-Nya telah memberikan kekuatan dan kelancaran sehingga penulis dapat menyelesaikan Laporan Tugas Akhir yang berjudul "**PENERAPAN MODEL MACHINE LEARNING UNTUK MENINGKATKAN EFISIENSI ALGORITMA CLOCK PADA CACHE REPLACEMENT**" dengan baik dan tepat waktu.

Penelitian dan penyusunan Laporan Tugas Akhir ini merupakan salah satu syarat untuk memperoleh gelar Sarjana Teknik pada Program Studi Teknik Komputer, Departemen Teknik Komputer, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember. Proses ini tidak akan berjalan lancar tanpa adanya bimbingan, dukungan, dan doa dari berbagai pihak. Oleh karena itu, pada kesempatan ini, penulis ingin mengucapkan terima kasih yang tulus kepada:

1. Keluarga tercinta, terutama Ayah, Ibu, dan seluruh saudara, yang telah memberikan dukungan moral, materiel, serta doa yang tiada henti sepanjang perjalanan studi penulis.
2. Bapak Reza Fuad Rachmadi, S.T., M.T., Ph.D, selaku Dosen Pembimbing I, dan Bapak Dr. Arief Kurniawan, S.T., M.T, selaku Dosen Pembimbing II, yang telah dengan sabar memberikan bimbingan, arahan, dan masukan yang sangat berharga sejak awal hingga akhir penyusunan tugas akhir ini.
3. Bapak Dr. Arief Kurniawan, S.T., M.T, selaku Kepala Departemen Teknik Komputer, serta seluruh jajaran dosen dan staf di lingkungan departemen yang telah memberikan bekal ilmu dan kemudahan administrasi selama masa perkuliahan.
4. Seluruh sahabat dan rekan-rekan seperjuangan di Departemen Teknik Komputer yang telah menjadi teman diskusi, memberikan semangat, dan saling membantu dalam suka maupun duka selama menempuh pendidikan.
5. Semua pihak yang tidak dapat penulis sebutkan satu per satu, yang telah memberikan kontribusi dan dukungan dalam penyelesaian laporan ini.

Penulis menyadari bahwa Laporan Tugas Akhir ini masih jauh dari kesempurnaan. Oleh karena itu, segala bentuk kritik dan saran yang membangun akan penulis terima dengan lapang dada demi perbaikan di masa mendatang. Akhir kata, semoga laporan ini dapat memberikan manfaat dan kontribusi bagi perkembangan ilmu pengetahuan, khususnya di bidang *caching system* dan *machine learning*.

Surabaya, Desember 2025

Muhammad Haekal Muhyidin Al-Araby

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

ABSTRAK i

ABSTRACT iii

KATA PENGANTAR v

DAFTAR ISI vii

DAFTAR GAMBAR ix

DAFTAR TABEL xi

1 PENDAHULUAN 1

1.1 Latar Belakang 1

1.2 Rumusan Masalah 2

1.3 Tujuan Penelitian 2

1.4 Batasan Masalah 3

1.5 Sistematika Penulisan 3

2 TINJAUAN PUSTAKA 5

2.1 Penelitian Terdahulu 5

2.2 Konsep Dasar 5

2.2.1 Cache 5

2.2.2 Algoritma Eviction 6

2.2.3 FIFO (First-In, First-Out) 6

2.2.4 LRU (Least Recently Used) 6

2.2.5 CLOCK 6

2.2.6 Offline CLOCK 6

2.2.7 Machine Learning 7

2.2.8 Supervised Learning 7

2.2.9 Trace Replay 7

2.2.10 LibCacheSim 7

2.2.11 ONNX (Open Neural Network Exchange) 7

2.2.12 ONNX Runtime 7

3 METODOLOGI **9**

3.1 Datasets 9

3.2 Pengembangan Perangkat Lunak Simulasi 9

3.3 Pengumpulan Data dan Labelling 11

3.4 Pengembangan Model Machine Learning 11

3.5 Implementasi Model pada Perangkat Lunak Simulasi 11

3.6 Evaluasi Model 12

3.7 Perangkat Keras dan Lunak 14

3.7.1 Perangkat Keras 14

3.7.2 Perangkat Lunak 14

4 PENGUJIAN DAN ANALISIS **15**

4.1 Perbandingan Kinerja Keseluruhan 15

4.2 Analisis Karakteristik Algoritma Pembanding 15

4.2.1 Optimal Zipf Promotion (Oracle Heuristik) 15

4.2.2 Offline CLOCK (Batas Atas Teoretis) 16

4.2.3 CLOCK Standar (Baseline) 16

4.3 Evaluasi Model Machine Learning 16

4.4 Analisis Fitur dan Konfigurasi Model 16

5 PENUTUP **19**

5.1 Kesimpulan 19

5.2 Saran 20

DAFTAR PUSTAKA **21**

BIOGRAFI PENULIS **25**

DAFTAR GAMBAR

3.1 Diagram alur dari program simulasi yang dikembangkan. 10
3.2 Diagram proses pelatihan model *machine learning*. 12
3.3 Arsitektur model *machine learning* yang digunakan. 12
3.4 Diagram alur implementasi model pada simulator. 13

4.1 Perbandingan Miss Ratio dan Total Promosi pada Ukuran Cache 1% 17
4.2 Perbandingan Miss Ratio dan Total Promosi pada Ukuran Cache 10% 17
4.3 Perbandingan Miss Ratio dan Total Promosi pada Ukuran Cache 20% 18
4.4 Perbandingan Miss Ratio dan Total Promosi pada Ukuran Cache 40% 18

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

4.1 Perbandingan Komprehensif: ML-CLOCK vs. Referensi 15

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Penelitian ini didorong oleh kemajuan pesat dalam teknologi komputasi modern, yang menuntut adanya teknik-teknik yang lebih canggih untuk manajemen *cache* objek. Dalam beberapa tahun terakhir, *machine learning* telah menjadi topik yang sangat populer dalam riset ilmu komputer. Berbagai studi telah dilakukan untuk memahami sejauh mana kapabilitas *machine learning* dapat diterapkan dalam bidang *caching* [1], [2], [3], [4], [5], [6]. Penelitian ini secara spesifik menyelidiki bagaimana *machine learning* dapat diimplementasikan dalam sistem *cache* untuk meningkatkan efisiensi algoritma populer seperti CLOCK terhadap distribusi zipfian [7].

1.1 Latar Belakang

Perkembangan teknologi komputasi telah melaju dengan kecepatan yang luar biasa selama dekade terakhir. Pada masa-masa awal, komputer terbatas pada prosesor *single-core* [8], [9], [10], [11]. Keterbatasan ini memaksa para insinyur dan pemrogram untuk merancang perangkat lunak yang dioptimalkan untuk eksekusi *single-threaded*. Seiring waktu, ketika peningkatan kinerja mulai melambat karena batasan fisik dan termal, industri beralih ke arsitektur *multi-core* [12], [13], [14], [15] dan paralel [16], [17], [18]. Transisi ini memperkenalkan tantangan baru dalam desain perangkat lunak, menuntut pengembang untuk mengadopsi model pemrograman konkuren dan paralel untuk memanfaatkan sepenuhnya kapabilitas perangkat keras modern. Akibatnya, pemahaman dan optimalisasi aplikasi *multi-threaded* telah menjadi aspek penting dalam riset dan pengembangan komputasi modern.

Caching telah lama menjadi bagian integral dari sistem komputer Mekanisme ini memainkan peran krusial dalam meningkatkan kinerja sistem dengan menyimpan data yang sering diakses di lokasi penyimpanan yang lebih cepat [8], [19], [20]. Selama bertahun-tahun, berbagai strategi *caching* telah dikembangkan untuk mengoptimalkan efisiensi pengambilan data di berbagai lingkungan, mulai dari sistem operasi dan server web hingga sistem terdistribusi dan infrastruktur *cloud* [21], [22], [23].

Arsitektur *multi-core*, yang telah menjadi standar industri, juga menghadirkan tantangan signifikan bagi teknik *caching* yang ada [24], [25], [26]. Dalam lingkungan seperti ini, beberapa inti prosesor sering kali mengakses dan memodifikasi data *cache* yang sama secara bersamaan, yang dapat menyebabkan kondisi balapan (*race conditions*) dan inkonsistensi data [13]. Untuk menjaga kebenaran data, sebagian besar implementasi *cache* tradisional menekankan penggunaan kunci (*locks*) selama operasi penyisipan, promosi, dan modifikasi [27], [28], [29]. Namun, penggunaan *lock* yang berlebihan dapat menimbulkan persaingan (*contention*) dan mengurangi paralelisme, yang pada akhirnya membatasi kinerja sistem pada prosesor *multi-core* modern.

Berbagai kebijakan penggantian *cache* (*cache replacement policies*) telah diusulkan untuk mencapai tujuan desain yang berbeda. Kebijakan **Least Recently Used (LRU)** umum digunakan karena kemampuannya untuk mencapai *miss ratio* yang rendah dengan mempertahankan data yang sering diakses [30], [31], [32]. Namun, LRU memerlukan pembaruan dan sinkronisasi yang sering untuk memelihara informasi kebaruan data secara presisi, yang meningkatkan

overhead dari *locking*. Di sisi lain, kebijakan yang lebih sederhana seperti **First-In First-Out (FIFO)** sering kali lebih disukai karena skalabilitasnya dan biaya sinkronisasi yang lebih rendah, karena hampir tidak memerlukan *locking* [33], [34], [35].

