Implementasi *Load Balancing* pada *Platform Artificial Intelligence* Berbasis *Website* Menggunakan HAProxy

Sri Ayu Rahmadani[1], Yuris Mulya Saputra[1,*]
[1]Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada;
sri.ayu.rahmadani@mail.ugm.ac.id
[*]Korespondensi: ym.saputra@ugm.ac.id;

Abstract – As the need for social media content grows, verifying the truthfulness of news becomes increasingly challenging due to the spread of fake news. To address this, AI technology for news classification is being developed and implemented through websites using virtual machine infrastructure. However, high request volumes can reduce performance or cause downtime. This research examines using HAProxy for load balancing on web-based AI platforms to maintain performance and responsiveness under high traffic. The primary objective is to develop an AI platform for news classification that handles high traffic efficiently. Load testing with Apache JMeter demonstrated that the system effectively handled loads from 100 to 10,000 users without errors, with response times remaining within acceptable limits despite increasing with higher user loads. The AI platform successfully classified news, verified by external sources, demonstrating that HAProxy can enhance platform performance and reliability, significantly contributing to the accuracy of public information.

Keywords – Virtual machine (VM), Load Balancing, Artificial Intelligence, News Classification.

Intisari – Seiring meningkatnya kebutuhan akan konten media sosial, tantangan dalam memverifikasi kebenaran berita yang dikonsumsi publik juga semakin tinggi akibat penyebaran berita palsu. Permintaan akan teknologi AI untuk klasifikasi berita meningkat sebagai solusi atas masalah ini. AI untuk klasifikasi berita diimplementasikan melalui *website* dengan infrastruktur *virtual machine*. Namun, tingginya permintaan dapat menyebabkan penurunan kinerja atau *downtime*. Penelitian ini mengkaji penggunaan HAProxy sebagai solusi *load balancing* pada *platform* AI berbasis *web*. Tujuan utamanya adalah mengembangkan *platform* AI yang dapat mengklasifikasikan berita dan menerapkan teknologi *load balancing* agar *platform* dapat menangani lalu lintas tinggi dengan efisiensi dan responsivitas memadai. Pengujian menggunakan Apache JMeter menunjukkan bahwa sistem dapat menangani peningkatan beban dari 100 hingga 10.000 pengguna tanpa kesalahan, dengan waktu respons rata-rata 95 ms untuk 100 pengguna, 104 ms untuk 1,000 pengguna, dan 225 ms untuk 10,000 pengguna. Meskipun waktu respons meningkat seiring bertambahnya pengguna, tetap dalam batas yang dapat diterima. Model AI ini terbukti berhasil mengklasifikasikan berita dengan dukungan verifikasi dari sumber berita lain, menunjukkan bahwa HAProxy efektif meningkatkan kinerja dan keandalan *platform* AI, berkontribusi pada akurasi informasi publik.

Kata kunci – Mesin Virtual (VM), *Load balancing*, *Artificial Intelligence*, Klasifikasi Berita.

I. PENDAHULUAN

Meningkatnya volume konten digital membuat verifikasi kebenaran berita yang dikonsumsi publik semakin sulit, terutama akibat penyebaran berita palsu yang dapat menyesatkan masyarakat. Skala persoalan ini tidak kecil: tinjauan sistematis yang diterbitkan WHO menemukan proporsi *miss*-informasi mencapai 51% pada unggahan terkait vaksin dan hingga 60% pada konten terkait pandemi [1], sementara analisis terhadap sekitar 126.000 rumor di Twitter menunjukkan berita palsu menyebar lebih jauh dan 70% lebih mungkin dibagikan ulang dibanding berita benar [2]. Kondisi ini mendorong kebutuhan akan solusi teknologi yang mampu mengklasifikasikan berita sebagai asli atau palsu sehingga publik dapat memperoleh informasi yang terpercaya [3].

Untuk menjawab tantangan tersebut, para peneliti mulai memanfaatkan sistem *Artificial Intelligence* (AI) guna memverifikasi dan mengklasifikasikan konten berita [3]. Salah satu pendekatan yang menjanjikan adalah algoritma verifikasi berbasis AI yang menilai kredibilitas berita dari berbagai faktor, seperti rekam jejak sumber media, keahlian peneliti, hingga kecenderungan politiknya [3]. Seiring berkembangnya metode AI diskriminatif maupun generatif, teknologi ini semakin berperan dalam mendeteksi dan menekan disinformasi [4], [5]. Namun, solusi berbasis AI yang ada masih memiliki keterbatasan dalam mengimbangi pesatnya pertumbuhan volume konten daring [6].

Di sisi lain, kebutuhan terhadap layanan berbasis AI sendiri melonjak pesat; ChatGPT, misalnya, diperkirakan mencapai 100 juta pengguna aktif bulanan hanya dalam dua bulan setelah diluncurkan, menjadikannya aplikasi konsumen dengan pertumbuhan tercepat dalam sejarah [7]. Lonjakan permintaan semacam inilah yang membuat sistem AI rentan kelebihan beban dan mengalami gangguan layanan [8], sehingga kemampuan platform AI dalam menangani beban tinggi menjadi persoalan yang krusial.

Masalah yang umum dihadapi oleh *platform* AI berbasis *website* adalah kemampuannya untuk menangani beban lalu lintas yang besar dan berfluktuasi. Ketika ribuan atau bahkan jutaan pengguna mengakses layanan secara bersamaan, beban lalu lintas *server* dapat meningkat secara dramatis. Semakin banyak lalu lintas yang mencapai *server*, semakin buruk kinerja situs *website*, yang mengarah pada waktu respons yang lambat dan bahkan kegagalan sistem jika beban menjadi berlebihan [5].

Sistem yang *overload* menyebabkan *server down* karena ketidakmampuannya dalam menjalankan *request* yang berlebihan, upaya untuk menanggulanginya adalah dengan menggunakan teknologi *load balancing*. *Load balancing* sendiri merupakan komponen krusial dalam aplikasi *website* modern, yang memastikan distribusi lalu lintas masuk secara efisien di berbagai *server* untuk meningkatkan skalabilitas, ketersediaan, dan kinerja. Di antara berbagai solusi *load balancing* yang tersedia, HAProxy telah mendapatkan pengakuan signifikan dalam industri karena keandalannya, kinerja tinggi, dan fleksibilitasnya. Dibandingkan dengan *load balancing* lainnya, seperti Nginx dan Apache, HAProxy telah menunjukkan beberapa keunggulan. Sebuah studi oleh Singh dan Sharma menemukan bahwa HAProxy mengungguli Nginx dalam hal *throughput*, latensi, dan penanganan koneksi, terutama dalam skenario beban tinggi. Selain itu, analisis komparatif oleh Jain dan Bhatia mengungkapkan bahwa HAProxy menunjukkan kemampuan toleransi kesalahan dan *failover* yang *superior*, memastikan kontinuitas layanan yang mulus bahkan dalam kejadian kegagalan *server*. Selain itu, fleksibilitas HAProxy dalam mendukung berbagai protokol, termasuk HTTP, TCP, dan SSL/TLS, menjadikannya pilihan yang serbaguna untuk berbagai lingkungan aplikasi. Sharma dan Kumar melaporkan bahwa kemampuan HAProxy dalam menangani konfigurasi *load balancing* yang kompleks, seperti persistensi sesi dan pemeriksaan kesehatan, berkontribusi pada adopsi yang luas dalam implementasi tingkat perusahaan [9].

