BAB 2
LANDASAN TEORI

2.1 Edge AI dan On-Device Machine Learning

Edge computing adalah pendekatan komputasi yang memindahkan pemrosesan data dari *cloud data center* ke lokasi yang lebih dekat dengan pengguna atau sumber data [20]. Pendekatan ini lahir sebagai respons atas keterbatasan arsitektur cloud, yaitu tingginya latensi, konsumsi bandwidth yang besar, serta kerentanan privasi data yang tidak lagi memadai untuk menangani meningkatnya jumlah perangkat *Internet of Things* (IoT) [21].

Edge Artificial Intelligence (Edge AI) menggabungkan *artificial intelligence* dengan *edge computing* dengan mengeksekusi model machine learning langsung pada perangkat di ujung jaringan, tanpa mengirim data ke server pusat [20, 21, 22]. Pendekatan ini mengurangi latensi, menghemat bandwidth, dan menjaga keamanan data karena data pengguna tidak perlu keluar dari perangkat.

On-device machine learning merupakan bentuk konkret dari Edge AI, di mana model machine learning dijalankan langsung pada perangkat pengguna seperti ponsel, sensor IoT, atau embedded system menggunakan daya komputasi lokal [3, 23]. Untuk mendukung eksekusi pada perangkat dengan resource terbatas, berbagai framework lintas-platform telah dikembangkan seperti TensorFlow Lite, PyTorch Mobile, dan OpenVINO [22], di samping framework spesifik dari masing-masing platform mobile.

2.2 Inferensi Machine Learning

Inferensi (*inference*) merupakan fase eksekusi model machine learning yang telah terlatih, di mana model digunakan untuk menghasilkan prediksi dari data baru melalui proses *forward-pass* tanpa melakukan pembaruan bobot model [24]. Fase ini secara fundamental berbeda dari fase pelatihan, yang membutuhkan alokasi memori untuk bobot, aktivasi, gradien, serta data pelatihan dalam ukuran ratusan megabyte hingga gigabyte [25]. Karena kebutuhan komputasi yang jauh lebih ringan, inferensi dirancang untuk berjalan secara efisien pada perangkat dengan resource terbatas, termasuk perangkat mobile dan *embedded systems* [24].

Pada konteks edge AI, fase inferensi inilah yang secara langsung menghasilkan karakteristik runtime yang dapat diukur pada perangkat, meliputi

latensi, konsumsi energi, dan perilaku termal [17, 26]. Karakteristik tersebut sangat menentukan keberlanjutan sistem dan pengalaman pengguna, terutama pada perangkat mobile dengan sumber daya komputasi dan termal yang terbatas. Oleh karena itu, inferensi menjadi subjek utama dalam analisis karakteristik runtime pada penelitian ini.

2.2.1 Inferensi Berkelanjutan

Inferensi berkelanjutan (*continuous inference*) merupakan salah satu bentuk eksekusi inferensi pada perangkat mobile, di mana model dijalankan secara berulang dalam durasi yang panjang sehingga menghasilkan beban komputasi yang konstan pada perangkat. Skenario ini merepresentasikan beban kerja nyata pada berbagai aplikasi seperti pemrosesan video real-time, asisten suara, dan sistem *computer vision* [11].

Beban kerja yang konstan memunculkan efek-efek kumulatif yang tidak teramati pada eksekusi singkat. Konsumsi daya yang terus-menerus menyebabkan akumulasi panas pada chip SoC, dan ketika suhu mencapai ambang batas tertentu, sistem operasi akan menurunkan frekuensi kerja prosesor guna mencegah kerusakan perangkat keras [11, 17]. Penurunan frekuensi ini secara langsung memperpanjang waktu eksekusi dan menurunkan konsistensi latensi inferensi seiring waktu [11]. Karakteristik runtime tersebut, khususnya perilaku termal dan stabilitas latensi, hanya dapat diamati secara komprehensif pada inferensi yang dijalankan dalam durasi yang cukup panjang. Oleh karena itu, penelitian ini menggunakan inferensi berkelanjutan sebagai desain beban kerja eksperimen.

2.3 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) merupakan jenis arsitektur *neural network* yang dirancang khusus untuk mengolah data berbentuk grid seperti citra, dan menjadi pendekatan yang dominan pada tugas *computer vision* di perangkat mobile [27, 16]. Berbeda dari *neural network* konvensional yang menghubungkan seluruh neuron antar *layer*, CNN memanfaatkan operasi konvolusi untuk mengekstraksi fitur spasial secara lokal, sehingga jumlah parameter dapat ditekan dan komputasi menjadi lebih efisien [16].