Untuk mengatasi keterbatasan LRU dan FIFO, algoritma **CLOCK** dikembangkan sebagai aproksimasi dari LRU yang tetap mempertahankan skalabilitas tinggi teTang1976Cache. CLOCK menggunakan *buffer* melingkar (*circular buffer*) dari entri *cache*, di mana setiap entri memiliki bit referensi yang menandakan akses terkini. Alih-alih memelihara daftar yang terurut sepenuhnya seperti LRU, algoritma ini menggerakkan ”jarum jam” untuk menemukan entri yang akan diganti, sambil membersihkan bit referensi di sepanjang jalan. Desain ini secara signifikan mengurangi kebutuhan akan *locking* yang berlebihan dan pembaruan yang sering, sambil tetap mempertahankan manfaat dari LRU. Hasilnya, CLOCK menawarkan keseimbangan praktis antara mencapai *miss ratio* yang rendah dan menjaga skalabilitas serta kesederhanaan yang menjadi ciri khas FIFO [36].

Namun, terlepas dari keunggulannya, algoritma CLOCK masih mengalami masalah promosi yang tidak perlu [36], [37]. Karena setiap akses akan mengatur bit referensi tanpa memandang frekuensi atau kepentingan akses, bahkan akses tunggal atau yang jarang terjadi dapat mencegah sebuah entri untuk digusur. Akibatnya, *cache* dapat mempertahankan item yang memberikan sedikit manfaat sambil menggusur item lain dengan potensi penggunaan kembali yang lebih tinggi. Hal ini menyebabkan inefisiensi dalam pemanfaatan *cache*.

Perkembangan terkini dalam *machine learning* telah membuka kemungkinan untuk menggunakannya sebagai komponen pengambilan keputusan tambahan untuk mengurangi promosi yang tidak perlu dalam sistem *cache*. Dengan belajar dari pola akses dan mengidentifikasi objek yang kemungkinan besar tidak akan digunakan kembali, model *machine learning* dapat membantu mencegah *cache* mempromosikan entri yang memberikan sedikit manfaat. Pendekatan ini memungkinkan mekanisme *caching* untuk membuat keputusan yang lebih selektif dan terinformasi tentang objek mana yang layak dipertahankan. Dengan demikian, *machine learning* dapat melengkapi algoritma tradisional seperti CLOCK dengan meningkatkan efisiensi, mengurangi *overhead* manajemen, dan beradaptasi secara dinamis terhadap karakteristik beban kerja yang berubah [38], [39], [40], [41].

1.2 Rumusan Masalah

Berdasarkan pembahasan pada Latar Belakang, masalah utama yang dibahas dalam penelitian ini adalah bagaimana mengintegrasikan teknik *machine learning* ke dalam algoritma penggantian *cache* CLOCK untuk mengurangi promosi yang tidak perlu dan meningkatkan efisiensi sistem secara keseluruhan. Secara spesifik, studi ini bertujuan untuk mengeksplorasi bagaimana pengambilan keputusan berbasis data dapat membantu algoritma CLOCK untuk lebih mendekati perilaku penggantian yang optimal. Dengan demikian, penelitian ini berupaya untuk meningkatkan akurasi dan skalabilitas dari manajemen *cache* berbasis CLOCK.

1.3 Tujuan Penelitian

Tujuan utama dari penelitian ini adalah sebagai berikut:

1. Mengintegrasikan model *machine learning* ke dalam algoritma penggantian *cache* CLOCK untuk menyempurnakan proses pengambilan keputusannya.
2. Mengevaluasi efisiensi dan efektivitas algoritma CLOCK berbasis *machine learning* dalam

mengurangi jumlah promosi dan meningkatkan kinerja *cache* secara keseluruhan.

1.4 Batasan Masalah

Studi ini berfokus pada integrasi teknik *machine learning* ke dalam algoritma penggantian *cache* CLOCK untuk meningkatkan efisiensinya. Penelitian ini terbatas pada lingkungan simulasi *cache* berbasis perangkat lunak, bukan implementasi perangkat keras. Studi ini mengevaluasi metrik kinerja seperti *miss ratio* dan tingkat promosi.

Ruang lingkup penelitian tidak mencakup pengembangan arsitektur *machine learning* baru atau perbandingan dengan sistem *caching* berbasis *reinforcement learning* [42] yang canggih. Sebaliknya, penelitian ini menekankan analisis tentang bagaimana model ringan yang ada dapat meningkatkan pengambilan keputusan di CLOCK tanpa meningkatkan kompleksitas secara signifikan. Penelitian ini hanya berfokus pada promosi dan miss ratio serta tidak menekankan performa asli pada perangkat keras nyata. Penelitian ini juga terbatas pada *datasets* berbasis zipfian.

1.5 Sistematika Penulisan

Laporan tugas akhir ini dibagi menjadi lima bab sebagai berikut:

1. **BAB I Pendahuluan**
   Bab ini menyajikan latar belakang studi, masalah utama yang akan dibahas, tujuan penelitian, dan ruang lingkup studi. Bab ini memberikan gambaran umum tentang motivasi dan signifikansi topik penelitian.

2. **BAB II Tinjauan Pustaka**
   Bab ini membahas karya-karya sebelumnya dan studi terkait yang menjadi landasan penelitian ini. Bab ini menyoroti algoritma penggantian *cache* yang ada, pengembangan algoritma CLOCK, dan aplikasi *machine learning* sebelumnya dalam sistem *caching*.

3. **BAB III Perancangan dan Implementasi Sistem**
   Bab ini menjelaskan desain dan implementasi sistem yang diusulkan. Bab ini menjelaskan arsitektur sistem secara keseluruhan, alur data, integrasi model *machine learning*, dan proses adaptasi model ke algoritma penggantian *cache* CLOCK.

4. **BAB IV Pengujian dan Analisis**
   Bab ini menyajikan prosedur pengujian, pengaturan eksperimental, dan metrik evaluasi yang digunakan untuk mengukur kinerja sistem. Bab ini juga memberikan analisis hasil, membandingkan pendekatan yang diusulkan dengan metode dasar dalam hal efisiensi dan pengurangan promosi.

5. **BAB V Penutup**
   Bab ini merangkum temuan dan kontribusi penelitian. Bab ini juga membahas kemungkinan perbaikan dan rekomendasi untuk pekerjaan di masa depan terkait manajemen *cache* berbasis *machine learning*.

[Halaman ini sengaja dikosongkan]

BAB II
TINJAUAN PUSTAKA

Bab ini menyajikan tinjauan mendalam mengenai konsep-konsep dasar dan penelitian terdahulu yang menjadi landasan bagi studi ini. Pembahasan mencakup prinsip-prinsip fundamental *caching*, algoritma penggantian yang sudah mapan, serta berbagai upaya sebelumnya dalam meningkatkan sistem manajemen *cache*.

2.1 Penelitian Terdahulu

Algoritma CLOCK memiliki sejarah panjang dalam dunia komputasi. Algoritma ini pertama kali dikembangkan pada awal tahun 1970-an oleh para peneliti IBM untuk memenuhi kebutuhan sistem penggantian halaman (*page-replacement*) pada komputer *mainframe* yang memiliki *overhead* lebih rendah dibandingkan dengan algoritma LRU yang populer pada masanya [43].

Sejak diperkenalkan, beberapa pengembangan signifikan telah dibangun di atas algoritma CLOCK asli. Salah satu contoh yang paling terkenal adalah **Clock with Adaptive Replacement (CAR)**. Algoritma ini secara cerdas memanfaatkan dua buah daftar terpisah untuk mengelola objek yang baru diakses dan objek yang sering diakses. Keunggulan utama CAR terletak pada kemampuannya untuk secara dinamis menyeimbangkan kedua daftar ini berdasarkan pada kebaruan (*recency*) dan frekuensi (*frequency*) akses objek. Hal ini memungkinkan CAR untuk beradaptasi secara lebih efektif terhadap perubahan pola akses data dibandingkan dengan algoritma CLOCK standar atau LRU [44].

Pengembangan penting lainnya adalah algoritma **Clock-Pro** [45]. Algoritma ini memperkenalkan pendekatan yang lebih kompleks dengan mengkategorikan objek ke dalam tiga daftar: *Hot*, *Cold*, dan *Test*. Objek yang sering diakses akan dipromosikan ke daftar *Hot*, di mana mereka mendapatkan perlindungan dari proses penggusuran (*eviction*). Namun, jika sebuah objek di daftar *Hot* tidak diakses dalam periode yang lama, objek tersebut akan diturunkan ke daftar *Cold*. Daftar *Cold* berisi objek-objek yang baru diakses dan menjadi kandidat utama untuk digusur. Yang paling unik adalah daftar *Test*, yang tidak menyimpan objek secara langsung, melainkan hanya metadata dari objek yang baru saja digusur. Jika sebuah objek yang metadatanya ada di daftar *Test* diakses kembali dalam waktu singkat setelah penggusurannya, objek tersebut akan langsung dipromosikan ke daftar *Hot*. Mekanisme ini memungkinkan Clock-Pro untuk mencapai keseimbangan yang baik antara kebaruan dan frekuensi, beradaptasi dengan berbagai pola akses, dan mencapai *hit ratio* yang sebanding dengan LRU namun dengan *overhead* yang tetap rendah seperti CLOCK.

2.2 Konsep Dasar

2.2.1 Cache

Cache adalah sebuah lapisan penyimpanan berukuran kecil namun berkecepatan sangat tinggi yang berfungsi untuk menyimpan data yang sering diakses. Tujuan utama dari penggunaan *cache* adalah untuk mempercepat waktu akses data, yang pada akhirnya akan meningkatkan

kinerja sistem secara keseluruhan [21], [22], [23]. Mekanisme *caching* diimplementasikan di hampir semua sistem komputasi modern untuk memastikan bahwa objek yang baru atau sering digunakan dapat diakses secara instan tanpa harus mengambilnya dari media penyimpanan yang lebih lambat atau melalui proses komputasi ulang yang memakan waktu.

