GIỚI THIỆU GIẢI PHÁP AI VOICE

ASR & Text-to-Speech

MỤC LỤC

Contents

1. TỔNG QUAN

Tài liệu này mô tả kiến trúc và các thành phần kỹ thuật của giải pháp AI VOICE do đội ngũ Sphinx phát triển

- ASR (Automatic Speech Recognition) – Chuyển đổi giọng nói sang văn bản.
- TTS (Text-to-Speech) – Tổng hợp giọng nói từ văn bản với khả năng clone giọng.

Cả hai giải pháp đều được expose ra ngoài thông qua API Gateway, sử dụng Python FastAPI làm tầng giao tiếp HTTP/WebSocket. Dữ liệu inference thực tế chạy trên GPU NVIDIA cao cấp, được đề xuất sử dụng A6000 Ada hoặc L40S.

1.1. Phạm vi tài liệu

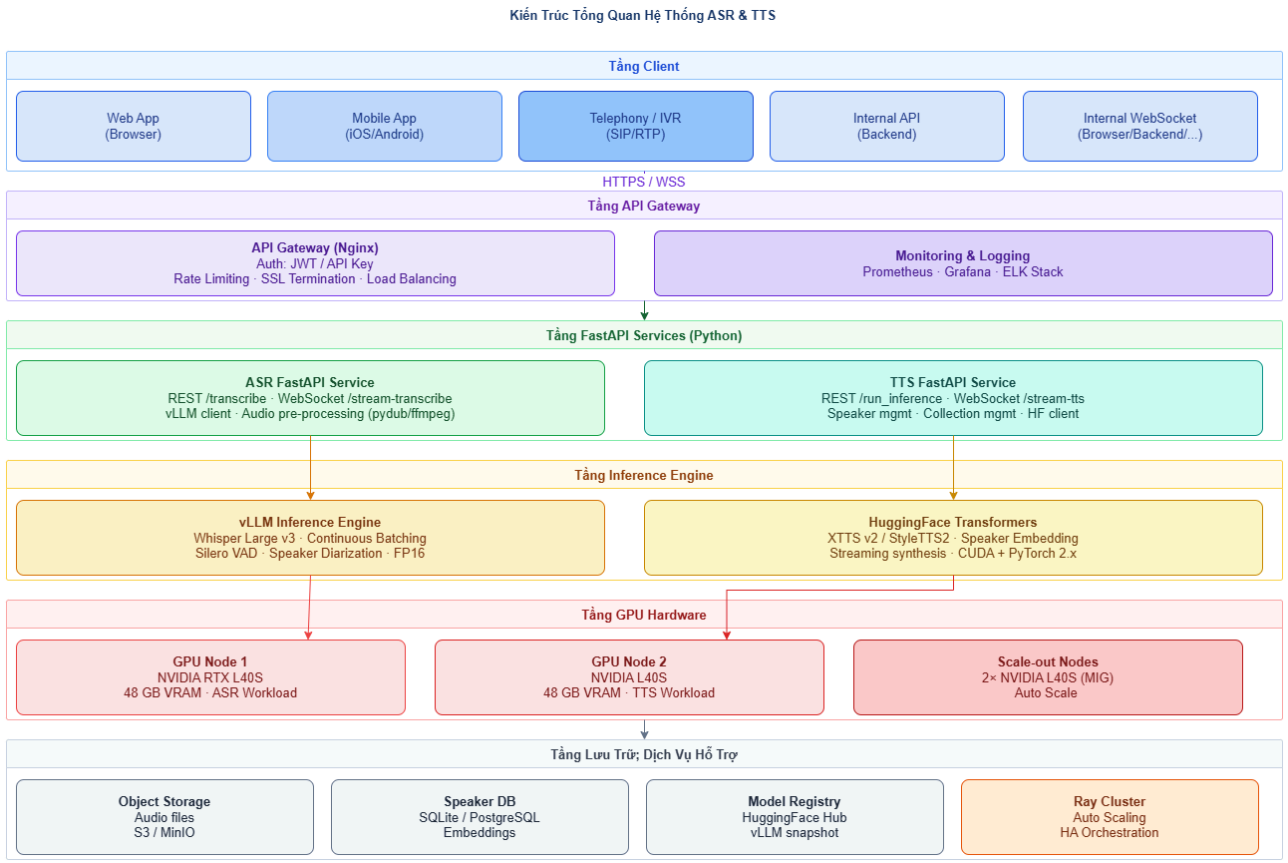
Tài liệu tập trung vào:

- Kiến trúc triển khai chi tiết của từng giải pháp ASR và TTS, bao gồm sơ đồ tổng quan hệ thống.
- Stack công nghệ: Inference Engine, API Layer, thư viện sử dụng.
- Đề xuất phần cứng GPU (A6000 Ada / L40S) kèm ước tính CCU (Concurrent Users).
- Danh sách API endpoint cung cấp cho bên tích hợp.
- Khả năng Auto Scaling với Ray Serve và đảm bảo High Availability (HA) cho ASR và TTS.
- Phương án phối hợp với Microsoft Fabric MCP trong trường hợp triển khai tích hợp dữ liệu.

2. KIẾN TRÚC TRIỂN KHAI TỔNG THỂ

Hai giải pháp ASR và TTS được triển khai dưới dạng microservice độc lập, đứng sau một API Gateway chung. Luồng dữ liệu tổng quát được thể hiện trong sơ đồ kiến trúc dưới đây.

2.1. Sơ đồ Kiến Trúc Tổng Quan



Hình 1: Kiến trúc tổng quan hệ thống ASR & TTS

2.2. Diễn Giải Chi Tiết Các Tầng Kiến Trúc

Tầng Client

Đây là tầng tiếp nhận yêu cầu từ các ứng dụng đầu cuối bao gồm:

- Web App (trình duyệt): giao diện người dùng web gửi audio/text qua HTTPS hoặc WebSocket
- Mobile App (iOS/Android): ứng dụng di động gọi API qua REST hoặc WebSocket.
- Telephony / IVR (SIP/RTP): hệ thống tổng đài, nhận audio từ cuộc gọi điện thoại và chuyển qua gateway.
- Internal API (Backend Services): các service nội bộ tích hợp qua REST API.
- Internal WebSocket: các dịch vụ nội bộ gọi qua nhau thông qua giao thức WebSocket

Tầng API Gateway

API Gateway đảm nhận:

- Xác thực (Authentication): kiểm tra API Key và JWT token từ header request.
- Rate Limiting: giới hạn số request/giây để bảo vệ hệ thống khỏi DDoS và overload.

- Load Balancing: phân phối tải đến nhiều instance FastAPI theo thuật toán round-robin hoặc least-connection.
- SSL Termination: xử lý HTTPS/WSS tại gateway, các service nội bộ giao tiếp qua HTTPS thuần.
- Monitoring & Logging: thu thập metrics qua Prometheus, hiển thị trên Grafana, log tập trung qua ELK Stack.

Tầng FastAPI Services

Mỗi service (ASR và TTS) được triển khai độc lập bằng Python FastAPI:

- ASR FastAPI Service: tiền xử lý audio (convert format, normalize), gọi vLLM inference engine, trả kết quả text + timestamps.
- TTS FastAPI Service: quản lý speaker embedding, gọi HuggingFace Transformers inference, stream audio về client.
- Cả hai hỗ trợ REST (synchronous) và WebSocket (streaming realtime).

Tầng Inference Engine

Tầng này chứa model AI thực tế chạy trên GPU:

- vLLM (ASR): framework inference GPU hiệu năng cao, sử dụng continuous batching để tối ưu throughput, chạy model ASR của Sphinx.
- HuggingFace Transformers (TTS): framework linh hoạt cho các model TTS thế hệ mới (XTTS v2, StyleTTS2), tối ưu streaming realtime, chạy model TTS của Sphinx

Tầng GPU Hardware

- GPU Node 1 (L40S, 48GB VRAM): phục vụ workload ASR – compute intensive, throughput cao.
- GPU Node 2 (L40S, 48GB VRAM): phục vụ workload TTS – latency sensitive, hỗ trợ MIG.
- Scale-out Nodes (2x L40S với MIG): mở rộng tự động khi đạt trigger points.

Tầng Lưu Trữ & Dịch Vụ Hỗ Trợ

- Object Storage (S3/MinIO): lưu trữ file audio input/output.
- Speaker DB (SQLite/PostgreSQL): lưu speaker metadata và embedding vector.
- Model Registry (HuggingFace Hub + vLLM snapshot): quản lý phiên bản model AI.
- Ray Cluster: điều phối auto-scaling và HA (chi tiết tại Mục 5).