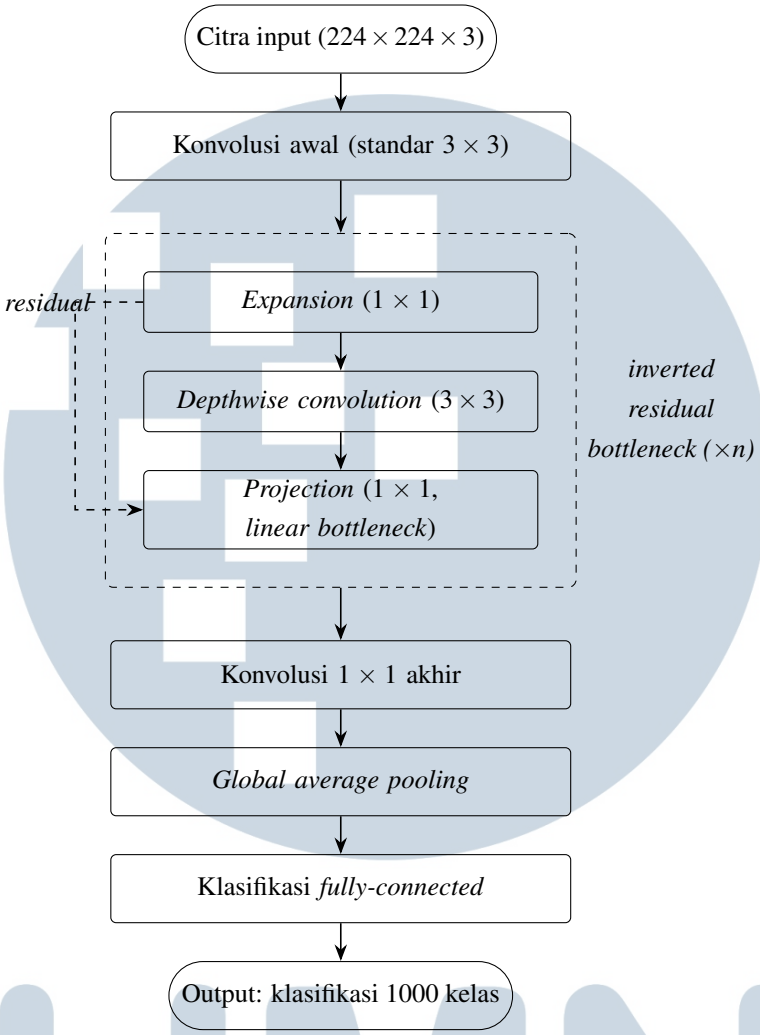
Sebuah CNN umumnya tersusun atas tiga jenis layer utama. Layer konvolusi (*convolution layer*) mengekstraksi fitur spasial seperti tepi, tekstur, dan bentuk

dari citra input menggunakan filter yang digeser ke seluruh bagian citra. Layer *pooling* mereduksi dimensi spasial fitur sambil mempertahankan informasi yang penting, sehingga mengurangi beban komputasi pada layer berikutnya. *Fully connected layer* menggabungkan fitur yang telah diekstraksi untuk menghasilkan output klasifikasi akhir [16, 27]. Melalui susunan layer tersebut, CNN mempelajari representasi fitur secara hierarkis, mulai dari fitur sederhana pada layer awal hingga fitur yang lebih kompleks pada layer yang lebih dalam, tanpa memerlukan rekayasa fitur secara manual. Karakteristik inilah yang menjadikan CNN sebagai arsitektur yang banyak digunakan pada inferensi machine learning di perangkat mobile, dan model yang digunakan pada penelitian ini merupakan salah satu arsitektur dari kelas ini.

2.4 MobileNetV2

MobileNetV2 adalah arsitektur CNN yang dirancang khusus untuk efisiensi pada perangkat dengan resource terbatas [15]. Arsitektur ini memanfaatkan *depthwise separable convolutions*, yang memecah operasi konvolusi standar menjadi tahapan yang lebih ringan, serta *inverted residuals* dengan *linear bottlenecks* untuk menjaga aliran informasi antar layer tanpa menambah beban komputasi secara signifikan [15]. Dibandingkan arsitektur pendahulunya, rancangan ini menurunkan jumlah parameter dan kebutuhan komputasi secara substansial pada tingkat akurasi yang sebanding [24], sehingga MobileNetV2 banyak digunakan sebagai model acuan pada studi inferensi machine learning di perangkat mobile [25]. Arsitektur MobileNetV2 secara keseluruhan ditunjukkan pada Gambar 2.1.

