ChatGPT

Windows Ortamında Offline Video Dublaj Uygulaması: Bileşenler ve Uygulama Detayları

Giriş ve Genel İş Akışı

Tamamen **offline** çalışan bir video dublaj uygulaması geliştirmek için ihtiyaç duyulan bileşenler ve teknolojiler, birbiriyle entegre bir **pipeline** oluşturur. Bu pipeline özetle şu adımlardan oluşur:

1. **Ses Ayrıştırma:** Yüklenen video veya ses dosyasından ham ses akışı çıkarılır (FFmpeg ile).
2. **Konuşmadan Metne Çeviri (STT):** Elde edilen ses, konuşma tanıma motoru ile yazılı metne dönüştürülür. Tercihen model, konuşulan dili otomatik algılar.
3. **Metin Çevirisi (MT):** Transkript edilen metin, hedef dile çevrilir. Bu işlem offline çalışan çok dilli bir çeviri motoruyla yapılır.
4. **Metinden Konuşmaya (TTS):** Çevrilen metin, hedef dilde bir ses sentezleyici ile ses dosyasına dönüştürülür (dublaj sesi).
5. **Çıktı Oluşturma:** Ortaya çıkan hedef dildeki ses, tercihen orijinal video ile birleştirilir veya bağımsız bir ses dosyası (`.mp3`) olarak kullanıcıya sunulur.

Yukarıdaki adımlar tamamen kullanıcının kendi bilgisayarında, internet bağlantısı olmadan gerçekleşir. Bu sayede gizlilik korunur ve çevrimdışı ortamlarda dahi çok dilli dublaj mümkündür [1] [2]. Aşağıda, bu adımlar için gerekli açık kaynak araçlar, kurulumları, entegrasyon ipuçları ve optimizasyon teknikleri detaylandırılmaktadır.

1. Konuşmayı Metne Çevirme Teknolojileri (Speech-to-Text)

Konuşmadan metne (STT) dönüşümde en güncel ve başarılı açık kaynak çözüm OpenAI'nin **Whisper** modelidir. Whisper, 680.000 saatlik çok dilli veriye eğitilmiş bir sinir ağıdır ve çeşitli dillerde yüksek doğrulukla metin transkribe edebilir [3]. Toplam **99 dili** desteklemesiyle oldukça esnek bir modeldir [4]. Whisper'ın eğitimi ağırlıklı İngilizce veriye dayansa da (%65 İngilizce, %17 diğer diller) pek çok dilde robust sonuçlar verebilmektedir [4]. Ayrıca model, gürültülü ortamlara ve aksanlara karşı dayanıklı olacak şekilde geniş bir veriyle eğitilmiştir.

Whisper'ın önemli bir özelliği, **dil algılama** ve **çeviri** yeteneğinin modele entegre olmasıdır. Model, bir ses dosyasını işlerken otomatik olarak dilini tespit edebilir ve istersek doğrudan İngilizce'ye çeviri modunda da çalışabilir [5]. Bu, çok dilli bir dublaj uygulamasında kaynak dilin ne olduğunu tespit etmeyi kolaylaştırır.

Whisper.cpp: OpenAI Whisper modeli, varsayılan halinde Python (PyTorch) ile çalışsa da, **whisper.cpp** projesi sayesinde modele C++ ile çok daha hafif ve hızlı şekilde, GPU gerektirmeden CPU'da erişmek mümkündür. Whisper.cpp, **hiçbir harici bağımlılığı olmadan** tamamen C/C++ ile yazılmış bir implementasyondur ve Apple silikon (ARM) ile x86 işlemciler için optimize edilmiştir [6] [7]. Bu sayede düşük bellek kullanımıyla CPU üzerinde real-time'a yakın hızlarda transkripsiyon yapabilir. Whisper.cpp aynı zamanda 16-bit ve 32-bit karışık hassasiyet kullanarak (f16/f32) performansı artırır [8]. Python uygulamanıza whisper.cpp'yi entegre etmenin bir yolu, mevcut **Python bağlayıcılarını** kullanmaktır.

Örneğin, PyPI üzerindeki `pywhispercpp` paketi bu amaçla geliştirilmiştir ve whisper.cpp'nin tüm özelliklerini Python üzerinden kullanmaya imkan tanır [9].

Whisper.cpp ile kullanmak için öncelikle uygun boyutta bir model dosyasını (ggml formatında, örn. `base` veya `small` modeli) indirip uygulamaya dahil etmek gerekir. Ardından basitçe Python'da aşağıdaki gibi kullanılabilir:

```
import pywhispercpp
# Örnek: Küçük boyutlu modeli yükleyelim (base modeli, 95 MB civarında)
model = pywhispercpp.Whisper.from_pretrained("base")
result = model.transcribe("ornek_kayit.wav")
metin = result["text"]
kaynak_dil = result["language"]
print(f"Transkript: {metin}\nDil: {kaynak_dil}")
```

Yukarıdaki kod, `ornek_kayit.wav` ses dosyasını işleyip içinde geçen konuşmayı metin olarak döndürür. `result["language"]` Whisper'ın tahmin ettiği dil kodudur (ör. `"en"` veya `"tr"`). Not: `pywhispercpp` yerine, OpenAI'nin orijinal `whisper` kütüphanesi de `pip install openai-whisper` ile kurulup benzer şekilde kullanılabilir; ancak bu durumda PyTorch ve model ağırlıklarının kurulumu gerektiği için daha ağırdır.

Alternatif STT Motorları: Whisper güncel olarak en iyi doğruluk ve dil desteğini sunsa da, daha hafif veya farklı lisanslı alternatifler de vardır:

- **Vosk:** Kaldi tabanlı bir offline STT kütüphanesidir. 20'den fazla dili destekler (İngilizce, Türkçe, Çince, Rusça vb.) ve düşük kaynak tüketimiyle çalışabilir [10]. Her bir dil modeli yaklaşık 50 MB boyutundadır, Raspberry Pi gibi cihazlarda bile çalışabilir [11]. Python'dan `vosk` paketiyle kullanılabilir.
- **Mozilla DeepSpeech / Coqui STT:** Mozilla'nın açık kaynak STT modeli DeepSpeech, Coqui STT adıyla geliştirilmeye devam etmektedir. Bu modeller bazı dillerde iyi sonuç verir ancak Whisper kadar geniş dil desteği veya doğruluk sunmayabilir.
- **Google'ın NeMo, Wav2Vec2 vb.:** Akademik ve açık kaynak camiada ortaya çıkan Wav2Vec 2.0 tabanlı modeller veya NVIDIA NeMo platformu da STT için kullanılabilir. Fakat bunlar genellikle tek bir dilde uzmanlaşmıştır veya kurulumları daha zordur.
- **PocketSphinx:** Carnegie Mellon University'nin Sphinx projesinin devamı olan PocketSphinx, eski bir teknoloji olsa da offline ve hızlıdır. Ancak doğruluk açısından modern derin öğrenme tabanlı modellerin gerisindedir.

Dil Algılama: Eğer Whisper kullanılmazsa, transkript edilen metnin hangi dilde olduğunu belirlemek için ayrıca bir dil tanıma aracına ihtiyaç duyulabilir. Örneğin, `langdetect` veya `fastText` tabanlı dil tespiti kütüphaneleri metin üzerinden dili tanıyabilir. Whisper kullanıldığında ise bu ekstra adıma gerek kalmadan modelden dil çıktısı alınabilir.

STT İçin Optimizasyonlar

- **Model Boyutu Seçimi:** Whisper modeli farklı boyutlarda mevcuttur (tiny, base, small, medium, large). Küçük modeller daha hızlı ve hafif çalışırken doğrulukları büyük modellere göre bir miktar düşüktür. Uygulamanızın ihtiyacına göre model seçimi yapabilirsiniz. Örneğin, gerçek zamanlıya yakın hız için `tiny` (39 MB) veya `base` (95 MB) modeli yeterli olabilir. En yüksek doğruluk için `large` model (~1.5 GB) gerekebilir.