2.3. Bảng Phân Loại Tài Nguyên

Model	Inference Engine	Interface	Phân nhóm GPU	Ghi chú
ASR	vLLM	REST + WebSocket	High GPU (Compute Intensive)	Xử lý batch audio, cần throughput cao
TTS	HF Transformers	REST + WebSocket	Medium GPU (Latency Sensitive)	Streaming realtime, ưu tiên latency thấp
SV / AS	ONNX Runtime	REST	Low GPU / CPU	(Ngoài phạm vi tài liệu này)

3. GIẢI PHÁP ASR (AUTOMATIC SPEECH RECOGNITION)

3.1. Mô tả chức năng

Giải pháp ASR chuyển đổi file audio (WAV, MP3, M4A,...) thành văn bản. Hỗ trợ:

- Tự động phát hiện ngôn ngữ (Vietnamese, English,...).
- Tách đoạn im lặng tự động bằng Silero VAD.
- Trả về text kèm word-level timestamps và speaker diarization (phân biệt người nói).
- Xử lý streaming theo WebSocket để giảm latency.

3.2. Thành phần kỹ thuật

3.2.1. Inference Engine – vLLM

ASR được phục vụ bởi vLLM – framework inference GPU hiệu năng cao, chuyên tối ưu throughput cho các model lớn.

3.2.2. API Layer – Python FastAPI

Tầng API được xây dựng bằng Python FastAPI, đóng vai trò:

- Nhận request từ API Gateway (HTTP multipart/form-data hoặc JSON).
- Tiền xử lý audio (convert format, normalize sample rate).
- Gọi vLLM inference engine qua localhost.
- Trả kết quả JSON (text, segments, timestamps, speaker labels).

Stack thư viện Python chính:

```
fastapi >= 0.110     – HTTP framework, async-native
uvicorn              – ASGI server chạy FastAPI
vllm >= 0.4          – Inference engine cho model ASR
silero-vad           – Voice Activity Detection
pydub / ffmpeg       – Xử lý, convert audio format
python-multipart     – Parse file upload (multipart/form-data)
```

3.3. API Endpoints

Tất cả endpoint dùng chung base URL và header xác thực:

```
Base URL: https://gateway.brighto.ai/gateway/transcribe-service/
Headers bắt buộc:
  apiKey: <API_KEY_DO_HE_THONG_CAP>
  Origin: <IP_hoac_domain_cua_client>
```

Method	Endpoint	Chức năng	Ghi chú
POST	/transcribe	Transcribe audio	Upload file audio, trả về text + segments + duration

POST	/transcribe-with-property	Transcribe nâng cao	Trả thêm word timestamps + speaker diarization

Ví dụ request /transcribe

```
curl -X POST 'https://gateway.brighto.ai/gateway/transcribe-service/transcribe?language=vi' \
    -H 'apiKey: YOUR_API_KEY' \
    -H 'Origin: https://yourdomain.com' \
    -F 'file=@audio.mp3'
// Response mẫu:
// { "text": "Xin chào, tôi cần hỗ trợ...",
//   "segments": [{"start": 0.0, "end": 3.2, "text": "Xin chào"}],
//   "duration": 15.4, "processing_time": 1.2 }
```

3.4. Khuyến nghị GPU & Ước tính CCU

Do tính chất Compute Intensive (nhiều phép tính float trên audio dài), ASR yêu cầu GPU có VRAM lớn và throughput FP16 cao.

Thông số	NVIDIA RTX A6000 Ada	NVIDIA L40S	Ghi chú
VRAM	48 GB GDDR6	48 GB GDDR6	Tương đương
TDP	300W	350W	L40S tiêu thụ điện hơn
FP16 TFLOPS	~362 TFLOPS	~366 TFLOPS	Ngang nhau
Kiến trúc	Ada Lovelace	Ada Lovelace	Cùng thế hệ
Tối ưu cho	Workstation / Data Center (dual-slot)	Data Center (server rack)	L40S phù hợp server rack hơn
CCU ASR ước tính (audio ~30s/request)	30-40 CCU	30-40 CCU	vLLM continuous batching; L40S cao hơn nhờ MIG support

Giả định CCU:

- Mỗi request xử lý audio ~30 giây, inference time ~1-2 giây (realtime factor ~15-20x).
- LLM continuous batching gom 8-16 request/batch.
- Mô hình Sphinx ASR chiếm ~7-12 GB VRAM; còn dư VRAM cho batching.
- L40S có hỗ trợ MIG (Multi-Instance GPU) – có thể chia GPU phục vụ workload phụ.
- Nếu cần tăng CCU: scale horizontal thêm node GPU.

