DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova – Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

Relazione per la prova finale

Progettazione ed implementazione di un sistema Speech-to-text con Smart Chunking

Relatore: Prof. Di Buccio Emanuele

Laureando: Bruno Marco
Matricola n.: 2113965

Padova, 21/07/2026

AZIENDA E CONTESTO INDUSTRIALE (Piazza Copernico s.r.l.)

- **Profilo Aziendale**: Leader nel settore dell'e-learning e della formazione digitale avanzata.
- **Core Business**: Integrazione di soluzioni didattiche all'interno di un unico ecosistema formativo.
- **Offerta Principale**:
  - Piattaforma **LMS** (Learning Management System) erogata in modalità **Cloud SaaS.**
  - Sviluppo di **Learning Object** personalizzati.
  - Ampio catalogo di corsi online personalizzati.

INTRODUZIONE E SCOPO DEL PROGETTO

- **Il Contesto**: La formazione digitale moderna richiede la gestione di vasti archivi multimediali e contenuti interattivi.

- **L'Obiettivo**: Progettare un'architettura backend avanzata per l'automazione dei processi di *Speech-to-Text.*

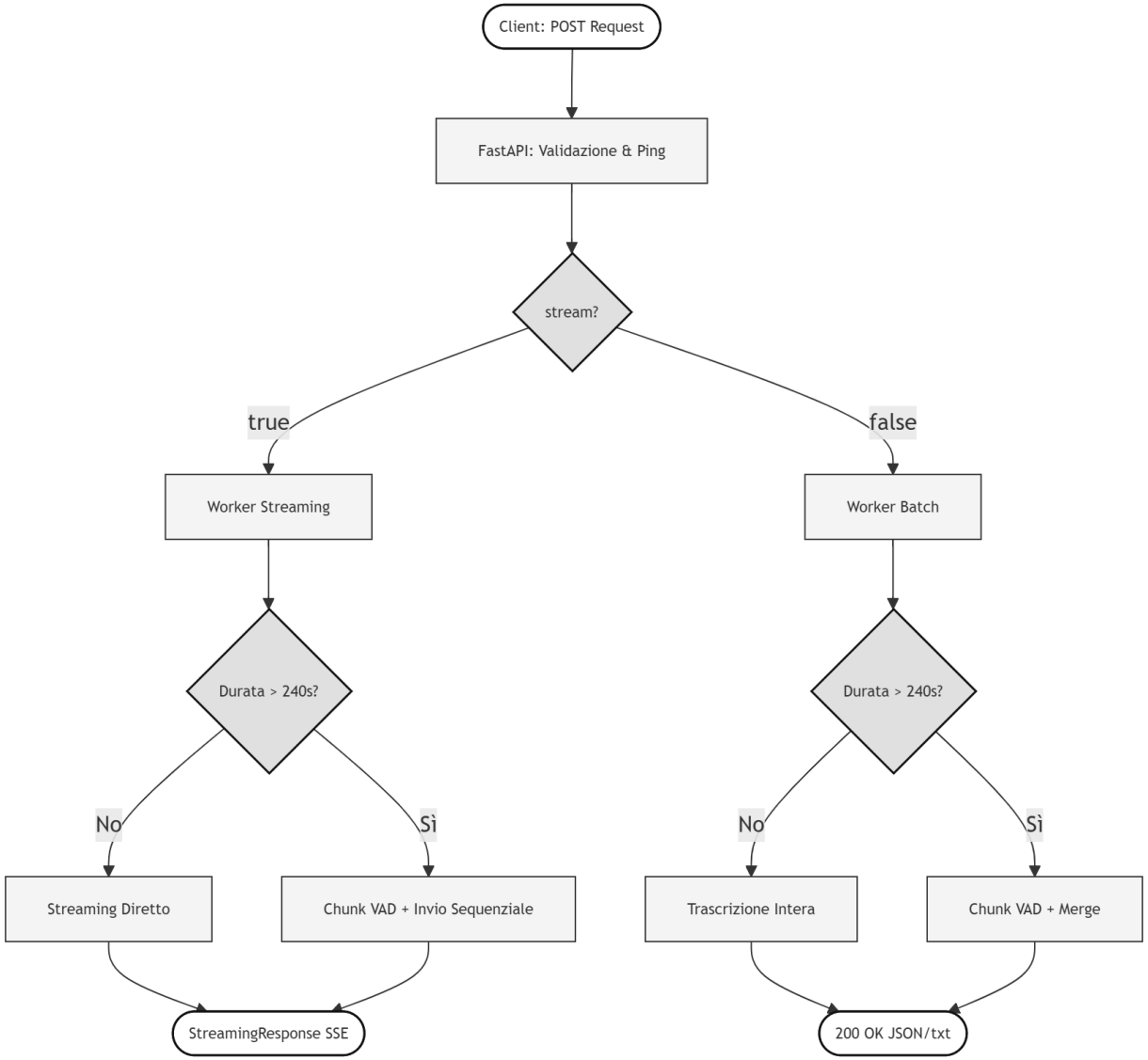
- **Il Progetto**: Sviluppo di un servizio software ibrido, capace di gestire sia l'elaborazione massiva di file (Offline) sia la restituzione progressiva del testo per contesti interattivi (Streaming).

