BAB II

LANDASAN TEORI

2.1 Penelitian Terdahulu

Berikut merupakan penelitian terdahulu yang dapat mendukung dilakukannya penelitian ini:

Tabel 2. 1 Penelitian Terdahulu

No	Judul dan peneliti Artikel	Nama Jurnal	Metode	Hasil Penelitian
1	**Judul Artikel:** *Real-time Avatar-Based Speech-to-Speech Conversational AI Tutor on AI PC* Nama Peneliti: Mee Sim Lai et al.	IEEE 15th Symposium on Computer Applications & Industrial Electronics (ISCAIE 2025)	Arsitektur multimodal *cascaded* yang mencakup *Noise Reduction* → ASR (Whisper) → RAG → LLM (Llama3 8B) → TTS (Piper) → Avatar (Wav2Lip). Sistem menggunakan *pipeline* modular dengan *frontend–backend* terpisah, model via HTTP API, dan proses *avatar* sinkron menggunakan *multithreading*.	Arsitektur *Speech-to-Speech cascaded* dapat berjalan *real-time* di perangkat *edge* (AI PC). Integrasi semua komponen menghasilkan *voice tutor* interaktif berbasis *avatar* yang responsif, dengan *usability score* tinggi (CUS = 2.72). Sistem berhasil menggabungkan pengenalan suara, *reasoning* berbasis LLM, dan TTS dengan *lip-sync avatar* secara *real-time*.
2	Judul Artikel: *VoiceTalk: A No-Code Approach for Creating Voice-Controlled Smart Home Applications*	IEEE Open Journal of the Computer Society	ASR (Whisper, Web Speech API) → *open-source* SLM (Llama 3.2 3B) → TTS; integrasi *no-*	Whisper Medium + Llama 3.2 3B menghasilkan *error voice-to-text* hampir nol; arsitektur mudah

No	Judul dan peneliti Artikel	Nama Jurnal	Metode	Hasil Penelitian
	Nama Peneliti: Yun-Wei Lin et al.		*code smart home*	diintegrasikan untuk aplikasi *smart home*
3	Judul Artikel: *EmoSDS: Unified Emotionally Adaptive Spoken Dialogue System Using Self-Supervised Speech Representations* Nama Peneliti: Jaehwan Lee et al.	Future Internet	ASR → LLM *open source* (SpeechGPT, Align-SLM, Gemma/Mistral /Zephyr) → TTS (VITS/HiFi-GAN); integrasi SSL & *emotional adaptation*	*Pipeline cascaded* meningkatkan ekspresivitas dan *naturalness* dialog; arsitektur modular, mudah diadaptasi untuk berbagai *domain*
4	**Judul Artikel:** *DQ-Whisper: Joint Distillation and Quantization for Efficient Multilingual Speech Recognition* **Nama Peneliti:** Hang Shao et al.	IEEE SLT 2024	*Quantization* dan *distillation*	Model Whisper dikompresi hingga 5,18x dengan penurunan performa minimal; *inference* lebih efisien tanpa mengorbankan akurasi multibahasa
5	**Judul Artikel:** *Benchmarking Emerging Deep learning Quantization Methods for Energy Efficiency* **Nama Peneliti:** Saurabhsingh Rajput & Tushar Sharma	IEEE ICSA-C 2024	GGUF (GPT-Generated Unified Format)	GGUF terbukti sebagai salah satu metode *quantization* paling efisien secara energi untuk *inference model* besar
6	**Judul Artikel:** *Conversational Payments on UPI Apps: A Pipeline Approach Leveraging ASR and NLP Techniques*	ACM CODS-COMAD 2024	CTranslate2 untuk percepatan *inference*	Penggunaan CTranslate2 mempercepat *inference* model ASR (Whisper) dan NMT, menurunkan *latency* hingga ~600 ms pada aplikasi nyata

No	Judul dan peneliti Artikel	Nama Jurnal	Metode	Hasil Penelitian
	Nama Peneliti: Sai Kasyap Kamaraju et al.			
7	**Judul Artikel:** *Toward Real-time and Efficient Perception Workflows in Software-Defined Vehicles* **Nama Peneliti:** Reza Sumaiya et al.	IEEE Internet of Things Journal	*Pruning*, *multiprecision quantization* (FP32, FP16, INT8), ONNX *Runtime*, TensorRT	Meningkatkan kecepatan *inference* hingga 18x dan *throughput* 16,5x, mengurangi penggunaan GPU/memori hingga 30% dengan dampak minimal pada akurasi
8	**Judul Artikel:** I-*ViT: Integer-only Quantization for Efficient Vision Transformer Inference* **Nama Peneliti:** Zhikai Li & Qingyi Gu	IEEE/CVF ICCV 2022	*Integer-only* INT8 *quantization* dengan *dyadic arithmetic*, approximasi operasi non-linear (Shiftmax, ShiftGELU)	Kecepatan *inference* 3,7–4,1x lebih cepat dibanding FP, akurasi setara atau lebih baik dari model *full precision*
9	**Judul Artikel:** *ESPnet-ONNX: Bridging a Gap Between Research and Production* **Nama Peneliti:** Masao Someki et al.	APSIPA ASC 2022	Konversi model PyTorch ke ONNX, *node fusion*, *quantization parameter*	Percepatan *inference* 1,3–2x pada tugas ASR, TTS, dan lain-lain tanpa pelatihan ulang dan tanpa penurunan performa
10	**Judul Artikel:** *Using Quantized Neural Network for Speaker Recognition on Edge Computing Devices* **Nama Peneliti:** Tongwei Dai	Journal of Physics Conference Series	*Post-training quantization* dengan FBGEMM (per-channel) dan QNNPACK (per-tensor). Konversi FP32 ke INT8 dengan *linear mapping*. Arsitektur VGGNet pada	Pengurangan ukuran model 3x lebih kecil, *inference* speed 1.5x lebih cepat dibandingkan *floating-point model*. Analisis *power consumption* menunjukkan *edge devices* (ASIC) mencapai <1W dengan *throughput* 10-100 GOPS.

No	Judul dan peneliti Artikel	Nama Jurnal	Metode	Hasil Penelitian
			dataset VoxCeleb.	
11	**Judul Artikel:** *Empowering Large Language Models to Edge Intelligence: A Survey of Edge Efficient LLMs and Techniques* **Nama Peneliti:** Rui Wang, et al.	ScienceDirect	Survey komprehensif teknik *model compression* (*quantization, pruning, distillation, low-rank decomposition*), *inference optimization, on-device inference engines*, dan *cloud-edge collaborative frameworks*.	Identifikasi 4 tantangan utama *edge deployment (computational resources, memory footprint, thermal constraints, connectivity)*. Keunggulan *edge*: *latency reduction, privacy enhancement, bandwidth efficiency, offline capability*.
12	**Judul Artikel:** *A Survey on Optimization Techniques for Edge Artificial Intelligence (AI)* **Nama Peneliti:** Rui Wang, et al.	Sensors (MDPI)	Kategorisasi komprehensif optimasi: *hardware optimization* (CPU/GPU/FPGA/ASIC/TPU), *federated learning, model optimization (pruning, quantization, weight sharing, matrix decomposition), hyperparameter tuning*, dan *energy efficiency methods*.	INT8/INT4/mixed-precision *quantization trade-off*s terdokumentasi. CPU *inference* dengan INT8 memberikan *best performance-per-watt* untuk *edge devices*.