Arsitektur internal HAProxy memungkinkan distribusi permintaan yang efisien di antara beberapa *server*, yang dapat secara signifikan mengurangi waktu respons dan memastikan bahwa permintaan *client* dilayani dalam rentang waktu yang diinginkan. Selain itu, *HAProxy* menyediakan kemampuan untuk menunjuk *server* cadangan, yang dapat diaktifkan dalam kejadian kegagalan *server* utama, memastikan ketersediaan tinggi dan mengurangi dampak serangan *denial of service* [5]. Dengan mendistribusikan beban kerja secara merata di antara proses-proses yang bersaing, *HAProxy* dapat meningkatkan efisiensi keseluruhan sistem dan meminimalkan waktu respons, biaya, serta memaksimalkan *throughput*. Hal ini sangat penting dalam lingkungan yang kritis, seperti perawatan kesehatan, di mana waktu respons yang lambat dapat memiliki konsekuensi yang serius. Secara keseluruhan, fitur *load balancing* yang ditawarkan oleh HAProxy, seperti *failover*, *backup server designation*, dan distribusi permintaan yang efisien, dapat signifikan meningkatkan kinerja dan keandalan *server*, menjadikannya *tools* yang berharga bagi organisasi yang ingin meningkatkan infrastruktur *server* mereka.

II. DASAR TEORI

Peneliti melakukan riset terkait penelitian yang menyinggung layanan *Load Balancing* dan AI. Dari 10 penelitian yang telah dilakukan, peneliti mengambil beberapa sampel metodologi dan menggunakannya untuk mempelajari sistem yang akan dibuat pada penelitian ini.

Pertama, Penelitian yang dilakukan oleh Vivek Sharma, seorang Assistant Professor di GLA University, pada tahun 2022 dengan judul "*Object Detection and Recognition using Amazon Recognition with Boto3*" bertujuan memberikan tinjauan mendalam tentang penggunaan Amazon Recognition untuk deteksi dan pengenalan objek. Metode yang digunakan adalah *Frame Differencing*, yang memungkinkan deteksi objek dalam berbagai skenario dengan efisiensi tinggi. Hasil penelitian menunjukkan bahwa Amazon Recognition dapat mengenali objek secara akurat, menjadikannya alat yang berguna untuk berbagai aplikasi yang memerlukan deteksi dan pengenalan objek secara otomatis [10]. Penelitian pertama yang peneliti tinjau mempunyai judul, "*Website server Load balancing Based on Number of Client Connections on Docker Swarm*". Pada penelitian ini peneliti melakukan analisis kebutuhan untuk menentukan apa yang dibutuhkan dalam sistem *load balancing* yang efektif untuk meningkatkan stabilitas *server website*. Strategi *load balancing* yang digunakan adalah *Least Connection Algorithm* dengan *Round Robin*, yang diterapkan pada Docker Swarm. Docker Swarm adalah *platform containerization* yang memungkinkan pengembangan sistem distribusi. *Load balancing* yang digunakan adalah Nginx, yang dikenal dengan performa yang lebih baik dibandingkan dengan Apache dalam beberapa aspek, seperti meng*handle request* dan menghubungkan data yang diperlukan oleh *client*.

Penelitian kedua yang peneliti tinjau mempunyai judul "*Weighted Response Time Algorithm for Website server Load balancing in Software Defined Network*". Pada penelitian ini peneliti melakukan analisis pengembangan algoritma *load balancing* berbasis waktu tanggapan berat (*Weighted Response Time Algorithm*) untuk jaringan *Software Defined Network* (SDN). Algoritma ini dirancang untuk membagi beban *server* secara lebih rata dan meningkatkan kinerja sistem dengan mempertimbangkan spesifikasi *server* dan waktu tanggapan. Penelitian ini menunjukkan bahwa algoritma ini dapat meningkatkan kinerja sistem dengan mengurangi waktu tanggapan dan meningkatkan *throughput* serta stabilitas penggunaan CPU dibandingkan dengan algoritma *load balancing* berbasis waktu tanggapan biasa. [11].

Penelitian ketiga yang peneliti tinjau mempunyai judul, "*Design of Artificial Intelligence AI-based User Experience Websitesites for E-commerce Application and Future of Digital Marketing*". Pada penelitian ini peneliti melakukan analisis pengembangan aplikasi *website* berbasis AI untuk pengalaman pengguna yang disesuaikan dan meningkatkan efisiensi dalam pemasaran digital. Penelitian ini membahas tentang penggunaan AI dalam desain aplikasi *website*, integrasi API, dan metode *load balancing* untuk meningkatkan kinerja dan keamanan aplikasi. Selain itu, penelitian ini juga membahas tentang pentingnya *privacy* data dan *cookies consent management* dalam pengembangan

aplikasi *website* yang personal. Pada *platform* AI tersebut menggunakan Azure sebagai *platform cloud* untuk mengintegrasikan API menggunakan metode SOAP dan mengawasi privasi data.

Penelitian keempat yang peneliti tinjau mempunyai judul "*Research and Application of server Cluster Load balancing Technology*". Pada penelitian ini peneliti melakukan analisis tentang penerapan teknologi *load balancing cluster server* dengan menggunakan Nginx sebagai *tools load balancing*-nya. Peneliti mengusulkan strategi konfigurasi penyeimbangan beban *cluster server* berdasarkan Nginx, yang memastikan tingkat respons dan stabilitas layanan bus, meningkatkan kemampuan pemrosesan permintaan akses *webiste* bersamaan yang eksplosif, dan mengurangi waktu tunggu *client*. Hasil pengujian menunjukkan bahwa waktu respons *cluster server* jauh lebih baik dibandingkan *server* tunggal, dengan waktu respons rata-rata sekitar 5.2 ms untuk 300 koneksi bersamaan. Peneliti juga membahas pentingnya penyeimbangan beban dalam sistem *cluster server*, menyoroti perannya dalam meningkatkan kinerja dan daya tanggap sistem [13].

Penelitian kelima yang peneliti tinjau mempunyai judul "*Session-Persistent Load balancing for Clustered Web servers without Acting as a Reverse-proxy*". Pada penelitian ini peneliti melakukan analisis pengembangan metode *load balancing* yang *persistent* pada tingkat aplikasi tanpa berfungsi sebagai *reverse-proxy*. Peneliti ingin meningkatkan kinerja *load balancing* dalam meng-*handle* sesi aplikasi dengan cara menggunakan data sesi TLS (*Transport Layer Security*) untuk mengidentifikasi sesi *client* dan mengarahkan permintaan berikutnya ke *server* yang sama yang telah dipilih untuk sesi tersebut. Mereka menggunakan metode *load balancing* L4 yang tidak berfungsi sebagai *reverse-proxy* dan menggunakan *web server* untuk menampung data sesi aplikasi. Hasil penelitian menunjukkan bahwa metode *load balancing* yang digunakan dapat meningkatkan kinerja *load balancing* dalam meng-*handle* sesi aplikasi dengan tingkat permintaan yang lebih tinggi dan respons waktu yang lebih cepat. Peneliti juga membahas tentang perbedaan antara *load balancing* L4 dan L7 serta menguji performansi *load balancing* menggunakan *tools Siege* [7].