- **Quantization (Nicemleme):** Whisper.cpp, modelleri 8-bit gibi daha düşük çözünürlükte çalıştırma (quantize etme) imkânı sunar. Bu, hız ve bellek kullanımını iyileştirir (fakat doğruluğu biraz düşürebilir). Örneğin, ggml formatlı modeli `quantize` aracıyla int8'e dönüştürmek %50'ye yakın RAM tasarrufu sağlar.
- **Çoklu İş Parçacığı:** Whisper.cpp, CPU çekirdeklerini verimli kullanmak için çoklu iş parçacığını destekler. `n_threads` parametresi ile işlemci çekirdek sayınıza uygun bir değer vererek transkripsiyon süresini kısaltabilirsiniz.
- **Ses Ön-işleme:** STT'ye beslemeden önce sesin 16 kHz, mono kanal PCM wav formatında olması önerilir. FFmpeg ile ses dönüştürme adımında sample rate ve kanallar zaten ayarlanacağından (bakınız FFmpeg bölümü), Whisper'ın beklediği formata uygun veri girişi sağlanacaktır. Giriş sesinde gürültü varsa, basit bir filtre uygulamak (ör. sox veya librosa ile) tanıma doğruluğunu artırabilir.

2. Offline Çok Dilli Çeviri Motorları

STT adımından elde edilen metin, istenen hedef dile çevrilmelidir. Bulut tabanlı API'ler (Google Translate, Azure vs.) kullanmadan **offline** çeviri yapabilen açık kaynak çözümler sınırlıdır ancak mevcut bazı projeler oldukça başarılıdır.

Argos Translate: Öne çıkan seçeneklerden biri **Argos Translate** kütüphanesidir. Argos, Python ile yazılmış açık kaynak bir çeviri kütüphanesidir ve **OpenNMT** çerçevesini kullanarak sinirsel makine çevirisi gerçekleştirir [12]. **SentencePiece** tokenizasyonu ve **Stanza** cümle dilimleme araçlarını entegre eder, böylece metinleri uygun biçimde parçalar [12]. En önemli özelliği ise, internet gerektirmeden **nöral çeviri modellerini lokal olarak çalıştırmasıdır**. Desteklediği her dil çifti için bir model paketi (`.argosmodel` uzantılı) indirilebilir ve Argos ile yüklenebilir [13]. Örneğin İngilizce-Türkçe veya İngilizce-İspanyolca gibi popüler dil çiftleri için hazır eğitimli modeller bulunmaktadır.

Argos Translate, birçok dil arasında dolaylı çeviriyi de destekler. Yani yüklediğiniz model çiftleri doğrudan gerekli olmasa bile, **pivot dili** kullanarak çevrim yapabilir. Örneğin elimizde sadece İngilizce→Türkçe ve Türkçe→Almanca modelleri varsa, Argos bunları zincirleyerek İngilizce bir metni önce Türkçeye, ardından Türkçeden Almancaya çevirip **İngilizce→Almanca** sonucu verebilir [14]. Bu sayede desteklenen model sayısıyla sınırlı kalmayıp çok daha fazla dil kombinasyonunu offline olarak gerçekleştirebilir (bazı kalite kayıpları pahasına) [14]. Argos'un kullanımına dair önemli noktalar:

- **Kurulum:** `pip install argostranslate` ile kütüphane kurulabilir. (Not: Argos Translate, TensorFlow/CTranslate2 tabanlı bir backend kullandığından, ilk kurulumda birkaç bağımlılık indirebilir).
- **Model Yükleme:** Argos modellerini programatik olarak indirebilirsiniz. Örneğin, İngilizce'den Türkçeye çeviri modeli için:

```
import argostranslate.package, argostranslate.translate
argostranslate.package.update_package_index()  # model listesi güncelle
packages = argostranslate.package.get_available_packages()
# İngilizce (en) -> Türkçe (tr) modelini bul
pkg = list(filter(lambda x: x.from_code == "en" and x.to_code == "tr",
packages))[0]
download_path = pkg.download()
argostranslate.package.install_from_path(download_path)
```

Yukarıdaki kod, ilgili modeli internetten indirip Argos'a yükler [15] [16]. Offline ortam için, `.argosmodel` dosyasını önceden temin edip `install_from_path` ile yüklemek de mümkün.

- **Çeviri Kullanımı:** Gerekli model yüklendikten sonra, Argos ile çeviri yapmak oldukça basittir:

```
# Yüklü diller listelenir
langs = argostranslate.translate.get_installed_languages()
src_lang = list(filter(lambda x: x.code == "en", langs))[0]
tgt_lang = list(filter(lambda x: x.code == "tr", langs))[0]
translation = src_lang.get_translation(tgt_lang)
sonuc = translation.translate("Hello, how are you?")
print(sonuc)  # "Merhaba, nasılsın?"
```

Bu örnekte İngilizce bir cümle Türkçeye çevrilmiştir.

Argos Translate, hızlı ve kurulumu görece kolay bir çözümdür. Karşılaştırmalı testlerde **Marian NMT** gibi alternatiflere göre daha düşük kaynakla daha yüksek hız sunar, ancak çeviri kalitesi bazı durumlarda bir nebze geride kalabilir [17]. **Marian** (Helsinki NLP'nin OPUS-MT modelleri) ise yüzlerce dil çifti için hazır modeller sunar ve daha fazla dil desteği/kalite sağlayabilir [17]. Marian kullanımı için Hugging Face Transformers kütüphanesinden ilgili modeli indirmek yeterlidir. Örneğin:

```
from transformers import MarianMTModel, MarianTokenizer
model_name = "Helsinki-NLP/opus-mt-en-tr"  # İngilizce->Türkçe Marian modeli
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)
text = "Hello, how are you?"
translated = model.generate(**tokenizer(text, return_tensors="pt",
padding=True))
turkish_text = tokenizer.decode(translated[0], skip_special_tokens=True)
```

Bu kod, benzer şekilde İngilizce metni Türkçeye çevirecektir. Marian modelleri PyTorch tabanlı olduğundan Argos'a göre daha fazla bellek kullanabilir ve GPU yoksa yavaş çalışabilir. Dolayısıyla **Argos vs. Marian** seçiminde bir **hız-kalite** dengesi söz konusudur: *"Argos Translate kurulumu daha kolay ve hızlıdır; Marian ise daha fazla dil ve bazen daha iyi çeviri kalitesi sunar"* [17].

Diğer Alternatifler:

- **NLLB-200 (No Language Left Behind):** Meta AI tarafından yayınlanan devasa bir modeldir. *200 farklı dil* arasında, herhangi bir dil çiftinde çeviri yapabilir [18]. Ancak NLLB-200 tam modeli 54 milyar parametreli dev bir modeldir ve pratikte çalıştırmak için güçlü GPU'lar gerektirir [18]. Daha küçük alt modelleri veya distilled versiyonları da mevcuttur (3.3B parametre gibi), fakat yine de sıradan bir PC için ağırdır. İhtiyaç halinde, Hugging Face üzerinden `facebook/nllb-200-distilled-600M` gibi bir model indirip Transformers ile kullanmak mümkün olsa da, bu proje için NLLB muhtemelen fazla büyük kalacaktır.
- **M2M-100:** Facebook'un başka bir çoklu dil modeli olan M2M100, tek bir modelle 100'den fazla dilde çeviri yapabilir. Örneğin 418M parametrelik sürümü, birçok dil arasında kabul edilebilir çeviriler sunar. Bu da Transformers aracılığıyla kullanılabilir (`facebook/m2m100_418M`).
- **Bergamot (Mozilla):** Firefox tarayıcısına entegre edilen Bergamot projesi, istemci tarafında (offline) çeviri sağlamayı amaçlıyordu. Temelde marian-nmt modellerini WebAssembly formatına

çevirerek çalışır. Python ortamında doğrudan kullanımı yoktur ancak konsept olarak Argos ile benzerdir.