Arsitektur *voice conversational AI* berbasis *pipeline cascaded* (ASR, LLM/SLM, dan TTS) merupakan pendekatan yang efektif dan fleksibel dalam berbagai skenario aplikasi. Sistem *speech-to-speech tutor* berbasis *avatar real-time*

telah membuktikan bahwa kombinasi modul *Noise Reduction*, Whisper, RAG–LLM, serta TTS dapat menghasilkan interaksi suara yang natural, responsif, dan dapat dijalankan pada perangkat *edge* secara efisien [32]. Pada aplikasi *smart home*, integrasi Whisper dengan SLM *open-source* dalam *pipeline* ASR–SLM–TTS menghasilkan performa transkripsi yang sangat akurat serta kemudahan integrasi dengan sistem kendali perangkat rumah pintar [33]. Sementara itu, pengembangan *emotionally adaptive dialogue systems* berbasis *pipeline* ASR, LLM, dan TTS menunjukkan bahwa penambahan representasi suara berbasis *self-supervised learning* mampu meningkatkan *naturalness* dan ekspresivitas interaksi suara [34]. Secara keseluruhan, temuan-temuan tersebut menegaskan bahwa arsitektur *cascaded* ASR, LLM/SLM, dan TTS adalah fondasi yang kuat, modular, serta dapat disesuaikan untuk berbagai kebutuhan pengembangan *voice conversational* AI modern.

Beberapa penelitian menunjukkan bahwa optimasi *inference* pada Whisper dapat dilakukan melalui berbagai pendekatan yang berfokus pada kompresi model, efisiensi energi, dan percepatan eksekusi di perangkat dengan sumber daya terbatas. Pendekatan *quantization-distillation* terbukti mampu mengurangi ukuran model secara signifikan tanpa menurunkan akurasi secara berarti, sehingga menghasilkan performa *inference* yang lebih ringan dan sesuai untuk *deployment* pada lingkungan komputasi rendah [35]. Penelitian lain juga menunjukkan bahwa penggunaan format model yang dioptimalkan seperti GGUF dapat meningkatkan efisiensi memori dan konsumsi energi selama *inference*, menjadikannya salah satu pilihan terbaik untuk menjalankan model berukuran besar pada perangkat *edge* [36]. Selain itu, percepatan *inference* melalui *framework* seperti CTranslate2 telah menghasilkan *latency* yang sangat rendah dalam aplikasi *real-time*, menunjukkan bahwa optimasi berbasis *runtime* dapat memberikan peningkatan performa yang substansial pada sistem yang mengandalkan pemrosesan suara secara langsung [37]. Temuan-temuan ini menegaskan bahwa optimasi Whisper melalui *quantization*, format model ringkas, dan percepatan *runtime* merupakan landasan penting dalam meningkatkan efisiensi ASR pada sistem *voice conversational* AI modern.

Optimasi *inference* pada *small language models* (SLM) dapat dicapai melalui teknik *post-training quantization* yang tidak memerlukan pelatihan ulang model. GGUF (GPT-Generated Unified Format) terbukti sebagai salah satu metode *quantization* paling efisien secara energi untuk *inference* model besar, dengan kemampuan mengurangi presisi numerik bobot model sambil mempertahankan kualitas *output* yang *acceptable* [36]. Format GGUF dirancang khusus untuk mendukung berbagai tingkat *quantization* seperti INT8, INT4, dan mixed-precision, memungkinkan *trade-off* yang fleksibel antara ukuran model, kecepatan *inference*, dan akurasi prediksi. Implementasi GGUF pada SLM seperti Gemma 3 1B dapat dilakukan melalui konversi model ke format binary yang teroptimasi, disertai dengan metadata *quantization* yang memfasilitasi loading dan eksekusi model secara efisien pada perangkat dengan sumber daya komputasi terbatas. Secara keseluruhan, temuan-temuan tersebut menunjukkan bahwa optimasi Gemma 3 1B dapat dilakukan tanpa melakukan pelatihan ulang model, melainkan melalui penerapan *weight-only quantization* berbasis GGUF yang mendukung *deployment* efisien pada lingkungan CPU-*only*.

Optimasi *inference* pada model TTS berbasis VITS dapat dicapai melalui teknik kompresi dan percepatan *runtime* yang berfokus pada efisiensi eksekusi di perangkat dengan sumber daya terbatas. Pendekatan yang menggabungkan *pruning* dan *multiprecision quantization* dengan *backend* seperti ONNX *Runtime* dan TensorRT terbukti mampu meningkatkan kecepatan *inference* secara signifikan tanpa mengorbankan kualitas keluaran *audio*, sehingga relevan untuk *deployment* VITS dalam aplikasi *real-time* [31]. Teknik *integer-only quantization* juga telah ditunjukkan mampu mengurangi kompleksitas komputasi secara substansial dengan menjalankan seluruh operasi menggunakan aritmatika *integer*, sambil mempertahankan stabilitas akustik model, sehingga dapat diterapkan pada komponen VITS yang mengandalkan arsitektur transformer [38]. Selain itu, optimasi melalui konversi model PyTorch ke ONNX dilakukan dengan peningkatan seperti *node fusion*, *graph optimization*, dan *quantization* yang telah terbukti memberikan percepatan *inference* yang konsisten tanpa menurunkan kualitas suara [39], menjadikannya teknik yang sangat praktis untuk pengembangan dan produksi

model VITS. Secara keseluruhan, temuan-temuan ini menegaskan bahwa kombinasi *quantization* dan optimasi ONNX *Runtime* merupakan strategi efektif untuk meningkatkan efisiensi dan performa model VITS dalam sistem TTS *real-time* maupun lingkungan komputasi berdaya rendah.

Deployment voice conversational AI pada lingkungan CPU-*only* memiliki landasan ilmiah berdasarkan tiga aspek fundamental, yaitu aksesibilitas infrastruktur, kebutuhan privasi dan *compliance*, serta efisiensi operasional. Penelitian menunjukkan bahwa mayoritas *edge devices* dan *local server*s di lingkup *enterprise* hanya dilengkapi dengan CPU tanpa akselerasi GPU atau TPU khusus [25], [26]. Dalam kebutuhan *deployment*, khususnya untuk aplikasi telekomunikasi, layanan kesehatan, dan layanan finansial, pemrosesan data sensitif secara lokal menjadi keharusan untuk memenuhi regulasi seperti HIPAA, GDPR, dan PCI-DSS yang melarang transmisi *raw audio* ke *cloud* [25]. *Edge*-based voice AI processing pada CPU memungkinkan *latency* reduction yang signifikan, yaitu dari rata-rata 200ms *cloud latency* menjadi <10ms local processing yang krusial untuk conversational flow yang natural [25], [40]. Studi terbaru membuktikan bahwa dengan teknik *quantization* yang tepat, khususnya INT8 *quantization* pada CPU, performa *inference* dapat mencapai 3-4x lebih cepat dibandingkan *floating-point* model dengan penurunan akurasi yang minim (<1%) [26], [41].

Kontribusi utama penelitian optimasi *end-to-end pipeline* pada CPU-*only* terletak pada integrasi sistematis multi-component *quantization* yang belum dieksplorasi secara komprehensif dalam literatur yang sudah ada. Penelitian sebelumnya cenderung fokus pada optimasi single-component atau *deployment* hybrid cloud-*edge* [25], [26], sementara optimasi setiap ketiga komponen (ASR, LLM, TTS) dengan teknik *quantization* yang berbeda-beda dalam satu *pipeline* terpadu masih terbatas [40]. Hasil optimasi setiap komponen menunjukkan performa yang sangat baik, namun integrasi antar-komponen belum terdokumentasi baik.

Meskipun penelitian-penelitian sebelumnya telah menunjukkan efektivitas optimasi individual pada komponen ASR, LLM/SLM, dan TTS, terdapat beberapa

research gap yang belum terpenuhi secara komprehensif. Mayoritas penelitian terdahulu berfokus pada optimasi *single-component* dalam lingkungan *hybrid cloud-edge* atau GPU-*accelerated*. Meskipun arsitektur *cascaded* telah terbukti efektif, belum terdapat studi sistematis yang mengintegrasikan optimasi *inference*-level secara keseluruhan komponen dengan teknik *quantization* yang beragam dalam satu *pipeline* terpadu. Oleh karena itu, penelitian ini berkontribusi untuk untuk *research gap* tersebut dengan mengimplementasikan dan mengevaluasi sistem *voice conversational* AI berbasis arsitektur *cascaded* yang dioptimasi secara *inference-level* pada lingkungan CPU-*only*. Kontribusi penelitian mencakup implementasi strategi optimasi multi-komponen dengan beberapa teknik berbeda, validasi bahwa optimasi *inference*-level mampu mencapai pengurangan *latency* dengan tetap mempertahankan kualitas, serta membuktikan kapabilitas *deployment* pada sistem berbasis CPU ataupun *local hosting*. Sehingga, penelitian mampu menjawab kebutuhan industri dengan perhatian terhadap sensitivitas data dan spesifikasi sistem yang minim, yang dapat digunakan pada kebutuhan aplikasi seperti telekomunikasi, layanan kesehatan, dan layanan finansial.

