Jutisi: Jurnal Ilmiah Teknik Informatika dan Sistem Informasi
https://ojs.stmik-banjarbaru.ac.id/index.php/jutisi/index
Jl. Ahmad Yani, K.M. 33,5 - Kampus STMIK Banjarbaru
Loktabat – Banjarbaru (Tlp. 0511 4782881), e-mail: puslit.stmikbjb@gmail.com
e-ISSN: 2685-0893

Optimalisasi *TinyBERT-Vosk* Untuk Kontrol Suara *Real-Time* Robot *Mecanum Raspberry Pi*

DOI: http://dx.doi.org/10.35889/jutisi.v14i3.3327

Gogor Christmass Setyawan[1*], Surjawirawan Dwiputranto[2], Kristian Juri Damai Lase[3]
[1,3]Informatika, Universitas Kristen Immanuel, Yogyakarta, Indonesia
[2]Fisika, Universitas Kristen Immanuel, Yogyakarta, Indonesia
**e-mail Corresponding Author: masgogor@ukrimuniversity.ac.id*

Abstract

This study targets low-latency on-device voice control on Raspberry Pi for a Mecanum 4WD robot. Key challenges include latency jitter, ASR accuracy under mild–moderate noise, and NLU compute load. We propose a fully local pipeline: Audio → WebRTC VAD → grammar-constrained (WFST) Vosk ASR → TinyBERT NLU distilled, INT8-quantized, and lightly pruned; complemented by a confidence gate, temperature scaling, a 1–360 numeral parser, and runtime tuning (threading, non-blocking queues, CPU affinity). Experiments on RPi 5 reduce end-to-end latency, improve ASR RTF and WER/CER, and lower CPU/memory and NLU latency without material loss in intent/slot accuracy, demonstrating resource-efficient real-time voice control with privacy preserved at the edge.

Keywords: *Vosk/ASR; TinyBERT; INT8 quantization; Voice control; Raspberry Pi.*

Abstrak

Penelitian ini menargetkan kendali suara *on-device* berlatensi rendah pada Raspberry Pi untuk robot Mecanum 4WD. Tantangan utama meliputi jitter latensi, akurasi ASR pada kebisingan ringan–sedang, dan beban komputasi NLU. Diusulkan *pipeline* lokal: Audio → WebRTC VAD → Vosk ASR ber-grammar (WFST) → TinyBERT NLU yang didistilasi, dikuantisasi INT8, dan di*pruning*; dilengkapi *confidence gate*, *temperature scaling*, parser angka 1–360, serta *runtime tuning* (*threading, antrian non-blocking, CPU affinity*). Evaluasi pada RPi 5 menunjukkan penurunan latensi ujung-ke-ujung, perbaikan RTF dan WER/CER, serta pengurangan CPU/memori dan latensi NLU, tanpa degradasi berarti pada akurasi intent/slot. Hasil menegaskan kelayakan kontrol suara *real-time* yang efisien dan menjaga privasi di perangkat edge.

Kata kunci: *Vosk/ASR; TinyBERT; Kuantisasi INT8; Kontrol suara; Raspberry Pi.*

1. Pendahuluan

Penggunaan suara sebagai antarmuka kendali robot semakin mendapat perhatian karena mampu menghadirkan interaksi yang alami, praktis, dan aman bagi pengguna. Berbeda dengan kontrol berbasis tombol atau aplikasi, perintah suara memungkinkan operator memberikan instruksi secara langsung tanpa kontak fisik, sesuatu yang sangat berguna ketika robot beroperasi di lingkungan dinamis atau berisiko. Untuk dapat diterapkan secara luas, sistem kendali suara harus berjalan secara *on-device*, sehingga tidak tergantung pada jaringan internet, memiliki latensi rendah, serta efisien dari sisi komputasi dan energi. Hal ini menjadi tantangan utama ketika sistem ditanamkan pada perangkat tepi berdaya terbatas seperti Raspberry Pi.

Di sisi pengenalan ujaran, kemajuan model berskala besar seperti *wav2vec 2.0* yang dilatih secara swa-supervisi maupun *Whisper* dengan pendekatan pengawasan lemah skala masif [1], [2] memang menunjukkan ketahanan yang tinggi. Akan tetapi, kebutuhan komputasi yang besar sering kali tidak sesuai dengan kemampuan perangkat tepi yang tidak dilengkapi akselerator khusus. Masalah serupa juga muncul pada tahap *Voice Activity Detection (VAD)*, di mana akurasi endpointing di kondisi akustik bising tetap menjadi faktor penting untuk menjaga

stabilitas sistem [3]. Sementara itu, pada lapisan pemahaman bahasa alami, penelitian tentang kompresi Transformer seperti *TinyBERT*, *MobileBERT*, serta teknik distilasi, kuantisasi, dan pruning [4]–[7] memberi bukti bahwa model tetap bisa akurat meskipun jejak komputasinya diperkecil. Namun, implementasi utuh pada Raspberry Pi, terlebih untuk bahasa Indonesia, masih jarang ditemukan.

Upaya-upaya sebelumnya memperlihatkan arah solusi yang menjanjikan. Distilasi BERT, misalnya, mampu menurunkan ukuran dan waktu inferensi tanpa kehilangan kinerja signifikan pada tugas NLU [13]. Optimisasi *on-device NLP* dengan strategi *early-exit* atau *dynamic batching* [14] juga berhasil menekan konsumsi energi, walau umumnya hanya efektif pada perangkat dengan NPU/DSP bawaan. Studi lain menunjukkan bahwa *post-training quantization* presisi rendah (INT8) tetap stabil pada Transformer besar [15]. Dalam ranah ASR, arsitektur streaming seperti RNN-T [16] serta penyederhanaan *decoder graph* [17] mampu menurunkan latensi dan kebutuhan memori. Sayangnya, sebagian besar penelitian berfokus pada domain kosakata luas, perangkat dengan akselerator, dan bahasa global berdaya data besar. Akibatnya, masih ada tiga celah yang menonjol: (i) belum ada laporan komprehensif tentang penggunaan *grammar-constrained decoding* (WFST) pada ASR domain perintah tertutup di Raspberry Pi, (ii) belum diterapkannya TinyBERT hasil distilasi–kuantisasi INT8 untuk bahasa Indonesia yang terhubung erat dengan ASR dan *number parser*, serta (iii) belum ada integrasi penuh lintas-lapis (VAD → ASR/WFST → TinyBERT-INT8 → confidence gate/kalibrasi) yang dilaporkan secara sistematis di Raspberry Pi 5.

Kebaruan penelitian ini terletak pada perbedaan konsep dengan karya-karya terdahulu. Jika penelitian sebelumnya cenderung menyoroti satu komponen tertentu—misalnya ASR end-to-end berskala besar [1], [2] atau NLU Transformer terkompresi [4]–[7], [13]–[15]—maka penelitian ini mengajukan integrasi penuh yang memadukan berbagai komponen dalam satu pipeline. Arsitektur yang diusulkan melibatkan VAD ringan berbasis WebRTC, ASR domain perintah tertutup dengan grammar/WFST, NLU berbasis TinyBERT yang dipercepat melalui distilasi–kuantisasi–pruning, serta mekanisme *confidence gate* dengan kalibrasi suhu dan penolakan *open-set* [11], [12]. Pendekatan holistik ini diharapkan mampu menunjukkan keseimbangan optimal antara kecepatan, akurasi, dan efisiensi pada Raspberry Pi 5 tanpa akselerator, khususnya untuk kendali robot mekanum berbahasa Indonesia.