4. GIẢI PHÁP TTS (TEXT-TO-SPEECH)

4.1. Mô tả chức năng

Giải pháp TTS tổng hợp giọng nói tự nhiên từ văn bản. Các tính năng nổi bật:

- Voice cloning – học và tái tạo giọng nói từ audio mẫu của người dùng.
- Hỗ trợ đa ngôn ngữ thông qua cơ chế Collection (map ngôn ngữ → speaker_id).
- Streaming realtime – trả audio ngay khi sinh được, không cần đợi xong toàn bộ.
- Xử lý song song với tham số num_tasks.

4.2. Thành phần kỹ thuật

4.2.1. Inference Engine – HuggingFace Transformers

TTS sử dụng HuggingFace Transformers làm inference engine, phù hợp với các model TTS thế hệ mới (XTTS v2, Bark, StyleTTS2,...).

4.2.2. API Layer – Python FastAPI

Tầng FastAPI của TTS service đảm nhiệm:

- Quản lý Speaker: upload audio mẫu, extract embedding, lưu DB.
- Quản lý Collection: nhóm speaker theo ngôn ngữ cho usecase đa ngôn ngữ.
- Inference endpoint: nhận text, lấy speaker embedding, gọi TTS model, stream audio.
- WebSocket endpoint: test real-time streaming trong môi trường dev.

Stack thư viện Python chính:

```
fastapi >= 0.110        – HTTP/WebSocket framework
uvicorn                 – ASGI server
transformers >= 4.40    – Load và run TTS model (HuggingFace)
torch >= 2.2            – PyTorch backend, tối ưu CUDA cho A6000/L40S
torchaudio              – Xử lý audio tensor, resample
numpy / scipy           – Tiền/hậu xử lý waveform
sqlalchemy / sqlite     – Lưu trữ speaker metadata và embeddings
python-multipart        – Parse file upload
```

4.3. API Endpoints

```
Base URL: https://gateway.brighto.ai/gateway/transcribe-service/
Headers bắt buộc:
  apiKey: <API_KEY_DO_HE_THONG_CAP>
  Origin: <IP_hoac_domain_cua_client>
```

Method	Endpoint	Chức năng	Ghi chú
POST	/api/speakers/upload	Tạo speaker mới	Upload audio mẫu + reference_text để extract giọng
GET	/api/speakers	Liệt kê speakers	Lọc theo user_id

DELETE	/api/speakers/{id}	Xóa speaker	Không thể hoàn tác
POST	/api/collections	Tạo collection	Nhóm speaker theo ngôn ngữ (vi, en, ja,...)
PUT	/api/collections/{id}	Cập nhật collection	Thay đổi speaker mapping theo ngôn ngữ
POST	/run_inference	TTS Inference	Endpoint chính – nhận text, trả audio streaming

Ví dụ request /run_inference (TTS)

```
curl -X POST 'https://gateway.brighto.ai/gateway/transcribe-service/run_inference' \
  -H 'apiKey: YOUR_API_KEY' -H 'Content-Type: application/json' \
  -d '{"target_text": "Xin chào quý khách, tôi là trợ lý ảo của đơn vị abc.",
       "speaker_id": "spk_001", "language": "vi", "is_realtime": true}'
```

4.4. Khuyến nghị GPU & Ước tính CCU

TTS thuộc nhóm Latency Sensitive – ưu tiên thời gian đến audio đầu tiên (Time-to-First-Audio) thấp.

Thông số	NVIDIA RTX A6000 Ada	NVIDIA L40S	Ghi chú
VRAM	48 GB GDDR6	48 GB GDDR6	Tương đương
Memory Bandwidth	864 GB/s	864 GB/s	Quan trọng cho TTS latency
FP16 TFLOPS	~362 TFLOPS	~366 TFLOPS	Ngang nhau
CCU TTS ước tính (text ~200 ký tự/request)	30 – 50 CCU	40 – 60 CCU	HF Transformers không batching mạnh như vLLM; L40S cao hơn

Giả định CCU:

- Mỗi request synthesis ~200 ký tự, inference time ~0.8-1.5 giây (với streaming).
- XTTS v2 chiếm ~6-8 GB VRAM; còn buffer cho speaker embedding cache.
- is_realtime=true: client nhận audio chunk đầu tiên trong ~300-500ms.
- Nếu cần thêm CCU: scale horizontal.