Penelitian keenam yang peneliti tinjau mempunyai judul "*Server Load balancing Using Linux*" Pada penelitian ini peneliti melakukan analisis yang berisi pembahasan tentang penggunaan Linux Virtual *server* (LVS) untuk mengoptimalkan kinerja *server* dalam penggunaan *load balancing*. Penelitian ini menggunakan metode eksperimen untuk menguji efektivitas LVS dan menunjukkan bahwa penggunaan LVS dapat meningkatkan kinerja *server* secara signifikan dan mengatasi masalah *overload server* dengan efektif. Hasil penelitian ini menunjukkan bahwa LVS dapat menjadi teknologi *load balancing* yang efektif dalam meningkatkan kinerja *server*.

Penelitian ketujuh yang peneliti tinjau mempunyai judul "*Weighted Response Time Algorithm for Web server Load balancing in Software Defined Network*" Pada penelitian ini peneliti melakukan analisis tentang pengembangan algoritma *load balancing* berbasis waktu tanggapan yang diperbarui dengan mempertimbangkan spesifikasi *server*. Algoritma ini menghitung rasio waktu tanggapan dan nilai berat yang mewakili spesifikasi *server*. *Server* dengan rasio nilai berat dan waktu tanggapan terkecil akan menerima permintaan dari *client*. Implementasi algoritma dilakukan pada *Software Defined Network* (SDN) menggunakan Mininet, sebuah emulator SDN yang mendukung OpenFlow *switches* dan *controller*. Hasil pengujian menunjukkan bahwa jaringan yang menggunakan algoritma *load balancing* berbasis waktu tanggapan yang diperbarui memerlukan waktu tanggapan yang lebih rendah dan menghasilkan *throughput* yang lebih baik dibandingkan dengan jaringan yang menggunakan algoritma berbasis waktu tanggapan sebelumnya. Hasil pengujian juga menunjukkan bahwa CPU *utilization* pada jaringan yang menggunakan algoritma *load balancing* berbasis waktu tanggapan yang diperbarui lebih stabil dan memiliki *throughput* yang lebih tinggi dibandingkan dengan jaringan yang menggunakan algoritma berbasis waktu tanggapan sebelumnya [14].

Penelitian kedelapan yang peneliti tinjau mempunyai judul "*Server Load balancing Using Availability Information Based on Fuzzy logic Decision*" Pada penelitian ini peneliti melakukan analisis tentang penggunaan metode *fuzzy logic* dalam sistem *load balancing* untuk meningkatkan kinerja dan ketersediaan aplikasi serta situs *web*. Penelitian ini menggunakan data log untuk mengukur kinerja sistem *load balancing* yang menggunakan *fuzzy logic*. Data log tersebut mencakup indikator seperti *throughput*, respons *time*, *request loss*, dan *availability of server resources*. Hasil penelitian menunjukkan bahwa sistem *load balancing* yang menggunakan *fuzzy logic* dapat meningkatkan kinerja dan ketersediaan aplikasi serta situs *web*. Peneliti membahas tentang penggunaan *fuzzy logic* dalam sistem *load balancing* untuk meningkatkan kinerja dan ketersediaan aplikasi serta situs *web*. Penelitian ini menggunakan metode eksperimen untuk menguji kinerja sistem *load balancing* yang menggunakan *fuzzy logic*. Mereka menggunakan data log untuk mengukur kinerja sistem *load balancing* yang menggunakan *fuzzy logic*. Data log tersebut mencakup indikator seperti *throughput*, *response time*, *request loss*, dan *availability of server resources*. Hasil penelitian menunjukkan bahwa sistem *load balancing* yang menggunakan *fuzzy logic* dapat meningkatkan kinerja dan ketersediaan aplikasi serta situs *web* [12].

Penelitian kesembilan yang peneliti tinjau mempunyai judul "*Load balancing Tuning: Comparative Analysis of HAProxy Load balancing Methods*" pada penelitian ini peneliti melakukan analisis tentang optimalisasi performa dan efisiensi *load balancing* menggunakan HAProxy. Peneliti melakukan analisis *comparative* antara berbagai algoritma

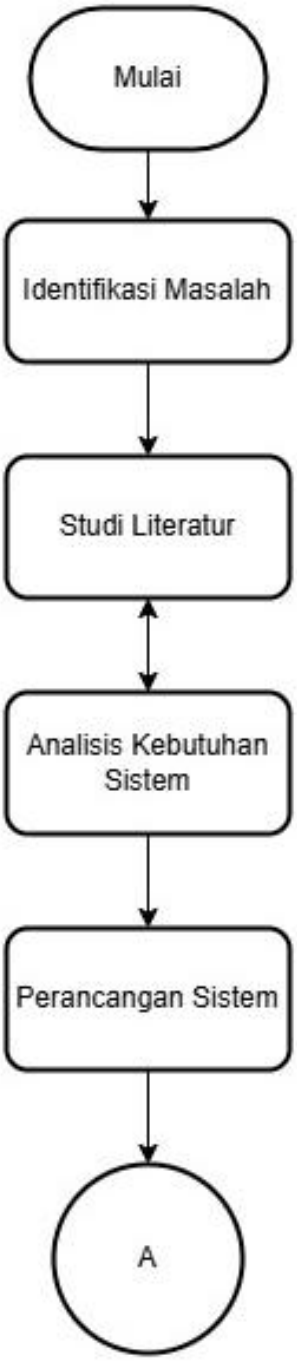
load balancing (ALB) dan *non-load balancing* (NLB), serta mengoptimalkan konfigurasi parameter HAProxy. Peneliti juga mengembangkan algoritma *load balancing custom* yang mempertimbangkan informasi tambahan tentang tugas yang datang, seperti URL yang diminta, untuk membuat keputusan *load balancing* yang lebih tepat. Penelitian ini menggunakan berbagai metode, termasuk analisis *comparative*, *tuning* HAProxy, implementasi algoritma *load balancing* custom, pengujian dengan Apache JMeter, penggunaan *virtual* dan *containerized machines*, serta penggunaan *libvirt*. Mereka juga menggunakan *cloud provider* seperti AWS untuk menguji performa *load balancing* dalam lingkungan *cloud*. Hasil penelitian menunjukkan bahwa algoritma *load balancing* yang mempertimbangkan informasi tambahan tentang tugas yang datang dapat meningkatkan performa *load balancing*. Mereka juga menemukan bahwa penggunaan *server heterogen* dapat meningkatkan performa *load balancing*. Penelitian ini juga membahas tentang cara mengoptimalkan konfigurasi parameter HAProxy untuk meningkatkan performa dan efisiensi dalam penggunaan sumber daya.

Penelitian kesepuluh yang peneliti tinjau mempunyai judul "*Mitigating Denial of Service Attacks with Load balancing*" pada penelitian ini peneliti melakukan analisis yang berfokus pada penggunaan *load balancing* HAProxy untuk menghadapi serangan *Denial of Service* (DoS) pada layer 7. Penelitian ini menggunakan *tools* seperti *Slow* HTTP *Test* dan *Golden Eye* untuk mengirimkan lalu lintas HTTP ke *server web* dan menguji efektivitas HAProxy dalam menghadapi serangan DoS. Hasil penelitian menunjukkan bahwa HAProxy dapat menghadapi berbagai serangan DoS, termasuk *Slow POST attack*, *Slow READ attack,* dan *Apache Range Header attack*, dengan cara mengumpulkan data dan menggunakan algoritma *load balancing* untuk menyebarluaskan lalu lintas ke *server web backend*. Penelitian ini juga menunjukkan bahwa penggunaan HAProxy dapat meningkatkan keamanan *server web* terhadap serangan DoS dan mengurangi dampaknya pada *server web*. [15].