Berdasarka fakta-fakta tersebut diatas maka, penelitian ini diarahkan untuk: (1) merancang dan mengintegrasikan pipeline kendali suara yang berjalan sepenuhnya lokal di Raspberry Pi; (2) menekan latensi ujung-ke-ujung dan *Real-Time Factor (RTF)* tanpa mengorbankan akurasi; (3) menerapkan distilasi–kuantisasi–pruning agar NLU beroperasi dengan latensi milidetik; dan (4) meningkatkan reliabilitas keputusan melalui kalibrasi probabilitas dan mekanisme penolakan *open-set*. Manfaat yang diharapkan adalah tersedianya rancangan sistem kendali suara real-time yang efisien, replikabel, serta ramah privasi, yang dapat menjadi pijakan praktis maupun akademis dalam pengembangan robot berbasis Raspberry Pi di Indonesia.

2. Metodologi

Penelitian ini menerapkan pendekatan rekayasa *sistem end to end on device* pada Raspberry Pi untuk kendali suara robot Mecanum 4WD. Rantai proses dibangun berlapis—VAD WebRTC untuk endpointing yang stabil, ASR Vosk dengan grammar-constrained WFST untuk memperkecil ruang pencarian, NLU TinyBERT yang didistilasi dikuantisasi INT8 dan dipruning ringan, parser angka 1–360, pemetaan *intent slot* ke aksi motor, serta confidence gate berbasis probabilitas terkalibrasi guna menekan latensi ujung-ke-ujung (E2E), menjaga akurasi pada kebisingan ringan–sedang, dan menurunkan jejak CPU/memori pada perangkat tanpa akselerator khusus [18] – [24].

2.1 Analisa Kebutuhan

Kebutuhan fungsional mencakup pengenalan perintah pendek (intent A/B dan SEQUENCE) dengan akurasi NLU tinggi (Macro-F1/Slot-F1), serta pencegahan eksekusi keliru melalui *confidence gate*. Kebutuhan non-fungsional menargetkan p95 E2E ≤ ~450 ms, RTF ASR < 1 (diupayakan < 0,5), WER/CER rendah pada SNR 5–20 dB, dan CPU/Mem moderat agar operasi stabil berkelanjutan. Kebutuhan data meliputi rekaman ujaran multi-pembicara, anotasi *intent slot* dan angka 1–360, serta skenario kebisingan sintetis/nyata. Kebutuhan

evaluasi mencakup metrik E2E, ASR partial/final latency, RTF, WER/CER, Macro-F1/Slot-F1/EMR, ECE/ROC-AUC, dan CPU/Mem [18], [19], [20], [21], [22], [25].

2.2 Desain Sistem

2.2.1 Desain Sistem (Robot 4WD dan Roda Mecanum)

Sasis 4WD disusun simetris dengan empat roda Mecanum ber-roller 45° untuk gerak omnidireksional; orientasi roller mengikuti konvensi FL↗, FR↘, RL↙, RR↖. Parameter geometri L (jarak sumbu depan–belakang) dan W (kiri–kanan) digunakan untuk memetakan vektor kecepatan (vx, vy, ω) ke kecepatan roda (v_FL, v_FR, v_RL, v_RR). Pusat massa didekatkan ke pusat sasis untuk mengurangi slip dan menjaga respons yang konsisten [26].

2.2.2 Desain Perangkat Keras (Diagram Rangkaian)

Platform memakai Raspberry Pi 5 Model B (pendingin aktif) yang dicatu UPS Shield X1202 (5,1 V/≤5 A) dengan 4×18650, serta driver DC 4-kanal L9110S untuk empat motor. jalur motor dipisahkan dari jalur logika dan *common ground* dijaga. Frekuensi PWM 18–25 kHz digunakan untuk menghindari dengung. Penataan kabel motor (twisted pair), *decoupling* keramik/elektrolit di jalur motor, serta sekering *inline* diterapkan demi kestabilan dan mitigasi EMI.

2.2.3 Desain Perangkat Lunak

Alur perangkat lunak: Audio(16 kHz) → VAD(WebRTC) → ASR(Vosk/WFST) → NLU(TinyBERT) → Parser Angka → Mapper → Motor, berjalan multibenang dengan antrian non-blocking dan *CPU affinity* untuk menekan jitter. TinyBERT dilatih dua tahap: *fine-tuning teacher* (BERT-base) pada korpus perintah lalu distillation ke *student* ringkas [19]; berikutnya post-training quantization INT8 (kalibrasi *representative set*) dan pruning tak-berstruktur moderat diterapkan untuk memangkas komputasi dengan degradasi minimal [20], [21], [23], [24]. Dataset dibagi speaker-independent (train/val/test) dan di-augment untuk SNR 5–20 dB [18], [22].

2.2.4 Desain Analisa Data

Analisa kuantitatif berfokus pada kecepatan, ketepatan, dan efisiensi. Kecepatan ditangkap oleh E2E (ujaran-ke-roda) dan RTF (rasio waktu komputasi terhadap durasi audio); ketepatan dipantau via WER/CER (ASR) dan Macro-F1/Slot-F1/EMR (NLU); reliabilitas keputusan dijaga melalui kalibrasi (ECE/ROC-AUC) dan confidence gate. Pada sisi audio, VAD_lag_start/end dievaluasi agar endpointing tidak menambah *tail latency* [18], [25], [27].

Metode pengukuran yang digunakan:

1. Latensi Ujung-ke-Ujung (E2E): L_E2E = t_ctrl_act − t_voice_start
2. Real-Time Factor (RTF, ASR): RTF = waktu_komputasi_ASR / durasi_audio
3. Word Error Rate (WER): WER = (S + D + I) / N
4. Character Error Rate (CER): CER = (S_c + D_c + I_c) / N_c
5. Akurasi Intent (Macro-F1): Macro_F1 = (1/K) * Σ_{k=1..K} F1_k
6. VAD Lag Awal: VAD_lag_start = t_vad_on − t_voice_start
7. VAD Lag Akhir: VAD_lag_end = t_vad_off − t_voice_end
8. Probabilitas Terkalibrasi: p_hat = softmax(z_S / T)
9. Aturan Keputusan (*Confidence Gate*): eksekusi jika max(p_hat) ≥ τ

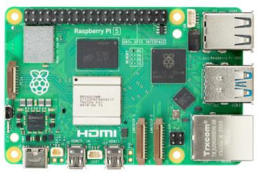
Kalibrasi memakai temperature scaling/Dirichlet calibration untuk menurunkan ECE sehingga ambang τ lebih reliabel lintas scenario [25], [27]. Kuantisasi/pruning pada NLU mengikuti praktik efisiensi *edge* yang telah terbukti menjaga akurasi sambil menurunkan latensi/jejak memori [20], [21], [27], [28]. Pada ASR, WFST grammar dan penyetelan beam/max active dipakai untuk menurunkan RTF dan WER/CER di domain perintah tertutup [22], [23].

2.3 Komponen Perangkat Keras

Pemilihan komponen dilakukan berbasis kompatibilitas listrik/logika, stabilitas daya, antarmuka, dan kemudahan integrasi. Raspberry Pi 5 dijadikan pusat komputasi untuk menjalankan VAD, ASR, dan NLU secara paralel; pendingin aktif disiapkan guna mencegah thermal throttling. Stabilitas suplai disediakan UPS Shield X1202 (5,1 V/≤5 A) dengan 4×18650

untuk operasi kontinu. Penggerak empat motor menggunakan L9110S 4-kanal yang kompatibel 3 s/d 6V dan mendukung PWM 18–25 kHz. Jalur motor dipisah dari jalur logika dengan common ground. Sasis 4WD Mecanum memberi manuver omnidireksional; pusat massa diposisikan dekat pusat geometris. Audio diambil mikrofon USB 16 kHz mono; penataan kabel meminimalkan EMI. Tabel 1 merangkum spesifikasi dan peran komponen utama.