Çeviri Optimizasyonları:

- **Model Boyutu ve Dili:** Sadece gerekli dil çiftlerinin modellerini dahil edin. Her ek model dosyası boyut ve bellek açısından yük getirir. Örneğin Argos İngilizce-Türkçe modeli ~50 MB ise, 10 dil desteği için toplam birkaç yüz MB veriyi göze almak gerekir.
- **Pivot/Gerçek Çeviri:** Mümkün olan durumlarda doğrudan model kullanın. Örneğin Argos'ta İngilizce → Almanca için doğrudan model yoksa, İngilizce → Fransızca → Almanca gibi dolaylı yollara başvurmak kaliteyi düşürür [14]. En sık gereken dil çiftleri için modelleri önceden yüklemek iyi bir stratejidir.
- **CTranslate2 Kullanımı:** Argos Translate, kaputun altında CTranslate2 kullanır; bu kütüphane CPU üzerinde int8 quantization ve multi-threading destekler. Argos ile gelen modeller zaten optimize edilmiştir. Marian modellerini de CTranslate2 formatına çevirip (çevrim araçları mevcut) daha performanslı çalıştırmak mümkündür.
- **Batch Çeviri:** Bir seferde çok miktarda metin çevirilecekse (örneğin uzun bir transkript), bunları cümle bazında çevirmek yerine mini bloklar halinde batch olarak modele vermek daha hızlı olabilir. Marian NMT bunu destekler; Argos arka planda kendisi bölüp birleştirebilir.

3. Offline Çok Dilli Metinden Konuşmaya (TTS) Motorları

Pipeline'ın son aşaması, çevrilen metnin hedef dilde seslendirilmesidir. **Text-to-Speech (TTS)** motorları, yazıyı mümkün olduğunca doğal bir konuşmaya çevirmeyi amaçlar. Offline bir dublaj uygulamasında bulut tabanlı TTS (Google, Amazon Polly vb.) kullanılamayacağı için, yerelde çalışabilen çözümlere odaklanmalıyız.

Windows SAPI ve Sistem Sesleri: Windows işletim sisteminde, Microsoft'un SAPI (Speech API) altyapısı üzerinden çeşitli dillerde konuşma yapabilen sesler bulunur. Windows 10+ sürümlerinde pek çok dil için ücretsiz sistem sesleri indirilebilir. Bu sesleri kullanmanın bir yolu, **Balabolka** gibi bir arayüz program veya doğrudan SAPI kullanmaktır. Balabolka, sistemde yüklü tüm sesleri görebilen ve metni WAV/MP3 gibi formatlarda kaydedebilen ücretsiz bir yazılımdır [19]. Nitekim Balabolka'nın kendisi bir GUI olsa da, yanında **"balcon"** adlı bir komut satırı aracı sunar. Balcon aracı ile bir metni seslendirmek veya ses dosyasına kaydetmek komutlarla mümkündür [20] [21]. Örneğin, sistemde Mary adında bir İngilizce sesi kullanarak metni MP3'e kaydetmek için şu komut verilebilir:

```
balcon -n "Mary" -f input.txt -w output.wav
lame output.wav output.mp3  # isteğe bağlı MP3 dönüşümü için LAME kullanımı
```

Yukarıda `-n "Mary"` ilgili sesin adını (veya dil kimliğini) seçer, `-f input.txt` okunacak metin dosyasını belirtir, `-w output.wav` ise çıktı ses dosyasının ismini belirler [21]. Balabolka, SAPI4, SAPI5 ve Microsoft Speech Platform seslerini destekler [22] [23]. Tüm SAPI uyumlu sesleri listelemek için `balcon -l` komutu da kullanılabilir [24].

Balabolka aracını Python içinden `subprocess` ile çağırarak entegre edebilirsiniz. Fakat daha **Pythonik bir yaklaşım**, doğrudan SAPI'yi veya benzer API'leri Python'dan kullanmaktır. Bu amaçla yaygın olarak `pyttsx3` kütüphanesi tercih edilir. `pyttsx3`, platform bağımlı olarak en iyi motoru seçen offline bir TTS kütüphanesidir. Windows'ta SAPI5'i, Linux'ta eSpeak'i kullanır. "`pyttsx3` *Python kütüphanesi internetsiz çalışır ve İngilizce, Fransızca, Almanca, Hintçe gibi birden çok sesi/dili destekler*" [25]. Kullanımı oldukça kolaydır:

```
import pyttsx3
engine = pyttsx3.init()
voices = engine.getProperty('voices')
# Örneğin Türkçe bir ses varsa onu seçelim:
for v in voices:
    if "Turkish" in v.name or v.languages.count('tr_TR'):
        engine.setProperty('voice', v.id); break
engine.setProperty('rate', 180)  # konuşma hızını ayarla (isteğe bağlı)
engine.save_to_file("Merhaba dünya", "output.mp3")  # metni seslendirip mp3
kaydet
engine.runAndWait()
```

Yukarıda, mevcut seslerden Türkçe olan bulunup seçiliyor, konuşma hızı ayarlanıyor ve `save_to_file` ile doğrudan bir MP3 dosyasına yazdırılıyor. (Not: `save_to_file` bazı sistemlerde WAV olarak kaydedip LAME ile MP3'e çevirmeyi gerektirebilir; pyttsx3 bunu arka planda halledebilir veya sadece WAV üretip ayrık dönüşüm yapılabilir.)

`pyttsx3` ile SAPI seslerini kullanmak, projenin Windows'a özgü kalmasını sağlar ancak Windows'un sunduğu tüm dillere erişim verir. Microsoft'un desteklediği diller arasında Türkçe, İngilizce çeşitleri, Almanca, Fransızca, İspanyolca, Rusça, Çince ve daha birçok dil bulunur. Kullanıcı, işletim sistemine yeni bir dil ses paketi ekleyerek uygulamanın o dilde de konuşma yapmasını sağlayabilir.

Açık Kaynak TTS Motorları: Sistem seslerine bağımlı kalmadan, tamamen uygulamayla beraber gelen TTS motorları da değerlendirilebilir. Ancak burada dikkat edilmesi gereken husus, ses kalitesi ile dosya boyutu arasında ciddi bir trade-off olduğudur:

- **eSpeak NG:** Çok hafif ve hızlı bir çözümdür, 100'den fazla dil ve aksanı destekler [26] [27]. Formant sentezi kullandığı için konuşma kalitesi oldukça robotik ve tekdüzedir, fakat net anlaşılır. Özellikle boyut kısıtlı projelerde veya erişilebilirlik amaçlı kullanılır [28]. eSpeak'i doğrudan komut satırı ile çağırabilir veya Python'da `subprocess` ile kullanabilirsiniz (`espeak` komutu metni seslendirebilir, `-w output.wav` ile WAV çıktısı verebilir). Windows için dll olarak da kullanımı vardır.
- **Pico TTS:** Android'de kullanılan Pico TTS (SVOX) motorunun açık sürümü, birkaç dili destekleyen ufak bir motordur. Kalitesi eSpeak'ten biraz iyidir ama diller sınırlıdır.
- **Festival / Flite:** Festival, C++ tabanlı klasik bir TTS motoru; İngilizce başta olmak üzere birkaç dilde fena olmayan sesler sunar. Flite (Festival-lite) ise onun hafifleştirilmiş sürümü.
- **MaryTTS:** Java ile geliştirilmiş bir TTS sistemidir. Oldukça esnek ve özelleştirilebilir, çeşitli dillere (İngilizce, Almanca, Fransızca, İtalyanca, Rusça, Türkçe vs. için bazı sesler mevcut) destek verir [29]. MaryTTS ile kendi sesinizi eğitmeniz bile mümkün. Ancak Java ortamı gerektirdiği ve entegre etmesi karmaşık olduğu için, bu projede ek bir servis olarak çalıştırılıp REST API ile kullanılması düşünülmediği sürece ağır kalabilir.
- **Coqui TTS (Mozilla TTS):** Son yıllarda derin öğrenme tabanlı TTS modelleri de açık kaynak olarak yayınlanıyor. Coqui TTS, Tacotron2, FastSpeech gibi mimarilerle yüksek kaliteli konuşma sentezleyebilir [30]. Hatta tek bir modelin birden fazla dilde konuşması veya ses klonlaması (transfer learning) bile mümkün [30]. Coqui'nin hazır modelleri arasında çok dilli veya tek dilliler bulunuyor. Örneğin Tacotron2 tabanlı bir Türkçe modeli veya multi-lingual bir model HuggingFace'den indirilebilir. Ancak bu yaklaşımla her dil için yüzlerce MB boyutunda model dosyaları ekleneceği ve çalıştırmak için güçlü CPU/GPU gerekeceği unutulmamalıdır [31].
- **Silero ve Diğerleri:** Silero modeli (Snakers4 tarafından geliştirilen) belirli diller için oldukça doğal sesler üreten, optimize edilmiş bir TTS'tir. Python'da `silero` paketini veya doğrudan onnx

modellerini kullanabilirsiniz. Örneğin İngilizce, Rusça, İspanyolca, Deutsch gibi dilleri var. Her biri ~30-50 MB civarı bir onnx modeliyle çalışır ve CPU'da gerçek zamana yakın sentez yapabilir.