2.2.2 Algoritma Eviction

Algoritma *eviction* atau *replacement* (penggusuran) adalah metode yang digunakan untuk menentukan data mana yang harus dikeluarkan dari *cache* ketika kapasitasnya sudah penuh dan ada data baru yang perlu dimasukkan [19]. Algoritma ini sangat krusial untuk menjaga efisiensi *cache* dengan memastikan bahwa data yang paling relevan tetap tersimpan sementara data yang sudah jarang digunakan dapat digantikan. Contoh umum dari algoritma ini adalah FIFO, LRU, dan CLOCK.

2.2.3 FIFO (First-In, First-Out)

FIFO adalah salah satu algoritma *eviction* yang paling sederhana [33], [46]. Sesuai dengan namanya, kebijakan ini akan mengeluarkan data yang pertama kali masuk ke dalam *cache*. FIFO tidak mempertimbangkan frekuensi atau kebaruan akses data, sehingga data yang paling lama berada di dalam *cache* akan selalu menjadi yang pertama untuk digantikan. Meskipun memiliki *overhead* paling rendah, algoritma ini umumnya memiliki *miss ratio* yang paling tinggi dibandingkan algoritma populer lainnya.

2.2.4 LRU (Least Recently Used)

LRU adalah algoritma *eviction* yang menggantikan data yang paling lama tidak diakses [33]. Algoritma ini beroperasi berdasarkan asumsi bahwa data yang baru saja diakses memiliki kemungkinan besar untuk diakses kembali dalam waktu dekat. LRU biasanya diimplementasikan menggunakan struktur data antrian (*queue*) di mana setiap objek yang diakses akan dipindahkan ke kepala antrian, sehingga melindunginya dari proses *eviction*.

2.2.5 CLOCK

Algoritma CLOCK adalah variasi dari kebijakan *eviction* yang memberikan ”kesempatan kedua” pada data sebelum dikeluarkan dari *cache* [36]. Setiap entri dalam *cache* memiliki sebuah bit referensi. Ketika proses *eviction* diperlukan, algoritma ini akan memeriksa bit referensi dari entri yang ditunjuk oleh ”jarum jam”. Jika bit referensi bernilai 0, data tersebut akan diganti. Jika bernilai 1, bit tersebut akan di-reset menjadi 0 dan jarum jam akan bergerak ke entri berikutnya. Proses ini terus berulang hingga ditemukan entri dengan bit referensi 0. Karena mekanisme ini, CLOCK sering disebut sebagai *Second Chance Algorithm*. Algoritma ini pada dasarnya berbasis FIFO, namun dengan modifikasi di mana item akan dimasukkan kembali ke kepala antrian jika bit referensinya 1, sehingga sering juga disebut sebagai *FIFO-Reinsertion*. Algoritma ini mengkombinasikan *hit ratio* tinggi dari LRU dengan *overhead* rendah dari FIFO.

2.2.6 Offline CLOCK

Offline CLOCK bukanlah algoritma praktis, melainkan sebuah tolok ukur teoretis. Dalam skenario ini, algoritma CLOCK dijalankan secara berulang-ulang pada set data dan konfigurasi yang sama. Pada setiap iterasi, setiap promosi objek yang ternyata tidak diakses kembali (promosi sia-sia) akan dicatat. Pada iterasi berikutnya, promosi sia-sia tersebut akan diabaikan. Proses ini secara dramatis mengurangi jumlah total promosi yang dilakukan dan digunakan

sebagai batas atas (*upper bound*) dari kinerja optimal yang bisa dicapai [36].

2.2.7 Machine Learning

Machine learning adalah cabang dari kecerdasan buatan (AI) yang berfokus pada pengembangan algoritma dan model yang memungkinkan komputer untuk ”belajar” dari data dan membuat prediksi atau keputusan tanpa diprogram secara eksplisit. Dalam *machine learning*, sebuah sistem mempelajari pola atau hubungan dari data latih (*training data*) untuk kemudian diterapkan pada data baru yang belum pernah dilihat sebelumnya [38], [39], [42].

2.2.8 Supervised Learning

Supervised learning (pembelajaran terarah) adalah salah satu pendekatan utama dalam *machine learning* di mana sebuah model dilatih menggunakan data yang sudah berlabel. Artinya, setiap data masukan (*input*) dipasangkan dengan keluaran (*output*) yang sudah diketahui. Tujuannya adalah untuk menciptakan sebuah model yang mampu memetakan *input* ke *output* secara akurat, sehingga dapat digunakan untuk melakukan prediksi pada data baru [38].

2.2.9 Trace Replay

Trace replay adalah sebuah metode yang umum digunakan dalam evaluasi sistem komputer, khususnya dalam penelitian terkait *cache*, memori, atau sistem file [47]. Dalam metode ini, urutan akses data (disebut *trace*) dari sebuah sistem nyata direkam terlebih dahulu, kemudian ”diputar ulang” (*replayed*) pada sistem atau model yang sedang diuji. *Trace replay* memungkinkan peneliti untuk menganalisis kinerja sebuah algoritma dengan skenario akses data yang realistis tanpa harus menjalankan aplikasi aslinya secara langsung.

2.2.10 LibCacheSim

LibCacheSim adalah sebuah pustaka (*library*) yang digunakan untuk melakukan simulasi *cache* pada sistem komputer [47]. Pustaka ini memungkinkan peneliti dan pengembang untuk menguji dan menganalisis kinerja berbagai kebijakan penggantian *cache* (seperti LRU, FIFO, CLOCK) dalam berbagai skenario akses data.

2.2.11 ONNX (Open Neural Network Exchange)

ONNX adalah sebuah format *open-source* yang dirancang untuk merepresentasikan model *machine learning* [48], [49], [50]. Format ini berfungsi sebagai standar terbuka yang memungkinkan pengembang untuk memindahkan model antar-kerangka kerja (-framework) *machine learning* yang berbeda, seperti PyTorch dan TensorFlow [51]. Tujuannya adalah untuk memfasilitasi interoperabilitas, sehingga model yang dilatih di satu *framework* dapat dengan mudah dijalankan di *framework* lain [50].

2.2.12 ONNX Runtime

ONNX Runtime adalah sebuah *inference engine* berkinerja tinggi yang dikembangkan oleh Microsoft untuk menjalankan model yang disimpan dalam format ONNX [48], [49], [50], [51]. Tujuannya adalah untuk memaksimalkan kinerja inferensi (proses penggunaan model yang sudah dilatih) di berbagai jenis perangkat keras (CPU, GPU, dll.) dan sistem operasi.

[Halaman ini sengaja dikosongkan]

BAB III
METODOLOGI

Bab ini merinci metodologi yang digunakan dalam penelitian ini, mulai dari datasets, perancangan dan pengembangan perangkat lunak simulasi, proses pengumpulan dan pengolahan data, hingga implementasi dan evaluasi model *machine learning*.

3.1 Datasets

Dalam penelitian ini, penulis menggunakan dataset berupa trace sintetis yang mengikuti distribusi Zipfian. Pemilihan distribusi Zipfian ini didasarkan pada karakteristiknya yang sangat menyerupai pola akses pada web trace di dunia nyata, di mana sebagian kecil konten diakses jauh lebih sering dibandingkan sisanya (popularitas yang tidak merata) [7].

Seluruh dataset dibuat secara prosedural menggunakan utilitas skrip dari pustaka libCacheSim. Pendekatan ini dipilih penulis karena menawarkan fleksibilitas yang lebih baik dibandingkan menggunakan log server mentah. Dengan menggunakan data sintetis, penulis dapat menghilangkan noise atau gangguan yang sering terdapat pada data riil, serta memiliki kontrol penuh terhadap parameter distribusi untuk mensimulasikan berbagai kondisi beban kerja yang spesifik.

Secara struktural, dataset ini disimpan menggunakan format oracleGeneral. Dalam format ini, data disusun sebagai urutan akses di mana setiap entri merepresentasikan satu permintaan. Komponen utama dalam struktur ini adalah kolom object_id, yang berisi identifikasi numerik unik untuk setiap objek yang diminta. Format ini memastikan kompatibilitas langsung dengan simulator tanpa memerlukan langkah pra-pemrosesan yang kompleks.

Untuk keperluan eksperimen, penulis membuat beberapa set trace Zipfian yang berbeda secara terpisah. Pembagian antara data untuk pelatihan (training) dan pengujian (testing) dilakukan secara ketat dan berbeda file. Hal ini krusial untuk memastikan bahwa evaluasi performa model dilakukan pada pola akses baru yang belum pernah dilihat sebelumnya. Tujuannya adalah untuk menguji kemampuan generalisasi sistem dalam menangani trafik, bukan sekadar kemampuan model dalam menghafal urutan data latih.

3.2 Pengembangan Perangkat Lunak Simulasi

Langkah pertama dalam penelitian ini adalah pengembangan perangkat lunak simulasi kustom yang mampu memenuhi kebutuhan analisis. Perangkat lunak ini dirancang dengan beberapa persyaratan utama:

1. **Integrasi Model Machine Learning:** Perangkat lunak harus mampu memanggil dan menjalankan model *machine learning* secara efisien di dalam fungsi *eviction* dari algoritma *cache*.

2. **Dukungan Algoritma Pembanding:** Selain algoritma yang diusulkan, perangkat lunak harus dapat menjalankan algoritma CLOCK standar dan Offline CLOCK sebagai tolok ukur perbandingan kinerja.