2.2 Teori yang Berkaitan

2.2.1 *Artificial Intelligence* (AI)

Artificial Intelligence (AI) adalah bidang multidisiplin yang berfokus pada pengembangan sistem komputer yang mampu melakukan tugas-tugas yang biasanya memerlukan kecerdasan manusia, seperti memahami bahasa, mengenali suara, memecahkan masalah, dan membuat keputusan [42]. AI telah membawa manfaat signifikan di berbagai sektor, termasuk peningkatan efisiensi, otomatisasi proses, dan personalisasi layanan [43]. Dalam konteks teknologi *voice conversational* AI, seperti sistem *Automatic Speech Recognition* (ASR), *Large Language Model* (LLM), dan *Text-to-Speech* (TTS), AI berperan sebagai inti yang memungkinkan komputer memahami, memproses, dan merespons percakapan manusia secara alami [44]. Sistem ini menggabungkan kemampuan pengenalan suara, pemrosesan bahasa alami dan penalaran, serta sintesis suara untuk menciptakan interaksi yang lebih

manusiawi dan responsif, yang telah diadopsi luas dalam layanan pelanggan, asisten virtual, dan aplikasi edukasi [45].

Secara teknis, AI bekerja dengan membangun sistem cerdas yang mampu belajar dari data dan mengambil keputusan secara otonom. Proses ini melibatkan penggunaan algoritma *machine learning* dan *deep learning*, di mana sistem dilatih menggunakan data besar untuk mengenali pola, membuat prediksi, dan meningkatkan kinerjanya seiring waktu tanpa instruksi eksplisit [46]. Sistem AI modern memanfaatkan jaringan saraf tiruan atau *artificial neural networks* yang meniru cara kerja otak manusia untuk memproses informasi kompleks, seperti pengenalan suara dan pemahaman bahasa alami. Kemampuan pengambilan keputusan AI didasarkan pada analisis data, penalaran logis, dan pembelajaran berkelanjutan, sehingga AI dapat memberikan rekomendasi, memecahkan masalah, dan menyesuaikan respons sesuai konteks [47]. Dengan demikian, AI tidak hanya meniru kecerdasan manusia, tetapi juga memperluas kapasitas pengambilan keputusan berbasis data di berbagai bidang aplikasi.

2.2.2 *Deep learning*

Deep learning adalah cabang dari *machine learning* yang menggunakan arsitektur *deep neural networks* untuk mempelajari representasi data secara otomatis dan hierarkis [48]. Peranannya sangat penting dalam pengembangan model-model modern, karena mampu mengekstraksi fitur kompleks dari data mentah tanpa memerlukan rekayasa fitur manual. Dalam sistem *voice conversational* AI, khususnya ASR dan TTS, *deep learning* telah merevolusi performa dan akurasi [49]. Model seperti Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), dan Transformer telah digunakan secara luas untuk mengenali pola suara, memahami konteks percakapan, serta menghasilkan suara sintetis yang alami, sehingga meningkatkan kualitas interaksi manusia-mesin [50].

Secara teknis, *deep learning* bekerja dengan membangun jaringan saraf berlapis-lapis, di mana setiap lapisan (*layer*) bertugas mengekstraksi

representasi fitur yang semakin abstrak dari data *input* [51]. Proses pembelajaran dilakukan melalui mekanisme *forward propagation*, di mana data mengalir dari *input* ke *output*, dan *backward propagation*, di mana bobot jaringan diperbarui berdasarkan *error* yang dihitung dari *output* menggunakan algoritma *backpropagation* [52]. Dengan pendekatan ini, model *deep learning* mampu secara otomatis belajar dari data besar, menyesuaikan *parameter* internalnya untuk meminimalkan kesalahan, dan menemukan pola-pola penting tanpa intervensi manusia. Kemampuan ini menjadikan *deep learning* sangat efektif dalam menangani data kompleks seperti suara dan bahasa, serta memungkinkan pengambilan keputusan berbasis representasi fitur yang dihasilkan secara otomatis [53].

2.2.3 Inference-Level Optimization

Inference-level optimization merupakan serangkaian teknik yang diterapkan pada model *deep learning* untuk meningkatkan efisiensi komputasi selama fase inferensi tanpa mengubah arsitektur dasar model atau melakukan pelatihan ulang [54]. Optimasi pada tingkat inferensi menjadi sangat krusial karena fase inferensi adalah tahap di mana model *deep learning* digunakan secara aktif dalam aplikasi produksi, memproses data dalam jumlah besar dengan kebutuhan *latency* rendah dan *throughput* tinggi [55]. Berbeda dengan optimasi pada fase pelatihan yang berfokus pada konvergensi model, optimasi inferensi bertujuan untuk mengurangi *computational footprint, memory bandwidth*, dan *inference latency* tanpa mengorbankan akurasi prediksi secara signifikan [56]. Teknik-teknik optimasi inferensi mencakup *graph-level optimization* seperti *operator fusion* dan *constant folding*, serta *algorithm-level optimization* seperti *quantization* dan *pruning* yang secara kolektif dapat menghasilkan peningkatan performa hingga 3-5x pada berbagai arsitektur *neural network* [57].

Inference-level optimization bekerja melalui transformasi pada *computational graph* dan *parameter* model yang telah terlatih untuk memaksimalkan efisiensi eksekusi pada target *hardware* tertentu. *Post-training optimization* (PTO) merupakan pendekatan yang paling umum digunakan karena dapat diterapkan pada model *pre-trained* tanpa memerlukan siklus

pelatihan tambahan, dengan teknik utama meliputi *graph optimization* yang menggabungkan beberapa operasi menjadi *single kernel* untuk mengurangi kebutuhan *memory bandwidth*, dan *post-training quantization* yang mengkonversi representasi numerik dari *floating-point* 32-bit (FP32) menjadi *integer* 8-bit (INT8) atau bahkan presisi lebih rendah [58]. Operator fusion sebagai salah satu teknik *graph optimization* dapat mengeliminasi alokasi *intermediate tensor* dan mengurangi *kernel launch overhead*, sementara *quantization* mampu mengurangi ukuran model hingga 75% dengan degradasi akurasi yang minim melalui pengurangan *bit-width* untuk *weights* dan *activations* [59]. Implementasi *inference optimization* yang efektif mempertimbangkan karakteristik target *hardware* dan profil *workload* aplikasi, sehingga menghasilkan *trade-off* optimal antara *latency, throughput, memory usage,* dan akurasi untuk *deployment* pada *resource-constrained environments* seperti *edge devices* dan *mobile platforms* [60].

2.2.4 *Small Language Model* (SLM) / *Large Language Model* (LLM)

Small Language Model (SLM) dan *Large Language Model* (LLM) adalah dua kategori model bahasa berbasis *neural network* yang digunakan untuk pemrosesan bahasa alami. SLM umumnya memiliki jumlah *parameter* yang lebih sedikit, sehingga lebih efisien secara komputasi dan cocok untuk aplikasi dengan sumber daya terbatas, namun cenderung memiliki kapasitas generalisasi dan pemahaman bahasa yang lebih terbatas dibandingkan LLM [61]. Sebaliknya, LLM memiliki ratusan juta hingga miliaran *parameter*, dilatih pada data teks dalam skala besar, dan mampu memahami serta menghasilkan teks yang sangat kompleks dan kontekstual [62]. Dalam *pipeline voice conversational* AI, baik SLM maupun LLM berperan sebagai komponen pemrosesan bahasa alami yang menginterpretasi hasil transkripsi ASR, memahami maksud pengguna, dan menghasilkan respons yang kemudian diubah kembali menjadi suara oleh TTS. SLM menawarkan keunggulan efisiensi dan kemudahan *deployment,* sedangkan LLM unggul dalam kualitas respons dan kemampuan generalisasi lintas *domain*.

