BAB 2
TINJAUAN PUSTAKA

2.1 Arsitektur Backend untuk ML Inference

Arsitektur *backend* untuk *machine learning inference* dirancang untuk mendukung pemrosesan prediksi secara efisien, skalabel, dan terisolasi antar komponen. Pendekatan ini memisahkan tanggung jawab antara *request handling*, *routing*, dan eksekusi *model inference* guna meningkatkan performa dan stabilitas sistem [11]. Dengan menerapkan *layered architecture*, sistem mampu mengelola beban *I/O-bound* dan *compute-intensive* secara terpisah sesuai karakteristik masing-masing proses [8]. Selain itu, pemanfaatan Docker memungkinkan penerapan arsitektur *hybrid* yang fleksibel dan efisien pada lingkungan *deployment* terpusat [9].

2.1.1 Layered Architecture

Arsitektur *backend* sistem AIRA dirancang dengan pendekatan *layered architecture* yang memisahkan tanggung jawab berdasarkan fungsi [12]. Sistem dibagi menjadi empat *layer*, yaitu *presentation layer* untuk interaksi pengguna melalui *frontend*, *application layer* berupa *Cancer Gateway* yang menangani *routing* dan validasi, *service layer* berupa AI *Backend* yang menjalankan *model inference*, serta *data layer* berupa MySQL *database* untuk penyimpanan metadata dan *history*. Pemisahan ini membuat setiap *layer* dapat dikembangkan dan diuji secara independen, serta memudahkan *scaling* pada bagian yang bersifat lebih *compute-intensive* [13]. Pada implementasinya, *backend* dibagi menjadi dua komponen utama, yaitu *Cancer Gateway* dan AI *Backend*, yang berkoordinasi untuk menangani alur prediksi.

2.1.2 Cancer Gateway

Cancer Gateway dibangun menggunakan Node.js dan Express.js sebagai API *gateway* yang menjadi *entry point* dari *frontend*. Komponen ini bertugas melakukan *request routing* berdasarkan jenis kanker dan tipe data, menerapkan *rate limiting* untuk melindungi AI *Backend* dari *overload*, serta melakukan validasi input dan *logging* untuk keperluan monitoring. Node.js dipilih karena menggunakan

event-driven dan *non-blocking I/O model* yang efisien untuk menangani *concurrent connections* [9][11].

2.1.3 AI Backend

AI *Backend* dibangun menggunakan FastAPI untuk menangani *loading* dan manajemen *multiple TensorFlow models*, melakukan *preprocessing* input CSV (misalnya normalisasi dan *feature scaling*), menjalankan *inference* menggunakan model yang sesuai, dan mengembalikan hasil prediksi dalam format JSON. FastAPI dipilih karena performanya tinggi, dukungan *native async*, serta kompatibel dengan ekosistem ML seperti TensorFlow, NumPy, dan Pandas. Dalam sistem AIRA, model TensorFlow dimuat saat *startup* dan disimpan di memori untuk mendukung *fast inference*, sesuai dengan praktik umum dalam *AI model serving* [3][14].

2.1.4 Pemisahan Komponen

Pemisahan *Cancer Gateway* dan AI *Backend* dilakukan berdasarkan pertimbangan teknis. Node.js lebih efisien untuk operasi yang bersifat *I/O-bound* seperti *routing requests* dan *database queries*, sedangkan Python memiliki ekosistem yang matang untuk operasi ML dengan *library* seperti TensorFlow dan NumPy. Dengan pemisahan ini, *compute-intensive service* dapat dilakukan *scale* secara independen tanpa mempengaruhi *gateway service*, sebagaimana direkomendasikan dalam arsitektur *microservices* [11][12]. Selain itu, isolasi kesalahan lebih terjaga karena *crash* pada satu *service* tidak langsung mempengaruhi *service* lain.

2.1.5 Hybrid Architecture dengan Docker

Sistem di-*deploy* menggunakan Docker *containerization* dengan tiga *container* terpisah untuk *Cancer Gateway*, AI *Backend*, dan MySQL *database*. Ketiga *container* berjalan dalam satu *server* dan saling terhubung melalui Docker *bridge network* [15]. Pendekatan *hybrid* ini menggabungkan modularitas *microservices* dengan efisiensi *co-location* untuk mengurangi *network latency* antar-*services*, sekaligus menyederhanakan proses *deployment* dan *maintenance*. Docker menyediakan lingkungan yang terisolasi dan konsisten dari tahap *development* hingga *production*.