Pada penelitian ini, MobileNetV2 digunakan sebagai satu-satunya model inferensi yang dijalankan pada ketiga konfigurasi compute unit. Dengan demikian, penelitian ini menganalisis karakteristik runtime dari algoritma CNN dengan arsitektur model MobileNetV2, dan setiap perbedaan karakteristik runtime yang teramati dapat dikaitkan dengan perbedaan konfigurasi eksekusi, bukan dengan perbedaan arsitektur model.



Gambar 2.1. Arsitektur MobileNetV2 dengan struktur *inverted residual bottleneck*

2.5 Arsitektur Perangkat Mobile

Perangkat mobile modern umumnya dibangun di atas arsitektur *System on Chip* (SoC), di mana beberapa unit komputasi dengan karakteristik berbeda diintegrasikan ke dalam satu chip yang sama. Pada perangkat iOS yang menjadi fokus penelitian ini, ketiga unit komputasi utama yang berperan dalam eksekusi inferensi machine learning adalah *Central Processing Unit* (CPU), *Graphics Processing Unit* (GPU), dan Apple Neural Engine (ANE) [18]. Setiap unit memiliki arsitektur dan karakteristik kinerja yang berbeda, sehingga distribusi beban kerja inferensi ke unit yang berbeda dapat menghasilkan perilaku runtime yang berbeda pula.

2.5.1 CPU

CPU pada perangkat mobile modern umumnya dibangun di atas arsitektur ARM dengan konfigurasi *big.LITTLE*, yang menggabungkan inti berperforma tinggi dan inti hemat daya dalam satu prosesor untuk menyeimbangkan kecepatan dan efisiensi konsumsi energi. CPU bersifat fleksibel dan mampu menjalankan berbagai jenis algoritma maupun operasi logika kompleks. Namun, arsitektur CPU pada dasarnya bersifat sekuensial dan memiliki keterbatasan dalam mengeksekusi operasi matriks paralel berskala besar, yang mendominasi komputasi pada model *neural network*. Akibatnya, eksekusi inferensi yang sepenuhnya bergantung pada CPU sering kali menghasilkan latensi yang lebih tinggi dibandingkan eksekusi pada akselerator khusus. Meskipun demikian, CPU tetap berperan penting sebagai layer komputasi cadangan (*fallback*) untuk operasi yang tidak didukung oleh akselerator lain [28].

2.5.2 GPU

GPU pada perangkat mobile dirancang dengan arsitektur paralel yang memungkinkan eksekusi ribuan operasi aritmatika secara bersamaan melalui kumpulan unit komputasi yang disebut *compute units*. Sifat paralel ini membuat GPU sangat efektif dalam mempercepat operasi perkalian matriks yang menjadi inti dari komputasi inferensi, sehingga *throughput* inferensi pada GPU dapat meningkat secara signifikan dibandingkan eksekusi pada CPU [18]. Namun, pengerahan inti GPU secara bersamaan untuk komputasi machine learning berdampak langsung pada konsumsi daya yang tinggi, yang berpotensi memicu peningkatan suhu yang signifikan pada perangkat mobile dengan sistem pendinginan pasif [17]. Karakteristik *throughput* yang tinggi dengan konsekuensi termal yang signifikan inilah yang menjadikan GPU sebagai salah satu unit komputasi yang dianalisis secara terpisah pada penelitian ini.

2.5.3 Apple Neural Engine (ANE)

Pada ekosistem Apple Silicon, *Apple Neural Engine* (ANE) merupakan *Neural Processing Unit* (NPU) yang dirancang sebagai arsitektur spesifik domain (*Domain-Specific Architecture*) untuk mengakselerasi komputasi machine learning. ANE dioptimalkan untuk operasi matriks dengan presisi rendah seperti FP16, yang umum digunakan pada inferensi *neural network* modern [18]. Pengukuran

empiris pada generasi terbaru Apple Silicon menunjukkan bahwa ANE mampu menghasilkan kinerja yang sebanding dengan GPU pada beban kerja perkalian matriks, namun dengan konsumsi daya yang jauh lebih rendah. Sebagai ilustrasi, pada chip M4 operasi GEMM yang sama dieksekusi pada konsumsi daya sekitar 5 watt di ANE dibandingkan 24 watt di GPU [18].