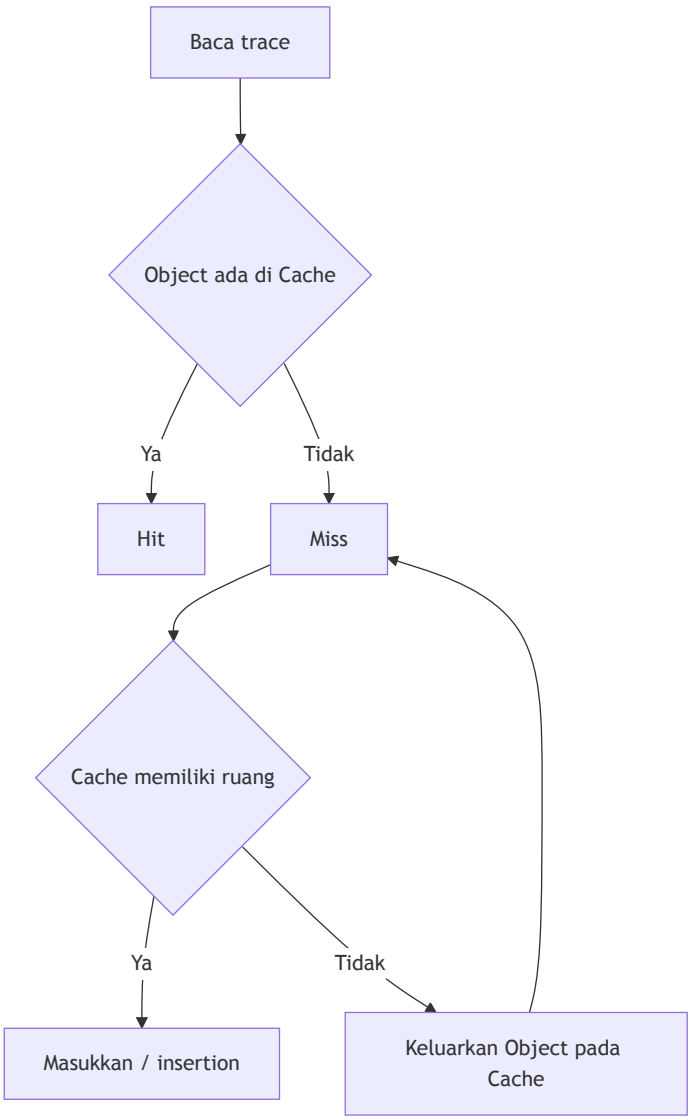
3. **Pengumpulan Data dari Offline CLOCK:** Perangkat lunak harus mampu menjalankan mode Offline CLOCK untuk mengumpulkan data-data krusial, antara lain:

   (a) Nilai *miss ratio* dan jumlah promosi untuk setiap iterasi simulasi.

   (b) Metadata dari setiap objek pada saat keputusan promosi dibuat, beserta label apakah promosi tersebut pada akhirnya sia-sia (*wasted*) atau tidak.

   (c) Kemampuan untuk mengekspor semua data yang terkumpul ke dalam format CSV (*Comma-Separated Values*) untuk memfasilitasi proses pengolahan dan analisis data lebih lanjut.

Untuk mempercepat proses pengembangan, penulis memanfaatkan **LibCacheSim**, sebuah pustaka sumber terbuka yang dirancang khusus untuk pengembangan simulator *cache*. Pustaka ini ditulis dalam bahasa C/C++, yang menjamin kinerja komputasi yang sangat efisien, terutama untuk tugas-tugas berat seperti pengumpulan data dari *trace* berukuran besar.



Gambar 3.1: Diagram alur dari program simulasi yang dikembangkan.

3.3 Pengumpulan Data dan Labelling

Pengumpulan data awal dilakukan sepenuhnya menggunakan perangkat lunak simulasi yang telah dikembangkan. Penulis menjalankan simulasi ini dengan tujuan utama untuk mendapatkan metrik kinerja dasar seperti jumlah promosi total dan *miss ratio* yang dihasilkan oleh algoritma CLOCK standar. Angka-angka ini berfungsi sebagai *baseline* untuk mengukur efektivitas model yang diusulkan nantinya.

Bersamaan dengan simulasi baseline tersebut, penulis melakukan proses pembuatan dataset yang akan digunakan untuk melatih model *machine learning*. Dataset ini disusun dengan cara mencatat (*logging*) metadata relevan dari setiap objek tepat pada saat objek tersebut menjadi kandidat untuk keputusan promosi. Fitur-fitur yang dicatat meliputi statistik akses masa lalu dan status objek dalam antrian saat itu.

Pemberian label pada setiap sampel data dilakukan dengan merujuk pada simulasi *Offline CLOCK* yang dijalankan setelahnya. Karena *Offline CLOCK* memiliki visibilitas terhadap seluruh *trace* masa depan, algoritma ini dapat menentukan keputusan optimal secara retrospektif. Penulis memberikan label `wasted` pada objek yang dipromosikan tetapi tidak memberikan *hit* sebelum akhirnya dikeluarkan (*evict*) kembali dari cache. Sebaliknya, label `not wasted` diberikan jika promosi tersebut terbukti berguna dan menghasilkan *hit*.

3.4 Pengembangan Model Machine Learning

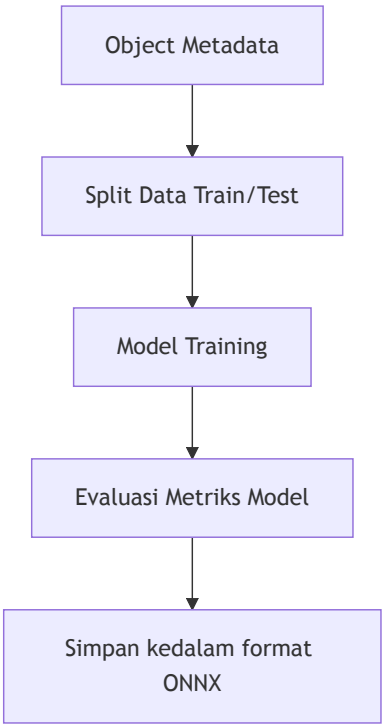
Berdasarkan dataset terlabel yang telah disiapkan sebelumnya, penulis mengembangkan sebuah model *machine learning* sebagai mesin inferensi utama. Model ini didesain secara spesifik untuk memproses vektor fitur yang diekstraksi dari metadata objek secara *real-time* tanpa membebani latensi sistem. Variabel masukan yang digunakan meliputi frekuensi akses historis, durasi waktu sejak akses terakhir atau *recency*, serta ukuran objek dalam byte.

Sebagai keluaran, model menghasilkan skor probabilitas kontinu dengan rentang nilai antara 0 hingga 1. Dalam konteks klasifikasi biner ini, nilai probabilitas ditafsirkan sebagai tingkat keyakinan terhadap label `wasted`. Artinya, nilai keluaran yang mendekati 1 mengindikasikan prediksi kuat bahwa objek tersebut tidak akan memberikan manfaat atau *hit* jika dipromosikan ke *cache* utama. Sebaliknya, nilai yang rendah menandakan bahwa objek tersebut memiliki potensi utilitas yang tinggi dan layak untuk mendapatkan promosi.

3.5 Implementasi Model pada Perangkat Lunak Simulasi

Setelah tahap pelatihan selesai, model *machine learning* yang dihasilkan diintegrasikan kembali ke dalam perangkat lunak simulasi untuk pengujian kinerja. Tantangan teknis utama pada tahap ini adalah menjembatani perbedaan lingkungan antara proses pelatihan yang berbasis Python dengan lingkungan simulasi yang ditulis dalam C++ demi efisiensi memori dan waktu eksekusi. Solusi yang diterapkan penulis adalah dengan mengekspor model tersebut ke dalam format **ONNX (Open Neural Network Exchange)**. Format ini dipilih karena standar interoperabilitasnya yang tinggi, memungkinkan model yang dilatih pada *framework* tingkat tinggi untuk dijalankan pada lingkungan C++ tanpa perlu menulis ulang logika model secara manual.

Eksekusi model di dalam simulator dilakukan menggunakan **ONNX Runtime**, sebuah *inference engine* yang dioptimalkan untuk kinerja tinggi dan latensi rendah. Secara arsitektural,



Gambar 3.2: Diagram proses pelatihan model *machine learning*.

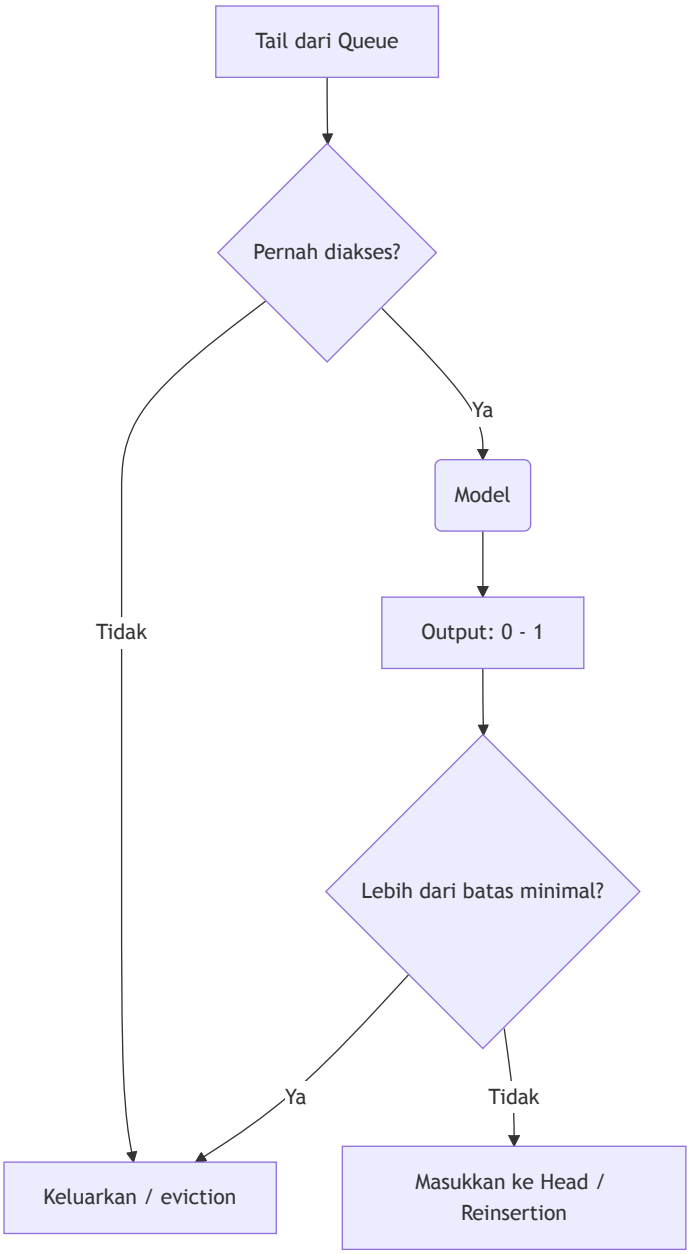
Gambar 3.3: Arsitektur model *machine learning* yang digunakan.