Secara teknis, model bahasa modern bekerja melalui beberapa tahapan utama. Proses dimulai dengan tokenisasi, yaitu memecah teks menjadi unit-unit kecil (token) yang dapat diproses oleh model. Selanjutnya, token-token ini diubah menjadi vektor representasi (*embedding*) dan diproses melalui arsitektur transformer, yang terdiri dari lapisan-lapisan *self-attention* dan *feed-forward neural network* [63]. Mekanisme *attention* memungkinkan model untuk menimbang relevansi antar token dalam satu konteks, sehingga dapat memahami hubungan semantik dan sintaksis secara dinamis. Model kemudian menghasilkan respons bahasa secara autoregresif, yaitu memprediksi token berikutnya berdasarkan urutan token sebelumnya, dengan teknik *decoding* seperti *greedy search*, top-k, atau top-p sampling untuk mengontrol kreativitas dan koherensi respons. Dengan pendekatan ini, baik SLM maupun LLM dapat menghasilkan teks yang relevan dan kontekstual, meskipun LLM umumnya lebih unggul dalam menangani konteks yang panjang dan kompleks [64].

2.2.5 *Speech-to-Text* (STT) / *Automatic Speech Recognition* (ASR)

Automatic Speech Recognition (ASR) atau *Speech-to-Text* (STT) adalah teknologi yang memungkinkan sistem komputer untuk mengkonversi sinyal suara manusia menjadi teks secara otomatis [65]. ASR berperan penting sebagai komponen awal dalam *pipeline voice conversational* AI, di mana hasil transkripsi dari suara menjadi teks digunakan untuk pemrosesan bahasa alami dan interaksi lanjutan dengan pengguna. Manfaat utama ASR meliputi peningkatan aksesibilitas bagi penyandang disabilitas, efisiensi dalam layanan pelanggan, transkripsi otomatis untuk dokumentasi, serta mendukung berbagai aplikasi seperti asisten virtual, sistem dikte, dan layanan terjemahan. Dengan kemajuan *deep learning*, akurasi dan keandalan ASR semakin meningkat, meskipun tantangan seperti kebisingan lingkungan, variasi aksen, dan kecepatan bicara masih menjadi perhatian utama.

Secara teknis, proses kerja ASR dimulai dari tahap *preprocessing audio*, di mana sinyal suara diolah melalui segmentasi, *filtering*, dan normalisasi untuk mengurangi *noise* dan menyiapkan data mentah [66]. Selanjutnya, dilakukan ekstraksi fitur menggunakan teknik seperti *Mel-Frequency Cepstral*

Coefficients (MFCC) atau *spectrogram*, yang mengubah sinyal *audio* menjadi representasi numerik yang lebih mudah dianalisis oleh mesin. Fitur-fitur ini kemudian diproses oleh model *neural network* modern, seperti arsitektur *encoder–decoder* berbasis *attention*, LSTM, atau Transformer, yang mampu mempelajari hubungan temporal dan pola kompleks dalam data suara. Model ini secara otomatis memetakan urutan fitur akustik menjadi urutan teks, dengan proses pelatihan yang mengoptimalkan *parameter* jaringan untuk minimalisir kesalahan transkripsi. Pendekatan *end-to-end* ini terbukti meningkatkan akurasi dan efisiensi ASR dalam berbagai kondisi dan bahasa [67].

2.2.6 *Text-to-Speech* (TTS)

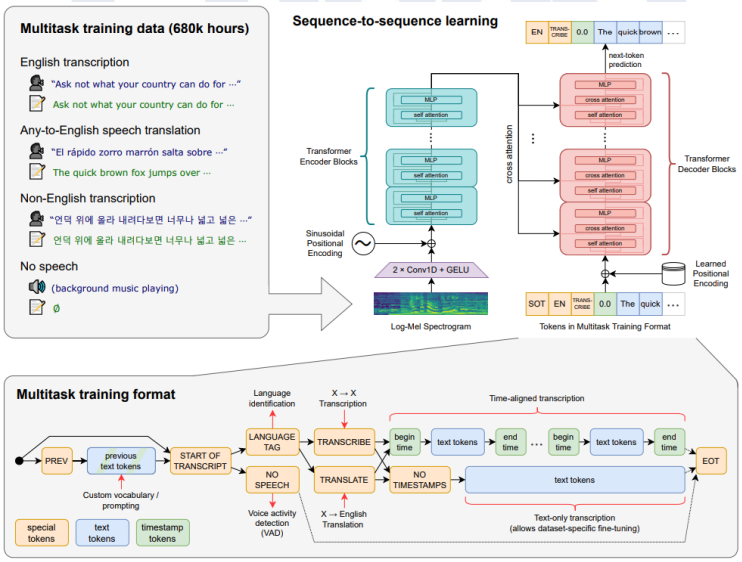
Text-to-Speech (TTS) adalah teknologi yang mengubah teks tertulis menjadi suara manusia buatan yang terdengar alami. TTS berperan sebagai komponen akhir dalam sistem *voice conversational* AI, memungkinkan sistem untuk memberikan respons verbal yang mudah dipahami dan interaktif bagi pengguna [68]. Manfaat utama TTS meliputi peningkatan aksesibilitas bagi penyandang disabilitas, otomatisasi layanan pelanggan, pembacaan buku digital, serta personalisasi asisten virtual. Dengan kemajuan *deep learning*, kualitas suara yang dihasilkan TTS semakin mendekati suara manusia asli, sehingga interaksi antara manusia dan mesin menjadi lebih natural dan efektif.

Secara konseptual, sistem TTS modern terdiri dari beberapa tahapan utama. Proses dimulai dengan *text analysis*, yaitu analisis linguistik untuk mengidentifikasi struktur kalimat, pengucapan fonem, intonasi, dan fitur prosodi lainnya [69]. Hasil analisis ini kemudian diproses oleh *acoustic model*, biasanya berbasis *neural network* seperti *encoder–decoder* atau transformer, untuk menghasilkan representasi akustik berupa *mel-spectrogram* atau fitur serupa. Tahap akhir adalah *vocoder-based waveform synthesis*, di mana *vocoder*, seperti *WaveNet*, HiFi-GAN, atau WaveGlow mengubah representasi akustik menjadi gelombang suara digital yang dapat diperdengarkan. Pendekatan *end-to-end* dan penggunaan arsitektur *deep learning* telah meningkatkan naturalitas, ekspresivitas, dan fleksibilitas sistem TTS secara signifikan.

2.3 *Framework*/Algoritma yang digunakan

2.3.1 Whisper

Whisper merupakan *automatic speech recognition* (ASR) yang dikembangkan oleh OpenAI dengan pendekatan *large-scale weak supervision training* menggunakan 680,000 jam data *audio* multilingual dan *multitask* yang dikumpulkan dari internet [70]. Model ini dirancang untuk mengatasi keterbatasan ASR tradisional yang memerlukan *dataset-specific fine-tuning* dengan kemampuan generalisasi melalui *zero-shot transfer learning* ke berbagai *benchmark datasets* tanpa memerlukan pelatihan tambahan. Whisper dilatih secara *multitask* untuk menangani beragam *speech processing tasks* dalam *single unified model*, termasuk *multilingual speech recognition* yang mendukung lebih dari 90 bahasa, *speech translation* ke bahasa Inggris, *spoken language identification*, dan *voice activity detection*, di mana seluruh tugas tersebut direpresentasikan sebagai *sequence of tokens* yang diprediksi oleh komponen *decoder*. Karakteristik dari Whisper adalah *robustness* yang tinggi terhadap variasi aksen, *background noise*, dan *technical language*, yang membuatnya mampu berperforma mendekati *human-level accuracy* pada berbagai kondisi akustik yang menantang [70]. Model ini tersedia dalam berbagai ukuran mulai dari *Tiny* (39M *parameters*) hingga *Large* (1550M *parameters*).