5. AUTO SCALING VỚI RAY VÀ HIGH AVAILABILITY

Phần này mô tả cơ chế tự động mở rộng (Auto Scaling) và đảm bảo tính sẵn sàng cao (High Availability – HA) cho cả hai dịch vụ ASR và TTS, sử dụng Ray Serve làm nền tảng điều phối.

5.1. Tổng Quan Ray Serve

Ray Serve là framework phục vụ ML model của Ray AI Runtime, cung cấp:

- HTTP Proxy tích hợp: mỗi deployment tự expose endpoint HTTP, không cần reverse proxy riêng.
- Replica auto-scaling: tự động tăng/giảm số replica dựa trên số request đang chờ.
- Zero-downtime rollout: cập nhật model mà không làm gián đoạn traffic.
- Actor isolation: mỗi replica chạy trong Ray Actor độc lập – lỗi một replica không ảnh hưởng replica khác.
- Global Control Store (GCS): trạng thái cluster được persisted, đảm bảo recovery sau sự cố.

5.2. Chiến Lược High Availability

Health Probes

Ray Serve tích hợp health check tự động:

- Liveness probe: gửi GET /healthz mỗi 10 giây; replica không phản hồi trong 5 giây sẽ bị restart.
- Readiness probe: gửi GET /ready; replica chưa warmup xong sẽ không nhận traffic.
- Model warmup: khi replica khởi động, chạy 1 request dummy để load model vào VRAM trước khi nhận request thật.

Failover Policy

- max_restarts: 5 – Ray Serve tự restart replica lỗi tối đa 5 lần trước khi báo lỗi.
- Exponential backoff: thời gian chờ giữa các lần restart tăng theo cấp số nhân (1s → 2s → 4s → ...).
- Graceful shutdown: khi scale down, replica được thông báo trước 30 giây để hoàn thành request đang xử lý.
- Request retry: load balancer tự động retry sang replica khác nếu replica hiện tại lỗi.

Persistent State

- Ray Global Control Store (GCS) được backup mỗi 5 phút vào Redis/etcd.
- Speaker embedding cache được lưu vào Object Storage, replica mới có thể reload nhanh chóng.
- Checkpoint định kỳ cho phép cluster khôi phục trạng thái sau sự cố ngay cả khi Head Node gặp lỗi.