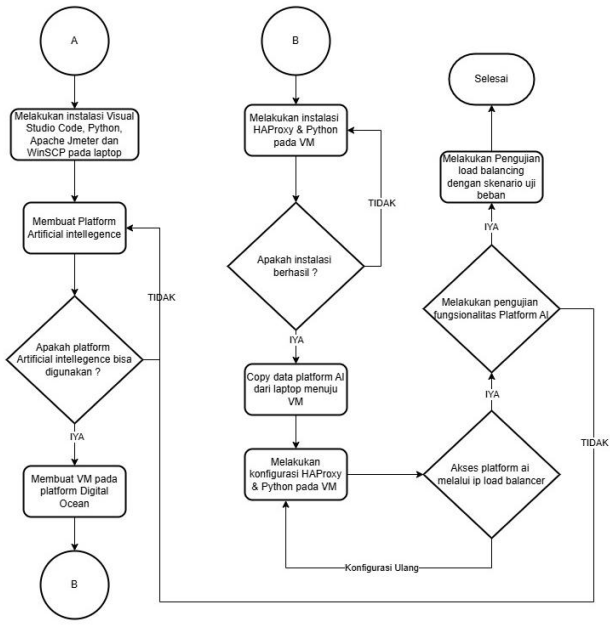
Dalam penelitian ini, peneliti menitikberatkan pada implementasi dan evaluasi kinerja HAProxy dalam *platform* AI berbasis *website*, yang berbeda dari beberapa penelitian sebelumnya. Sebagian besar penelitian terdahulu menggunakan Docker Swarm dan Nginx, [14] dengan algoritma *Weighted Response Time* dalam jaringan SDN, dan [3] yang menggunakan *fuzzy logic*, serta yang berfokus pada pengembangan aplikasi *e-commerce* berbasis AI, berfokus pada aspek atau teknologi tertentu dalam konteks *load balancing*. Dalam penelitian ini, peneliti berfokus pada analisis konfigurasi HAProxy untuk meningkatkan kinerja dan stabilitas *server* dalam konteks *platform* AI berbasis *website*. Hal ini berbeda dari penelitian yang menggunakan teknologi lain seperti Linux Virtual *server* (LVS), metode persistent *load balancing* tanpa *reverse-proxy* [7]. Dengan demikian, penelitian peneliti memberikan kontribusi unik dan wawasan baru tentang penerapan HAProxy untuk *load balancing* dalam konteks aplikasi AI berbasis *website.*

III. METODOLOGI

Berdasarkan Gambar 1 dan Gambar 2, peneliti memulai penelitian dengan beberapa langkah untuk mencapai hasil dari penelitian.



Gambar 1. Gambar Diagram Alur Penelitian



Gambar 2. Gambar Diagram Sistem Penelitian

1. Tahap Identifikasi Masalah

Pada tahap awal penelitian, dilakukan identifikasi terhadap permasalahan dan kebutuhan yang relevan sebagai dasar latar belakang penelitian ini. Langkah ini mencakup evaluasi menyeluruh terhadap kondisi saat ini, analisis tantangan yang dihadapi, serta eksplorasi potensi solusi yang dapat diimplementasikan. Hasil dari proses identifikasi ini kemudian digunakan untuk menetapkan fokus penelitian, dengan tujuan untuk mengembangkan solusi yang efektif dan efisien dalam menjawab permasalahan yang telah diidentifikasi. Pendekatan ini memastikan bahwa penelitian didasarkan pada kebutuhan nyata dan berorientasi pada hasil yang bermanfaat.

2. Tahap Pengumpulan Data Melalui Studi Literatur

Pada tahap ini, peneliti melakukan tinjauan pustaka yang komprehensif untuk membandingkan berbagai penelitian terdahulu yang relevan. Tujuan dari tinjauan ini adalah untuk mendapatkan wawasan mendalam mengenai kemajuan yang telah dicapai serta kekurangan yang masih ada dalam penelitian sebelumnya. Peneliti menggunakan hasil dari tinjauan pustaka ini sebagai landasan untuk memilih literatur yang tepat, yang kemudian akan digunakan sebagai dasar dalam merancang solusi terhadap masalah yang telah diidentifikasi. Pendekatan ini memungkinkan peneliti untuk membangun penelitian yang lebih kuat dan berkontribusi dalam memperkaya pengetahuan di bidang terkait.

3. Tahap Perancangan Sistem

Pada tahap perancangan sistem, peneliti secara mendetail merancang struktur dan komponen-komponen dari sistem yang akan dikembangkan, berdasarkan hasil identifikasi masalah dan tinjauan pustaka yang telah dilakukan sebelumnya. Perancangan ini mencakup berbagai aspek penting seperti spesifikasi teknis, arsitektur sistem, alur kerja, serta rencana implementasi sistem. Dengan merancang secara rinci, peneliti bertujuan untuk memastikan bahwa sistem yang dikembangkan tidak hanya memenuhi kebutuhan yang telah diidentifikasi, tetapi juga mampu secara efektif mengatasi masalah yang ada. Tahap perancangan ini sangat krusial untuk menjamin bahwa sistem yang dihasilkan akan berfungsi sesuai harapan dan memberikan solusi yang optimal terhadap permasalahan yang telah teridentifikasi sebelumnya.

4. Tahap Instalasi, Konfigurasi, dan Implementasi

Pada tahap ini, peneliti melakukan serangkaian langkah untuk menginstal, mengonfigurasi, dan mengimplementasikan sistem *load balancing* menggunakan HAProxy pada *platform* berbasis *web* yang menjadi fokus penelitian ini. Langkah pertama adalah instalasi Python pada laptop peneliti, yang digunakan untuk mengembangkan *platform Artificial Intelligence* (AI). Peneliti memastikan bahwa seluruh dependensi yang diperlukan untuk pengembangan telah terpasang dengan benar. Selain itu, Apache JMeter juga diinstal pada laptop peneliti untuk digunakan dalam pengujian beban dan analisis kinerja *load balancing* yang akan dilakukan pada tahap selanjutnya. Pada sisi lain, peneliti melakukan instalasi HAProxy dan Python pada *Virtual machine* (VM) yang dibuat di Digital Ocean. Peneliti membuat dua VM untuk *server backend* dan satu VM untuk *server load balancing*. Instalasi Python pada VM diperlukan untuk menjalankan *platform* AI yang telah dikembangkan, sementara HAProxy diinstal pada VM yang akan berfungsi sebagai *load balancing* untuk mengatur distribusi lalu lintas jaringan ke *server backend*. Setelah instalasi selesai, tahap implementasi sistem dimulai dengan memindahkan *file platform* AI dari laptop peneliti ke VM yang telah memiliki instalasi Python. Proses ini dilakukan untuk memastikan bahwa *platform* AI dapat dijalankan di lingkungan *server* yang telah dipersiapkan. Kemudian, peneliti mengonfigurasi HAProxy untuk mengatur lalu lintas jaringan ke *server backend*. Konfigurasi ini mencakup pengaturan *frontend* dan *backend*, serta monitoring untuk memantau kinerja sistem. Setelah konfigurasi selesai, peneliti melakukan pengujian fungsionalitas untuk memastikan bahwa semua *server backend* dapat diakses melalui HAProxy. Tahap implementasi ini merupakan langkah penting untuk memastikan bahwa sistem *load balancing* menggunakan HAProxy dapat berfungsi dengan baik dan memenuhi kebutuhan yang telah diidentifikasi. Implementasi yang berhasil memberikan dasar yang kuat untuk pengujian lebih lanjut dan analisis kinerja sistem. Peneliti menggunakan Apache JMeter untuk melakukan pengujian beban dan mengumpulkan data yang akan dianalisis untuk mengevaluasi kinerja dan keandalan sistem yang telah dikembangkan.