Gambar 2. 1 Arsitektur Model Whisper [70]

Sesuai dengan Gambar 2.1, arsitektur Whisper mengimplementasikan *encoder-decoder* Transformer *architecture* dengan desain yang sederhana namun efektif untuk *speech-to-text conversion*. *Audio processing pipeline* dimulai dengan segmentasi *input audio* menjadi *chunks* berdurasi 30 detik, yang kemudian dikonversi menjadi log-Mel *spectrogram representation* melalui Short-Time Fourier Transform (STFT) yang dilakukan pada *windowed segments* dari *audio waveform* [70]. Komponen *Encoder* terdiri dari *multiple layers* Transformer *blocks* yang menerima log-Mel *spectrogram* sebagai *input* dan menghasilkan *high-dimensional audio feature embeddings* yang meng-*capture acoustic* dan *phonetic information* dari *spoken content*. Komponen *Decoder* menggunakan *autoregressive generation mechanism* untuk memprediksi transkripsi teks secara *sequential token-by-token*, di mana setiap *generated token* di-*condition* pada *generated tokens* sebelumnya dan *audio features* dari *encoder* melalui *cross-attention mechanism*. Whisper memanfaatkan *special tokens system* untuk mengontrol *multitask behavior*, termasuk *language identification tokens, task specification tokens* untuk membedakan antara *transcription* dan *translation tasks, timestamp prediction tokens* untuk *phrase-level temporal alignment*, dan *end-of-text token* untuk menandakan selesainya dari *generation process*. *Training objective* menggunakan *standard cross-entropy loss* untuk *next-token prediction* dengan *teacher forcing strategy*, di mana model belajar untuk memaksimalkan kesamaan dari transkripsi *audio input* dan *generated tokens* sebelumnya. Arsitektur ini memungkinkan Whisper untuk berfungsi sebagai *single unified model* yang menggantikan banyak komponen spesifik dalam *speech processing pipeline* tradisional, sekaligus menjaga fleksibilitas untuk mengatasi perbedaan bahasa dan kondisi akustik tanpa membutuhkan modifikasi arsitektur atau *task-specific adaptations*.

2.3.2 Gemma 3 1B

Gemma 3 1B merupakan *lightweight small language model* yang dikembangkan oleh Google DeepMind sebagai bagian dari Gemma 3 model *family* yang dirilis pada tahun 2025, dengan ukuran *parameter* 1 miliar yang

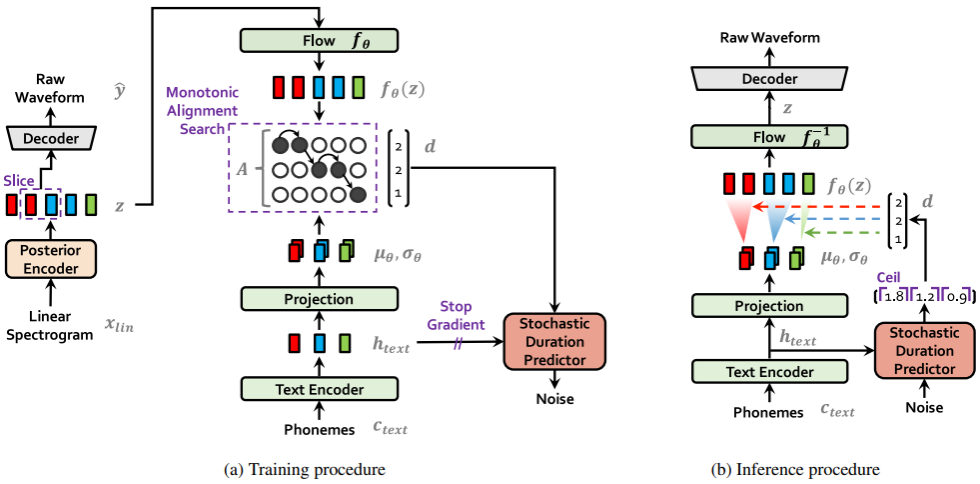
dirancang untuk *deployment* pada *resource-constrained environments* seperti *edge devices* dan *consumer-grade hardware* [71]. Model ini dilatih menggunakan 2 triliun *tokens* dari beragam korpus teks yang mencakup *web documents* dalam lebih dari 140 bahasa, *code repositories*, dan *mathematical content* dengan *knowledge cutoff date* pada Agustus 2024, sehingga memiliki beragam *linguistic coverage* dan kemampuan multilingual yang superior dibandingkan models pendahulunya. Gemma 3 1B mengimplementasikan improvisasi arsitektur termasuk *interleaved local-global attention mechanism* yang didesain untuk mengurangi konsumsi KV-*cache memory* pada *long-context inference*, serta mendukung *context window* hingga 32,000 *tokens* yang memungkinkan *processing* dari sejarah obrolan lebih panjang atau dokumen [71]. *Training process* melibatkan teknik seperti knowl*edge distillation* dari model yang lebih besar, serta fase *reinforcement learning* yang menggunakan *multiple reward functions* untuk peningkatan performa dan *reasoning*.

Arsitektur Gemma 3 1B mengadopsi *decoder-only* Transformer *architecture* dengan total 1,152 *hidden dimensions* dan 8 *attention heads* yang mengimplementasikan Grouped-Query Attention (GQA) untuk meningkatkan efisiensi *inference* dengan mengurangi kebutuhan *memory bandwidth* selama *autoregressive generation* [71]. Model menggunakan Gemini 2.0 SentencePiece *tokenizer* dengan *vocabulary size* 262,000 *tokens* yang dioptimasi untuk menangani *multilingual text*, digit numerikal, dan *whitespace characters*, sehingga meningkatkan efisiensi *tokenization*. *Layer normalization* menggunakan RMSNorm yang diaplikasikan pada *pre-norm* dan *post-norm positions* untuk menstabilkan *training dynamics*, dengan perubahan arsitektural dari Gemma 2 yaitu penggantian dari *soft-capping activation* dengan QK-*normalization mechanism* yang menormalisasi *query* dan *key matrices* sebelum *attention computation* untuk mencegah ketidakstabilan *numerical* pada *large-scale training*. *Input processing* melibatkan *tokenization* dari *text input* menjadi barisan *token IDs*, diikuti dengan *embedding lookup tokens* ke *high-dimensional vector representations*, kemudian melewati tumpukan Transformer *decoder layers* yang terdiri dari *self-attention mechanism* untuk *modeling contextual*

relationships dan *feed-forward network* untuk *feature transformation*. Proses generasi menggunakan *autoregressive decoding* di mana model memprediksi *token* berikutnya berdasarkan pada *tokens* yang di-*generate* sebelumnya.

2.3.3 VITS MMS

VITS (*Variational Inference with adversarial learning for end-to-end Text-to-Speech*) MMS *Indonesian* merupakan *text-to-speech synthesis model* yang dikembangkan sebagai bagian dari project *Massively Multilingual Speech* (MMS) Facebook yang bertujuan untuk menyediakan teknologi *speech* dalam lebih dari 1,000 bahasa termasuk bahasa Indonesia [72]. Model ini mengimplementasikan *end-to-end speech synthesis architecture* yang mengatasi keterbatasan *two-stage* TTS *systems* yang tradisional dengan menggabungkan *acoustic modeling* dan *vocoding* dalam *single unified framework*, sehingga menghilangkan *intermediate representation* seperti mel-spectrogram dan mengaktifkan *direct waveform generation* dari teks *input*. VITS memanfaatkan *conditional variational autoencoder framework* yang *augmented* dengan normalisasi *flows* dan *adversarial training* untuk meningkatkan kekuaran ekspresif dari *generative modeling*, memungkinkan sintesis dari *speech waveforms* yang natural dan ekspresif [73]. Model ini dilatih secara *monolingual* untuk setiap bahasa dengan beberapa *checkpoint*, di mana VITS MMS *Indonesian* di-*trained* menggunakan data *speech* berbahasa Indonesia yang dikumpulkan melalui MMS-lab yang memanfaatkan pembacaan teks sebagai sumber data utama. Karakteristik dari VITS adalah kemampuannya untuk membahas *one-to-many nature* dari masalah TTS di mana teks *input* yang sama dapat diucapkan dengan beberapa cara dengan nada dan ritme yang berbeda, melalui implementasi dari *stochastic duration predictor* yang *generate* pola ucapan yang berbeda dari teks *input* serupa.