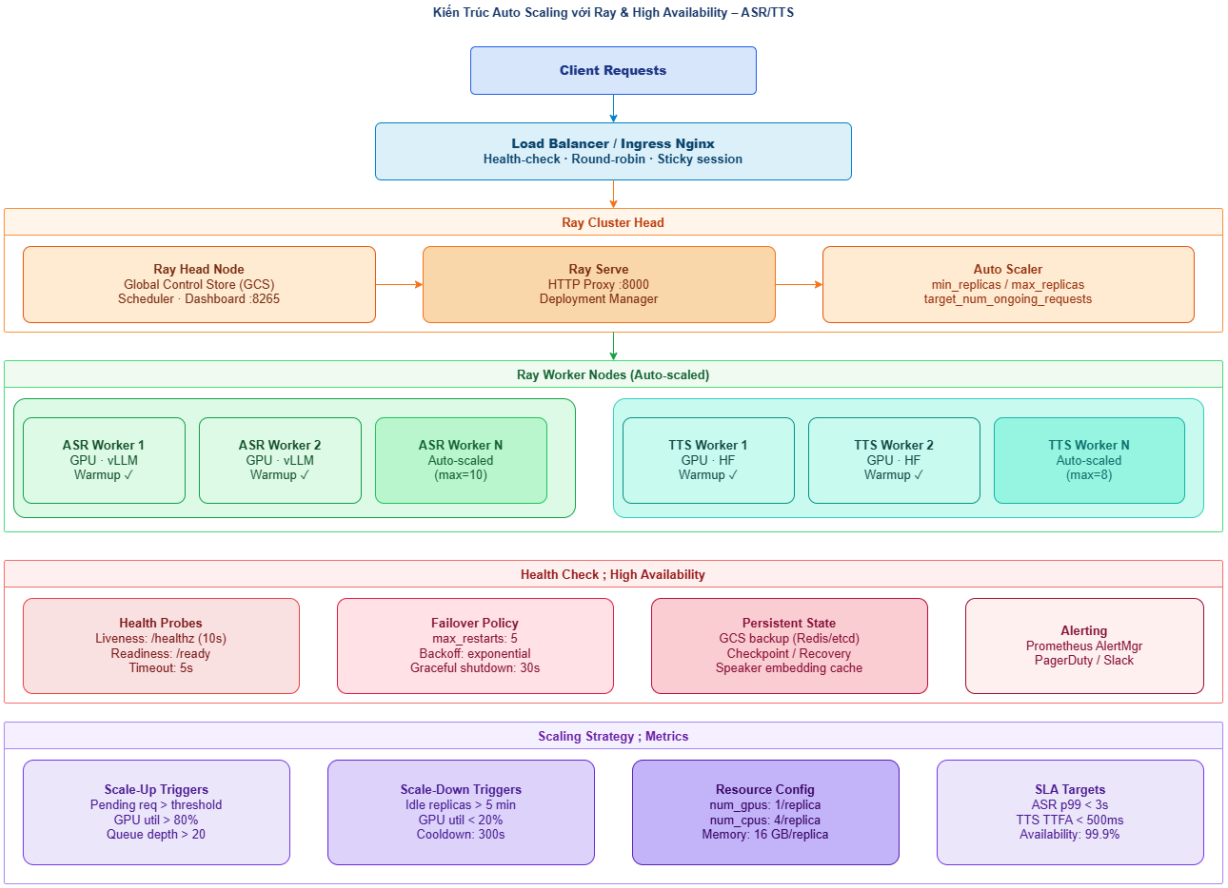
Monitoring & Alerting

- Prometheus scrapes metrics từ Ray Dashboard (:8265) và Ray Serve (:8000/metrics).
- Grafana hiển thị: số replica active, queue depth, GPU utilization, p50/p99 latency.
- AlertManager gửi cảnh báo qua PagerDuty khi: GPU > 90%, queue > 50 requests, replica restart > 3 lần.

5.3. Bảng Thông Số Scaling

Tham số	ASR	TTS	Ghi chú
min_replicas	1	1	Luôn sẵn sàng nhận request
max_replicas	10	8	Giới hạn GPU budget
target_ongoing_requests	8	5	TTS nhạy cảm latency hơn
upscale_delay_s	30s	15s	TTS scale nhanh hơn
downscale_delay_s	300s	300s	Tránh scale down quá nhanh
GPU per replica	1x L40S (48GB)	1x L40S (48GB)	Dedicated GPU
SLA p99 latency	< 3 giây	< 500ms TTFA	Time to First Audio (TTS)
CCU tối đa (10 replicas)	~300 CCU	~240 CCU	Với target=8/5 mỗi replica

5.4. Sơ Đồ Kiến Trúc Auto Scaling & High Availability



Hình 2: Kiến trúc Auto Scaling với Ray Serve và High Availability cho ASR/TTS

Diễn giải luồng Auto Scaling

Khi lưu lượng request tăng đột biến (ví dụ giờ cao điểm sử dụng):

1. Client gửi request đến Load Balancer (Nginx Ingress).
2. Ray Serve HTTP Proxy phân phối request đến các replica đang chạy.

3. AutoScaler theo dõi số request đang xử lý/replica. Khi vượt ngưỡng target, AutoScaler yêu cầu Ray Head Node tạo thêm replica.
4. Replica mới khởi động trên GPU node sẵn có (hoặc node mới được provision), chạy warmup, rồi bắt đầu nhận traffic.
5. Sau khi traffic giảm và duy trì thấp 300 giây, AutoScaler thu hồi replica dư thừa.

6. PHỐI HỢP VỚI MICROSOFT FABRIC MCP

Phần này mô tả phương án tích hợp hệ thống ASR/TTS của SphinxJSC với Microsoft Fabric thông qua giao thức Model Context Protocol (MCP), cho phép các AI Agent (Claude, GPT-4) trực tiếp gọi dịch vụ nhận dạng giọng nói và tổng hợp giọng nói như một tool.

6.1. Tổng Quan Microsoft Fabric MCP

Microsoft Fabric MCP (Model Context Protocol) là chuẩn mở do Anthropic khởi xướng và được Microsoft tích hợp vào Microsoft Fabric, cho phép:

- AI Agent sử dụng các dịch vụ dữ liệu và xử lý như tool call, không cần tích hợp trực tiếp API.
- Chuẩn hóa giao tiếp giữa LLM và các service backend thông qua JSON-RPC 2.0 over stdio hoặc SSE.
- Tool registry: định nghĩa các tool (functions) với schema đầu vào/đầu ra rõ ràng.
- Context management: MCP server giữ context cuộc hội thoại, cho phép multi-turn interaction.