modul inferensi ini ditanamkan langsung ke dalam logika internal algoritma CLOCK, spesifiknya pada fungsi yang menangani mekanisme *eviction* dan promosi. Setiap kali sistem mempertimbangkan sebuah objek untuk dipromosikan ke segmen memori utama, simulator secara *real-time* mengekstrak metadata objek tersebut dan mengubahnya menjadi tensor input yang sesuai dengan spesifikasi model ONNX.

Keputusan akhir ditentukan berdasarkan perbandingan antara keluaran model dengan nilai ambang batas (*threshold*) sensitivitas yang telah dikonfigurasi sebelumnya. Model menghasilkan skor probabilitas yang merepresentasikan kecenderungan objek untuk menjadi `wasted`. Jika skor probabilitas ini melampaui ambang batas, sistem akan mengambil tindakan preventif dengan menggagalkan promosi dan langsung menggusur objek tersebut (*bypass*). Mekanisme ini bertujuan untuk mencegah polusi *cache* oleh objek-objek yang diprediksi tidak akan memberikan keuntungan kinerja di masa depan.

3.6 Evaluasi Model

Tahap final dari rangkaian penelitian ini adalah evaluasi komprehensif terhadap kinerja model yang telah diintegrasikan ke dalam simulator. Untuk mengukur efektivitas solusi yang diusulkan, penulis melakukan perbandingan langsung (*benchmarking*) antara algoritma CLOCK



Gambar 3.4: Diagram alur implementasi model pada simulator.

yang dimodifikasi dengan tiga standar referensi utama.

Referensi pertama adalah algoritma CLOCK standar tanpa modifikasi. Ini berfungsi sebagai *baseline* untuk melihat peningkatan kinerja riil dibandingkan metode konvensional yang ada saat ini. Referensi kedua adalah *Offline CLOCK*, sebuah algoritma teoretis yang diasumsikan memiliki pengetahuan sempurna akan masa depan. Perbandingan ini krusial untuk mengetahui seberapa dekat kinerja model prediksi dengan batas optimal yang mungkin dicapai (*upper bound*).

Referensi ketiga adalah strategi *Optimal Zipf Promotion*. Metode ini memanfaatkan pengetahuan *oracle* mengenai distribusi popularitas global dataset. Dalam strategi ini, promosi hanya diberikan kepada sebagian kecil objek yang berada di puncak kurva popularitas Zipfian (objek-

objek terpopuler). Perbandingan ini bertujuan untuk memverifikasi apakah strategi berbasis popularitas statis sudah cukup efisien, atau apakah pendekatan dinamis berbasis *machine learning* mampu menangkap pola akses yang lebih kompleks yang terlewatkan oleh metrik popularitas semata.

Metrik evaluasi difokuskan pada dua indikator utama. Indikator pertama adalah *miss ratio*, yang merepresentasikan efisiensi dasar *cache* dalam melayani permintaan pengguna. Indikator kedua adalah jumlah total promosi ke segmen utama. Metrik ini menjadi fokus khusus karena berkorelasi lurus dengan operasi tulis (*write operations*). Keberhasilan model didefinisikan sebagai kemampuannya menekan jumlah promosi secara signifikan untuk menghemat *bandwidth* dan umur perangkat, tanpa menyebabkan degradasi yang berarti pada *miss ratio*.

3.7 Perangkat Keras dan Lunak

Penelitian ini didukung oleh infrastruktur komputasi dan tumpukan perangkat lunak sebagai berikut:

3.7.1 Perangkat Keras

Testbed Utama PC server berkinerja tinggi yang ditenagai oleh prosesor Intel Xeon Silver 4114 (40 *logical cores* @ 3.00 GHz) dan memori RAM sebesar 192 GiB. Perangkat ini didedikasikan untuk menjalankan simulasi paralel yang intensif dan memuat dataset *trace* berukuran besar ke dalam memori.

Unit Pengembangan Lokal Laptop pribadi yang digunakan untuk tahap pengembangan kode awal, *prototyping*, serta analisis hasil pasca-simulasi.

3.7.2 Perangkat Lunak

C++23 Bahasa pemrograman utama untuk implementasi simulator. Dipilih untuk memaksimalkan efisiensi eksekusi waktu nyata dan kontrol memori tingkat rendah.

Python 3.13 Bahasa skrip yang digunakan untuk orkestrasi eksperimen, pemrosesan data awal (*preprocessing*), serta pelatihan model *machine learning*.

libCacheSim Pustaka simulator *cache* yang menjadi fondasi dasar pengembangan sistem, menyediakan implementasi struktur data *cache* yang efisien.

ONNX Runtime Mesin inferensi (*inference engine*) berkinerja tinggi yang digunakan untuk menjalankan model prediksi di dalam lingkungan C++ dengan latensi minimal.

Pandas & Plotly Pustaka Python yang digunakan untuk analisis statistik dataset dan visualisasi data hasil eksperimen.

OpenSSH Digunakan untuk akses jarak jauh, manajemen *job*, dan transfer data antara unit lokal dan *testbed*.

BAB IV
PENGUJIAN DAN ANALISIS

Bagian ini menguraikan hasil evaluasi kuantitatif dari model ML-CLOCK yang diusulkan. Kinerja model dibandingkan dengan tiga standar: CLOCK Standar (Baseline), Optimal Zipf Promotion (Heuristik Oracle Sederhana), dan Offline CLOCK (Batas Atas Teoretis). Pengujian dilakukan pada empat variasi ukuran *cache*: 1%, 10%, 20%, dan 40% dari total ukuran dataset unik.

4.1 Perbandingan Kinerja Keseluruhan

Tabel 4.1 menyajikan rangkuman lengkap kinerja seluruh metode. Nilai negatif pada *Delta Miss Ratio* menandakan perbaikan (penurunan miss), sedangkan nilai positif menandakan degradasi kinerja.

Tabel 4.1: Perbandingan Komprehensif: ML-CLOCK vs. Referensi

Ukuran Cache	Metode	Delta Miss Ratio	Reduksi Promosi
1%	ML-CLOCK (Usulan)	-0.05%	10%
	Optimal Zipf	**-0.20%**	10%
	Offline CLOCK	-0.35%	40%
10%	ML-CLOCK (Usulan)	-0.10%	10%
	Optimal Zipf	**-0.90%**	13%
	Offline CLOCK	-1.80%	43%
20%	ML-CLOCK (Usulan)	-0.20%	10%
	Optimal Zipf	**-1.67%**	15%
	Offline CLOCK	-3.50%	43%
40%	ML-CLOCK (Usulan)	+0.15% (Regresi)	8%
	Optimal Zipf	**-3.30%**	17%
	Offline CLOCK	-8.00%	43%

4.2 Analisis Karakteristik Algoritma Pembanding

Untuk menempatkan kinerja ML-CLOCK dalam perspektif yang objektif, karakteristik fundamental dari ketiga algoritma pembanding harus dipahami sebagai berikut:

4.2.1 Optimal Zipf Promotion (Oracle Heuristik)

Strategi ini bertindak sebagai algoritma *Oracle* yang ”curang” karena memanfaatkan **pengetahuan masa depan yang lengkap** mengenai distribusi data. Algoritma ini secara eksplisit

membaca identitas objek (`objectId`) dan hanya mempromosikan objek yang diketahui berada di puncak kurva popularitas Zipfian. Dalam skenario dunia nyata, sistem tidak memiliki akses ke ”kunci jawaban” berupa distribusi probabilitas global ini. Oleh karena itu, Optimal Zipf memberikan batas kinerja heuristik yang sangat tinggi namun tidak praktis untuk implementasi nyata.

4.2.2 Offline CLOCK (Batas Atas Teoretis)

Offline CLOCK merupakan representasi dari kesempurnaan teoretis (mirip dengan algoritma Belady's OPT). Algoritma ini bekerja dengan cara memindai seluruh *trace* akses di masa depan sebelum simulasi dimulai. Keputusan promosi diambil dengan logika deterministik sempurna: sebuah objek hanya akan dipromosikan jika dan hanya jika objek tersebut dipastikan akan diakses kembali (*hit*) sebelum waktu penggusurannya tiba. Kesenjangan kinerja yang masif antara ML-CLOCK dan Offline CLOCK (43% reduksi promosi) menunjukkan batas fisik maksimal yang bisa dicapai jika prediksi masa depan memiliki akurasi 100%.