OBBIETTIVO DEL TIROCINIO

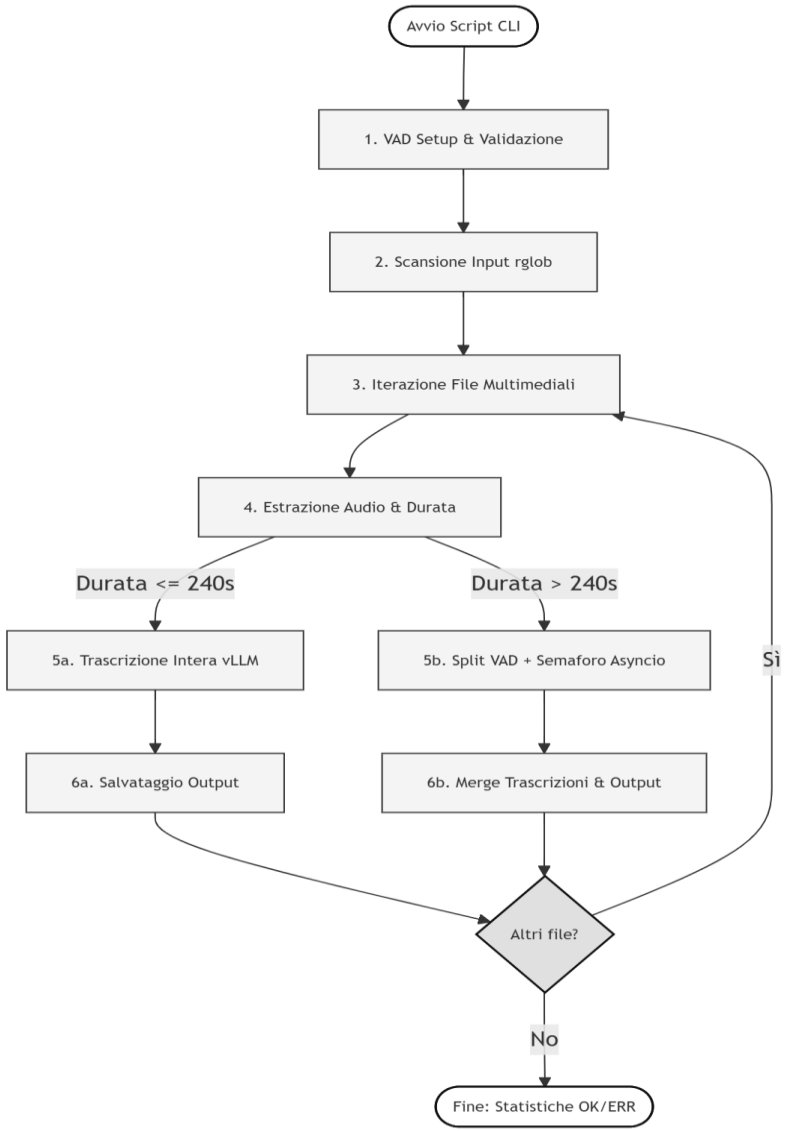
L'attività si è focalizzata sull'ingegnerizzazione di un backend *Speech-to-Text* perseguendo tre obiettivi tecnologici chiave:

- **Flessibilità Ibrida:** Implementazione di due modalità operative: *Batch Offline* e *Streaming.*
- **Standardizzazione:** Adozione dello standard API OpenAI per garantire la massima compatibilità e facilità di integrazione con i sistemi esistenti.
- **Orchestrazione**: Sviluppo di un **Proxy FastAPI** per mediare i flussi di dati, isolare il carico del motore di inferenza e gestire in sicurezza la manipolazione dei file multimediali.