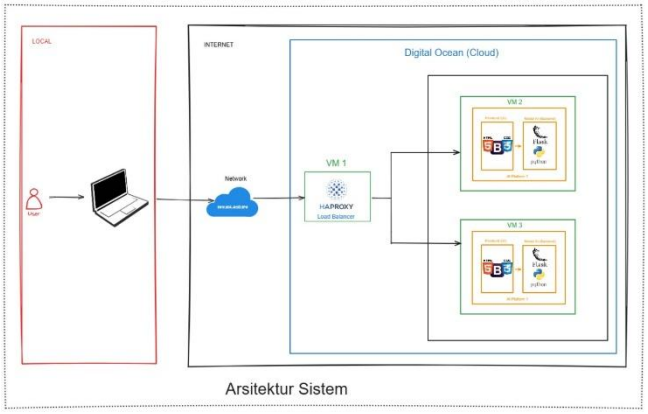
5. Tahap Pengujian Sistem

Dalam tahap Pengujian Sistem ini, peneliti melakukan dua jenis pengujian utama. Pertama, pengujian dilakukan terhadap fungsi *load balancing* menggunakan Apache JMeter untuk mengevaluasi performa sistem dalam menangani lalu lintas yang diarahkan oleh HAProxy ke berbagai *server backend*.

Pengujian ini bertujuan untuk mengukur *throughput*, *latency*, dan kestabilan sistem saat menghadapi beban yang beragam. Kedua, pengujian dilakukan terhadap fungsionalitas *platform* AI dalam memverifikasi berita untuk menentukan apakah berita tersebut termasuk berita hoaks atau berita benar. Pengujian ini bertujuan untuk mengevaluasi kemampuan *platform* AI dalam mengidentifikasi dan mengklasifikasikan berita dengan akurat, memastikan bahwa sistem dapat memberikan hasil yang dapat diandalkan dalam berbagai kondisi penggunaan.

A. Desain Arsitektur Sistem

Bagian ini menggambarkan desain arsitektur sistem *Platform* AI yang akan diimplementasikan, dengan alur infrastruktur yang tersaji dalam Gambar 3.



Gambar 3. Arsitektur Sistem

Arsitektur sistem ini menggambarkan implementasi *load balancing* pada *platform Artificial Intelligence* (AI) berbasis *website* menggunakan HAProxy. Dalam sistem ini, pengguna mengakses layanan AI melalui internet menuju alamat IP publik yang terdaftar (188.166.250.178). Permintaan pengguna kemudian diteruskan ke HAProxy yang berfungsi sebagai *load balancing*. HAProxy bertanggung jawab untuk mendistribusikan lalu lintas jaringan secara merata ke dua *instance* aplikasi AI yang tersedia, yaitu AI *APP* 1 dan AI *APP* 2. Proses dalam arsitektur ini melibatkan beberapa langkah. Pertama, pengguna mengirimkan permintaan ke layanan AI melalui jaringan internet. Permintaan ini mencapai HAProxy, yang berperan sebagai *load balancing* untuk mengelola distribusi lalu lintas. HAProxy kemudian menganalisis dan memutuskan *instance* aplikasi AI mana yang akan memproses permintaan berdasarkan algoritma *load balancing* yang digunakan. Setelah itu, salah satu *instance* aplikasi AI, AI *APP* 1 atau AI *APP* 2, menerima dan memproses permintaan tersebut. Selama pemrosesan, model AI dalam *instance* tersebut akan bekerja sesuai dengan tugas yang diminta. Setelah pemrosesan selesai, hasilnya dikirimkan kembali ke HAProxy, yang kemudian meneruskan hasil tersebut ke pengguna.

Pada arsitektur sistem yang terdapat pada gambar 3.3 terdapat 3 *virtual machine*, yang mana *virtual machine* pertama sebagai load balancer dan *virtual machine* 2 dan 3 sebagai backend server. Pada *virtual machine* pertama (*load balancer*) terdapat aplikasi yang diinstal yaitu HAProxy, sebagai *load balancer* utama yang mendistribusikan lalu lintas ke *backend server*, dan Apache JMeter yang digunakan untuk melakukan pengujian kinerja terhadap *load balancer*. Pada *virtual machine* yang kedua dan ketiga terdapat aplikasi yang diinstal yaitu sistem operasi ubuntu *server* untuk kompatibilitas dengan Python dan aplikasi *chatbot*, kemudian terdapat Python dan Flask, instalasi Python sebagai *runtime* utama untuk aplikasi *chatbot*, Flask untuk membangun *chatbot*. Kemudian terdapat instalasi dependensi dan pustaka yang dibutuhkan oleh aplikasi *chatbot*. Pustaka yang digunakan antara lain *Pandas* dan *NumPy* untuk pengolahan data, *Matplotlib* untuk visualisasi, serta *Pickle* untuk menyimpan model yang telah dilatih. Selain itu, peneliti juga mengimpor pustaka NLTK dan *Sastrawi* untuk proses tokenisasi, penghilangan kata umum (*stopwords*), dan *stemming* pada teks. Pustaka *Scikit-learn* juga digunakan untuk berbagai keperluan *machine learning*, seperti vektorisasi teks menggunakan *TfidfVectorizer*, seleksi fitur menggunakan *SelectKBest*, serta model *Naive Bayes* untuk klasifikasi.

Cara kerja arsitektur ini memungkinkan distribusi permintaan secara merata, mencegah *overload* pada satu *server* tertentu. Selain itu, arsitektur ini mendukung penambahan *instance* aplikasi lebih lanjut (seperti AI *APP* 3, 4, dan seterusnya) untuk menangani lebih banyak permintaan seiring pertumbuhan jumlah pengguna. Jika salah satu *instance* aplikasi mengalami kegagalan, HAProxy dapat mengalihkan permintaan ke *instance* lain yang masih aktif, sehingga layanan tetap tersedia. Dengan demikian, arsitektur ini menunjukkan bagaimana HAProxy dapat digunakan untuk mendistribusikan lalu lintas jaringan di antara beberapa aplikasi AI di *platform cloud* (Digital Ocean), untuk meningkatkan ketersediaan, kinerja, dan skalabilitas layanan AI berbasis *website*.