Özetle, eğer **geniş dil desteği ve yeterli kalite** isteniyorsa, Windows'un kendi TTS seslerini kullanmak en pratik yoldur. eSpeak gibi bir motor yedek olarak dahil edilebilir (örneğin desteklenmeyen bir dil için son çare). Yüksek kaliteye ihtiyaç duyulan özel dil(ler) için ise opsiyonel olarak Coqui veya Silero gibi modele dayalı çözümler entegre edilebilir.

TTS Optimizasyonları ve İpuçları:

- **Ses Kalitesi vs. Boyut:** Her dil için yüksek kaliteli bir neural TTS modeli dahil etmek proje boyutunu hızla artırır. Bu nedenle, uygulamanın asıl hedefi olan diller belirlenmeli. Örneğin sadece İngilizce→Türkçe dublaj için Türkçe kaliteli bir sesi (eğer Windows'ta mevcut değilse) dahil etmek mantıklı olabilir. Diğer 80+ dil için ise kullanıcıdan o dilin sistem sesini kurmasını istemek veya eSpeak gibi bir düşük kaliteli seçeneği kullanmak gerekebilir.
- **Konuşma Hızı ve Tonlama:** Özellikle eSpeak veya sistem seslerinde, konuşma hızını (`rate`) ve sesin perdesini (`pitch`) ayarlamak önemli olabilir. Farklı dillerde doğru telaffuz için dil kodu ayarlamak da gerekir (pyttsx3 genelde bunu seçilen sesle otomatik halleder).
- **Cümle Dilimleri:** Uzun metinleri tek seferde TTS'e vermek bellek kullanımını artırabilir. Bunun yerine transkript metnini cümle cümle veya paragraflara bölüp sırayla seslendirmek gerekebilir. Bu aynı zamanda daha doğal bir duraklama yapısı sağlar.
- **Ses Dosyası Formatı:** Çıktı sesi genellikle MP3 isteniyor (uygulama gereksiniminde .mp3 belirtilmiş [32]). Pyttsx3 WAV üretirse, LAME ile MP3'e çevirme yapılabilir. Veya doğrudan MP3 desteği olan bir yol (Balabolka doğrudan MP3 kaydedebilir [19]) kullanılabilir. FFmpeg de WAV'ı MP3'e çevirmede kullanılabilir (`ffmpeg -i output.wav output.mp3`).
- **Dublaj Senkronizasyonu:** Gelişmiş senaryolarda, üretilen konuşma orijinal videodaki konuşmanın hızına/uzunluğuna uymayabilir. Basit MVP'de bu önemsenmeyebilir (tüm konuşma içeriği yeni seste de vardır, videoyu belki pause/play ile idare edebilir kullanıcı). Fakat gelecekte senkronizasyon istenirse, cümle zaman damgaları (Whisper bunları üretebilir) ile TTS sentezini o süreye yaymak, ya da FFmpeg ile zamanı uzatıp kısaltmak (time-stretch) gibi çözümler gerekebilir. Bu ileri seviye bir optimizasyon konusudur.

4. FFmpeg ile Ses Ayrıştırma ve İşleme

FFmpeg, video ve ses işlemede neredeyse her projede başvurulan, çok güçlü bir komut satırı aracıdır. Bu uygulamada FFmpeg'in iki temel rolü olacaktır:

1. **Videodan Ses Ayrıştırma:** Kullanıcı bir video dosyası (.mp4 gibi) yüklediğinde, önce bunun içindeki ses akışını çıkarmamız gerekir. FFmpeg ile yeniden kodlama yapmadan ses çıkarmak mümkündür. Örneğin:

   ```
   ffmpeg -i giris.mp4 -vn -acodec copy cikti_audio.aac
   ```

   Bu komut, videodaki ses akışını orijinal formatıyla (ör. AAC ise AAC olarak) `cikti_audio.aac` dosyasına yazar. Burada `-vn` video içeriğini devre dışı bırakır, `-acodec copy` ise ses akışını yeniden kodlamadan kopyalar [33]. Ancak STT motorumuzun beklentisine uygun bir format almak genellikle daha iyidir. Whisper gibi modeller çoğunlukla **PCM WAV (16-bit, 16kHz, mono)** formatında girdi bekler. Bu nedenle, pratikte FFmpeg ile çıkarırken dönüştürme de yapabiliriz:

```
ffmpeg -i giris.mp4 -vn -ar 16000 -ac 1 -c:a pcm_s16le cikti.wav
```

Bu komut, videodan sesi çıkarırken 16 kHz örnekleme oranına, tek kanala indirger ve PCM 16-bit WAV olarak kaydeder. Bu dosya artık STT için idealdir.

2. **Ses Dosyalarını Birleştirme / Kodlama:** Pipeline sonunda elde edilen dublaj sesini (`.wav` veya `.mp3`) kullanıcının istediği formatta sunmak gerekebilir. MVP hedeflerinde çıktının .mp3 olacağı belirtilmiştir [32], bu yüzden TTS adımında direkt mp3 elde edilemezse FFmpeg ile WAV'ı mp3'e çevirebiliriz:

```
ffmpeg -i dublaj.wav -c:a libmp3lame -q:a 2 dublaj.mp3
```

Yukarıda `-q:a 2` yüksek kaliteli (yaklaşık 190 kbps) mp3 oluşturur [34]. Alternatif olarak bitrateleri de sabit ayarlayabilirsiniz (ör. `-b:a 192k`).

Ayrıca videoya entegre etmek istenirse, FFmpeg kullanarak orijinal videonun görüntü akışı ile yeni ses akışını birleştirmek mümkündür. Örneğin:

```
ffmpeg -i giris.mp4 -i dublaj.mp3 -c:v copy -map 0:v:0 -map 1:a:0 cikti.mp4
```

Bu komut, `giris.mp4` içindeki video akışını ve `dublaj.mp3` içindeki ses akışını alıp `cikti.mp4` içinde birleştirir. `-c:v copy` ile video kısmı yeniden kodlanmaz, sadece kopyalanır. (Not: MP4 konteyneri içinde MP3 ses resmî olarak desteklenmeyebilir; bu durumda `-c:a aac` ile mp3'ü AAC olarak kodlamak daha uyumlu olur). Eğer orijinal videoda ses de varsa, `-map 0:v:0` orijinal videonun ilk video akışını, `-map 1:a:0` ise yeni ses dosyasının ilk ses akışını seçer; böylece eski sesi dışarıda bırakıp yenisini eklemiş oluruz [35] [36]. Bu yöntemle kalite kaybı olmadan videoya yeni ses eklenebilir.

MVP sürümünde belki video montajı yapılmayıp sadece mp3 çıktı verileceği düşünülmüş [32]. Ancak gelecekte "video dublaj" tam anlamıyla istenirse, orijinal videonun arka plan müziğini koruyup sadece konuşmayı değiştirmek gibi ihtiyaçlar doğacaktır. O durumda daha ileri ses işlemleri gerekebilir (özellikle orijinal sesten vokali izole edip, altyapı müziğini yeni sesle mikslemek gibi). Bu tür işlemler FFmpeg'in ötesinde, ses ayırma kütüphaneleri (spleeter gibi) gerektirir. MVP kapsamında bunlar yoktur.