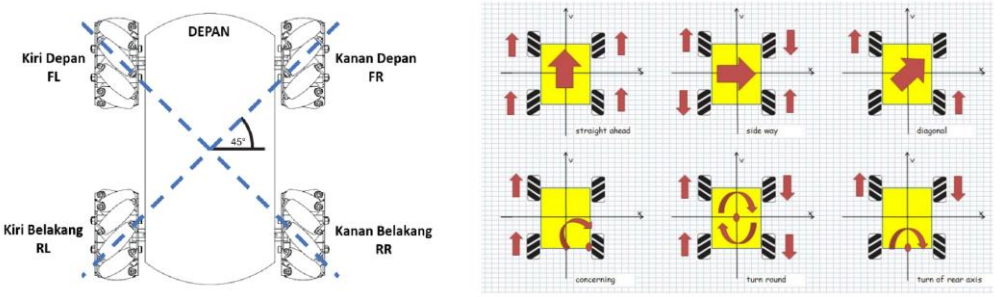
Tabel 1. Daftar komponen perangkat keras

Komponen	Spesifikasi singkat	Peran	**Ilustrasi**
Raspberry Pi 5 Model B	CPU Cortex-A76, 64-bit, pendingin aktif	Komputasi on-device (VAD, ASR, NLU, kontrol)	
UPS Shield X1202 + 4×18650	5,1 V/≤5 A, proteksi, *hot-swap*	Catu daya stabil, *soak test*	
L9110S (4-Ch)	Logika 3,3 V; 4 kanal/chip	Penggerak 4 motor DC (PWM)	
4× Motor + Roda Mecanum	Roller 45°, *hub* seragam	Gerak omni (vx, vy, ω)	
Mikrofon USB	16 kHz mono	Akuisisi audio/VAD	

2.4 Implementasi

2.4.1 Implementasi Perangkat Robot (Platform 4WD Mecanum)

Sasis 4WD dipasang simetris (persegi panjang) dengan empat roda mecanum sudut roller 45° dan orientasi industri: FL ↗, FR ↘, RL ↙, RR ↖. Titik berat (baterai + UPS) ditempatkan sedekat mungkin dengan pusat geometris untuk menjaga distribusi beban dan mengurangi slip. Dudukan motor disetel agar poros sejajar (toe-in/out ≈ 0°); shim tipis dipakai bila perlu. Kabel motor dibuat **twisted pair** dan diberi strain-relief; jalur kabel audio/mikrofon dijauhkan dari kabel motor untuk menekan EMI. Clearance sasis dipastikan tidak membuat roller menyentuh rangka saat rotasi penuh. Verifikasi kinematika dilakukan dengan uji gerak dasar (maju, geser kiri/kanan, rotasi CCW/CW, dan diagonal), sambil memantau kesesuaian arah perintah dengan arah resultan kecepatan roda. Konfigurasi tataletak roda mekanum dan arah putaran roda untuk menghasilkan Gerakan robot dijelaskan pada gambar 1.



A. Konfigurasi roda mekanum pada chasis 4WD sistem robot

B. Diagram arah putaran roda mekanum dan representasi arah gerakan sistem robot

Gambar 1. Implementasi roda mekanum pada system robot

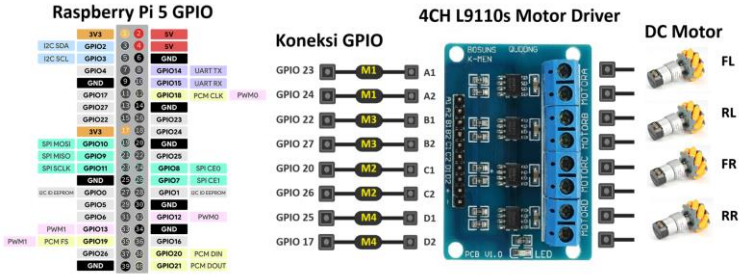
2.4.2 Implementasi Perangkat Keras (Daya, I/O, Audio, Penggerak)

Catu daya & termal. Raspberry Pi 5 dipasang pada UPS X1202 (5,1 V/≤5 A) dengan 4 sel 18650 sejenis. Jalur motor dipisah dari jalur 5 V Pi; seluruh GND disatukan (common ground). Dipasang kapasitor decoupling 0,1 µF dekat IC dan 470–1000 µF di rail motor setiap papan driver. Pendinginan aktif (heatsink + kipas) diterapkan untuk mencegah throttling.

Antarmuka penggerak. Driver DC L9110S 4-kanal dipakai untuk empat motor. PWM direkomendasikan 18–25 kHz agar di luar pita audio. Pemetaan GPIO (Raspberry Pi 5) koneksi yang dipakai adalah:

- FL: IN1=GPIO23, IN2=GPIO24
- FR: IN1=GPIO22, IN2=GPIO20
- RL: IN1=GPIO20, IN2=GPIO26
- RR: IN1=GPIO25, IN2=GPIO17

Konfigurasi pengkabelan antara Soket GPIO Raspbery Pi 5 dan motor DC *Driver* 4CH-L9100s berikut pengkabelan motor-motor di jelaskan pada gambar 2.



Gambar 2. Koneksi GPIO dan Driver 4CH-L9100s serta DC motor

Audio I/O. Mikrofon USB (mono, 16 kHz, 16-bit PCM) dipilih demi stabilitas driver. Tingkat pengambilan disetel tetap, AGC sistem dimatikan. Latensi buffer ALSA/PortAudio diturunkan bertahap hingga tidak terjadi *xrun*; jika *xrun* terdeteksi, buffer dinaikkan satu langkah.

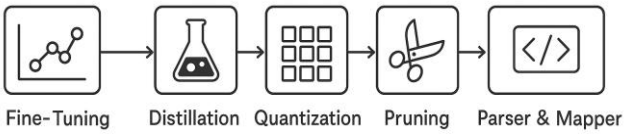
2.4.3 Implementasi Perangkat Lunak

Lingkungan dan dependensi meliputi: Raspberry Pi OS 64-bit (*headless*), Python ≥3.10. Paket utama yang digunakan: *pyaudio* (I/O), *webrtcvad* (VAD), *vosk* (ASR), *onnxruntime* atau *tflite-runtime* (TinyBERT INT8), numpy/pandas (analisis), serta skrip *utilitas logging*.

VAD (*WebRTC*). Frame 20 ms, mode 2–3 (*trade off miss/false*), *pre-roll* ≈80 ms dan *hangover* ≈200–300 ms. *Event vad_on/vad_off* digunakan sebagai penanda segmen untuk ASR dan sebagai jangkar perhitungan *lag*.

ASR (Vosk + WFST *grammar*). Leksikon perintah + angka 1–360 disiapkan, LM kecil (3-gram) dibuat dan dikompilasi menjadi HCLG. Parameter decoding tipikal: *beam* 8–12, *lattice-beam* 6–8, *max-active* 1500–2000, *partial result* diaktifkan. Grafik dan parameter disimpan sebagai profil sehingga dapat diganti tanpa *rebuild*.