Meskipun unggul dalam efisiensi energi, arsitektur ANE bersifat tertutup dan hanya mendukung jenis operasi tertentu secara *native* [8]. Apple tidak mendokumentasikan secara publik aturan kompatibilitas operasi pada ANE, sehingga model yang mengandung operasi tidak standar tidak dapat sepenuhnya dieksekusi oleh ANE dan akan dialihkan ke CPU sebagai *fallback* [8]. Sifat tertutup ini menjadi salah satu faktor yang menyulitkan prediksi perilaku runtime pada konfigurasi yang melibatkan ANE.

Pada perangkat iOS modern, ketiga unit komputasi tersebut beroperasi secara bersamaan dan saling melengkapi. Mendistribusikan beban kerja inferensi ke unit yang paling sesuai bukanlah proses yang sederhana, proses tersebut melibatkan *scheduling*, sinkronisasi, dan alokasi memori antar komponen yang kompleks [17, 28]. Untuk menyederhanakan kompleksitas tersebut bagi pengembang, dibutuhkan sebuah framework yang mampu memetakan beban kerja inferensi ke unit komputasi yang tepat secara otomatis. Pada ekosistem iOS, peran tersebut dijalankan oleh Core ML.

2.6 Core ML Framework

Core ML adalah framework machine learning *native* yang dirancang Apple untuk memfasilitasi eksekusi inferensi machine learning secara on-device pada perangkat dalam ekosistem Apple Silicon [29]. Secara arsitektural, Core ML bertindak sebagai layer abstraksi (*abstraction layer*) yang menjembatani model machine learning dengan ketiga unit komputasi pada perangkat CPU, GPU, dan ANE, sehingga pengembang dapat mengintegrasikan model ke dalam aplikasi tanpa perlu mengelola perangkat keras secara langsung [8]. Melalui abstraksi ini, Core ML secara otomatis membagi beban komputasi model dan mengalokasikan setiap bagian ke unit komputasi yang dianggap paling sesuai pada saat eksekusi [30].

Tingginya tingkat abstraksi ini membawa konsekuensi penting bagi pengembang. Meskipun Core ML menyediakan opsi untuk membatasi unit komputasi yang diizinkan menjalankan model melalui parameter `MLComputeUnits`, pengembang tidak memiliki kendali atas bagaimana Core ML mendistribusikan

operasi di antara unit-unit yang tersedia. Pada konfigurasi yang melibatkan lebih dari satu unit, Core ML beroperasi sebagai *black-box scheduler* yang menentukan secara runtime apakah suatu operasi akan dijalankan pada CPU, GPU, atau ANE, dengan mekanisme scheduling internal yang tidak didokumentasikan secara publik oleh Apple [8]. Pengembang juga tidak dapat memprediksi perilaku runtime yang akan dihasilkan oleh setiap konfigurasi secara analitis [8]. Oleh karena itu, pengukuran empiris menjadi pendekatan utama untuk memahami perilaku runtime Core ML di bawah berbagai konfigurasi eksekusi [19].

2.6.1 Konfigurasi Compute Unit

API Core ML menyediakan parameter `MLComputeUnits` yang memungkinkan pengembang menentukan kebijakan alokasi unit komputasi yang diizinkan untuk mengeksekusi model [31]. Opsi `.cpuOnly` membatasi seluruh eksekusi model agar hanya diproses oleh CPU, sedangkan `.cpuAndGPU` memperluas alokasi dengan turut memanfaatkan GPU. Opsi `.all` menginstruksikan Core ML untuk membagi beban inferensi ke seluruh unit komputasi yang tersedia, termasuk ANE, sesuai dengan keputusan internal penjadwal Core ML [31].

Ketiga opsi tersebut merepresentasikan variasi terkontrol dalam pemanfaatan perangkat keras, mulai dari eksekusi yang dibatasi pada CPU hingga pemanfaatan seluruh unit akselerator. Pada penelitian ini, ketiga konfigurasi tersebut berfungsi sebagai variabel independen yang digunakan untuk mengamati dan membandingkan perbedaan karakteristik runtime yang dihasilkan oleh masing-masing konfigurasi.