MySQL digunakan untuk menyimpan metadata sistem seperti *cancer types*, konfigurasi model, dan *prediction history* yang dibutuhkan CMS. MySQL dipilih karena memiliki *ACID compliance* yang menjamin integritas data serta ekosistem yang matang dengan banyak *tools* dan dokumentasi. Docker Compose dimanfaatkan untuk mengatur *services*, *networks*, dan *volumes* dalam satu berkas konfigurasi sehingga proses *deployment* menjadi lebih terstruktur.

2.2 Machine Learning untuk Deteksi Kanker

Machine Learning (ML) telah menunjukkan potensi besar dalam membantu diagnosis penyakit, termasuk deteksi dini kanker [1]. Dalam konteks kesehatan, ML dapat menganalisis data klinis dan genomik untuk mengidentifikasi pola yang mengindikasikan keberadaan sel kanker. Model dilatih menggunakan data historis pasien untuk membedakan kondisi sehat dan kondisi kanker, kemudian dimanfaatkan kembali pada tahap *inference* untuk menghasilkan prediksi terhadap data baru. Tantangan yang muncul bukan hanya pada pembuatan model, tetapi juga pada integrasi *multiple ML models* ke dalam sistem *web* yang stabil dan mudah diakses [7].

Sistem AIRA menggunakan data genomik untuk deteksi kanker prostat dan payudara. Data genomik yang digunakan meliputi *gene expression* yang mengukur tingkat aktivitas gen dalam sel, DNA *methylation* sebagai modifikasi epigenetik yang mempengaruhi ekspresi gen, serta *microRNA* (miRNA) sebagai molekul RNA kecil yang mengatur ekspresi gen. Sistem mengelola *multiple models* dengan jumlah dan jenis fitur yang berbeda, misalnya model kanker prostat dengan 18 fitur ekspresi gen, 25 fitur ekspresi gen, dan 90 fitur metilasi DNA. Keragaman ini menuntut arsitektur *backend* yang fleksibel dan mampu menangani variasi *payload* serta logika *model inference*.

2.3 API Gateway Pattern

API *Gateway pattern* adalah pola desain di mana satu *single entry point* menangani semua *client requests* dan meneruskannya ke *backend services* yang sesuai [12]. *Gateway* bertindak sebagai *facade* yang menyembunyikan kompleksitas *backend* dari *client*. Dalam sistem AIRA, *Cancer Gateway* mengimplementasikan pola ini dengan melakukan *request routing* ke AI *Backend*, menerapkan *rate limiting*, serta menyediakan *centralized logging* untuk keperluan

monitoring. Pendekatan ini membuat *client* hanya perlu berinteraksi dengan satu *endpoint*, sementara perubahan pada *backend services* dapat dilakukan tanpa mengganggu *client*.

Pada konteks ML *inference*, API *gateway* juga menambah *overhead latency* karena *request* harus melewati satu *layer* tambahan sebelum mencapai AI *Backend*. *Overhead* ini berasal dari aktivitas *routing*, validasi input, dan transformasi data. Oleh karena itu, perlu dilakukan evaluasi untuk memastikan *overhead* tersebut masih *acceptable* untuk kebutuhan *healthcare* yang menuntut *response time* cepat.

2.4 Performance Metrics dan Benchmarking

Evaluasi performa merupakan aspek penting dalam pengembangan sistem *backend*, khususnya untuk aplikasi yang melibatkan proses komputasi intensif seperti *machine learning inference*. Pengukuran performa diperlukan untuk memastikan bahwa sistem mampu memberikan waktu respons yang dapat diterima, menjaga stabilitas layanan, serta menangani peningkatan beban pengguna secara bertahap. Oleh karena itu, penggunaan metrik performa yang tepat dan metodologi pengujian yang terstruktur menjadi dasar dalam menilai kesiapan sistem sebelum digunakan pada skenario mendekati kondisi nyata. Pada bagian ini dibahas konsep metrik performa, metodologi *load testing*, serta pendekatan *benchmarking* yang digunakan sebagai acuan dalam evaluasi sistem AIRA.

2.4.1 Metrik Performa untuk Backend Systems

Evaluasi performa *backend systems* umumnya menggunakan beberapa kategori metrik seperti *response time*, *throughput*, reliabilitas, dan utilisasi *resource*. *Response time* mengukur waktu dari *request* dikirim hingga *response* diterima, sedangkan *throughput* menggambarkan jumlah *requests per second* (RPS) yang dapat diproses sistem [9][11]. Selain itu, *success rate* dan *error rate* digunakan untuk melihat reliabilitas sistem di bawah beban tertentu, sementara penggunaan CPU dan memori dipantau untuk memahami efisiensi pemakaian *resource*. Konsep-konsep dasar ini menjadi landasan perancangan *stress testing* yang dijelaskan lebih rinci pada bab metodologi.