6.2. Mô Hình Tích Hợp

Trong kịch bản tích hợp với khách hàng, các thành phần kết nối như sau:

- AI Agent Layer: Claude Sonnet (Anthropic), GPT-4 (Azure OpenAI), hoặc Client's Internal LLM đóng vai trò orchestrator.
- Fabric MCP Server: cung cấp hai tool chính là transcribe_audio và synthesize_speech, định tuyến request đến ASR/TTS service tương ứng.
- SphinxJSC ASR/TTS Services: nhận request từ MCP Server qua REST API nội bộ (API Key auth), thực hiện inference và trả kết quả.
- Microsoft Fabric Data Platform: lưu trữ audio files (OneLake), kết quả transcription (Lakehouse), và phân tích (Power BI).

6.3. Định Nghĩa Tool MCP

Tool: transcribe_audio

```
{
 "name": "transcribe_audio",
 "description": "Chuyển đổi file audio sang văn bản sử dụng ASR service",
 "inputSchema": {
  "type": "object",
  "properties": {
   "audio_url": {"type": "string", "description": "URL file audio (WAV/MP3/M4A)"},
   "language": {"type": "string", "enum": ["vi", "en", "auto"], "default": "auto"},
   "with_diarization": {"type": "boolean", "default": false},
   "with_timestamps": {"type": "boolean", "default": true}
  },
  "required": ["audio_url"]
 }
}
```

Tool: synthesize_speech

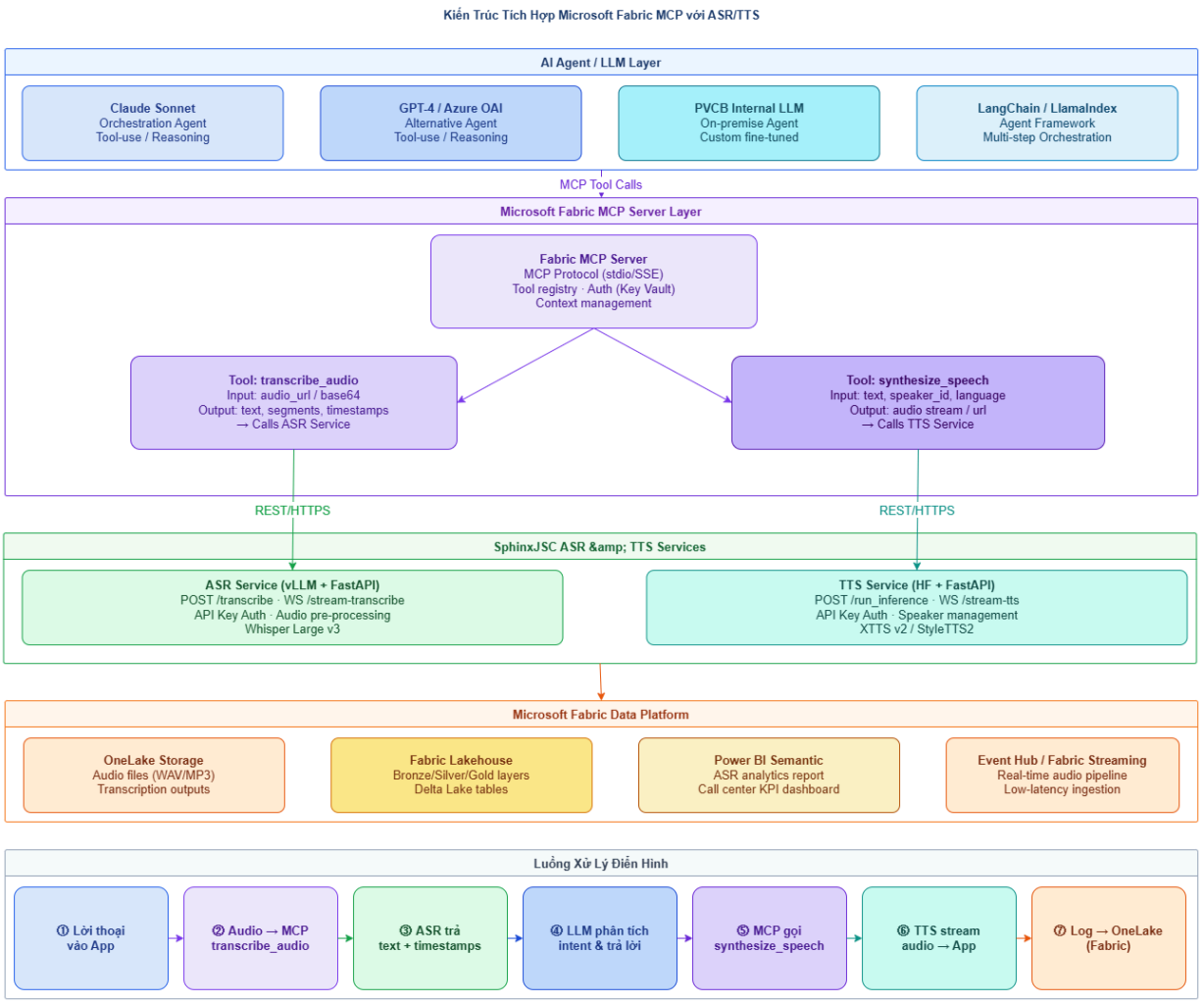
```
{
 "name": "synthesize_speech",
 "description": "Tổng hợp giọng nói từ văn bản sử dụng TTS service",
 "inputSchema": {
  "type": "object",
  "properties": {
   "text": {"type": "string", "description": "Văn bản cần đọc"},
   "speaker_id": {"type": "string", "description": "ID giọng nói (optional)"},
   "language": {"type": "string", "enum": ["vi", "en"], "default": "vi"},
   "is_realtime": {"type": "boolean", "default": true}
  },
  "required": ["text"]
 }
}
```