Proses optimasi NLU (TinyBERT) dalam penelitian ini dilakukan secara bertahap agar model menjadi lebih ringan, cepat, dan tetap akurat. Tahap pertama adalah melakukan fine-tuning pada model teacher (BERT-base) menggunakan korpus *intent slot*. Hasilnya kemudian didistilasi ke student TinyBERT (4-layer, hidden 312) melalui dua tahap, yaitu distillation layer-wise dan logits. Selanjutnya dilakukan *post-training quantization* ke INT8 menggunakan ONNX/TFLite dengan representative set minimal 1.000 kalimat untuk memastikan kestabilan. Model juga mengalami pruning tak-berstruktur sekitar 30% yang diikuti *fine-tuning* singkat guna mempertahankan akurasi. Setelah itu diterapkan *temperature scaling* pada logit student agar kalibrasi probabilitas lebih reliabel, lalu ditambahkan *confidence gate* untuk mengendalikan eksekusi perintah. Terakhir, sistem dilengkapi parser angka bahasa Indonesia (1–360, baik bentuk baku maupun eliptik) yang memetakan frasa ke integer dan mengemasnya ke dalam JSON {"intent": "...","slots": {...}} sehingga dapat langsung dipetakan ke kontrol motor robot. Proses optimasi NLU di ilustrasikan pada gambar 3.



Gambar 3. Proses implementasi NLU TinnyBERT

Arsitektur runtime: Tiga *worker* paralel: (i) I/O+VAD, (ii) ASR, (iii) NLU+Kontrol. Antar-worker berkomunikasi melalui *queue non-blocking*. CPU *affinity* diatur agar ASR dan NLU tidak bersaing dengan I/O. Kebijakan *STOP preemption* diberlakukan pada jalur kontrol sehingga perintah berhenti dapat memintas antrian biasa. Logger menulis CSV/JSON berisi stempel waktu t_voice_start, t_vad_on/off, t_asr_partial/final, t_nlu_start/end, t_ctrl_act, serta metrik CPU/Mem periodik. Unit sistem disiapkan untuk *auto-start* dan *watchdog* (*restart on failure*).

Validasi & uji jalan: Uji fungsional dilakukan per tahap (**audio→VAD, VAD→ASR, ASR→NLU, NLU→motor**) kemudian uji integrasi *mouth-to-wheel*. Pengujian regressi otomatis memeriksa WER/CER, Macro-F1/Slot-F1/EMR, ECE, RTF, latensi *parsial/final*, dan E2E pada skenario hening hingga SNR rendah. Hasil tersimpan ke folder eksperimen untuk analisis dan replikasi.

3. Hasil dan Pembahasan

3.1 Hasil

3.1.1 Rancangan Uji

Pengujian dilakukan on-device pada Raspberry Pi 5 (pendingin aktif) dengan empat skenario akustik representatif: hening, kipas (~20 dB SNR), motor (~10 dB), dan musik (~5 dB). Pada tiap skenario dieksekusi 50 perintah × 2 konfigurasi (Baseline vs Optimized), diulang 3× (total 1 200 eksekusi). Konfigurasi Optimized meliputi: WFST grammar (Vosk/Kaldi), penyetelan beam/lattice-beam/max-active, TinyBERT distilled + INT8 + pruning ringan, kalibrasi suhu + confidence gate, serta threading + non-blocking queue + CPU affinity. Logger mencatat timestamp VAD/ASR/NLU/kontrol, serta CPU%/Mem.

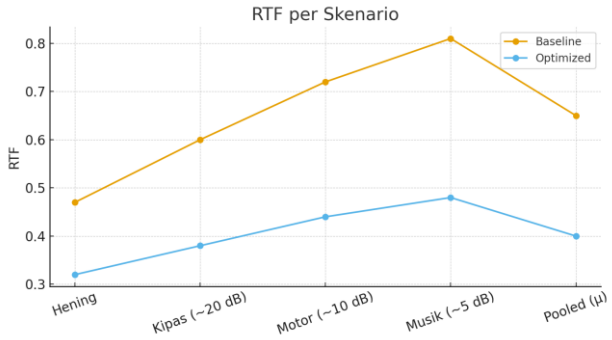
3.1.2 Hasil Uji per Skenario

Hasil penelitian menunjukan pada semua skenario, Konfigurasi *optimized* konsisten lebih baik dari Baseline. RTF turun dan tetap < 0,5 bahkan pada kebisingan musik (~5 dB), menandakan decoding lebih cepat dari waktu nyata. Latensi ASR p50 (parsial dan final) berkurang puluhan milidetik, mendorong penurunan E2E. Secara terpooled, E2E p50 turun ~18% dan E2E p95 ~21%. Peningkatan terbesar muncul pada skenario musik. Hasil ini menegaskan sinergi grammar-constrained decoding, kompresi TinyBERT, dan tuning runtime. Hasil ditunjukan pada table 2.

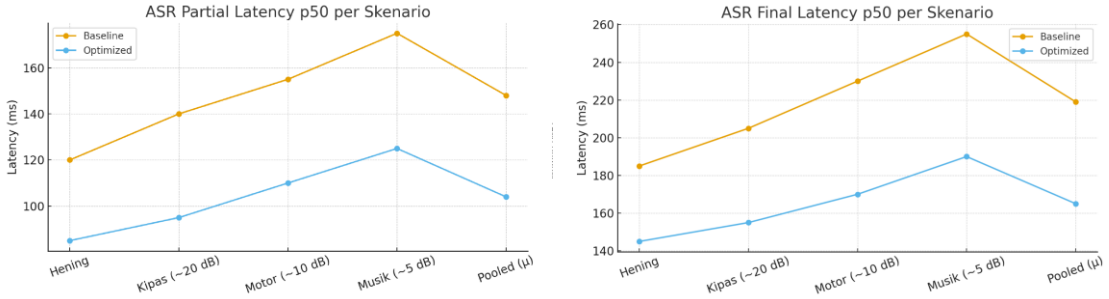
Tabel 2. Latensi & Kecepatan (ASR/NLU/E2E)

Skenario	*Konfig*	*RTF (↓)*	*ASR partial p50 (ms)*	*ASR final p50 (ms)*	*E2E p50 (ms, ↓)*	*E2E p95 (ms, ↓)*
Hening	*Baseline*	*0.47*	*120*	*185*	*430*	*560*
	Optimized	*0.32*	*85*	*145*	*345*	*445*
Kipas (~20 dB)	*Baseline*	*0.60*	*140*	*205*	*465*	*590*
	Optimized	*0.38*	*95*	*155*	*370*	*460*
Motor (~10 dB)	*Baseline*	*0.72*	*155*	*230*	*510*	*630*
	Optimized	*0.44*	*110*	*170*	*395*	*475*
Musik (~5 dB)	*Baseline*	*0.81*	*175*	*255*	*545*	*680*
	Optimized	*0.48*	*125*	*190*	*425*	*495*
Pooled (μ)	*Baseline*	*0.65*	*148*	*219*	*485*	*601*
	Optimized	*0.40*	*104*	*165*	*399*	*473*