2.6.2 Eksekusi Runtime Model

Proses eksekusi model pada Core ML mencakup serangkaian tahapan, mulai dari pemuatan model (*model loading*), kompilasi model ke representasi internal Apple yang disebut *Model Intermediate Language* (MIL), hingga scheduling eksekusi ke perangkat keras [8]. Pada tahap eksekusi, Core ML mempartisi graf komputasi model, yaitu rangkaian operasi seperti konvolusi, perkalian matriks, dan aktivasi yang menyusun model secara dinamis ke beberapa segmen, di mana setiap segmen dipetakan ke unit komputasi (CPU, GPU, atau ANE) yang dianggap paling sesuai untuk mengeksekusi operasi-operasi di dalamnya [8, 30].

Karena keputusan distribusi beban kerja dilakukan secara internal dan tidak

terlihat dari sisi pengembang, perilaku runtime, termasuk latensi, konsumsi energi, dan keterlibatan masing-masing unit komputasi tidak dapat diprediksi sebelum diukur. Pendekatan profiling empiris menjadi metode yang paling dapat diandalkan untuk mengkarakterisasi perilaku runtime Core ML yang sesungguhnya, baik melalui pengukuran latensi per iterasi maupun analisis konsumsi daya selama sesi inferensi berlangsung [18, 19].

2.7 Karakteristik Runtime

Karakteristik runtime adalah metrik perilaku sistem yang dapat diamati dan diukur secara langsung selama eksekusi model machine learning pada perangkat. Pada penelitian ini, empat karakteristik runtime menjadi fokus pengamatan, yaitu latensi, stabilitas, konsumsi energi, dan perilaku termal. Keempat karakteristik tersebut dipilih karena saling berkaitan dalam membentuk perilaku sistem secara menyeluruh dan menjadi metrik utama dalam mengevaluasi perbedaan perilaku runtime yang dihasilkan oleh masing-masing konfigurasi compute unit Core ML.

2.7.1 Latensi

Dalam konteks eksekusi machine learning, latensi inferensi didefinisikan sebagai total waktu yang dibutuhkan sistem untuk menyelesaikan satu iterasi inferensi penuh, terhitung sejak data input diterima hingga prediksi akhir dihasilkan [9]. Latensi merupakan salah satu metrik kinerja yang paling banyak diteliti pada inferensi machine learning di perangkat mobile karena nilainya secara langsung memengaruhi responsivitas aplikasi yang bergantung pada hasil prediksi [9]. Pada sistem *real-time* seperti pemrosesan video langsung dan aplikasi visi komputer, keterlambatan dalam hitungan milidetik dapat mengurangi kualitas pengalaman pengguna atau bahkan membatalkan fungsionalitas aplikasi [32].

Pengukuran latensi pada inferensi machine learning di Core ML telah menjadi fokus pada studi terdahulu. Bazso dan Ahremark mengukur latensi inferensi YOLOv5 pada konfigurasi compute unit Core ML yang berbeda dan menunjukkan bahwa pemanfaatan ANE dapat menurunkan latensi hingga sekitar 33 kali lipat dibandingkan eksekusi TorchScript pada CPU [12]. Studi tersebut mengukur latensi sebagai nilai rata-rata dari 500 iterasi setelah fase *warm-up*, dengan seluruh pengujian dijalankan secara berurutan pada kondisi termal normal, sehingga menghasilkan satu nilai representatif per konfigurasi. Pendekatan tersebut

menggambarkan latensi pada kondisi sesaat, namun belum mengamati bagaimana latensi berkembang seiring waktu pada beban kerja yang berkelanjutan, aspek yang menjadi fokus pengukuran latensi pada penelitian ini.

2.7.2 Stabilitas

Stabilitas runtime merepresentasikan tingkat konsistensi latensi ketika inferensi dijalankan secara repetitif dari waktu ke waktu. Variasi latensi antar iterasi yang dalam literatur sering disebut sebagai *jitter*, dapat muncul pada eksekusi berdurasi panjang akibat persaingan akses resource, intervensi scheduling dari sistem operasi, maupun dinamika perangkat keras seperti penyesuaian frekuensi prosesor [32]. Karena variabilitas tersebut tidak tertangkap pada metrik latensi rata-rata, evaluasi runtime yang lebih lengkap memerlukan pengukuran terhadap konsistensi latensi sepanjang sesi inferensi [32].