Gambar 2. 2 Arsitektur Model VITS [73]

Sesuai dengan Gambar 2.2, arsitektur VITS mengadopsi struktur *conditional variational autoencoder* yang terdiri dari tiga komponen utama: *posterior encoder* yang mengkodekan bentuk gelombang ucapan referensi menjadi representasi laten, *conditional prior* yang memodelkan distribusi variabel laten yang dikondisikan pada teks *input*, dan *decoder* yang menghasilkan bentuk gelombang ucapan dari variabel laten [73]. *Pipeline* pemrosesan teks dimulai dengan tokenisasi menggunakan VitsTokenizer yang disesuaikan untuk bahasa Indonesia, mengonversi teks menjadi representasi fonem yang di-*embed* menjadi vektor fitur berdimensi tinggi. Komponen *conditional prior* menggunakan *text encoder* berbasis Transformer untuk mengekstraksi informasi kontekstual, diikuti modul normalisasi *flow* dengan *multiple affine coupling layers* untuk meningkatkan fleksibilitas distribusi prior, dan *projection layer* untuk memetakan representasi ke dimensi ruang laten. *Stochastic duration predictor* mengimplementasikan model generatif berbasis *flow* yang memprediksi durasi fonem dengan menggabungkan variabilitas melalui dua variabel acak, memungkinkan model menghasilkan ucapan dengan variasi ritme natural. *Posterior encoder* memproses mel-spectrogram referensi yang dihitung dari bentuk gelombang *ground truth* menggunakan *Short-Time Fourier Transform* dengan tumpukan WaveNet *residual blocks*, di mana penyelarasan teks-*audio* dicapai melalui algoritma *Monotonic Alignment*

Search (MAS) yang secara otomatis menemukan korespondensi optimal tanpa memerlukan anotasi *forced alignment* eksternal. *Decoder* menggunakan arsitektur identik dengan generator HiFi-GAN, terdiri dari tumpukan *transposed convolutional layers* dengan *multi-receptive field fusion* untuk mengonversi representasi laten menjadi bentuk gelombang *audio* mentah pada *sampling rate* 16kHz.

2.3.4 *Quantization*

Quantization merupakan salah satu teknik kompresi model yang digunakan dalam *deep learning* untuk meningkatkan efisiensi komputasi dengan menurunkan presisi representasi numerik pada bobot maupun aktivasi jaringan saraf. Teknik ini mengubah representasi *floating-point* seperti FP32 atau FP16 menjadi representasi *integer* berdimensi rendah, misalnya INT8 atau INT4, sehingga ukuran model berkurang secara signifikan dan proses *inference* menjadi lebih cepat dengan konsumsi memori serta daya yang lebih rendah [74]. *Quantization* banyak digunakan dalam *deployment* model pada perangkat berdaya rendah seperti CPU, *edge device*, maupun perangkat *mobile* karena mampu menghasilkan percepatan komputasi tanpa penurunan akurasi yang berarti. Selain itu, *quantization* memungkinkan model *deep learning* tetap dapat dijalankan secara *real-time*, bahkan ketika lingkungan komputasi terbatas, sehingga menjadi salah satu pendekatan optimasi paling penting dalam implementasi model modern, termasuk model berbasis transformer, *convolutional network*, maupun model generatif.

Salah satu pendekatan *quantization* yang paling umum digunakan adalah *Post-training Quantization* (PTQ), yaitu metode yang diterapkan setelah model selesai dilatih tanpa memerlukan proses pelatihan ulang [75]. PTQ bekerja dengan memetakan bobot dan aktivasi model ke skala *integer* melalui proses *scaling* dan *rounding*, sehingga model dapat dijalankan menggunakan operasi aritmatika *integer* yang jauh lebih efisien dibanding representasi *floating-point*. Teknik ini mencakup beberapa varian seperti *dynamic quantization*, *static quantization*, dan *weight-only quantization*, yang masing-masing menawarkan kompromi berbeda antara akurasi dan efisiensi. Karena PTQ tidak memerlukan

akses ke data pelatihan dan dapat diaplikasikan secara langsung pada model terlatih, pendekatan ini menjadi pilihan utama untuk mengoptimalkan model yang kompleks atau berukuran besar, serta secara luas digunakan dalam sistem yang menuntut *inference* cepat pada perangkat dengan sumber daya terbatas.

Berikut merupakan beberapa teknik turunan yang termasuk sebagai bagian dari *post-training quantization*:

1. ***Dynamic Quantization***

   Dynamic quantization merupakan teknik yang menerapkan penurunan presisi numerik terutama pada bobot (*weights*) model, sementara aktivasi dikonversi secara dinamis selama proses *inference*. Teknik ini tidak memerlukan data kalibrasi dan dapat diterapkan langsung pada model terlatih tanpa proses tambahan, sehingga menjadi metode yang paling mudah digunakan dalam konteks *post-training quantization* [76]. Pada *dynamic quantization*, bobot model biasanya diturunkan ke representasi INT8, sedangkan aktivasi tetap dihitung dalam representasi *floating-point* sehingga menjaga stabilitas numerik sambil tetap memperoleh peningkatan efisiensi komputasi. Pendekatan ini memberikan peningkatan kecepatan *inference* pada CPU dan pengurangan ukuran model yang signifikan, meskipun hasilnya tidak setinggi *static quantization* dalam hal kompresi menyeluruh. *Dynamic quantization* banyak digunakan pada model berbasis transformer dan *recurrent neural networks* yang sensitif terhadap penurunan presisi aktivasi, sehingga memberikan keseimbangan antara efisiensi dan akurasi.

2. ***Static Quantization* (*Full Integer Quantization*)**

   Static quantization, atau *full integer quantization*, merupakan teknik yang mengonversi bobot dan aktivasi sekaligus ke representasi *integer* melalui proses kalibrasi menggunakan sampel data. Berbeda dengan *dynamic quantization*, teknik ini memerlukan data representatif untuk menentukan rentang nilai (*range calibration*) sehingga proses *quantization* lebih akurat dan stabil [75]. Karena aktivasi dan bobot keduanya dikonversi

ke INT8, perangkat *inference* dapat melakukan operasi komputasi *integer* penuh, sehingga memberikan peningkatan performa yang lebih besar dibanding *dynamic quantization*, terutama pada CPU dan *edge* device. *Static quantization* juga memungkinkan optimasi seperti *per-tensor* dan *per-channel quantization*, yang menghasilkan distribusi nilai *integer* yang lebih presisi. Meskipun teknik ini memberikan kompresi dan percepatan terbaik dalam *post-training*, akurasi model dapat menurun jika data kalibrasi tidak representatif atau jika model sangat sensitif terhadap perubahan presisi numerik.

3. *Quantization*-Aware *Training*

Quantization-Aware *Training* (QAT) merupakan teknik *advanced quantization* yang mensimulasikan efek *quantization* selama proses pelatihan, sehingga model dapat menyesuaikan bobotnya terhadap *noise* yang diperkenalkan oleh representasi *integer*[74]. Pada QAT, operasi *quantization* seperti *rounding* dan *clipping* disimulasikan dalam *forward pass*, tetapi *parameter* model tetap diperbarui menggunakan gradien *floating-point*, sehingga memungkinkan model mempertahankan akurasi yang jauh lebih tinggi dibandingkan *post-training quantization* konvensional. Teknik ini secara umum menghasilkan performa mendekati model *full-precision*, bahkan ketika bobot dan aktivasi dikompresi ke INT8 atau lebih rendah. QAT cocok untuk model dengan struktur kompleks seperti transformer, *convolutional networks*, atau *vocoder* TTS, tetapi memerlukan proses pelatihan ulang sehingga lebih mahal secara komputasi. Karena implementasinya lebih rumit, QAT biasanya diterapkan pada skenario industri atau model produksi yang sangat sensitif terhadap penurunan akurasi.