FFmpeg Entegrasyon İpuçları:

- FFmpeg'i kullanmak için sistemde yüklü olması gerekir. Uygulamayı PyInstaller ile paketlerken FFmpeg binary'sini de pakete dahil etmeyi planlayabilirsiniz. Alternatif olarak, kullanıcıya FFmpeg yüklemesini söyleyip PATH'de olması koşulunu kontrol edebilirsiniz. En kolayı, FFmpeg'in statik derlenmiş `.exe` dosyasını proje dizinine koyup subprocess ile onu çağırmaktır.
- Python'dan FFmpeg çağırmak için `subprocess.run([...])` kullanabileceğiniz gibi, `ffmpeg-python` gibi bir wrapper kütüphane de kullanabilirsiniz. Örneğin:

```
import ffmpeg
(
    ffmpeg
    .input('giris.mp4')
```

```
    .output('cikti.wav', **{'vn': None, 'ar': 16000, 'ac': 1, 'c:a':
'pcm_s16le'})
    .run()
)
```

şeklinde ffmpeg komutunu Python koduyla ifade etmek mümkün. Ancak `ffmpeg-python` de arka planda aynı ffmpeg binary'sine ihtiyaç duyacaktır.

- **Hata Yönetimi:** FFmpeg komutları yürürken hata olursa (`subprocess.CalledProcessError` fırlatabilir), bunları yakalayıp kullanıcıya anlaşılır mesaj vermek önemli. Örneğin, "Geçersiz dosya formatı" gibi bir durumda kullanıcıyı bilgilendirmek gerekir.
- **Performans:** Birkaç dakikalık videolarda ses ayrıştırma ve birleştirme işlemleri genelde çok hızlı (saniyeler mertebesinde) gerçekleşir. Yine de bu işlemlerin de ayrı thread/proseslerde yapılması, GUI'yi dondurmamak açısından gereklidir (aşağıda GUI bölümünde ele alınmıştır).
- **Ses Düzeyi ve Kalite:** FFmpeg ile sesin ses seviyesini normalize etmek istenirse `-af loudnorm` gibi filtreler kullanılabilir. Ayrıca çıktı kalitesi için mp3 yerine WAV veya FLAC vermek teknik olarak daha iyi olsa da, dosya boyutu büyük olacağı için mp3 mantıklı tercihtir.

5. Python GUI ile Entegrasyon (Tkinter vs. Alternatifleri)

Uygulamanın masaüstü arayüzü, kullanıcı deneyimi açısından basit ve anlaşılır olmalıdır. **Python'ın standart GUI kütüphanesi Tkinter**, bu iş için yeterlidir ve kullanımı kolaydır. Tkinter, Python ile birlikte gelen (Windows ve macOS'te kurulumda hazır, Linux'ta paket ile eklenebilen) standart GUI aracıdır [37]. Ek bir bağımlılık gerektirmediğinden, uygulamayı .exe yaparken de sorun çıkarmaz. *"Tkinter Python'un varsayılan GUI aracıdır ve öğrenmesi/başlaması en kolay seçenektir"* [38].

Arayüz Tasarımı: MVP için planlanan arayüz oldukça sade: Bir dosya seçme alanı, dil seçimi ve bir "Dublaj Başlat" düğmesi yeterli olacaktır [39] [40]. Örneğin şöyle bir akış olabilir: - Kullanıcı `.mp4/.mp3/.wav` dosyasını bir `tkinter.filedialog` ile seçer. - Hedef dil seçimi için bir `OptionMenu` veya `Combobox` bulunur (varsayılan Türkçe olabilir, listede Argos'un desteklediği diller yer alır). - **Başlat** düğmesine basıldığında, arka planda yukarıda belirtilen pipeline işlemleri sırayla tetiklenir. - İşlem ilerlerken kullanıcıya bir ilerleme göstergesi (Progress bar) veya basitçe "İşlem devam ediyor..." mesajı verilebilir. - Tamamlandığında çıktı dosyası için bir indirme/dinleme opsiyonu sunulur (örn. "Çıktıyı Kaydet" veya ses dosyasını masaüstüne kaydedip "Aç" seçeneği).

Çoklu Thread ve Async: GUI ana iş parçacığı üzerinde uzun süren işlemler (STT, çeviri, TTS gibi birkaç saniyeden uzun sürebilecek) kesinlikle çalıştırılmamalıdır, aksi halde arayüz donar ve Windows "yanıt vermiyor" şeklinde gözükecektir. Bu nedenle "Dublaj Başlat" tıklandığında yeni bir thread başlatıp tüm pipeline'ı onun içinde yürütmek iyi bir çözümdür. Python'da `threading.Thread` kullanarak veya daha basitçe `concurrent.futures.ThreadPoolExecutor` ile bir fonksiyonu arkaplanda çalıştırmak mümkündür. Böylece ana thread arayüzü güncel tutabilir (progress bar ilerletebilir vs.).

Tkinter, thread kullanırken dikkat edilmesi gereken bir kütüphane: Sadece **ana thread** içinden arayüz bileşenlerine dokunmak güvenlidir. Arkaplan thread'ler işi bitirince sonucu ana thread'e bir haber vermeli (ör. `queue` kullanarak mesaj iletmek veya `window.after` ile periyodik kontrol) ve ana thread'te GUI güncellenmelidir. Bu tasarımı düzgün yapmaya özen gösterilmeli.

Alternatif GUI Araçları: Eğer daha modern bir arayüz istenirse, **PyQt / PySide** veya **Kivy** gibi kütüphaneler düşünülebilir. PyQt5 güçlü bir arayüz sağlar ancak boyutu büyük ve ticari kullanımda lisans dikkat ister (PySide6 alternatifi LGPL olarak kullanılabilir). Kivy ise çapraz platform bir GUI framework'üdür ancak mobil odaklıdır, masaüstünde ağır kaçabilir. Bu proje özelinde, Tkinter yeterli

olacaktır zira yapılacak işler basit form işlemleri. Tkinter için tasarım bileşenlerinden bazıları: - `Tk` ana pencere, içinde `Label`, `Button`, `Entry` gibi widget'lar. - Dosya seçimi için `tkinter.filedialog.askopenfilename`. - Progress bar için `ttk.Progressbar`. - Dil listesi için `ttk.Combobox` (yazma kapalı, sadece liste). - Mesaj/sonuç göstermek için opsiyonel olarak `tkinter.messagebox` kullanılabilir (işlem bittiğinde "Tamamlandı, dosya kaydedildi" gibi).

Örnek Basit Arayüz Kodu:

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import threading

def baslat_dublaj():
    dosya = dosya_yolu.get()
    hedef = dil_var.get()
    if not dosya:
        messagebox.showwarning("Uyarı", "Lütfen bir dosya seçin"); return
    # Arkaplanda çalıştır:
    thread = threading.Thread(target=pipeline_calistir, args=(dosya, hedef))
    thread.daemon = True
    thread.start()
    progress.start()  # progressbar animasyonu baslat

def pipeline_calistir(dosya, hedef_dil):
    try:
        # 1. FFmpeg ile ses çıkar (örn. cikti.wav)
        # 2. STT yap (transkript)
        # 3. Çeviri yap (transkript -> ceviri)
        # 4. TTS yap (ceviri -> dublaj.mp3)
        sonuc_dosya = "dublaj.mp3"
        # Ana threade geri dön: (progress durdur, mesaj göster)
        root.after(0, lambda: tamamla(sonuc_dosya))
    except Exception as e:
        root.after(0, lambda: hata_goster(e))

root = tk.Tk()
root.title("Dublajör")
dosya_yolu = tk.StringVar()
dil_var = tk.StringVar(value="tr")
ttk.Label(root, text="Medya Dosyası:").grid(row=0, column=0, padx=5, pady=5)
ttk.Entry(root, textvariable=dosya_yolu, width=40).grid(row=0, column=1,
padx=5)
ttk.Button(root, text="Seç...", command=lambda:
dosya_yolu.set(filedialog.askopenfilename())).grid(row=0, column=2, padx=5)
ttk.Label(root, text="Hedef Dil:").grid(row=1, column=0, padx=5, pady=5)
ttk.Combobox(root, textvariable=dil_var, values=["tr","en","es","de","fr"],
state="readonly").grid(row=1, column=1, padx=5)
progress = ttk.Progressbar(root, mode='indeterminate')
progress.grid(row=2, column=0, columnspan=3, pady=10)
ttk.Button(root, text="Dublaj Başlat", command=baslat_dublaj).grid(row=3,
```