B. Pengembangan Aplikasi *Artificial Intelligence*

Langkah-langkah instalasi Python dimulai dengan mengunduh *installer* dari situs resmi Python, menyesuaikan dengan sistem operasi yang digunakan, dan memastikan opsi "*Add Python to PATH*" dipilih untuk menambahkan Python ke PATH sistem. Selanjutnya, pustaka-pustaka yang diperlukan seperti *Pandas, NumPy, Matplotlib, Pickle*, *NLTK, Sastrawi*, dan *Scikit-learn* diimpor untuk proses pelatihan model. *Dataset* dimuat dari *file CSV* dan disederhanakan dengan mempertahankan kolom judul dan label. Setelah itu, dilakukan *preprocessing* teks meliputi *case folding*, normalisasi kata, penghapusan *stopwords*, dan *stemming* menggunakan pustaka *Sastrawi*. Data hasil *preprocessing* disimpan dan dilakukan pemisahan fitur dan target. Representasi numerik teks dilakukan dengan metode TF-IDF dan seleksi fitur dilakukan menggunakan *Chi-Square*. Data dibagi menjadi *data training* dan *testing*, kemudian model

dilatih dan dievaluasi menggunakan data tersebut. Model yang telah dilatih disimpan dan digunakan dalam *aplikasi web Flask* untuk mendeteksi berita hoaks, di mana teks input diproses dan diklasifikasikan menggunakan model yang telah disimpan.

C. Instalasi & Konfigurasi Apache Jmeter pada *Host*

Untuk memulai penggunaan Apache JMeter, peneliti dapat mengunduh paket distribusi yang sesuai dari situs resmi Apache JMeter (https://jmeter.apache.org/download_jmeter.cgi) dan mengekstraknya ke direktori yang diinginkan, serta memastikan instalasi JDK dan pengaturan variabel lingkungan JAVA_HOME sudah dilakukan, kemudian membuka *command prompt* atau terminal, menavigasi ke direktori bin dalam instalasi JMeter, dan menjalankan skrip eksekusi (jmeter.bat untuk Windows atau ./jmeter.sh untuk Linux/macOS), sehingga antarmuka grafis pengguna JMeter terbuka untuk membuat, mengelola, dan menjalankan skenario pengujian beban guna mengevaluasi performa sistem secara optimal sesuai kebutuhan penelitian.

D. Instalasi dan Konfigurasi *WinSCP* pada *Host*

Sebelum memulai pembuatan *platform* AI, peneliti menginstal WinSCP dengan mengunduh *installer* dari situs resmi (https://winscp.net/eng/download.php) dan mengikuti proses instalasi standar, termasuk konfigurasi opsi seperti pembuatan *shortcut*, untuk mendukung *transfer file* yang aman antara komputer lokal dan server jarak jauh menggunakan protokol SCP atau SFTP, guna mempersiapkan lingkungan kerja yang optimal.

E. Membuat *Virtual Machine* di DigitalOcean

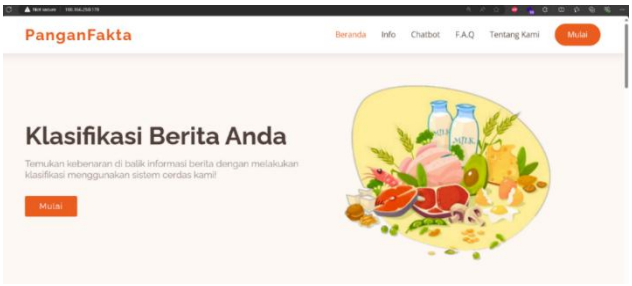
Untuk memulai implementasi pada *platform cloud*, peneliti membuat *Virtual Machine* (VM) di DigitalOcean sebagai infrastruktur dasar untuk menjalankan *platform* AI dan *load balancing*, dengan memilih *region* yang sesuai, Ubuntu 22.04 LTS sebagai OS, tipe droplet *SHARED CPU* dan *plan Basic*, serta CPU jenis "Regular" dengan biaya $18 per bulan, lalu memfinalisasi dengan membuat tiga droplet dengan konfigurasi serupa: satu untuk *load balancing* dan dua untuk menjalankan aplikasi AI.

F. Instalasi, Konfigurasi HAProxy dan Python pada *Virtual Machine*

Untuk menginstal dan mengonfigurasi HAProxy di VM sebagai layanan *load balancing* untuk *platform* AI, peneliti melakukan pembaruan paket sistem dengan `*sudo apt update*`, menginstal HAProxy menggunakan `*sudo apt install haproxy*`, lalu mengedit *file* konfigurasi utama di `*/etc/haproxy/haproxy.cfg*` untuk menyesuaikan pengaturan global, *default, frontend, backend*, dan monitoring sesuai kebutuhan aplikasi dan strategi *load balancing*, serta melanjutkan dengan menginstal Python dan mengatur lingkungan *virtual* untuk menjalankan *platform* AI.

G. Implementasi Sistem

Implementasi sistem dimulai dengan menjalankan *server backend* AI yang telah dikembangkan, mencakup pembuatan tiga VM di Digital Ocean: dua sebagai *backend* aplikasi dan satu sebagai *load balancer*. Setelah VM dibuat, instalasikan Python dan dependensi lainnya, serta pindahkan *file* aplikasi ke *backend* VM menggunakan SCP atau SSH. Konfigurasikan aplikasi untuk menggunakan *port* berbeda (misalnya, 6000 dan 6005) untuk menghindari konflik. Pada VM *load balancer*, instalasi Python dan dependensi juga dilakukan, kemudian konfigurasikan HAProxy untuk mendistribusikan lalu lintas HTTP secara merata ke *backend* VM. Uji sistem dengan mengakses IP *load balancer* melalui *browser* untuk memastikan aplikasi berfungsi dengan baik dan lalu lintas terdistribusi efektif. Hasil implementasi sistem disajikan pada Gambar 4.



Gambar 4. Hasil Implementasi Sistem

H. Pengujian Fungsionalitas Sistem

Pengujian ini bertujuan untuk menguji keberhasilan fungsionalitas sistem, meliputi fungsi *load balancing* dan layanan *platform* AI. Pengujian dilakukan pada infrastruktur tiga *virtual machine* (VM) di DigitalOcean, masing-masing bersistem operasi Ubuntu 22.04 LTS (x64) dengan 2 vCPU, RAM 2 GB, dan penyimpanan 60 GB. Dua VM difungsikan sebagai *backend server* yang menjalankan instansi aplikasi AI (AI APP 1 dan AI APP 2) pada *port* berbeda, sedangkan satu VM berfungsi sebagai *load balancer* dengan HAProxy yang diakses melalui alamat IP publik 188.166.250.178. Pengujian beban dijalankan dari perangkat peneliti menggunakan Apache JMeter, yang mengirimkan permintaan ke alamat *load balancer* untuk kemudian didistribusikan ke kedua *backend*.

- Fungsi Sistem *Load Balancing*:

Pengujian beban dilakukan dengan metode HTTP GET menggunakan Apache JMeter terhadap alamat IP *load balancer* (188.166.250.178). Pengujian dijalankan dalam tiga skenario berdasarkan jumlah pengguna simulasi, yaitu 100, 1.000, dan 10.000 pengguna, dengan parameter sebagaimana dirangkum pada Tabel 1. Setiap skenario menggunakan *Thread Group* dengan periode *ramp-up* 100 detik dan *loop count* 1, serta opsi *Delay Thread Creation Until Needed* yang diaktifkan, sementara permintaan diarahkan ke halaman

utama aplikasi melalui *load balancer*. Pada setiap skenario, sistem dievaluasi terhadap metrik waktu respons (rata-rata, minimum, dan maksimum), deviasi standar, tingkat kesalahan (*error rate*), *throughput*, serta volume data yang diterima dan dikirim (*received/sent* KB/detik). Metrik-metrik tersebut digunakan untuk menilai kemampuan HAProxy dalam mendistribusikan beban secara merata dan menjaga responsivitas sistem seiring meningkatnya jumlah permintaan.