4.2.3 CLOCK Standar (Baseline)

Algoritma ini mewakili pendekatan konvensional tanpa kecerdasan buatan. Ia bekerja secara reaktif murni berdasarkan prinsip *Recency* (kebaruan). Kelemahan fatalnya adalah ketidakmampuan membedakan objek populer dengan objek yang hanya lewat sekali (*one-hit wonders*). Akibatnya, CLOCK Standar sering melakukan promosi yang sia-sia (*wasted promotions*), yang memboroskan siklus penulisan memori dan mencemari *cache* dengan data sampah.

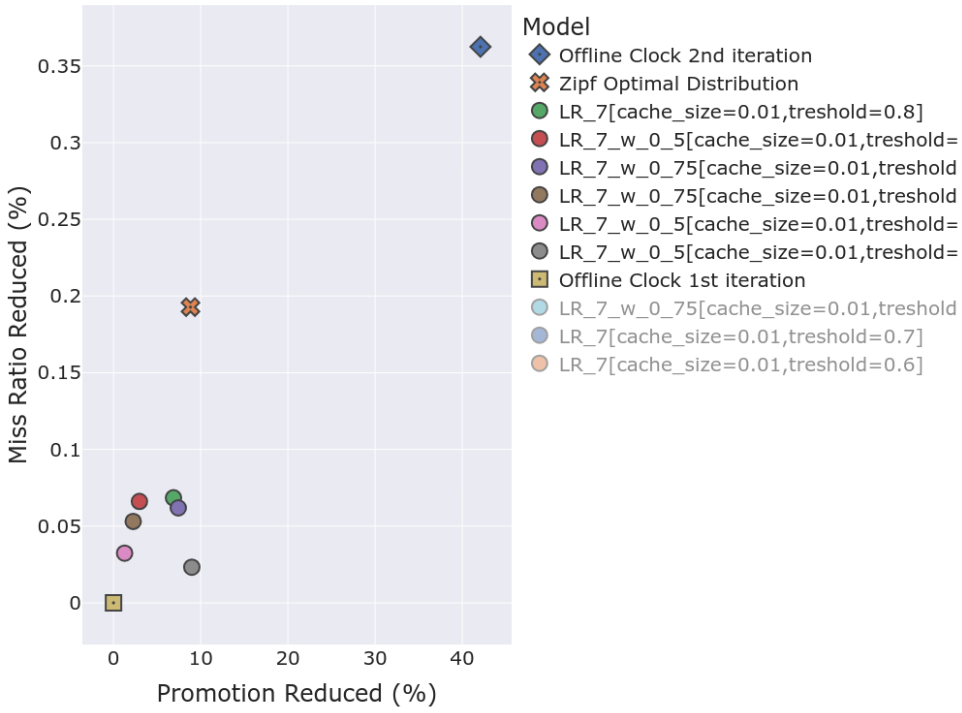
4.3 Evaluasi Model Machine Learning

Meskipun kalah dari strategi Oracle dan Offline yang memiliki pengetahuan masa depan, model ML-CLOCK menunjukkan karakteristik kinerja yang spesifik sebagai solusi *online*:

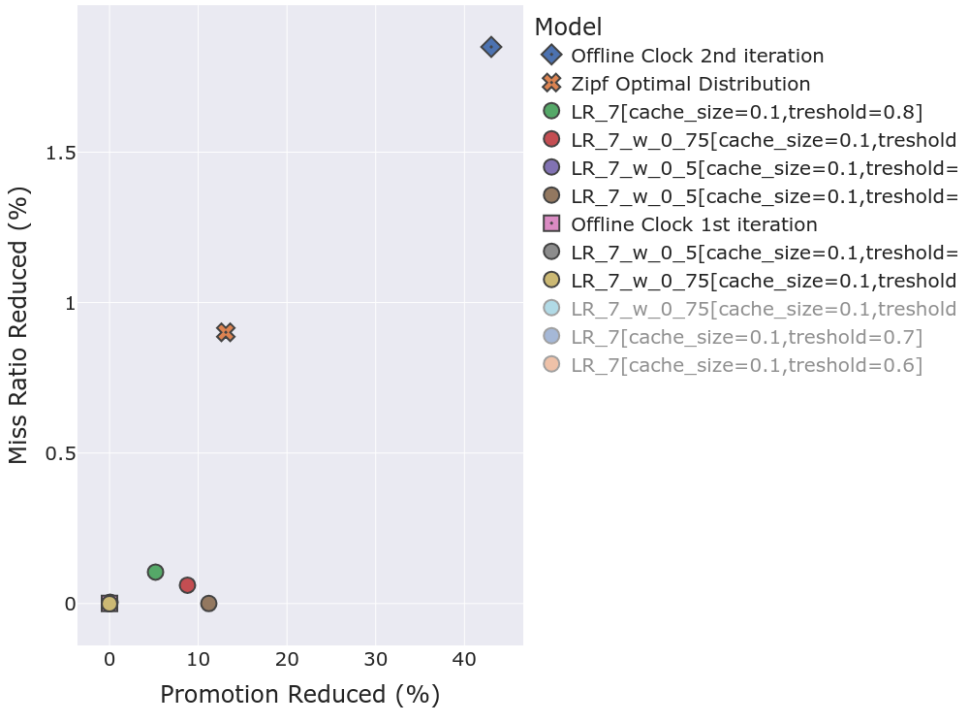
1. **Efektivitas Terbatas pada Cache Kecil:** Pada ukuran 1% hingga 20%, model berhasil memberikan kontribusi positif meskipun marjinal. Penurunan *miss ratio* sebesar 0.05% – 0.20% membuktikan bahwa model mampu belajar membedakan objek *wasted*.
2. **Kegagalan Generalisasi pada Cache Besar:** Pada ukuran 40%, terjadi regresi kinerja yang signifikan. Ambang batas (*threshold*) klasifikasi model mungkin kurang adaptif untuk kapasitas penyimpanan yang sangat besar.
3. **Peran Vital Bobot dan Threshold:** Tanpa penerapan bobot *loss* kecil (0.5) untuk kelas `wasted` dan ambang batas prediksi tinggi (0.6 – 0.8), model menjadi terlalu agresif menolak promosi (*aggressive eviction*). Mekanisme kontrol ini mencegah model membuang terlalu banyak objek potensial yang akan menghancurkan kinerja *miss ratio*.

4.4 Analisis Fitur dan Konfigurasi Model

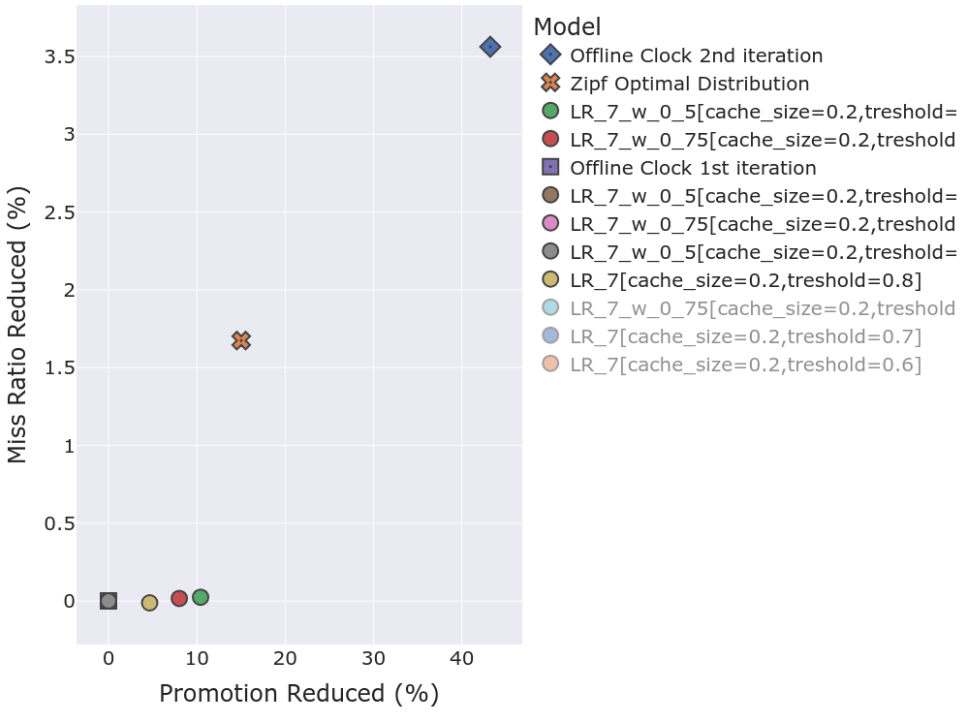
Model terbaik yang dihasilkan menggunakan kombinasi fitur frekuensi (`lifetime_freq`, `cache_freq`) dan kebaruan (`vtime_since_access`). Penting dicatat bahwa konfigurasi ambang batas tinggi (0.6 – 0.8) berfungsi sebagai mekanisme pertahanan (*safeguard*). Model memiliki bias inheren untuk mengklasifikasikan objek sebagai `wasted`. Tanpa ambang batas yang ketat ini, sistem akan mengalami *over-eviction*, di mana penghematan promosi mungkin melonjak drastis, namun dibayar mahal dengan degradasi performa layanan (*miss ratio*).