1.

6.5. Lợi Ích Khi Tích Hợp Fabric MCP

Khía cạnh	Không có MCP	Có Fabric MCP
Tích hợp AI Agent	Hard-code API call trong agent code	Tool call chuẩn hóa, agent dùng tool description
Quản lý context	Agent tự quản lý state	MCP Server quản lý conversation context tập trung
Bảo mật	API key trong agent code	MCP Server xử lý auth, agent không biết credential
Extensibility	Thêm service = sửa agent code	Thêm tool vào MCP registry, agent tự discove
Audit & Logging	Phân tán, khó tổng hợp	MCP ghi log tập trung vào Fabric Lakehouse
Multi-LLM support	Tích hợp riêng cho mỗi LLM	1 MCP Server phục vụ mọi LLM (Claude, GPT-4, Gemini)

6.6. Sơ Đồ Tích Hợp Fabric MCP



Hình 3: Kiến trúc tích hợp Microsoft Fabric MCP với ASR/TTS – AI Platform

Yêu cầu triển khai Fabric MCP

- Microsoft Fabric capacity: F64 hoặc cao hơn để sử dụng Fabric MCP Server.
- Network connectivity: SphinxJSC ASR/TTS endpoints phải accessible từ Fabric MCP Server (VPN hoặc public HTTPS).
- Credential management: API Key của ASR/TTS được lưu trong Azure Key Vault, MCP Server tự inject vào request.
- Tool schema validation: MCP Server validate input/output schema trước khi forward đến service.
- Error handling: MCP chuẩn hóa error response (JSON-RPC error code) cho Agent xử lý thống nhất.

7. KIẾN NGHỊ VÀ SO SÁNH GPU: A6000 Ada vs L40S

Cả hai GPU thuộc cùng thế hệ Ada Lovelace, có VRAM 48 GB và hiệu năng FP16 tương đương. Sự khác biệt nằm ở form factor và usecase tối ưu:

Tiêu chí	RTX A6000 Ada	L40S
Form factor	PCIe, dual-slot (workstation)	PCIe, single-slot passive (server rack)
VRAM	48 GB GDDR6	48 GB GDDR6
TDP	300W	350W
FP16 (Tensor Core)	~362 TFLOPS	~366 TFLOPS
INT8 / FP8	Hỗ trợ INT8	Hỗ trợ FP8 + INT8 (tốt hơn)
MIG Support	Không	Có (7 instances)
NVLink	Không (PCIe only)	Có (kết nối multi-GPU)
Khuyến nghị cho	Lab / workstation server nhỏ; dễ lắp, không cần rack chuyên dụng	Data center / rack server; scale production, hỗ trợ MIG và NVLink

7.1. Kết Luận Lựa Chọn GPU

Giải pháp	Khuyến nghị	Lý do	CCU ước tính
ASR	L40S (ưu tiên)	MIG support cho multi-workload, NVLink cho scale-out, server rack form factor	30-40 CCU/GPU ~300 CCU với 10 replicas
TTS	L40S (ưu tiên)	Bandwidth cao quan trọng cho latency TTS, CUDA MPS hỗ trợ concurrent inference	40-60 CCU/GPU ~240 CCU với 8 replicas

— Hết tài liệu —