Tabel 1. Parameter Pengujian Beban Apache JMeter

Parameter	Nilai
Metode permintaan	HTTP GET ke IP *load balancer* (188.166.250.178)
Jumlah pengguna (*thread*)	100; 1.000; 10.000
Periode *ramp-up*	100 detik
Loop count	1
Delay Thread Creation Until Needed	Aktif
Target permintaan	Halaman utama aplikasi melalui *load balancer*
Listener	*View Results Tree*, *Summary Report*

- Pengujian Fungsi Layanan *Platform* AI:

Pengujian ini bertujuan mengevaluasi ketepatan model dalam mengklasifikasikan berita. Model dibangun dari dataset sebanyak 4.231 entri berita yang terdiri atas 766 berita benar dan 3.465 berita hoaks. Teks melalui tahap praproses berupa *case folding*, normalisasi kata, penghapusan *stopword*, dan *stemming* dengan pustaka Sastrawi, lalu direpresentasikan secara numerik menggunakan TF-IDF. Seleksi fitur dilakukan dengan metode Chi-Square (*SelectKBest*) untuk memilih 3.000 fitur paling berpengaruh, dan ketidakseimbangan kelas pada data latih ditangani dengan teknik *oversampling* SMOTE. Model klasifikasi dilatih menggunakan algoritma *Multinomial Naive Bayes* dengan pembagian data latih dan uji sebesar 80:20. Kinerja model dievaluasi melalui *confusion matrix* serta metrik akurasi, *precision*, *recall*, dan *f1-score*, dengan akurasi yang diperoleh sebesar 92,36% pada data uji. Selain pengujian kuantitatif tersebut, sistem juga diuji secara langsung menggunakan sampel berita hoaks dan berita benar dari sumber asli, dan hasil klasifikasinya dibandingkan dengan verifikasi manual untuk memastikan kesesuaian.

IV. HASIL DAN PEMBAHASAN

A. Pengujian Fungsi Sistem *LoadBalancing*

Pengujian ini bertujuan mengevaluasi efektivitas dan kinerja sistem dalam mengimplementasikan fungsi *load balancing* pada *platform* AI berbasis *website*, dengan menganalisis distribusi lalu lintas antara beberapa VM sebagai *backend,* di mana satu VM berfungsi sebagai *load balancer,* serta waktu respons, *throughput*, dan tingkat kegagalan permintaan menggunakan Apache JMeter. Pengujian dilakukan dengan metode HTTP GET pada IP 188.166.250.178 menggunakan *Thread Group* konfigurasi dengan pengguna simulasi 100, 1.000, dan 10.000, periode *ramp-up* 100 detik, dan *loop count* 1. Hasil pengujian menunjukkan peningkatan waktu respons rata-rata dari 95 ms pada 100 pengguna menjadi 225 ms pada 10.000 pengguna, dengan minimum respons dari 87 ms menjadi 77 ms, dan maksimum respons dari 77 ms menjadi 571 ms. Deviasi standar meningkat dari 4,38 ms pada 100 pengguna menjadi 110,38 ms pada 10.000 pengguna. Tingkat kesalahan tetap 0,000%, dan *throughput* meningkat dari 0,99988 requests/s menjadi 99,75261 requests/s. *Received* KB/sec dan sent KB/sec masing-masing meningkat sebesar 9.877,26% dan 10.263,64%, sementara rata-rata *bytes* tetap stabil pada 15.318. Hasil pengujian menunjukkan bahwa sistem mampu menangani peningkatan beban pengguna dengan baik, meskipun ada peningkatan signifikan dalam waktu respons dan variasi waktu respons pada beban yang lebih tinggi, serta sistem *load balancer* efektif dalam mendistribusikan beban secara merata dan menjaga performa aplikasi di bawah beban pengguna yang beragam. Hasil pengujian disajikan pada Tabel 2.

Tabel 2. *Report* Pengujian Beban Apache JMeter

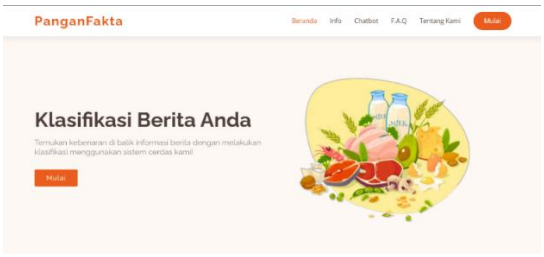
Matriks	100 Pengguna	1,000 Pengguna	10,000 Pengguna
Rata-rata Respons	95 ms	104 ms	225 ms
Minimum Respons	87 ms	90 ms	77 ms
Maksimum Respons	107 ms	139 ms	571 ms
Tingkat Kesalahan	0.000%	0.000%	0.000%
Throughput	0.99988 *Requests*/s	9.99400 *Requests*/s	99.75261 *Requests*/s
Received KB/sec	14.95	149.50	1492.20
Sent KB/sec	0.11	1.14	11.40
Avg. Bytes	15318.0	15318.0	15318.0

B. Pengujian Fungsi layanan *Platform* AI

Penelitian yang dilakukan melatih model *machine learning* dengan algoritma NLP. Hasil akurasi model sebesar 92% yang menunjukkan bahwa model ini efektif dalam mengklasifikasikan berita. Seperti terlihat pada Gambar 5.

Pada pengujian ini, peneliti menggunakan *chatbot* AI untuk melakukan pengecekan terhadap berita dari berbagai sumber dengan tujuan mengklasifikasikan apakah berita tersebut benar atau hoaks mengacu pada Gambar 5 yang merupakan *dashboard platform* AI, peneliti akan memulai

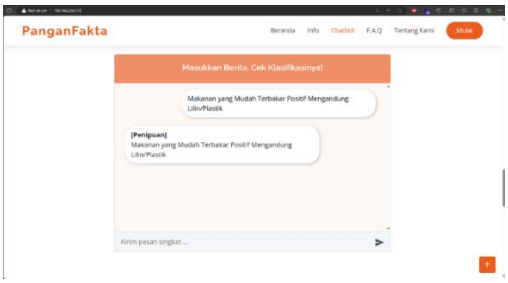
proses dengan mengajukan permintaan terkait berita kepada *chatbot* AI.



Gambar 5. Dashboard Platform AI

Evaluasi pada tahap ini melibatkan kejelasan interaksi, responsibilitas *chatbot* terhadap berbagai jenis pertanyaan dan kata kunci, serta kemudahan penggunaan antarmuka *chatbot*. Keberhasilan interaksi ini sangat bergantung pada kemampuan *chatbot* untuk memahami pertanyaan yang kompleks dan memberikan jawaban yang relevan. *Chatbot* AI melakukan pemrosesan terhadap pertanyaan yang diajukan oleh peneliti menggunakan teknik NLP. Hal ini mencakup analisis sintaksis dan semantik untuk mengurai permintaan peneliti sehingga dapat dipahami dengan benar. Evaluasi pada tahap ini mencakup keakuratan dalam memahami bahasa yang digunakan oleh peneliti, kemampuan untuk mengenali konteks dari setiap permintaan, dan responsibilitas dalam memberikan jawaban yang sesuai. Setelah memahami permintaan pengguna, *chatbot* AI melakukan pencarian dan verifikasi terhadap berita yang diminta. Ini melibatkan pengambilan data dari sumber-sumber berita yang terpercaya dan pengujian terhadap kebenaran informasi yang disajikan. Evaluasi pada tahap ini mencakup kehandalan dalam menentukan keabsahan berita, kualitas sumber berita yang digunakan, dan kemampuan untuk memverifikasi informasi secara akurat.