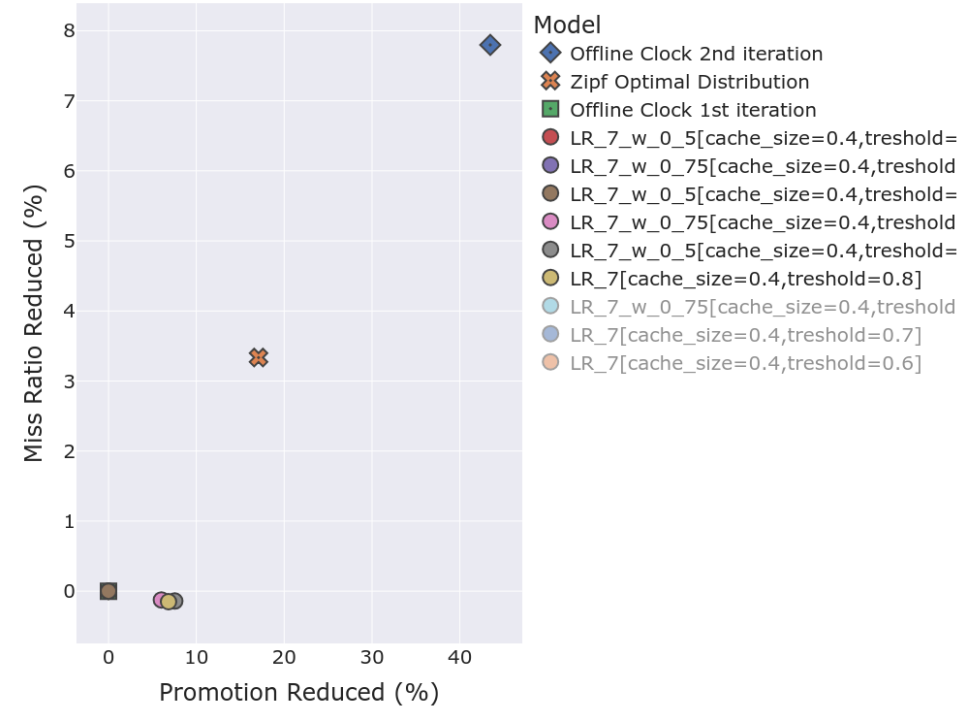
Gambar 4.1: Perbandingan Miss Ratio dan Total Promosi pada Ukuran Cache 1%



Gambar 4.2: Perbandingan Miss Ratio dan Total Promosi pada Ukuran Cache 10%



Gambar 4.3: Perbandingan Miss Ratio dan Total Promosi pada Ukuran Cache 20%



Gambar 4.4: Perbandingan Miss Ratio dan Total Promosi pada Ukuran Cache 40%

BAB V
PENUTUP

Bab ini merangkum temuan-temuan kunci yang diperoleh dari eksperimen integrasi *machine learning* ke dalam algoritma manajemen *cache* CLOCK, serta memberikan rekomendasi strategis untuk pengembangan penelitian selanjutnya.

5.1 Kesimpulan

Berdasarkan analisis data pengujian dan evaluasi kinerja yang telah dipaparkan pada bab sebelumnya, penulis menarik beberapa kesimpulan kritis sebagai berikut:

1. **Efisiensi Penulisan Terbukti Namun Terbatas.** Implementasi ML-CLOCK berhasil membuktikan hipotesis awal bahwa model prediktif dapat mengurangi operasi penulisan (*promotions*) ke memori utama. Pengurangan konsisten sebesar 8–10% tercapai di seluruh variasi ukuran *cache*. Meskipun angka ini signifikan untuk memperpanjang umur perangkat penyimpanan (*endurance*), capaian ini masih jauh di bawah batas optimal teoretis *Offline CLOCK* yang mencapai 43%. Hal ini menunjukkan bahwa model masih bersikap konservatif dalam pengambilan keputusan.

2. **Trade-off Kinerja pada Skala Besar.** Terdapat pola kinerja non-linear yang jelas. Pada lingkungan dengan tekanan memori tinggi (ukuran *cache* 1% – 20%), ML-CLOCK memberikan keuntungan ganda: pengurangan *miss ratio* dan penghematan promosi. Namun, terjadi regresi kinerja pada ukuran *cache* 40%, di mana model menjadi terlalu selektif (*over-selective*) sehingga membuang objek yang seharusnya disimpan saat ruang tersedia. Ini menegaskan bahwa pendekatan ML statis tanpa adaptasi dinamis terhadap ketersediaan ruang memiliki risiko pada kapasitas besar.

3. **Superioritas Oracle Mengungkap Batas Prediksi Lokal.** Dominasi strategi *Optimal Zipf Promotion* terhadap ML-CLOCK di seluruh metrik mengkonfirmasi bahwa dalam distribusi Zipfian, pengetahuan global mengenai popularitas objek jauh lebih berharga daripada pola akses lokal (*recency* dan frekuensi jangka pendek). ML-CLOCK yang hanya mengandalkan metadata lokal kesulitan untuk mereplikasi akurasi yang dimiliki oleh algoritma berbasis pengetahuan global tersebut.

4. **Kontribusi Arsitektur Integrasi ONNX.** Terlepas dari kinerja model prediksi spesifik yang digunakan, kontribusi teknis utama dari penelitian ini adalah keberhasilan pembangunan kerangka kerja (*framework*) simulasi hibrida. Mekanisme integrasi model Python ke dalam lingkungan C++ berkinerja tinggi menggunakan format **ONNX (Open Neural Network Exchange)** terbukti andal, stabil, dan minim latensi. Infrastruktur ini memecahkan hambatan bahasa pemrograman yang selama ini menyulitkan peneliti sistem operasi untuk mengadopsi model *deep learning* modern.

5.2 Saran

Penelitian ini membuka jalan bagi eksplorasi lebih lanjut dalam domain *AI-driven cache replacement*. Untuk mengatasi keterbatasan yang ditemukan, penulis mengajukan saran-saran berikut:

1. **Pemanfaatan Kerangka Kerja ONNX untuk Riset Lanjutan.** Arsitektur simulasi yang telah dibangun dengan integrasi *ONNX Runtime* dirancang agar bersifat modular dan agnostik terhadap jenis model. Peneliti lain sangat disarankan untuk menggunakan kembali (*reuse*) infrastruktur ini untuk menguji arsitektur model yang berbeda (seperti LSTM, Transformer, atau Reinforcement Learning) tanpa perlu membangun ulang simulator dari nol. Kode sumber simulator dapat dijadikan basis standar (*testbed*) untuk komunitas riset manajemen memori.

2. **Penggunaan Trace Dunia Nyata (Real-world Workloads).** Evaluasi pada dataset sintetis Zipfian memiliki keterbatasan dalam menangkap dinamika temporal yang kompleks, seperti *burstiness* atau perubahan fase popularitas. Penelitian selanjutnya disarankan menggunakan *trace* produksi dari CDN (Content Delivery Network) atau basis data skala besar. Data riil akan memberikan tantangan yang lebih adil bagi model ML dibandingkan distribusi statis Zipfian yang cenderung menguntungkan metode heuristik sederhana.

3. **Eksplorasi Fitur Non-Linear.** Fitur `vtime_since_access` terbukti kurang efektif dan berpotensi menjadi *noise* pada distribusi Zipfian. Disarankan untuk mengeksplorasi fitur yang lebih kompleks, seperti distribusi waktu antar-kedatangan (*inter-arrival time distribution*) atau fitur berbasis grafik relasi antar-objek, untuk menangkap pola akses yang tidak terdeteksi oleh statistik frekuensi sederhana.

4. **Mekanisme Threshold Adaptif.** Untuk mengatasi masalah regresi pada *cache* ukuran besar, model statis dengan ambang batas tetap (0.6 – 0.8) harus ditinggalkan. Disarankan pengembangan mekanisme *adaptive thresholding* di mana ambang batas penerimaan objek diturunkan secara otomatis ketika tekanan pada memori rendah (banyak ruang kosong) dan dinaikkan ketika memori penuh, menyerupai perilaku kontrol umpan balik (*feedback loop*).

DAFTAR PUSTAKA

[1] H. Herodotou, "AutoCache: Employing Machine Learning to Automate Caching in Distributed File Systems," 2019. DOI: https://doi.org/10.1109/icdew.2019.00-21.

[2] S. Sethumurugan, J. Yin, and J. Sartori, "Designing a Cost-Effective Cache Replacement Policy using Machine Learning," 2021. DOI: https://doi.org/10.1109/hpca51647.2021.00033.

[3] H. Sun, Q. L. Cui, J. Huang, and X. Qin, "NCache: A Machine-Learning Cache Management Scheme for Computational SSDs," 2022. DOI: https://doi.org/10.1109/tcad.2022.3208769.

[4] J. Shuja, K. Bilal, W. Alasmary, H. Sinky, and E. Alanazi, "Applying machine learning techniques for caching in next-generation edge networks: A comprehensive survey," 2021. DOI: https://doi.org/10.1016/j.jnca.2021.103005.

[5] H. Wang, X. Yi, P. Huang, B. Cheng, and K. Zhou, "Efficient SSD Caching by Avoiding Unnecessary Writes using Machine Learning," 2018. DOI: https://doi.org/10.1145/3225058.3225126.

[6] D. S. Berger, "Towards Lightweight and Robust Machine Learning for CDN Caching," 2018. DOI: https://doi.org/10.1145/3286062.3286082.

[7] L. Breslau, P. Cao, F. Li, G. M. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: evidence and implications," 1999. DOI: https://doi.org/10.1109/infcom.1999.749260.

[8] J. S. Liptay, "Structural aspects of the System/360 Model 85, II: The cache," 1968. DOI: https://doi.org/10.1147/sj.71.0015.

[9] A. Turing, "Lecture on the Automatic Computing Engine (1947)," 2004. DOI: https://doi.org/10.1093/oso/9780198250791.003.0015.

[10] M. Shima, "The Birth, Evolution and Future of the Microprocessor," 2005. DOI: https://doi.org/10.1109/cit.2005.182.

[11] T. Bäck, U. Hammel, and H.-P. Schwefel, "Evolutionary computation: comments on the history and current state," 1997. DOI: https://doi.org/10.1109/4235.585888.