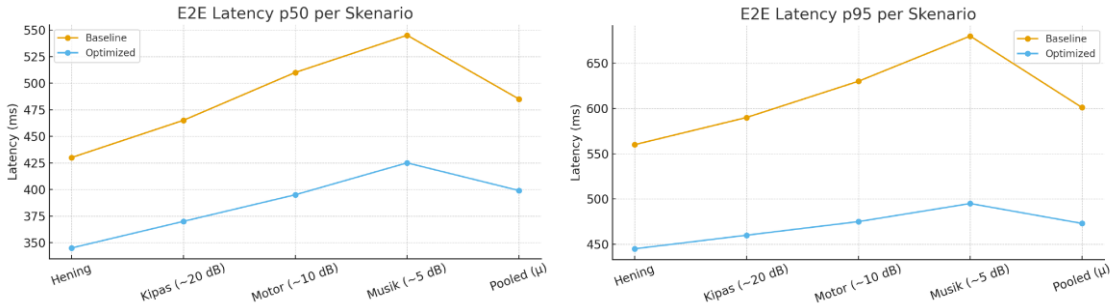
Penurunan E2E p95 paling besar muncul pada skenario bising (musik), sedangkan p50 turun konsisten di semua skenario. Uji *Wilcoxon* berpasangan menunjukkan perbedaan signifikan pada RTF dan E2E ($p < 0,01$). Visualisasi Uji RTF disajikan pada gambar 4, diikuti visualisasi uji latensi ASR p50 (parsial dan final) disajikan pada gambar 5 dan visualisasi uji latensi *end to end* E2E (p50 dan p95) disajikan pada gambar 6.



Gambar 4. Pengujian RTF per skenario



Gambar 5. Pengujian latensi ASR p50 (parsial dan final)



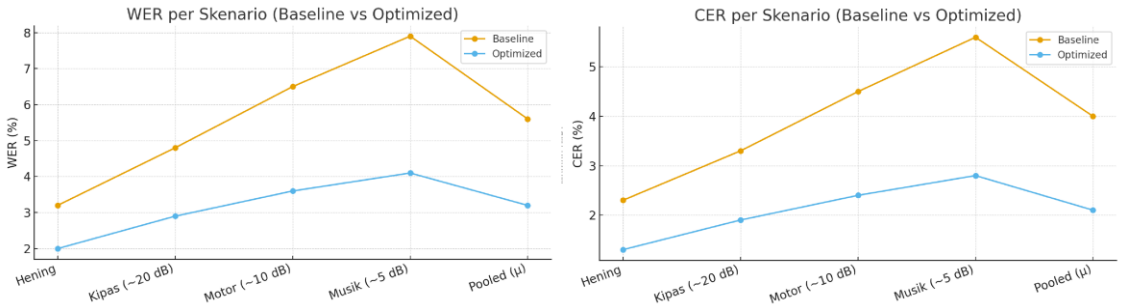
Ganbar 6. Pengujian latensi *end to end* E2E (p50 dan p95)

Pengujian akurasi pengenalan (ASR) menunjukan konfigurasi o*ptimized* menurunkan kesalahan di semua skenario: WER ter*pooled* 5,6→3,2% (≈−43%) dan CER 4,0→2,1% (≈−48%), dengan gain terbesar pada SNR terendah (musik: WER 7,9→4,1%; CER 5,6→2,8%). Ini menegaskan efektivitas WFST grammar dan tuning decoder pada kondisi bising. Hasil optimasi disajikan pada table 3.

Tabel 3. Akurasi Pengenalan (ASR)

Skenario	***Konfig***	***WER % (↓)***	***CER % (↓)***
Hening	*Baseline*	*3.2*	*2.3*
	Optimized	*2.0*	*1.3*
Kipas (~20 dB)	*Baseline*	*4.8*	*3.3*
	Optimized	*2.9*	*1.9*
Motor (~10 dB)	*Baseline*	*6.5*	*4.5*
	Optimized	*3.6*	*2.4*
Musik (~5 dB)	*Baseline*	*7.9*	*5.6*
	Optimized	*4.1*	*2.8*
Pooled (μ)	*Baseline*	*5.6*	*4.0*
	Optimized	*3.2*	*2.1*

Grammar-constrained decoding + penyetelan decoder menurunkan WER/CER secara konsisten, terutama saat SNR menurun. Visualisasi uji WER dan uji CER disajikan pada gambar 7.



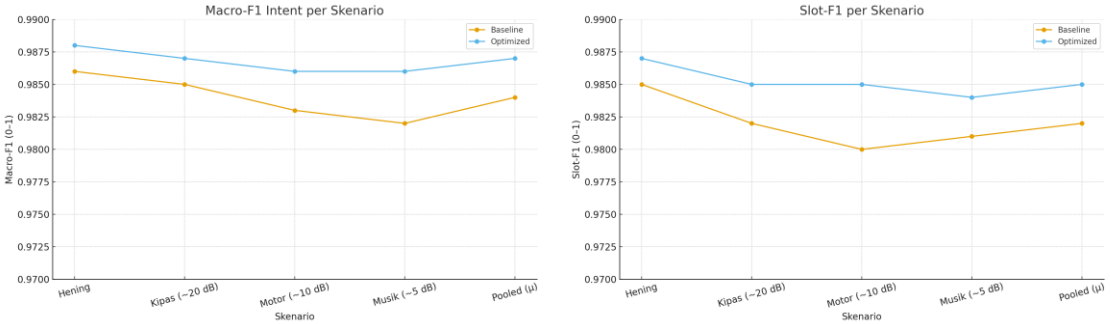
Gambar 7. Pengujian akurasi pengenalan (ASR)

Hasil pengujian NLU dan Kalibrasi menunjukan konfigurasi *optimized* memangkas latensi NLU hampir 50% (≈7,0→≈3,6 ms) sambil mempertahankan—bahkan sedikit meningkatkan—akurasi (Macro-F1 ≈0,987; Slot-F1 ≈0,985; EMR naik tipis). Kalibrasi membaik nyata (ECE 0,042→0,016) dan deteksi open-set meningkat (ROC-AUC 0,958→0,984). Tren konsisten di semua skenario, dengan keuntungan relatif terbesar pada kondisi paling bising. Pengujian NLU dan Kalibrasi disajikan pada table 4.

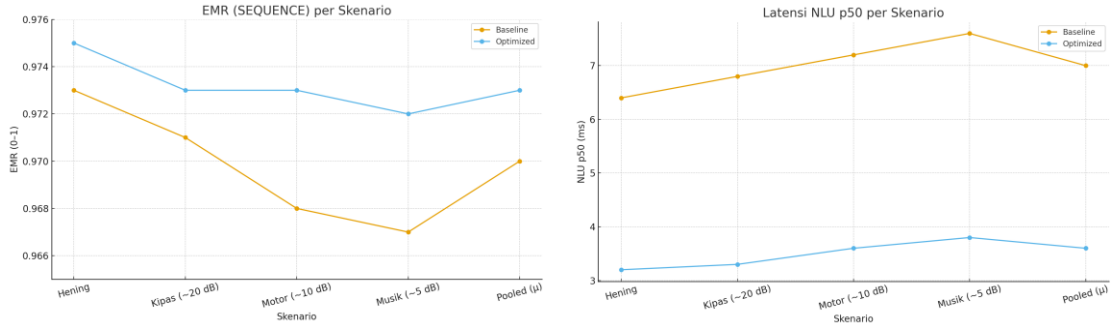
Tabel 4. NLU dam Kalibrasi

Skenario	***Konfig***	***Macro-F1 Intent (↑)***	***Slot-F1 (↑)***	***EMR SEQ (↑)***	***NLU p50 (ms, ↓)***	***ECE (↓)***	***ROC-AUC open-set (↑)***
Hening	*Baseline*	*0.986*	*0.985*	*0.973*	*6.4*	*0.034*	*0.962*
	Optimized	*0.988*	*0.987*	*0.975*	*3.2*	*0.014*	*0.984*
Kipas (~20 dB)	*Baseline*	*0.985*	*0.982*	*0.971*	*6.8*	*0.039*	*0.958*
	Optimized	*0.987*	*0.985*	*0.973*	*3.3*	*0.016*	*0.983*
Motor (~10 dB)	*Baseline*	*0.983*	*0.980*	*0.968*	*7.2*	*0.044*	*0.955*
	Optimized	*0.986*	*0.985*	*0.973*	*3.6*	*0.017*	*0.984*
Musik (~5 dB)	*Baseline*	*0.982*	*0.981*	*0.967*	*7.6*	*0.050*	*0.952*
	Optimized	*0.986*	*0.984*	*0.972*	*3.8*	*0.018*	*0.982*
Pooled (μ)	*Baseline*	*0.984*	*0.982*	*0.970*	*7.0*	*0.042*	*0.958*
	Optimized	*0.987*	*0.985*	*0.973*	*3.6*	*0.016*	*0.984*