ARCHITETTURA E DIAGRAMMI: FLUSSO ENDPOINT API



ARCHITETTURA E DIAGRAMMI: FLUSSO ESECUZIONE BATCH



DESCRIZIONE DEL LAVORO

Il percorso di sviluppo è stato articolato in quattro fasi:

1. **Analisi e Progettazione**: Studio approfondito delle tecnologie, definizione dei requisiti tecnici e stesura della documentazione preliminare.

2. **Gestione Vincoli Hardware:** Analisi delle criticità computazionali per ottimizzare il modello *Whisper*, bilanciando memoria, gestione delle code e logiche di *chunking*.

3. **Sviluppo e Testing:** Implementazione dei moduli core del software in parallelo a cicli di test unitari e di integrazione per la validazione del codice.

4. **Transazione ad Architettura Distribuita:** Evoluzione da prototipo locale (CLI) ad architettura distribuita, con lo sviluppo di un **Proxy FastAPI**.

STACK TECNOLOGICO E STRUMENTI

- **Linguaggio e Ambiente: Python 3** con **uv** per una gestione delle dipendenze.
- **Web Framework: FastAPI** e **Uvicorn** per la gestione asincrona (**asyncio**) dei task e dello streaming audio.
- **Inferenza e Modelli AI: vLLM** per l'esecuzione di modelli **Whisper** e **Silero VAD.**
- **Controllo Qualità Codice**: **Ruff** strumento per le attività di linting e formatazzione automatica.

STRUTTURA DELLA REPOSITORY


```
root/
├── scripts/                                  # Script operativi ed entrypoint dell'applicazione
│   ├── main_api.py                           # Entrypoint principale per l'avvio del server FastAPI
│   └── main_transcribe.py                    # Script per l'esecuzione manuale o batch delle trascrizioni
├── src/                                      # Codice sorgente principale del sistema
│   ├── api/                                  # Logica legata agli endpoint e ai servizi web
│   │   └── worker_api.py                     # Implementazione del client/worker per la gestione delle chiamate API
│   ├── __init__.py                           # Inizializzazione del package Python 'src'
│   ├── audio_normalization.py                # Logica di preprocessamento e normalizzazione dei file audio
│   ├── audio_validation.py                   # Validazione dei formati, dei metadati e dell'integrità dell'audio
│   └── offline_transcription.py              # Core per la gestione del modello di trascrizione in locale (Whisper)
├── tests/                                    # Suite di test automatizzati
│   ├── integration/                          # Test di integrazione tra i moduli e i servizi esterni
│   │   ├── mock_server.py                    # Server fittizio per simulare le risposte delle API esterne/vLLM
│   │   └── test_main_api.py                  # Test di integrazione degli endpoint dell'API principale
│   └── unit/                                 # Test unitari isolati sulle singole funzionalità
│       ├── test_audio_normalization.py       # Test per la corretta normalizzazione dei flussi audio
│       ├── test_offline_transcribe.py        # Test per i moduli di trascrizione offline
│       └── test_worker_api.py                # Test di unità sulla logica del worker API
├── .coveragerc                               # Configurazione dei criteri di analisi della code coverage
├── Makefile                                  # Comandi rapidi per automazione (build, test, linting, run)
└── pyproject.toml                            # Metadati del progetto e configurazione globale dei tool Python
```

PIPELINE DI VALIDAZIONE E OTTIMIZZAZIONE INPUT

- **Validazione dei dati (audio_validation.py):**
  - **Compatibilità Lingua**: Verifica di coerenza per il vincolo del modello Whisper.
  - **Integrità File:** Controllo rigoroso su esistenza, dimensione (> 0 byte) ed estensione dell'audio tramite una *whitelist* dedicata.
- **Isolamento degli Errori (main_transcribe.py):**
  - **Controlli Globali:** Interruzione immediata in caso di passaggio di parametri errati.
  - **Controlli Locali:** Gestione tramite *try…except* nel singolo worker. L'eventuale file corrotto viene loggato senza far crashare l'intero processo batch.

LOGICHE DI SMART CHUNKING E MERGE

Per elaborare file di lunga durata è stata sviluppata una pipeline di segmentazione e fusione:

- **Algoritmo di Smart Chunking;**
- **Algoritmo di Merge Temporale.**

ALGORITMO DI SMART CHUNKING

- **Segmentazione:** Taglio dei blocchi (target: 3600s) esclusivamente nei punti di silenzio naturale individuati dal modello **Silero VAD**.