Secara kuantitatif, stabilitas dapat dihitung sebagai variansi atau standar deviasi dari nilai latensi yang dikumpulkan sepanjang sesi inferensi, di mana nilai yang lebih rendah mengindikasikan konsistensi kinerja yang lebih baik. Untuk membandingkan stabilitas antar konfigurasi yang memiliki latensi rata-rata berbeda, koefisien variasi (*coefficient of variation*, CV) digunakan sebagai metrik yang ternormalisasi terhadap nilai rata-rata. Stabilitas yang rendah pada aplikasi nyata dapat menciptakan pengalaman pengguna yang buruk, seperti hilangnya konsistensi *frame rate* pada aplikasi visi komputer atau lonjakan waktu respons pada aplikasi interaktif.

2.7.3 Konsumsi Energi

Konsumsi energi merupakan salah satu metrik kritis pada eksekusi machine learning di perangkat mobile karena kapabilitas eksekusi sistem sepenuhnya dibatasi oleh kapasitas baterai perangkat [26]. Analisis energi pada perangkat mobile membedakan antara daya sesaat (*power draw*) yang ditarik oleh unit komputasi pada satu titik waktu dengan total energi kumulatif yang dibutuhkan untuk menyelesaikan suatu beban kerja inferensi sepanjang sesi.

Konsumsi energi memiliki relasi langsung dengan pemilihan unit komputasi yang menjalankan inferensi. Setiap unit memiliki karakteristik *performance-per-watt* yang berbeda, ANE umumnya lebih efisien secara energi pada operasi yang didukungnya secara native, sedangkan GPU dapat memberikan throughput

yang lebih tinggi dengan konsekuensi konsumsi daya yang juga lebih besar [18]. Dengan demikian, efisiensi energi menjadi salah satu dimensi penting dalam membandingkan perilaku runtime antar konfigurasi compute unit Core ML.

Meskipun pemilihan unit komputasi memengaruhi konsumsi energi secara langsung, pengukuran empiris terhadap karakteristik energi pada Core ML masih menjadi celah pada literatur saat ini. Bazso dan Ahremark secara eksplisit mengidentifikasi pengukuran konsumsi daya pada framework deep learning di iOS sebagai aspek penting yang belum dieksplorasi pada studi mereka dan menyarankannya sebagai arah penelitian lanjutan [12]. Penelitian ini menjawab celah tersebut dengan memasukkan konsumsi energi sebagai salah satu karakteristik runtime yang diamati pada setiap konfigurasi compute unit selama sesi inferensi berkelanjutan.

2.7.4 Perilaku Termal

Suhu perangkat merupakan implikasi fisik dari beban komputasi yang berkelanjutan pada arsitektur tertanam. Ponsel pintar umumnya menggunakan sistem pendinginan pasif, sehingga akumulasi panas dari SoC akan memicu sistem operasi untuk mengaktifkan mekanisme *thermal throttling*, yaitu penurunan paksa frekuensi kerja CPU dan GPU guna mencegah perangkat keras mencapai suhu kritis. Intervensi termal ini secara langsung menurunkan performa eksekusi inferensi, latensi memanjang dan stabilitas menurun seiring waktu [17].

Pada platform iOS, pengukuran suhu fisik secara langsung dalam satuan derajat Celsius umumnya tidak dapat diakses oleh aplikasi pihak ketiga. Oleh karena itu, perilaku termal pada penelitian ini diamati melalui dua indikator tidak langsung, yaitu status termal sistem yang dilaporkan melalui API `ProcessInfo.thermalState` [33] dan pola degradasi latensi seiring waktu sebagai manifestasi nyata dari *thermal throttling* [17].

2.7.5 Interaksi Antar Karakteristik

Keempat karakteristik runtime di atas tidak bersifat independen, melainkan terikat dalam rantai kausalitas yang membentuk perilaku sistem secara menyeluruh selama inferensi berkelanjutan. Beban komputasi yang intensif meningkatkan konsumsi daya pada unit-unit komputasi yang aktif, yang pada akhirnya menghasilkan peningkatan suhu. Ketika akumulasi panas mencapai ambang

batas tertentu, sistem operasi mengaktifkan *thermal throttling*, yang menurunkan frekuensi prosesor dan secara langsung memperpanjang latensi serta merusak stabilitas performa [17].

Rantai kausalitas ini: beban komputasi $\rightarrow$ konsumsi energi $\rightarrow$ akumulasi panas $\rightarrow$ *thermal throttling* $\rightarrow$ degradasi latensi dan stabilitas, membentuk model konseptual utama penelitian ini dan menjadi kerangka interpretasi dalam menganalisis hasil pengukuran di setiap konfigurasi compute unit.

Keseluruhan landasan teori dalam bab ini membentuk fondasi konseptual yang kohesif bagi penelitian ini.