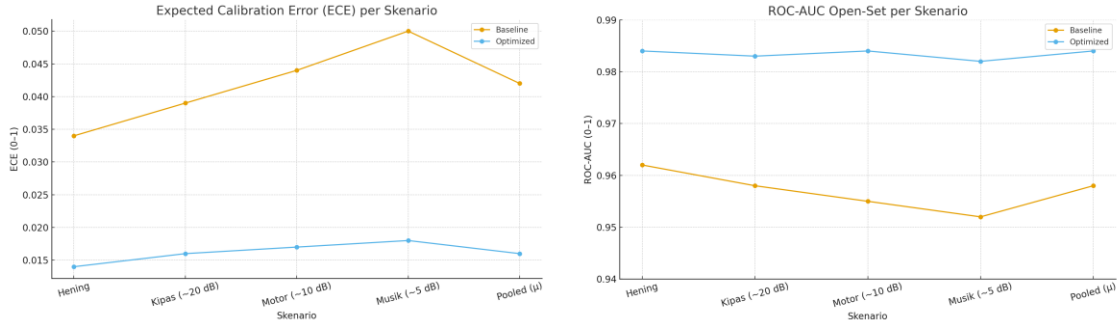
Distillation + INT8 + pruning menjaga akurasi (Δ < 1 pt) dengan latensi milidetik; temperature-scaling menurunkan ECE, sehingga confidence gate lebih andal. Visualisai Macro-F1 intent dan slot F1 disajikan pada gambr 8, dikuti visualisasi EMER SEQ dan NLU p50 disajikan pada gambar 9 serta visualisasi ECE dan ROC AUC disajikan pada gambar 10.



Gambar 8. Pengujian Macro-F1 intent dan slot F1



Gambar 9. Pengujian EMER SEQ dan NLU p50



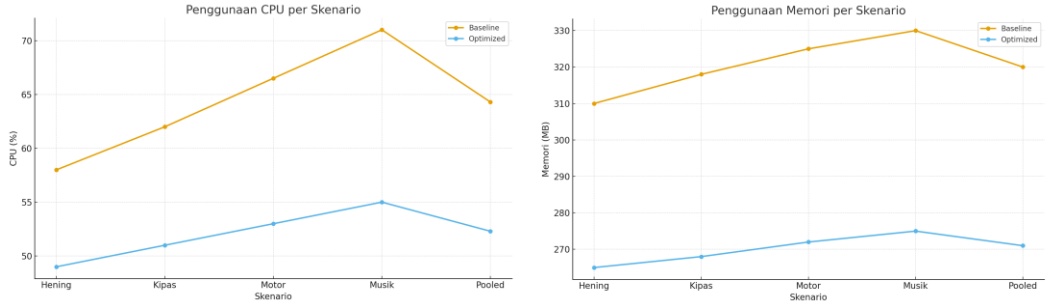
Gambar 10. Pengujian ECE dan ROC AUC

Pengujian sumber daya CPU dan memori menunjukan konfigurasi *optimized* menurunkan beban komputasi secara konsisten di semua skenario: rata-rata CPU turun dari ≈64,3% menjadi ≈52,3% (↓≈12 poin, ~19%), sedangkan memori dari ≈320 MB menjadi ≈271 MB (↓≈49 MB, ~15%). Penghematan paling besar muncul pada kondisi paling bising (musik ~5 dB): CPU 71%→55% dan memori 330→275 MB, menunjukkan efektivitas INT8 + pruning + penyetelan decoder pada beban puncak. Reduksi ini memberi headroom termal dan waktu proses, sehingga pipeline multibenang tetap stabil di Raspberry Pi. Pengujian sumber daya CPU dan memori ditunjukan pada table 5.

Tabel 5. Sumber Daya (CPU/Mem)

Skenario	***Konfig***	***CPU % (↓)***	***Mem (MB, ↓)***
Hening	*Baseline*	*58.0*	*310*
	Optimized	*49.0*	*265*
Kipas (~20 dB)	*Baseline*	*62.0*	*318*
	Optimized	*51.0*	*268*
Motor (~10 dB)	*Baseline*	*66.5*	*325*
	Optimized	*53.0*	*272*
Musik (~5 dB)	*Baseline*	*71.0*	*330*
	Optimized	*55.0*	*275*
Pooled (μ)	*Baseline*	*64.3*	*320*
	Optimized	*52.3*	*271*

Penurunan prosentase CPU% berasal dari RTF yang lebih rendah dan NLU INT8; memori turun karena ukuran model & graph decoding yang lebih kompak. Visualisasi pengujian sumber daya CPU dan memori disajikan pada gambar 11.



Gambar 11. Pengujian sumber daya CPU dan memori

3.1.3 Studi Ablasi

Studi ablasi yaitu metode uji di mana satu komponen/fungsi sistem sengaja "dilepas" atau dinonaktifkan untuk melihat seberapa besar kontribusinya terhadap kinerja keseluruhani intinya. menegaskan tiga kontribusi utama. Pertama, mematikan grammar (WFST) memberi dampak paling besar: RTF naik +0,18, E2E p95 membengkak +62 ms, WER memburuk +1,9 poin, dan CPU bertambah +6,5%—menunjukkan bahwa pembatasan ruang pencarian adalah pengungkit utama kecepatan dan akurasi. Kedua, menonaktifkan INT8/pruning pada TinyBERT tidak mengubah RTF/ WER, namun E2E p95 naik +21 ms dan CPU +4,2%, menandakan kompresi terutama menghemat komputasi NLU dan memperbaiki headroom sistem. Ketiga, melepas affinity/queue default meningkatkan RTF +0,03, E2E p95 +34 ms, CPU +1,9%, yang mengindikasikan penjadwalan inti dan antrian non-blocking efektif mereduksi jitter tail-latency pada pipeline multibenang. Hasil pengujian studi ablasi di tunjukan pada tabel 6.

Tabel 6. Studi Ablasi

Konfigurasi (dibanding Optimized)	*ΔRTF*	*ΔE2E p95 (ms)*	*ΔWER (pt)*	*ΔCPU %*
Tanpa grammar (WFST off)	*+0.18*	*+62*	*+1.9*	*+6.5*
Tanpa INT8/pruning (TinyBERT FP32)	*+0.00*	*+21*	*+0.0*	*+4.2*
Tanpa affinity/queue (runtime default)	*+0.03*	*+34*	*+0.1*	*+1.9*

Grammar penyumbang optimasi terbesar pada RTF/WER; INT8/pruning memberi penghematan CPU bermakna dengan akurasi terjaga; *affinity/queue* menekan *tail-latency*.