- **Caricamento "Lazy":** Lettura dal disco dei soli frammenti necessari all'analisi, minimizzando l'uso della RAM.

- **Margine di Sicurezza:** Arretramento automatico del punto di split (soglia < 0.5s) per evitare il troncamento di parole a fine blocco.

ALGORITMO DI MERGE TEMPORALE

- Ricomposizione sequenziale e differenziata per formati *.txt, .json e verbose_json.*

- **Riallineamento dei TimeStamp**: Calcolo dinamico dell'offset temporale cumulativo e riscrittura dei parametri interni di Whisper (start,end, seek e id) per garantire continuità.

TRANSIZIONE AD ARCHITETTURA PROXY

Evoluzione del sistema da prototipo locale (CLI) a servizio Proxy Server.

- **Endpoint (main_api.py):** Sviluppo dell'endpoint /v1/audio/transcriptions per accettare flussi multi/form-data.

- **Gestione del Ciclo Vita (lifespan):** Inizializzazione centralizzata del caricamento del modello Silerio VAD, istanziazione del client HTTP e configurazione semaforo asincrono.

- **Sicurezza del File System:**
  - **Mitigazione Path Traversal;**
  - **Isolamento Multi-utente;**
  - **Pulizia Asincrona.**

PARTIAL SUCCESS E MAPPATURA DEGLI ERRORI

Progettazione modulo di orchestrazione (worker_api.py) con tolleranza ai guasti.

- **Politica Preventiva:** Check esplorativo istantaneo (timeout 3.0s) verso il server vLLM.
- **Logica di Successo Parziale**: In caso di anomalia su un singolo frammento (*Smart Chunking*), l'intero processo non fallisce.
- **Salvataggio dei Risultati Elaborati:** Il server restituisce comunque un codice HTTP 200 con un payload contenente il testo elaborato con successo e l'elenco dei soli segmenti falliti.
- **Mappatura delle Eccezioni**: Intercettazione centralizzata degli errori interni e conversione automatica in codici HTTP standard (es. 400 Bad Request per lingua errata, 415 per formato non supportato).

PIPELINE DI STREAMING

Estensione delle funzionalità in modalità streaming

- **Generatore Asincrono a Basso Livello:** Ingegnerizzazione di una soluzione nativa con *httpx* per effettuare un reverse-proxy verso vLLM, intercettando lo stream tramite costrutti *yield*.

- **Vincolo Streaming HTTP:** nel protocollo HTTP, l'invio di una *Streaming Response* consolida lo status code 200 OK. Se si verifica un errore a flusso già avviato, è impossibile modificare la risposta.

RISOLUZIONE VINCOLI HTTP

- **Strategia Ibrida in Due Fasi:**

  - **Fase 1 (Pre-stream):** Ispezione della pipeline prima dell'avvio del flusso. Restituisce codici HTTP standard per errori iniziali.

  - **Fase 2 (In-Stream):** Gestione degli errori a connessione avviata tramite blocchi try…exception interni che formattano l'anomalia direttamente in un file JSON.

OBIETTIVI RAGGIUNTI – VALIDAZIONE DEL PROXY API

Replica dei risultati dello script locale sull'architettura Proxy ed esecuzione di test intensivi di carico e conformità.

- **Collaudo dell'interfaccia REST**: Simulazione di scenari operativi tramite **Swagger UI/Docs** di FastAPI.

- **Validazione della Logica Difensiva:** Intercettazione corretta e tempestiva di tutti gli scenari d'errore (file vuoti, estensioni non supportate, backend offline).

- **Conformità HTTP:** Instradamento dei codici di stato standard (es. 400, 415, 503) verso il client, azzerando gli sprechi di risorse sulla GPU.

CONCLUSIONI E SVILUPPI FUTURI

Conclusioni: Il progetto ha dimostrato come l'integrazione dei principi dell'Ingegneria del Software e modelli di Intelligenza Artificiale (AI) consenta di industrializzare servizi complessi.

- **Sviluppi Futuri:**
  - **Scalabilità e Load Balancing:** Evoluzione verso l'architettura su cluster di GPU distribuiti per gestire in modo efficiente flussi massivi di utenti concorrenti.
  - **Traduzione Multilingua Estesa:** Integrazione di modelli di traduzione dedicati per svincolare l'output dall'obbligo della sola lingua inglese.
  - **Fine-Tuning per i Dialetti:** Addestramento mirato del modello con dataset locali e regionali per migliorare l'accuratezza del riconoscimento vocale.