2.4.2 Metodologi Load Testing

Load testing adalah metode untuk mengevaluasi performa sistem dengan mensimulasikan banyak *concurrent users* atau *requests*. Berbagai skenario beban dapat disusun, mulai dari beban ringan hingga *stress* untuk melihat titik ketika sistem mulai mengalami degradasi performa [11]. Beragam *tools* seperti Apache JMeter, Locust, dan k6 umum digunakan karena menyediakan fitur *reporting* dan konfigurasi skenario yang fleksibel. Pada penelitian ini, konsep *load testing* dimanfaatkan terutama untuk melakukan *stress testing* terhadap arsitektur AIRA.

2.4.3 Benchmarking ML Inference Performance

Benchmarking ML *inference* memiliki karakteristik yang berbeda dibandingkan *general web services* karena melibatkan *workload* yang lebih *compute-intensive*. MLPerf, misalnya, berfokus mengukur *pure model inference* tanpa memasukkan *overhead* dari API *gateway* dan lapisan arsitektural lain [14]. Di sisi lain, studi lain melaporkan bahwa operasi CRUD pada aplikasi web umumnya memiliki *response time* sekitar 30–70 ms, sementara sistem *healthcare* multi-*tier* dapat berada pada rentang 200–800 ms, dan ML *inference* di *edge devices* berkisar 50–500 ms [9][16][17]. Rentang ini digunakan sebagai acuan awal untuk mengevaluasi apakah performa sistem AIRA berada pada kategori yang masih *acceptable*.

2.5 Penelitian Terkait

Penelitian terkait diperlukan untuk memberikan konteks terhadap posisi dan kontribusi penelitian yang dilakukan. Melalui tinjauan terhadap studi-studi sebelumnya, dapat diidentifikasi pendekatan yang umum digunakan dalam evaluasi performa *backend systems* serta keterbatasan yang masih ada, khususnya pada sistem yang melibatkan *machine learning inference*. Selain itu, kajian literatur membantu menunjukkan perbedaan karakteristik antara layanan web konvensional dan sistem ML di bidang kesehatan. Berdasarkan pemahaman tersebut, penelitian ini diposisikan untuk mengisi celah yang belum banyak dibahas dalam studi sebelumnya.

2.5.1 Performance Evaluation untuk Backend Systems

Blinowski et al. mengevaluasi performa arsitektur *monolithic* dan *microservices* untuk *general web services* seperti *e-commerce*, dan menemukan bahwa arsitektur *monolithic* memiliki *throughput* sedikit lebih tinggi untuk operasi CRUD [11]. Szewczyk dan Skublewska-Paszkowska membandingkan beberapa *framework* seperti Node.js, Django, dan ASP.NET, dan melaporkan *average response time* sekitar 30–70 ms untuk Express.js pada *payload* kecil [9]. Namun, kedua studi tersebut berfokus pada operasi CRUD umum dan tidak secara spesifik membahas *ML workloads* yang memiliki karakteristik *latency* dan *payload* berbeda.

2.5.2 ML Inference dan Healthcare Systems

Yan et al. menyusun *roadmap* implementasi ML di *healthcare* dan menyoroti bahwa banyak penelitian hanya berfokus pada akurasi model, sementara metrik performa sistem seperti *response time* dan *throughput* sering diabaikan [7]. Martinez-Garcia et al. menemukan bahwa hanya sebagian kecil studi ML di bidang kesehatan yang melaporkan metrik *system-level performance*, sehingga masih terdapat ruang untuk penelitian yang mengevaluasi performa arsitektur secara menyeluruh [8]. Mattson et al. melalui MLPerf juga menunjukkan pentingnya pengukuran *inference performance*, meskipun fokusnya masih pada *pure inference* tanpa *overhead* arsitektur [14].

2.5.3 Research Gap dan Positioning

Berdasarkan tinjauan literatur, masih terdapat *research gap* terkait evaluasi performa arsitektur *backend* untuk ML *inference* di konteks *healthcare*. Studi yang ada umumnya mengevaluasi operasi CRUD atau *pure model inference* tanpa memasukkan *overhead* dari API *gateway*, komunikasi *multi-tier*, dan *preprocessing pipeline* [9] [11][14]. Selain itu, hanya sedikit penelitian yang melaporkan metrik *system-level performance* yang relevan untuk *clinical deployment* [7][8]. Penelitian ini memosisikan diri untuk mengisi gap tersebut dengan menyediakan evaluasi empiris terhadap performa pola API *gateway* berbasis Node.js dan Express yang terintegrasi dengan AI *Backend* berbasis FastAPI dan TensorFlow pada sistem deteksi kanker berbasis data genomik.