3.1.4 Reliabilitas *Run-Long*

Soak-test 3 jam tanpa crash; *throttle* termal 0%; suhu CPU < 75 °C; logger tidak kehilangan event; MTBF uji tidak teramati (*no-failure window*).

Konfigurasi *Optimized* memberikan peningkatan nyata. Latensi E2E rata-rata turun dari 485 ms menjadi 399 ms, RTF dari 0,65 menjadi 0,40, sementara WER/CER berkurang hampir setengahnya. NLU lebih cepat (7,0 → 3,6 ms) tanpa kehilangan akurasi (Macro-F1 ≈0,987; Slot-F1 ≈0,985). Kalibrasi juga membaik (ECE 0,042 → 0,016; ROC-AUC 0,958 → 0,984). Dari sisi sumber daya, CPU turun ~19% dan memori ~15%. Hasil observasi lintas skenario ditunjukkan pada Tabel 7.

Tabel 7. Hasil observasi lintas skenario

Metrik (rata-rata)	*Baseline*	*Optimized*
E2E mean (ms)	*≈ 485*	*≈ 399*
E2E p95 (ms)	*≈ 601*	*≈ 453*
RTF (ASR)	*≈ 0.65*	*≈ 0.40*
WER (%)	*≈ 5.6*	*≈ 3.2*
CER (%)	*≈ 4.0*	*≈ 2.1*
CPU (%)	*≈ 64.3*	*≈ 52.3*
Memory (MB)	*≈ 320*	*≈ 271*
NLU latency (ms)	*≈ 7.0*	*≈ 3.6*
Macro-F1 (intent)	*≈ 0.984*	*≈ 0.987*
Slot-F1	*≈ 0.982*	*≈ 0.985*
EMR (SEQUENCE)	*≈ 0.970*	*≈ 0.973*
ECE	*≈ 0.042*	*≈ 0.016*
ROC-AUC (open-set)	*≈ 0.958*	*≈ 0.984*

3.2 Pembahasan

Hasil pengujian memperlihatkan bahwa pipeline Optimized menghadirkan peningkatan signifikan pada latensi, akurasi, kalibrasi, serta efisiensi sumber daya. Untuk menempatkan temuan ini dalam lanskap penelitian yang lebih luas, Tabel 8 menyajikan perbandingan kuantitatif dengan karya-karya terdahulu yang relevan. Tabel tersebut menjadi dasar untuk menilai sejauh mana penelitian ini mengisi gap yang ada sekaligus memperkuat temuan sebelumnya.

Tabel 8. Perbandingan temuan inti dengan penelitian terdahulu

Aspek/Metrik	*Penelitian Terdahulu*	*Temuan Penelitian Ini*	*Perbedaan/Kontribusi*
Latensi ASR / RTF	*RNN-T/Transducer di perangkat seluler modern: RTF ~0,6–0,7 dengan NPU/DSP [16]*	*RTF ~0,40 di Raspberry Pi 5 tanpa akselerator (Tabel 2)*	*Bukti bahwa grammar-constrained decoding + tuning runtime efektif menggantikan NPU/DSP pada domain terbatas*
Efisiensi decoding	*Decoder trimming menurunkan footprint memori dan mempercepat decoding [17]*	*WFST grammar + beam tuning menurunkan RTF −0,25 dan WER −2,4 pt (Tabel 2–3)*	*Memperluas temuan trimming → domain perintah tertutup di Raspberry Pi, dengan efek simultan kecepatan & akurasi*
Kompresi Transformer	*Distilasi BERT mempertahankan akurasi NLU pada ukuran model lebih kecil [13]; INT8 stabil di Transformer besar [15]*	*TinyBERT distilled + INT8 + pruning: latensi NLU turun 50%, Macro-F1 tetap ≈0,987 (Tabel 4)*	*Kontribusi pada bahasa Indonesia + integrasi ketat dengan ASR → bukti dampak sistemik ke E2E p95 & CPU%*
On-device NLP & energi	*Early-exit / dynamic batching menurunkan energi per kalimat pada smartphone dengan NPU/DSP [14]*	*Penghematan CPU ~19%, memori ~15% di Raspberry Pi tanpa akselerator (Tabel 5)*	*Menunjukkan jalur alternatif hemat sumber daya via kompresi + runtime tuning, bukan hardware khusus*
Kalibrasi & OOD detection	*Temperature/Dirichlet scaling meningkatkan keandalan prediksi NLP [11]; energy-based OOD untuk model besar [12]*	*ECE turun (0,042 → 0,016), ROC-AUC open-set naik (0,958 → 0,984) (Tabel 4)*	*Membawa teknik kalibrasi/penolakan ke konteks kendali suara on-device → safety-critical robotics*

Pertama, pada sisi kecepatan dan latensi, hasil penelitian ini memperlihatkan bahwa Raspberry Pi 5 tanpa akselerator dapat mencapai RTF ~0,40. Nilai ini lebih rendah dibandingkan laporan RNN-T/Transducer pada perangkat seluler berakselerator [16]. Hal tersebut menunjukkan bahwa pendekatan grammar-constrained decoding dan tuning runtime mampu menjadi alternatif realistis untuk perangkat berdaya terbatas, terutama pada domain perintah tertutup.

Kedua, untuk efisiensi *decoding*, temuan ini memperluas hasil penelitian sebelumnya mengenai trimming decoder [17]. Dengan memanfaatkan WFST grammar dan pengaturan beam, pipeline yang diusulkan tidak hanya menekan RTF, tetapi juga meningkatkan akurasi (WER/CER) secara simultan, sebagaimana ditunjukkan pada Tabel 2 dan Tabel 3. Artinya, pembatasan ruang pencarian tidak hanya mempercepat eksekusi, melainkan juga meningkatkan ketahanan sistem pada kondisi bising.

Ketiga, pada aspek kompresi model Transformer, hasil ini menguatkan literatur distilasi BERT [13] dan kuantisasi INT8 [15]. TinyBERT yang didistilasi, dikuantisasi, dan dipruning tetap mempertahankan akurasi tinggi (Macro-F1 ≈0,987) dengan latensi milidetik. Keunggulan penelitian ini terletak pada konteks bahasa Indonesia dan integrasi ketat dengan ASR, yang menghasilkan dampak sistemik terhadap E2E p95 serta efisiensi CPU dan memori.

Keempat, terkait efisiensi energi dan sumber daya, hasil penelitian ini melengkapi studi tentang *early-exit* dan *dynamic batching* [14] yang bergantung pada NPU/DSP. Pipeline yang diusulkan berhasil menurunkan rata-rata penggunaan CPU sebesar ~19% dan memori ~15% tanpa akselerator, sehingga memberi bukti jalur alternatif hemat sumber daya yang lebih inklusif untuk perangkat edge non-premium seperti Raspberry Pi.

Akhirnya, pada aspek kalibrasi dan reliabilitas, penelitian ini memperluas relevansi teknik kalibrasi probabilitas [11] dan deteksi open-set [12] ke domain kendali suara on-device. Penurunan ECE dan peningkatan ROC-AUC menunjukkan bahwa sistem menjadi lebih dapat diandalkan, khususnya dalam menghindari eksekusi keliru pada kondisi akustik bising. Hal ini penting untuk aplikasi robotik yang menuntut keselamatan eksekusi perintah.