4. *Weight-Only Quantization*

Weight-only quantization merupakan teknik yang hanya menurunkan presisi bobot model, sementara aktivasi tetap menggunakan presisi *floating-point*. Teknik ini banyak digunakan pada model bahasa dan model *encoder-decoder* besar karena memberikan efisiensi memori yang substansial tanpa

memengaruhi stabilitas aktivasi selama *inference*. Representasi bobot yang umum digunakan meliputi INT8, INT4, bahkan format *ultra-low precision* seperti GPTQ dan AWQ yang menggunakan skema *quantization* adaptif [77]. *Weight-only quantization* menghasilkan kompresi model yang sangat besar dengan penurunan akurasi minimal, dan sangat efektif ketika dikombinasikan dengan *runtime* yang mendukung kernel *integer* teroptimasi. Teknik ini telah menjadi praktik umum dalam *deployment* LLM di perangkat CPU maupun GPU *low-end* karena tidak memerlukan data kalibrasi dan dapat diaplikasikan secara langsung pada model terlatih.

2.3.5 ONNX *Runtime Optimization*

ONNX *Runtime* merupakan *high-performance inference engine* yang dirancang untuk mengeksekusi model *deep learning* secara efisien di berbagai *platform* komputasi. *Framework* ini mendukung model dengan format ONNX (Open Neural Network Exchange), sebuah standar terbuka yang memungkinkan interoperabilitas lintas *framework* seperti PyTorch, TensorFlow, dan JAX[55]. Keunggulan utama ONNX *Runtime* adalah kemampuannya mengoptimalkan eksekusi model melalui serangkaian teknik percepatan seperti *graph optimization*, *operator fusion*, dan pemilihan *execution provider* yang sesuai dengan perangkat keras yang digunakan. Dengan optimasi ini, model dapat berjalan lebih cepat, konsumsi memori berkurang, dan *latency inference* menjadi lebih rendah, sehingga ONNX *Runtime* banyak digunakan pada aplikasi *real-time* dan perangkat dengan sumber daya terbatas. Selain itu, ONNX *Runtime* mendukung akselerasi CPU maupun GPU serta memiliki ekstensi khusus seperti ONNX *Runtime Mobile*, yang semakin meningkatkan fleksibilitas penggunaan pada skenario *edge computing* dan *embedded systems*.

Secara teknis, ONNX *Runtime* melakukan optimasi model melalui beberapa tahapan yang mencakup *graph simplification*, *constant folding*, *node elimination*, serta *operator fusion*, yaitu penggabungan beberapa operasi menjadi satu kernel komputasi untuk mengurangi *overhead* eksekusi. Optimasi ini dijalankan pada *intermediate representation* dari

model ONNX, sehingga menghasilkan *optimized computation graph* yang lebih ringkas dan efisien. Selain itu, ONNX *Runtime* mendukung *execution provider* seperti CPUExecutionProvider, CUDAExecutionProvider, atau TensorRTExecutionProvider yang memilih *backend* eksekusi paling optimal sesuai perangkat keras. *Runtime* juga menyediakan dukungan untuk *quantization*, termasuk *dynamic quantization* dan *static quantization*, yang semakin menurunkan *latency* dan kebutuhan memori pada saat *inference*. Kombinasi teknik optimasi *graph*, pemilihan kernel yang efisien, dan integrasi *quantization* memungkinkan ONNX *Runtime* memberikan peningkatan performa signifikan pada model *deep learning*, terutama pada aplikasi TTS atau model sejenis dengan struktur komputasi kompleks.

2.3.6 CTranslate2 *Optimization*

CTranslate2 merupakan sebuah *high-performance inference engine* yang dirancang khusus untuk mengeksekusi model *sequence-to-sequence* pada tahap *deployment*, seperti ASR dan *neural machine translation*. Berbeda dengan *deep learning framework* umum seperti PyTorch yang dirancang untuk mendukung proses pelatihan dan inferensi secara fleksibel, CTranslate2 dikembangkan sebagai *standalone inference runtime* berbasis C++ yang berfokus pada efisiensi eksekusi model [78]. Pendekatan ini memungkinkan penghilangan berbagai *overhead* yang umum terdapat pada *training-oriented framework*s, seperti *autograd engine*, *dynamic computation graph*, dan abstraksi *tensor* tingkat tinggi, sehingga proses inferensi dapat dijalankan secara lebih ringan, deterministik, dan efisien. CTranslate2 mendukung berbagai arsitektur dan format model populer, termasuk Whisper dan model OpenNMT, serta menyediakan kompatibilitas dengan *backend* CPU dan GPU, menjadikannya sesuai untuk *deployment* pada lingkungan dengan keterbatasan sumber daya maupun aplikasi yang menuntut *latency* rendah dan *throughput* tinggi.

Keunggulan performa CTranslate2 dicapai melalui kombinasi implementasi *runtime* C++ yang teroptimasi dan pemanfaatan pustaka komputasi tingkat rendah yang disesuaikan dengan arsitektur perangkat

keras target. Pada eksekusi berbasis CPU, CTranslate2 memanfaatkan pustaka numerik seperti Intel MKL untuk mengakselerasi operasi linear melalui rutin BLAS dan vektorisasi SIMD, serta menerapkan *memory alignment* dan *efficient memory layout* untuk meminimalkan *cache miss* dan meningkatkan *data locality*. Selain itu, CTranslate2 mengintegrasikan berbagai teknik optimasi *inference* seperti *weight-only quantization* pasca-pelatihan, operator *fusion*, *decoder state caching*, dan *dynamic batching*, yang secara kolektif berkontribusi pada pengurangan *latency*, penurunan konsumsi memori, dan peningkatan *throughput* tanpa memerlukan proses pelatihan ulang atau fine-tuning model [78].

2.3.7 GGUF *Quantization Format*

GGUF (*General Graph Unified Format*) merupakan format representasi model yang digunakan untuk mendukung *inference Large Language Models* secara efisien, khususnya pada skenario *deployment* berbasis CPU dan sistem dengan keterbatasan sumber daya. Format ini digunakan sebagai representasi model hasil optimasi *post-training*, yang memungkinkan penyimpanan bobot terkuantisasi beserta metadata arsitektur dan informasi tensor dalam satu berkas terstruktur, sehingga memfasilitasi proses *model loading* dan eksekusi model yang lebih deterministik dan efisien [79], [80]. Penggunaan format GGUF telah dilaporkan dalam studi optimasi LLM yang mengevaluasi dampak *quantization* terhadap performa model pada skenario penggunaan nyata, terutama untuk *deployment* model berskala besar pada perangkat konsumen [80].

Secara teknis, GGUF merepresentasikan bobot model yang telah melalui proses *weight-only quantization*, seperti INT8 atau INT4, sementara aktivasi tetap diproses dalam presisi *floating-point* pada saat *inference*. Pendekatan ini memungkinkan pengurangan *memory footprint* model secara signifikan tanpa memerlukan proses *quantization-aware training*. Struktur file GGUF menyimpan *parameter quantization* dan metadata tensor secara eksplisit, sehingga karakteristik *quantization* dan implikasinya terhadap perilaku model dapat dianalisis secara sistematis pada *runtime*

inference [79]. Dengan demikian, GGUF diposisikan sebagai format model yang merepresentasikan hasil *post-training quantization* dan mendukung eksekusi LLM yang efisien pada lingkungan *inference* berbasis CPU [79], [80].