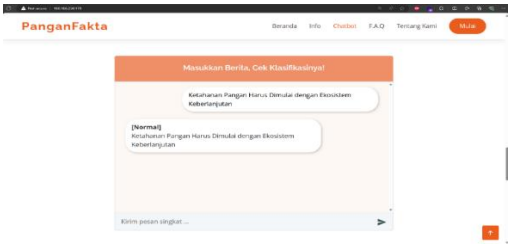
Klasifikasi ini didasarkan pada analisis konten berita dan faktor-faktor penentu lainnya. Salah satu contoh pengujian melibatkan berita dari Kominfo dengan judul "Makanan yang Mudah Terbakar Positif Mengandung Lilin/Plastik.". Hasil klasifikasi oleh sistem AI pada Gambar 6 menunjukkan bahwa berita tersebut juga teridentifikasi sebagai hoaks. Berdasarkan Gambar 7 yang merupakan artikel terbitan Kominfo, berita ini terbukti sebagai hoaks. Hal ini menunjukkan bahwa algoritma yang digunakan dalam *platform* AI dapat mengenali pola-pola tertentu yang sering muncul dalam berita hoaks, seperti penggunaan klaim yang tidak berdasar dan pemakaian istilah yang bersifat sensasional. Selain itu, hasil ini mengindikasikan bahwa sistem AI memiliki potensi untuk digunakan sebagai alat bantu dalam proses verifikasi berita. Dengan akurasi yang tinggi, *platform* AI dapat membantu mempercepat proses identifikasi berita.



Gambar 6. Pengujian Berita Hoaks

Gambar 7. Contoh Artikel Hoaks (KOMINFO, 2024)

Selanjutnya adalah melakukan pengujian pada berita yang berjudul "Mendag Zulhas: Ketahanan Pangan merupakan Prioritas Utama Pemerintah". Hasil verifikasi oleh sistem AI menunjukkan bahwa klaim dalam berita ini sebagai berita normal sesuai dari Gambar 8, kemudian dapat dilihat juga pada Gambar 9 yang merupakan sumber asli berita ini memberikan informasi yang sama secara akurat dan benar adanya.



Gambar 8. Hasil Pengujian Berita Normal

Gambar 9. Contoh Artikel Berita Normal (Kompas, 2023)

Keberhasilan sistem AI dalam mengklasifikasikan berita hoaks ini dapat dijelaskan melalui beberapa faktor penting. Pertama, kualitas *dataset* yang digunakan untuk melatih algoritma memiliki pengaruh yang signifikan terhadap kinerja sistem AI. Dalam penelitian ini, *dataset* yang digunakan terdiri dari berita hoaks dan berita yang benar, yang masing-masing telah diverifikasi kebenarannya. *Dataset* yang baik dan representatif memungkinkan algoritma untuk belajar dan mengenali pola-pola tertentu yang sering muncul dalam berita hoaks, seperti penggunaan klaim yang tidak berdasar dan istilah yang bersifat sensasional.

V. SIMPULAN

Berdasarkan data dan analisis dari penelitian implementasi *load balancing* pada *platform* AI berbasis *website* menggunakan HAProxy, dapat disimpulkan bahwa HAProxy efektif dalam mendistribusikan lalu lintas secara merata ke dua *backend server*, mencegah *overload; platform* AI menunjukkan waktu respons yang konsisten meskipun jumlah permintaan meningkat, dengan waktu respons rata-rata 95 ms pada 100 pengguna, 104 ms pada 1.000 pengguna, dan 225 ms pada 10.000 pengguna, menunjukkan kemampuan platform AI dalam menangani beban tinggi dengan performa memadai; dan *chatbot* AI untuk pengecekan berita berhasil mengklasifikasikan berita dengan akurasi 92,36%, memanfaatkan model yang telah dilatih untuk mendeteksi berita hoaks dan benar.

REFERENSI

[1] Borges do Nascimento, I. J., Pizarro, A. B., Almeida, J. M., Azzopardi-Muscat, N., Gonçalves, M. A., Björklund, M., & Novillo-Ortiz. D. (2022). Infodemics and health misinformation: A systematic review of reviews. Bulletin of the World Health Organization, 100(9), 544–561.

[2] Vosoughi, S., Roy, D., & Aral, S. (2018). The spread of true and false news online. Science, 359(6380), 1146–1151.

[3] Vinaykarthik, B. C. (2022, October). Design of artificial intelligence (AI) based user experience websites for e-commerce application and future of digital marketing. In 2022 3rd International Conference on Smart Electronics and Communication (ICOSEC) (pp. 1023–1029). IEEE.

[4] Mayorga, M. W., Hester, E. B., Helsel, E., Ivanov, B., Sellnow, T. L., Slovic, P., ... & Frakes, D. (2020). Enhancing public resistance to "fake news": A review of the problem and strategic solutions. The Handbook of Applied Communication Research, 197–212.

[5] Moharir, M., Shobha, G., Oppiliappan, A., GVL, R. M. K., Pandit, S. N., Akash, R., & Saxena, M. (2020, December). A study and comparison of various types of load balancers. In 2020 5th IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE) (pp. 1–7). IEEE.

[6] Rawls, C., & Salehi, M. A. (2022). Load balancer tuning: Comparative analysis of HAProxy load balancing methods. arXiv preprint arXiv:2212.14198.

[7] Hu, K. (2023, February 2). ChatGPT sets record for fastest-growing user base – analyst note. Reuters.

[8] Li, B., Shang, J., Dong, M., & He, Y. (2020, June). Research and application of server cluster load balancing technology. In 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (Vol. 1, pp. 2622–2625). IEEE.

[9] Hosseini, S. M., Jahangir, A. H., & Daraby, S. (2021, October). Session-persistent load balancing for clustered web servers without acting as a reverse-proxy. In 2021 17th International Conference on Network and Service Management (CNSM) (pp. 360–364). IEEE.

[10] Horvitz, E. (2022, November). On the horizon: Interactive and compositional deepfakes. In Proceedings of the 2022 International Conference on Multimodal Interaction (pp. 653–661).

[11] Chen, C. (2022). Server load balancing using Linux. In 2022 8th Annual International Conference on Network and Information Systems for Computers (ICNISC) (pp. 134–138). IEEE.

[12] Khan, T., Michalas, A., & Akhunzada, A. (2021). Fake news outbreak 2021: Can we stop the viral spread? Journal of Network and Computer Applications, 190, 103112.

[13] Koplin, J. J. (2023). Dual-use implications of AI text generation. Ethics and Information Technology, 25(2), 32.

[14] Ezenwe, A., Furey, E., & Curran, K. (2020). Mitigating denial of service attacks with load balancing. Journal of Robotics and Control (JRC), 1(4), 129–135.

[15] Nurwasito, H., & Rahmawati, R. (2021, July). Weighted response time algorithm for web server load balancing in software defined network. In 2021 3rd International Conference on Electronics Representation and Algorithm (ICERA) (pp. 143–148). IEEE.