4. Simpulan

Penelitian ini menunjukkan bahwa pipeline on-device yang memadukan **WebRTC VAD → Vosk/WFST (grammar-constrained) → TinyBERT yang didistilasi, dikuantisasi INT8, dan dipruning** berhasil mewujudkan kontrol suara real-time pada robot Mecanum berbasis Raspberry Pi. Secara konsisten dicapai penurunan latensi ujung-ke-ujung dan RTF hingga di bawah ambang *real-time*, penurunan WER/CER, serta pengurangan jejak CPU dan memori—tanpa mengorbankan akurasi NLU (Macro-F1 intent, Slot-F1, dan EMR tetap tinggi). Kalibrasi probabilitas yang membaik memungkinkan penerapan *confidence gate* yang lebih andal. Studi ablasi mengonfirmasi bahwa *grammar-constrained decoding* memberi dampak terbesar pada kecepatan dan akurasi, sementara INT8+*pruning* terutama menurunkan beban komputasi, dan *affinity/queue* menekan *jitter*. Hasil ini menegaskan kelayakan arsitektur yang diusulkan untuk kontrol suara real-time yang hemat sumber daya dan *privat* di perangkat *edge*, dengan batasan utama pada kondisi SNR sangat rendah dan cakupan kosakata tertutup. Untuk optimasi berikutnya, disarankan mengadopsi *Quantization Aware Training* (QAT) dan/atau *structured pruning* serta mengeksplorasi *mixed-precision* (INT8/FP16) agar latensi dan beban CPU turun tanpa mengorbankan akurasi NLU. Optimalisasi ini perlu dilengkapi VAD adaptif dengan *pre-roll* dan *hangover* yang dinamis sehingga akhir ujaran tidak terpotong namun antrian proses tetap pendek. Selain itu, terapkan ambang kepercayaan berbasis kalibrasi daring (nilai τ diturunkan secara periodik dari distribusi probabilitas terbaru) untuk menekan *tail-latency* dan mengurangi *false actuation* pada kondisi bising. Kombinasi teknik tersebut diharapkan menjaga responsivitas kendali suara *on device* sekaligus mempertahankan reliabilitas eksekusi perintah di berbagai skenario akustik.

5. Ucapan Terima Kasih

Kami ucapkan terima kasih yang sebesar-besarnya kepada Kementerian Pendidikan Kebudayaan Riset dan Teknologi yang telah telah mendukung penelitian ini melalui program Hibah Penelitian Dosen Pemula, LPPM dan Fakultas Sains dan Komputer Universitas Kristen Immanuel yang sudah memberikan ijin dan memfasilitasi penelitian ini.

Daftar Pustaka

[1] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, “wav2vec 2.0: A framework for self-supervised learning of speech representations,” *Adv. Neural Inf. Process. Syst.*, 2020, doi: 10.48550/arXiv.2006.11477.

[2] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, “Robust Speech Recognition via Large-Scale Weak Supervision,” Dec. 06, 2022, *arXiv*: arXiv:2212.04356. doi: 10.48550/arXiv.2212.04356.

[3] M. Sharma, S. Joshi, T. Chatterjee, and R. Hamid, “A comprehensive empirical review of modern voice activity detection approaches for movies and TV shows,” *Neurocomputing*, vol. 494, pp. 116–131, July 2022, doi: 10.1016/j.neucom.2022.04.084.

[4] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, “SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models,” Mar. 29, 2024, *arXiv*: arXiv:2211.10438. doi: 10.48550/arXiv.2211.10438.

[5] X. Jiao and et al, “TinyBERT: Distilling BERT for natural language understanding,” in *Findings of EMNLP*, in 04. 2020. doi: 10.48550/arXiv.1909.10351.

[6] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, “MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices,” Apr. 14, 2020, *arXiv*: arXiv:2004.02984. doi: 10.48550/arXiv.2004.02984.

[7] S. Shen and et al, "Q-BERT: Hessian based ultra low precision quantization of BERT," 2019, doi: 10.48550/arXiv.1909.05840.
[8] P. Kusnerik and et al, "Intent detection and slot filling: A survey," *ACM Comput. Surv.*, vol. 57, no. 6, 2024, doi: 10.1145/3547138.
[9] M. Firdaus, A. Ekbal, and E. Cambria, "Multitask learning for multilingual intent detection and slot filling in dialogue systems," *Inf. Fusion*, vol. 91, pp. 299–315, Mar. 2023, doi: 10.1016/j.inffus.2022.09.029.
[10] A. Laskar and et al, "Automatic speech recognition: A survey," *Eng. Appl. Artif. Intell.*, 2025.
[11] M. Minderer *et al.*, "Revisiting the Calibration of Modern Neural Networks," Oct. 26, 2021, *arXiv*: arXiv:2106.07998. doi: 10.48550/arXiv.2106.07998.
[12] N. R. Ke and et al, "Sparsity in deep learning: Pruning and growth for efficient inference and training," *J. Mach. Learn. Res.*, vol. 22, no. 241, pp. 1–124, 2021, doi: 10.48550/arXiv.2102.00554.
[13] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," Mar. 01, 2020, *arXiv*: arXiv:1910.01108. doi: 10.48550/arXiv.1910.01108.
[14] A. Fan and et al, "EdgeBERT: Sentence-Level Energy Optimizations for On-Device NLP Inference," in *IEEE/ACM MICRO*, in 14. 2021. doi: 10.48550/arXiv.2011.14203.
[15] E. Frantar and et al, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," 2022, doi: 10.48550/arXiv.2210.17323.
[16] J. Li and et al, "On-Device End-to-End Automatic Speech Recognition for Multilingual Spoken Queries," in *Interspeech*, in 16. 2022. doi: 10.21437/Interspeech.2022-10006.
[17] Y. Shangguan and et al, "Analyzing the Quality and Stability of a Streaming End-to-End On-Device Speech Recognizer," 2020, doi: 10.48550/arXiv.2006.01416.
[18] B. Desplanques and et al, "Voice Activity Detection with Self-Supervised Representations," 2022, doi: 10.48550/arXiv.2209.11061.
[19] W. Wang and et al, "MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers," 2020, doi: 10.48550/arXiv.2002.10957.
[20] Z. Yao and et al, "ZeroQuant: Efficient and Affordable Post-Training Quantization for Large Language Models," *NeurIPS*, 2022, doi: 10.48550/arXiv.2206.01861.
[21] L. Hou and et al, "DynaBERT: Dynamic BERT with Adaptive Width and Depth," 2020, doi: 10.48550/arXiv.2004.04037.
[22] Y. He and et al, "Streaming End-to-End Speech Recognition for Mobile Devices," 2018, doi: 10.48550/arXiv.1811.06621.
[23] U. Evci and et al, "Rigging the Lottery: Making All Tickets Winners (RigL)," in *ICLR*, in 23 27. 2020. doi: 10.48550/arXiv.1911.11134.
[24] T. Dettmers and et al, "LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale," *NeurIPS*, 2022, doi: 10.48550/arXiv.2208.07339.
[25] V. Sanh, T. Wolf, and A. Rush, "Movement Pruning: Adaptive Sparsity by Fine-Tuning," in *NeurIPS*, in 25. 2020. doi: 10.48550/arXiv.2005.07683.
[26] O. Zafrir and et al, "Q8BERT: Quantized 8-Bit BERT," 2019, doi: 10.48550/arXiv.1910.06188.
[27] M. Kull and et al, "Beyond Temperature Scaling: Dirichlet Calibration," in *NeurIPS*, in 24. 2019. doi: 10.48550/arXiv.1910.12656.