2.4 Rumus Metrik Evaluasi

Dalam evaluasi performa sistem *voice conversational* AI, berbagai metrik digunakan untuk menilai kualitas dan efisiensi komponen utama sistem, yaitu ASR, SLM, dan TTS. Metrik evaluasi ini mencakup pengukuran akurasi transkripsi, kecepatan pemrosesan, kualitas respons bahasa alami, serta kualitas keluaran *audio*. Berikut adalah metrik evaluasi yang digunakan dalam penelitian ini.

1. ***Word Error Rate* (WER)**

WER merupakan metrik utama yang digunakan untuk mengevaluasi akurasi model ASR dalam mentranskripsikan ujaran menjadi teks. WER mengukur jumlah kesalahan prediksi berdasarkan tiga jenis kesalahan: substitusi, penghapusan, dan penyisipan kata [81]. Nilai WER yang lebih rendah menunjukkan performa ASR yang lebih baik.

$$WER = \frac{S + D + I}{N} \times 100$$

Rumus 2. 1 Evaluasi Word *Error* Rate

Legend dari 2.1 Evaluasi *Word Error Rate* adalah sebagai berikut:

a. ***Substitution* (S):** Jumlah kata salah ganti
b. ***Deletions* (D):** Jumlah kata hilang
c. ***Insertions* (I):** Jumlah kata tambahan
d. **N:** Total kata dalam transkripsi referensi

2. ***Character Error Rate* (CER)**

Character Error Rate (CER) digunakan untuk mengevaluasi tingkat kesalahan transkripsi pada level karakter dengan membandingkan hasil transkripsi model terhadap transkrip *ground truth* [81]. Metrik ini menghitung

proporsi kesalahan karakter yang terjadi selama proses pengenalan ujaran. Nilai CER yang lebih rendah menunjukkan kualitas transkripsi yang lebih baik.

$$CER = \frac{N_{sub} + N_{del} + N_{ins}}{N_{char}} \times 100$$

Rumus 2. 2 Evaluasi Character *Error* Rate

Legend dari 2.2 Evaluasi Character *Error* Rate adalah sebagai berikut:

e. $\boldsymbol{N_{sub}}$**:** Jumlah kesalahan *substitution* pada level karakter
f. $\boldsymbol{N_{del}}$**:** Jumlah kesalahan *deletion* pada level karakter
g. $\boldsymbol{N_{ins}}$**:** Jumlah kesalahan *insertion* pada level karakter
h. $\boldsymbol{N_{char}}$**:** Jumlah total karakter pada transkrip *ground truth*

3. *Real-time Factor* (RTF)

RTF digunakan untuk mengukur kecepatan proses *inference*, terutama pada ASR dan TTS. RTF menunjukkan perbandingan antara waktu pemrosesan model terhadap durasi *audio* asli. Model dapat dikatakan berjalan secara *real-time* apabila nilai RTF sama dengan atau lebih kecil dari 1 [82].

$$RTF = \frac{T_{process}}{T_{audio}}$$

Rumus 2. 3 Evaluasi *Real-time* Factor

Legend dari rumus 2.3 Evaluasi *Real-time* Factor adalah sebagai berikut:

a. $\mathbf{T_{process}}$**:** Waktu proses *inference* (detik)
b. $\mathbf{T_{audio}}$**:** Durasi *audio* asli (detik)

4. *Latency* / *Inference Time* (Untuk SLM & TTS)

Latency mengukur waktu yang diperlukan sistem untuk menghasilkan keluaran setelah menerima *input*. Pada SLM, *latency* dihitung sejak teks masukan diterima hingga respons selesai dihasilkan, sementara pada TTS dihitung sejak teks diterima hingga *audio* selesai diproduksi [83]. *Latency*

menjadi indikator penting untuk aplikasi *real-time* seperti *voice conversational* AI.

$$Latency = T_{output} - T_{input}$$

Rumus 2. 4 Evaluasi *Latency*

Legend dari rumus 2.4 Evaluasi *Latency* adalah sebagai berikut:

- **a. T_{input}:** Waktu *input* diterima
- **b. T_{output}:** Waktu *output* selesai diproduksi

5. *Token Generation Rate* (Tokens per Second)

Token Generation Rate digunakan untuk mengevaluasi efisiensi model bahasa dalam menghasilkan teks. Metrik ini mengukur jumlah token bahasa yang dihasilkan model per detik [84]. Nilai yang lebih tinggi menunjukkan performa *inference* yang lebih cepat dan efisien.

$$TokenRate = \frac{N_{tokens}}{T_{generation}}$$

Rumus 2. 5 Evaluasi Token Generation Rate

Legend dari rumus 2.5 Evaluasi *Token Generation Rate* adalah sebagai berikut:

- **a. N_{tokens}:** Jumlah token yang dihasilkan
- **b. $T_{generation}$:** Waktu generasi respons (detik)

6. Model Size

Model size digunakan untuk mengevaluasi kebutuhan penyimpanan model pada tahap *deployment*. Metrik ini diukur sebagai ukuran file model setelah konversi dan/atau *quantization*. Nilai model size yang lebih kecil menunjukkan efisiensi memori yang lebih baik [58].

$$ModelSize = Size_{file}$$

Rumus 2. 6 Evaluasi Model Size

Legend dari rumus 2.6 Evaluasi *Model Size* adalah sebagai berikut:

a. $\boldsymbol{Size_{file}}$**:** Ukuran file model dalam satuan megabytes (MB) atau gigabytes (GB)

2.5 *Tools/software* yang digunakan

2.5.1 Python

Python merupakan bahasa pemrograman berorientasi objek yang banyak digunakan dalam pengembangan aplikasi modern karena sintaksnya yang sederhana, fleksibel, serta mudah dipahami oleh pemula maupun pengembang berpengalaman. Python memiliki ekosistem *library* yang sangat luas dan mendukung berbagai kebutuhan komputasi, termasuk *data processing*, *scientific computing*, *machine learning*, hingga pemrosesan suara, sehingga menjadikannya salah satu bahasa pemrograman paling populer dalam penelitian dan industry [85]. Kemampuan Python untuk berjalan lintas *platform*, didukung oleh komunitas global yang besar, serta keterhubungannya dengan berbagai *framework* dan *runtime engine* menjadikannya pilihan utama dalam pengembangan sistem berbasis AI dan pemodelan algoritmik. Selain itu, Python mengintegrasikan manajemen memori otomatis dan struktur data tingkat tinggi, memungkinkan pengembang membangun prototipe maupun aplikasi produksi secara cepat, efisien, dan hemat biaya, sesuai dengan tuntutan implementasi teknologi berbasis komputasi modern.

2.5.2 Visual Studio Code

Visual Studio Code (VS Code) merupakan *source-code editor* lintas *platform* yang dikembangkan oleh Microsoft dan telah menjadi salah satu lingkungan pengembangan paling populer karena ringan, fleksibel, serta memiliki dukungan *extension* yang luas. VS Code menyediakan fitur-fitur modern seperti *syntax highlighting*, *intellisense*, *debugger* terintegrasi, dan *version control* berbasis Git, sehingga memudahkan pengembang dalam membangun, menguji, dan memelihara aplikasi secara efisien. Editor ini juga mendukung berbagai bahasa pemrograman termasuk Python, JavaScript, dan

C++, serta memungkinkan integrasi langsung dengan *runtime environment* dan *tooling* lain melalui *marketplace open-source*. Selain itu, penelitian menunjukkan bahwa VS Code memberikan produktivitas tinggi bagi pengembang karena arsitekturnya yang modular, kustomisasi antarmuka yang fleksibel, dan performa yang optimal untuk proyek berskala kecil hingga menengah [86]. Dengan kombinasi fitur intuitif dan ekosistem pendukung yang kuat, VS Code menjadi pilihan utama dalam pengembangan perangkat lunak modern, termasuk dalam konteks penelitian dan implementasi sistem berbasis AI.

2.5.3 *Hardware* yang Digunakan

Penelitian ini dilakukan dengan ditenagai oleh beberapa komponen *hardware*, yaitu sebagai berikut:

1. CPU : Intel® Core™ i5-1235U
2. *Memory* : 16 GB RAM
3. OS : Windows 11
4. *Storage* : 512 GB SSD