[12] G. M. Papadopoulos and K. R. Traub, "Multithreading," 1991. DOI: https://doi.org/10.1145/115952.115986.

[13] M. Herlihy, "The art of multiprocessor programming," 2006. DOI: https://doi.org/10.1145/1146381.1146382.

[14] J. M. Borkenhagen, R. J. Eickemeyer, R. Kalla, and S. R. Kunkel, "A multithreaded PowerPC processor for commercial servers," 2000. DOI: https://doi.org/10.1147/rd.446.0885.

[15] A. B. Abdallah, "Advanced Multicore Systems-On-Chip," 2017. DOI: https://doi.org/10.1007/978-981-10-6092-2.

[16] A. C. Sodan, J. Machina, A. Deshmeh, K. Macnaughton, and B. Esbaugh, "Parallelism via Multithreaded and Multicore CPUs," 2010. DOI: https://doi.org/10.1109/mc.2010.75.

[17] J. Kim, S. Y. Yu, and J. Park, "Performance Evaluation of Multithreaded Computations for CPU Bounded Task," 2016. DOI: https://doi.org/10.1109/platcon.2016.7456816.

[18] R. J. Eickemeyer, R. E. Johnson, S. R. Kunkel, M. S. Squillante, and S. Liu, "Evaluation of multithreaded uniprocessors for commercial application environments," 1996. DOI: https://doi.org/10.1145/232973.232994.

[19] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," 1966. DOI: https://doi.org/10.1147/sj.52.0078.

[20] M. V. Wilkes, "Slave Memories and Dynamic Storage Allocation," 1965. DOI: https://doi.org/10.1109/pgec.1965.264263.

[21] J. Zhang, G. Wu, X. Hu, and X. Wu, "A Distributed Cache for Hadoop Distributed File System in Real-Time Cloud Services," 2012. DOI: https://doi.org/10.1109/grid.2012.17.

[22] K. Liu, J. Peng, J. Wang, Z. Huang, and J. Pan, "Adaptive and Scalable Caching With Erasure Codes in Distributed Cloud-Edge Storage Systems," 2022. DOI: https://doi.org/10.1109/tcc.2022.3168662.

[23] J. Zhang, Q. Li, and W. Zhou, "HDCache: A Distributed Cache System for Real-Time Cloud Services," 2016. DOI: https://doi.org/10.1007/s10723-015-9360-9.

[24] D. Eklöv, D. Black-Schaffer, and E. Hägersten, "Fast modeling of shared caches in multicore systems," 2011. DOI: https://doi.org/10.1145/1944862.1944885.

[25] J. M. Calandrino and J. H. Anderson, "On the Design and Implementation of a Cache-Aware Multicore Real-Time Scheduler," 2009. DOI: https://doi.org/10.1109/ecrts.2009.13.

[26] S. Seo, J. Lee, and Z. Sura, "Design and implementation of software-managed caches for multicores with local memory," 2009. DOI: https://doi.org/10.1109/hpca.2009.4798237.

[27] X. Vera, B. Lisper, and J. Xue, "Data cache locking for higher program predictability," 2003. DOI: https://doi.org/10.1145/781027.781062.

[28] W. Hu, W. Shi, Z. Tang, and M. Li, "A lock-based cache coherence protocol for scope consistency," 1998. DOI: https://doi.org/10.1007/bf02946599.

[29] C. Mohan and I. Narang, "Efficient locking and caching of data in the multisystem shared disks transaction environment," 2005. DOI: https://doi.org/10.1007/bfb0032448.

[30] C. Fricker, P. Robert, and J. W. Roberts, "A versatile and accurate approximation for LRU cache performance," 2012. DOI: https://doi.org/10.48550/arxiv.1202.3974.

[31] E. O'Neil, P. O'Neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering," 1993. DOI: https://doi.org/10.1145/170036.170081.

[32] E. O'Neil, P. E. ONeil, and G. Weikum, "An optimality proof of the LRU- ¡i¿K¡/i¿ page replacement algorithm," 1999. DOI: https://doi.org/10.1145/300515.300518.

[33] A. Dan and D. Towsley, "An approximate analysis of the LRU and FIFO buffer replacement schemes," 1990. DOI: https://doi.org/10.1145/98457.98525.

[34] M. Moir, D. A. Nussbaum, O. Shalev, and N. Shavit, "Using elimination to implement scalable and lock-free FIFO queues," 2005. DOI: https://doi.org/10.1145/1073970.1074013.

[35] C. E. Molnar, I. W. Jones, W. Coates, and J. Lexau, "A FIFO ring performance experiment," 2002. DOI: https://doi.org/10.1109/async.1997.587181.

[36] Q. Chen, M. H. M. Al-Araby, Z. Qiu, Z. Chen, R. Vinayak, and J. Yang, "Demystifying and improving lazy promotion in cache eviction," To appear in VLDB 2026.

[37] G. Eads, "NbQ-CLOCK: A Non-blocking Queue-based CLOCK Algorithm for Web-Object Caching," 2014. DOI: https://doi.org/10.13140/2.1.2919.2329.

[38] P. Cunningham, M. Cord, and S. J. Delany, "Supervised Learning," 2008. DOI: https://doi.org/10.1007/978-3-540-75171-7_2.

[39] O. Chapelle, B. Schlkopf, and A. Zien, "Semi-Supervised Learning," 2006. DOI: https://doi.org/10.7551/mitpress/9780262033589.001.0001.

[40] J. M. Hilbe, "Logistic Regression Models," 2009. DOI: https://doi.org/10.1201/9781420075779.

[41] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, "Applied Logistic Regression," 2013. DOI: https://doi.org/10.1002/9781118548387.

[42] R. S. Sutton and A. Barto, "Reinforcement Learning: An Introduction," 2005. DOI: https://doi.org/10.1109/tnn.2004.842673. [Online]. Available: https://doi.org/10.1109/tnn.2004.842673.

[43] C. Tang, "Cache system design in the tightly coupled multiprocessor system," 1976. DOI: https://doi.org/10.1145/1499799.1499901.

[44] S. Bansal and D. S. Modha, "CAR: Clock with Adaptive Replacement," 2004. DOI: None. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.138.583.

[45] S. Jiang, F. Chen, and X. Zhang, "CLOCK-Pro: an effective improvement of the CLOCK replacement," 2005. DOI: None. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.124.4250.

[46] J. Yang, Y. Zhang, Z. Qiu, Y. Yue, and R. Vinayak, "FIFO queues are all you need for cache eviction," 2023. DOI: https://doi.org/10.1145/3600006.3613147. [Online]. Available: https://doi.org/10.1145/3600006.3613147.

[47] J. Yang, Y. Yue, and K. V. Rashmi, "A Large-scale Analysis of Hundreds of In-memory Key-value Cache Clusters at Twitter," 2021. DOI: https://doi.org/10.1145/3468521. [Online]. Available: https://doi.org/10.1145/3468521.

[48] W.-F. Lin et al., "ONNC: A Compilation Framework Connecting ONNX to Proprietary Deep Learning Accelerators," 2019. DOI: https://doi.org/10.1109/aicas.2019.8771510. [Online]. Available: https://doi.org/10.1109/aicas.2019.8771510.

[49] J. Tian et al., "Compiling ONNX Neural Network Models Using MLIR," 2020. DOI: https://doi.org/10.48550/arxiv.2008.08272. [Online]. Available: http://arxiv.org/abs/2008.08272.

[50] P. Jajal et al., “Interoperability in Deep Learning: A User Survey and Failure Analysis of ONNX Model Converters,” 2024. DOI: https://doi.org/10.1145/3650212.3680374. [Online]. Available: https://doi.org/10.1145/3650212.3680374.

[51] M. Sever and S. Ogut, “A Performance Study Depending on Execution Times of Various Frameworks in Machine Learning Inference,” 2021. DOI: https://doi.org/10.1109/uyms54260.2021.9659677. [Online]. Available: https://doi.org/10.1109/uyms54260.2021.9659677.

BIOGRAFI PENULIS

Penulis, **Muhammad Haekal Muhyidin Al-Araby**, lahir di [Tempat Lahir] pada tanggal [Tanggal Lahir]. Penulis merupakan anak dari pasangan [Nama Ayah] dan [Nama Ibu].

Penulis menempuh pendidikan dasar di [Nama SD] dan lulus pada tahun [Tahun Lulus SD]. Kemudian, penulis melanjutkan pendidikan menengah pertama di [Nama SMP] dan lulus pada tahun [Tahun Lulus SMP]. Setelah itu, penulis menyelesaikan pendidikan menengah atas di [Nama SMA/SMK] pada tahun [Tahun Lulus SMA/SMK].

Pada tahun [Tahun Masuk Kuliah], penulis diterima sebagai mahasiswa di Program Studi Teknik Komputer, Departemen Teknik Komputer, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember (ITS) melalui jalur [Jalur Masuk, contoh: SNMPTN/SBMPTN/Mandiri].

Selama masa perkuliahan, penulis memiliki minat khusus pada bidang [Sebutkan Minat Bidang Anda, contoh: rekayasa perangkat lunak, kecerdasan buatan, sistem komputer]. Untuk memperdalam minat tersebut, penulis aktif dalam [Sebutkan kegiatan, contoh: penelitian di laboratorium, organisasi, atau proyek pribadi].

Untuk memenuhi salah satu syarat kelulusan dan memperoleh gelar Sarjana Teknik, penulis menyusun Tugas Akhir yang berjudul "" di bawah bimbingan Bapak Reza Fuad Rachmadi, S.T., M.T., Ph.D dan Bapak Dr. Arief Kurniawan, S.T., M.T.

Penulis dapat dihubungi melalui surel di [Alamat Email Anda]. Email Anda].

[Halaman ini sengaja dikosongkan]