```
column=0, columnspan=3, pady=5)
root.mainloop()
```

Yukarıdaki kod bir fikir vermek içindir. `pipeline_calistir` fonksiyonu, ele aldığımız FFmpeg+STT+MT+TTS adımlarını sırasıyla çağırmalıdır. Bu fonksiyon ayrı bir thread'de çalışır, işlemler bitince `root.after(0, ...)` ile ana thread içinde sonuç işlemlerini yapar (progress durdurma, mesaj gösterme gibi). `tamamla` fonksiyonunda `progress.stop()` ve `messagebox.showinfo("Tamamlandı", "Dublaj tamamlandı. Çıktı: "+sonuc_dosya)` gibi işlemler yapılabilir. Hata durumunda da benzer şekilde `hata_goster` ile kullanıcıya hatanın iletilmesi sağlanır.

Kullanıcı Deneyimi İyileştirmeleri: - Seçilen dosyanın geçerli format olup olmadığını kontrol etmek (uzantı kontrolü veya ffmpeg ile kısa bir test). - Hedef dil seçimi: Argos'un desteklediği diller listelenebilir. Belki dile özgü TTS imkânı kontrol edilip, eğer o dilde ses yoksa (pyttsx3'te) uyarı verilebilir. - İşlem sırasında Başlat düğmesini devre dışı bırakmak, hatta arzu edilirse "İptal" butonu eklemek (iptal için thread sonlandırma zor olsa da bir bayrak ile kibarca iptal denenebilir). - İşlem bitince çıktı dosyasını otomatik oynatmak veya klasörde göstermek.

Tkinter ile bunlar rahatlıkla yapılabilir. Arayüz tasarımında karışıklık olmaması için olabildiğince az kontrol ve temiz bir düzen kullanılmalıdır – ki bu da MVP'nin hedeflediği basitliktir [41].

6. PyInstaller ile .exe Paketleme

Geliştirilen Python uygulamasını son kullanıcı için tek tıklamayla çalışacak bir **.exe** dosyası haline getirmek adına **PyInstaller** kullanılacaktır. PyInstaller, Python betiklerini ve onların bağımlılıklarını toplayıp tek bir çalıştırılabilir dosya (one-file) veya bir klasör (one-dir) halinde paketler.

Temel Kullanım: Uygulamanın ana Python dosyası (örneğin `app.py`) için şöyle bir komut kullanılabilir:

```
pyinstaller --noconfirm --onefile --icon "app.ico" app.py
```

Burada `--onefile` tek bir exe yapacaktır, `--icon` ise isteğe bağlı simge dosyası ekler. Bu komutla `dist/app.exe` oluşturulur.

Dikkat Edilmesi Gerekenler: - **Harici Dosyalar (Veri Dosyaları):** Bizim projemizde Argos çeviri modelleri (`.argosmodel` dosyaları), Whisper model dosyaları (`ggml-base.bin` vb.), FFmpeg binary'si (`ffmpeg.exe`) gibi harici dosyalar olabilir. PyInstaller, bunları otomatik olarak almaz, elle belirtmek gerekir. `--add-data` parametresi ile her bir dosya/klasörü ekleyebiliriz [42]. Örneğin:

```
pyinstaller --onefile app.py --add-data "ffmpeg.exe;."
```

Bu komut `ffmpeg.exe` dosyasını exe içine ekler ve çalışma anında çıkartıp aynı dizine koyar. Birden fazla dosya için komutu tekrarlayabilirsiniz. (Not: Onefile modunda eklenen dosyalar çalışırken geçici bir `_MEIxxxx` klasörüne açılır [43], uygulama kapandığında silinir. Bu yüzden bu dosyaları değiştirici işlemler yapmamalı veya kullanım yolunu ona göre ayarlamalısınız).

Eklenen veri dosyalarını kod içinde bulmak için, PyInstaller çalıştığında bir ortam değişkeni kullanılır. Genelde şu yöntem kullanılır:

```
import sys, os
uygulama_yolu = getattr(sys, '_MEIPASS',
os.path.abspath(os.path.dirname(__file__)))
ffmpeg_path = os.path.join(uygulama_yolu, "ffmpeg.exe")
model_path = os.path.join(uygulama_yolu, "ggml-base.bin")
```

Bu, script paketlenmişken `_MEIPASS` içinden, normalde ise script dizininden dosyaları bulmayı sağlar. (PyInstaller bunları dokümanlarında "Run-time Information" bölümünde anlatır [44] [45] .) - **Gizli İçe Aktarmalar:** PyInstaller analiz aşamasında tüm import'ları bulmaya çalışır fakat dinamik import yapılan durumlar varsa (örneğin `importlib.import_module` ile veya plugin mantığıyla), bunları kaçırabilir. Bizim senaryomuzda Argos Translate, yüklenen modellere göre iç modüller kullanabilir, pek olası değil ama pyttsx3'te platforma göre farklı driver modülleri vardır. Bu nedenle bazen `--hidden-import <modul>` ile belirtmek gerekebilir. Örneğin pyttsx3 için `--hidden-import pyttsx3.drivers.sapi5` demek Windows'ta gerekebilir (PyInstaller genelde bunu yakalar fakat emin olmak iyi). - **Tek Dosya vs. Tek Klasör:** Tek dosya modunda her şey sıkıştırılmış halde .exe içine alınır, çalıştırılınca temp'e açılır. Tek klasör modunda ise bir dist klasörüne exe yanında tüm dll ve data dosyaları konur. Hata ayıklama ve çalışma anında dosyaları görme açısından tek klasör daha rahattır. Geliştirme sürecinde `--onedir` kullanıp, her şey yoluna girince `--onefile` yapmak iyi bir yaklaşımdır. - **Boyut Optimizasyonu:** Onefile exe boyutu, içerdiğimiz modeller yüzünden büyük olabilir. Örneğin Whisper base model 95 MB, Argos en-tr modeli ~50 MB, ffmpeg ~5-10 MB, Python runtime ~10 MB... toplandığında belki ~180 MB bir exe oluşabilir. Bu normaldir. Eğer boyutu azaltmak istersek, gereksiz kütüphaneleri dışarıda bırakabiliriz (`--exclude-module` ile) veya modellere ayrı paket muamelesi yapabiliriz. Bir yaklaşım, modelleri ayrı bir zip dosyası olarak verip, program ilk çalıştığında o zip'ten okuması olabilir. Fakat MVP için her şey dahil tek dosya, offline çalışmayı en sorunsuz garantileyen yöntemdir. - **UPX ve Diğer Araçlar:** PyInstaller, UPX ile exe içindeki bazı kitaplıkları sıkıştırabilir. Bu bazen antivirüs programlarının şüphelenmesine yol açabilir. İsterseniz `--noupx` ile kapatabilirsiniz. Ayrıca Windows Defender'ın yanlış pozitifini engellemek için uygulamanızı sertifikayla imzalamak profesyonel bir adımdır, ama başlangıç için şart değil.

Kurulum ve Çalıştırma: Elde edilen `app.exe`, hedef Windows bilgisayarında çift tıklandığında çalışmalıdır. Kullanıcıya ekstra bir yük kurdurmamak için, gerekli tüm dosyalar exe'de zaten mevcut olacaktır. Yine de **sistem gereksinimleri** konusunda bilgilendirme yapmak iyi olur: Örneğin Windows 10 veya üstü, x64 mimarisi, en az 4 GB RAM (büyük modeller kullanılıyorsa daha fazla), boş disk alanı vs. belirtilmeli. Özellikle STT veya TTS için CPU kullanımı yüksek olabilir, bu yüzden kullanıcıya işlem sırasında biraz beklemesi gerekebileceği açıklanabilir.

PyInstaller Alternatifleri: Kivy kullanılsa belki `PyInstaller` yerine Kivy-Packager, `cx_Freeze` vb. düşünülebilirdi. Ancak genel olarak PyInstaller en pratik çözümdür. Bir diğer alternatif, bu projeyi bir **Docker konteyner** içinde veya bir **CLI aracı** olarak dağıtmaktır, fakat masaüstü uygulama için .exe en uygun olanıdır.

7. Örnek `requirements.txt` ve `README.md`

Projenin bakımını kolaylaştırmak ve kullanıcıların/dıǵer geliştiricilerin kurulumu doğru yapabilmesi için, net bir `requirements.txt` ve açıklayıcı bir `README.md` sunulmalıdır.

`requirements.txt` Örneği

Aşağıda bu proje için bir gereksinimler dosyası örneği verilmiştir. Bu dosya, pip ile kurulması gereken paketleri listeler:

```
argostranslate==1.9.6        # Offline çeviri kütüphanesi
pywhispercpp==1.3.0         # Whisper.cpp Python bağlayıcısı (STT için)
# openai-whisper alternatif olarak kullanılacaksa yukarıdaki yerine:
# openai-whisper==20230314
pyttsx3==2.90               # Offline TTS (Windows SAPI kullanımı için)
vosk==0.3.45                # (Opsiyonel) Alternatif STT motoru Vosk
ffmpeg-python==0.2.0        # (Opsiyonel) FFmpeg için Python sarmalayıcı
sounddevice==0.4.6          # (Opsiyonel) pywhispercpp örnekleri için gerekli
olabilir
numpy==1.24.3               # Argos/Whisper gibi bazı paketlerin alt
bağımlılığı
comtypes==1.1.14            # pyttsx3'ün SAPI için ihtiyaç duyabileceği COM
arayüzü
```

Notlar: - Argos Translate belirli bir sürümle belirtildi, model uyumu açısından en güncel (1.9.x) kullanılabilir. - Whisper kısmı için ya `pywhispercpp` ya da `openai-whisper` seçilmeli; ikisi birden zorunlu değil. `whispercpp` daha hafif, ancak derleme adımı içerebilir. openai-whisper ise doğrudan çalışır ama PyTorch gerektirir. - Vosk ve ffmpeg-python opsiyoneldir; eğer Whisper ve subprocess ile ffmpeg kullanacaksak bunlar şart değil. Ama örnek diye dahil edildi. - `comtypes` Windows'da pyttsx3'ün ihtiyaç duyabildiği bir pakettir (bazı SAPI nesneleri için). Bazı pyttsx3 sürümleri pywin32 de gerektirebilir ama genelde kendi halleder. - Ek olarak belki `tkinter` yok çünkü bu Python ile gelen bir modül (pip ile kurulmaz). Yine de `pip install tk` gibi bir paket de vardır ama genelde gerekli değil.

Bu dosyayı kullanarak, geliştirme ortamında `pip install -r requirements.txt` komutuyla tüm gereksinimler kurulabilir. Kullanıcı açısından ise .exe verildiğinde bunları kurmasına gerek kalmayacak, ama açık kaynak kod sunulacaksa diğer geliştiriciler için bu liste değerlidir.

`README.md` Örnek Şablonu

Kullanıcı odaklı bir README, teknik detaya boğulmadan uygulamanın ne işe yaradığını, nasıl kurulup kullanıldığını anlatmalıdır. Aşağıda bu proje için olası bir README içeriği özetlenmiştir:

🎬 DUBLAJÖR – Offline Video Dublaj ve Çeviri Uygulaması

Dublajör, internet bağlantısına gerek duymadan çalışan bir masaüstü uygulamasıdır. Verdiğiniz video veya ses dosyasındaki konuşmaları yazıya çevirir, istediğiniz dile **çeviri yapar** ve hedef dilde size yeni bir **sesli dublaj** dosyası üretir. Tüm işlemler bilgisayarınızda gerçekleşir; gizli verileriniz buluta gönderilmez.

Özellikler

- **Offline Çalışma:** İnternet bağlantısı olmadan, tamamen yerel sistemde işlem yapar.
- **Desteklenen Girişler:** Video: `.mp4`, Ses: `.mp3`, `.wav` dosyaları.
- **Çıktı:** Seçilen dile çevrilmiş ses (standart olarak `output.mp3` formatında) dosyası.
- **Dil Desteği:** İngilizce başta olmak üzere yaklaşık 80+ dilde transkripsiyon ve çeviri. Örneğin İngilizce bir videodan Türkçe dublaj üretebilirsiniz. (Tam dil listesi için aşağıya bakınız.)
- **Arayüz:** Basit ve kullanıcı dostu. Tek pencereli arayüzde dosyanızı seçip **"Dublaj Başlat"** demeniz yeterli.
- **Tek .exe Kurulum:** Uygulama, gerekli bileşenlerle birlikte tek bir exe dosyası olarak gelir. Kurulum gerektirmez, çift tıklayıp çalıştırabilirsiniz.

📦 Kurulum ve Çalıştırma

Yöntem 1: Çalıştırılabilir Dosya (Önerilen)
Son kullanıcı için en kolay yol, yayınlanan `Dublajor_v1.0.exe` dosyasını indirmektir. Bu dosyayı uygun bir klasöre alın ve çift tıklayın. Uygulama açılacaktır. Herhangi bir kurulum adımı yoktur. *(Not: Windows SmartScreen ilk başta uyarı verebilir, "Yine de Çalıştır"ı seçmeniz gerekebilir. Çünkü uygulama yayıncı imzası içermiyor.)*

Yöntem 2: Kaynak Koddan Çalıştırma (Geliştiriciler için)
Python 3.9+ sürümü sisteminizde yüklüyse, bu depoyu klonlayıp komut satırından aşağıdaki adımları uygulayabilirsiniz:

```
pip install -r requirements.txt
python app.py
```

Bu şekilde de uygulama arayüzü açılacaktır. (FFmpeg'in sisteminizde kurulu ve PATH'de olması gerektiğini unutmayın.)

Kullanım Kılavuzu

1. Uygulamayı başlatın. Karşınıza gelen pencerede, **"Dosya Seç"** düğmesine tıklayın ve dublaj yapmak istediğiniz video veya ses dosyasını seçin.
2. **Hedef dili** açılır listeden belirleyin. (Örneğin Türkçe seçebilirsiniz. İngilizce dışındaki kaynak diller için uygulama otomatik algılamaya çalışacaktır.)
3. **"Dublaj Başlat"** butonuna tıklayın. İşlem süresince ilerleme çubuğu hareket edecektir. Bu esnada:
4. Video içeriğinden ses ayrıştırılır.
5. Konuşma metne çevrilir (transkript oluşturulur).
6. Transkript, seçtiğiniz dile çevrilir.
7. Çeviri metni, o dilde seslendirilir ve MP3 dosyası oluşturulur.
8. İşlem tamamlandığında, uygulama **çıktı dosyasını** size sunacaktır. Varsayılan olarak `output.mp3` adıyla, uygulamanın bulunduğu dizine kaydedilir. İsterseniz farklı bir konuma kaydedebilir veya hemen dinleyebilirsiniz (Windows'ta dosyaya çift tıklayınca varsayılan oynatıcıda açılır).
9. Yeni bir dosya için aynı adımları tekrarlayabilirsiniz.

Notlar: - İlk kullanımda, gerekli dil modelleri yüklenmemişse uygulama biraz gecikebilir veya model indirme işlemi yapabilir. Örneğin İngilizce→Türkçe çeviri modülü eksikse, uygulama bunu size soracak ve internet izni olduğu durumda indirecektir. Tamamen offline kullanım için lütfen Model İndirme bölümüne bakın. - Çıktı ses dosyası yalnızca konuşma içerir, orijinal videodaki arkaplan müziği veya efektler bu sürüme eklenmez (MVP kısıtı). İleriki sürümlerde daha gelişmiş dublaj özelliği eklenecektir. - Çok uzun videolarda (1 saat+ gibi) işlem süresi uzun olacaktır. Lütfen sabırlı olun ve işlem bitene kadar uygulamayı kapatmayın.

Desteklenen Diller

Transkripsiyon (STT) Desteklenen Diller: Uygulama, altyapısındaki Whisper modeli sayesinde 90+ dilde konuşmayı metne çevirebilir. Örnek diller: Türkçe, İngilizce, İspanyolca, Fransızca, Almanca, Rusça, Çince, Japonca, Arapça, Portekizce, Hollandaca, İtalyanca, Korece vb.

Çeviri (MT) Desteklenen Diller: Argos Translate altyapısı ile doğrudan veya dolaylı olarak birçok dil çiftinde çeviri mümkündür. En iyi sonuç alınan çiftler: İngilizce ↔ Türkçe, İngilizce ↔ İspanyolca, İngilizce ↔ Fransızca, Almanca ↔ İngilizce gibi yaygın kombinasyonlar. Daha az destekli diller için uygulama İngilizceyi ara dil olarak kullanabilir (Örneğin Çince → İngilizce → Türkçe şeklinde).

Seslendirme (TTS) Desteklenen Diller: TTS motoru olarak varsayılan Windows sistem sesleri kullanıldığından, Windows'un desteklediği tüm dillerde seslendirme yapılabilir. Örneğin Türkçe, İngilizce (ABD, İngiltere vs aksanları), Almanca, Fransızca, İspanyolca, İtalyanca, Lehçe, Portekizce, Rusça, Japonca, Korece, Çince (Mandarin) gibi birçok dil mevcuttur. Bilgisayarınızda ilgili dilin sesi yüklü değilse, [Ayarlar > Zaman ve Dil > Dil > Sesler] bölümünden çevrimdışı dil sesi paketini indirmeniz gerekebilir. Uygulama, eğer o dilde bir ses bulamazsa uyarı verecektir.

⚙ Teknik Detaylar (Geliştiriciler için)

(Bu bölümde, uygulamanın iç işleyişine dair teknik bilgiler verilir. Whisper.cpp, Argos Translate, SAPI gibi bileşenlerin nasıl kullanıldığı ve entegre edildiği anlatılır. Zaten bu bilgiler yukarıda kapsamlı biçimde işlendi.)

? Sık Sorulan Sorular

- **S: Gerçekten internet kullanılmıyor mu?**
  C: Kesinlikle hayır. Tüm işlemler (speech-to-text, translation, text-to-speech) cihazınızda çalışır. İlk defa kullanırken eksik model dosyalarını indirmeniz istenebilir (opsiyonel), onun dışında internet gerekmez.
- **S: Neden çıktıda orijinal videonun arka plan sesleri yok?**
  C: Şu anki sürüm, yalnızca konuşma sesini çevirip yeniden üretmeye odaklı. Gelecekte, orijinal videonun arka plan müziğini koruyup sadece konuşmaları değiştirme özelliği eklemeyi planlıyoruz.
- **S: X dilinden Y diline dublaj yapabilir miyim?**
  C: Kaynak dil veya hedef dil neredeyse herhangi bir dil olabilir. Ancak en iyi sonuçlar İngilizce ve Türkçe gibi çok desteklenen dillerle alınmaktadır. Örneğin Japonca bir videoyu Türkçeye dublaj yapabilirsiniz; uygulama Japoncayı metne çevirip, Türkçeye tercüme edip seslendirecektir.
- **S: Uygulama yavaş çalışıyor, ne yapabilirim?**
  C: İşlem süresi videonun uzunluğuna, bilgisayarınızın performansına ve seçilen model boyutlarına bağlıdır. Daha hızlı sonuç için "Ayarlar"dan (henüz eklenmediyse) daha küçük bir STT modeli veya daha hızlı bir çeviri modu seçilebilirdi. Mevcut sürümde böyle bir seçenek yok, ancak

güçlü bir CPU ve mümkünse GPU kullanımı işlemleri hızlandırır (gelecekte GPU desteği eklenebilir).

- **S: Mac/Linux için sürüm var mı?**
  C: Şimdilik Windows .exe olarak yayınlandı (Windows 10/11 64-bit). Talebe göre ve bileşenlerin uyumluluğuna göre diğer platformlara da uyarlanabilir. Proje zaten Python tabanlı olduğu için teorik olarak Linux/Mac'te de çalışır, ancak TTS kısmı Windows'a özgü (SAPI) olduğundan farklı bir TTS motoru entegre etmek gerekir.

Lisans ve Teşekkür

Bu proje açık kaynak olarak MIT lisansı ile dağıtılmaktadır. Whisper.cpp, Argos Translate, Balabolka gibi bileşenler kendi lisanslarına tabidir. Kullanılan teknolojilere ve açık kaynak topluluklarına teşekkür ederiz.

Yukarıdaki README taslağı, gerçek bir kullanıcıya yönelik anlaşılır bir dokümantasyon sağlamaktadır. Hem uygulamanın amacını ve kullanımını netleştirir, hem de gerektiğinde teknik detaya inen kısımları barındırır. Özellikle kurulum, kullanım adımları ve SSS bölümü, kullanıcıların sık takıldığı noktaları önceden cevaplayarak destek ihtiyacını azaltacaktır.

---

1 2 41 DUBLAJOR-Offline-Video-Dublaj-ve-Ceviri-Uygulamasi.pdf
file://file-2GWFMXPoedn6ZpcsN15uUA

3 5 Introducing Whisper | OpenAI
https://openai.com/index/whisper/

4 Gladia - What is OpenAI Whisper?
https://www.gladia.io/blog/what-is-openai-whisper

6 7 8 pywhispercpp · PyPI
https://pypi.org/project/pywhispercpp/

9 Python bindings · ggml-org whisper.cpp · Discussion #603 - GitHub
https://github.com/ggerganov/whisper.cpp/discussions/603

10 11 VOSK Offline Speech Recognition API
https://alphacephei.com/vosk/

12 argostranslate · PyPI
https://pypi.org/project/argostranslate/1.4.0/

13 14 Argos Translate Documentation — Argos Translate 1.0 documentation
https://argos-translate.readthedocs.io/

15 16 17 skeptric - Offline Translation in Python
https://skeptric.com/python-offline-translation

18 Meta Open-Sources 200 Language Translation AI NLLB-200 - InfoQ
https://www.infoq.com/news/2022/08/meta-translation-ai-nllb/

19 Balabolka Text To Audio, Always Latest Version - Blind Help Project
https://blindhelp.net/software/balabolka

20 21 22 23 24 Balabolka :: Command Line Utility
https://www.cross-plus-a.com/bconsole.htm

25 Text to Speech Python: A Comprehensive Guide | Speechify
https://speechify.com/blog/text-to-speech-python/?srsltid=AfmBOoq1ZuqCXxphsZsDJId71hGeeuV1iINzjvl7wJTLmd1VhHSCJVrs

26 27 28 29 30 31 Top Open Source Text to Speech Alternatives Compared
https://smallest.ai/blog/open-source-tts-alternatives-compared

32 39 40 DUBLAJÖR-STARTUP.txt
file://file-54gvqgfKYL1Yg2moc9iR9v

33 34 How can I extract audio from video with ffmpeg? - Stack Overflow
https://stackoverflow.com/questions/9913032/how-can-i-extract-audio-from-video-with-ffmpeg

35 36 ffmpeg - replace audio in video - Super User
https://superuser.com/questions/1137612/ffmpeg-replace-audio-in-video

37 tkinter — Python interface to Tcl/Tk — Python 3.13.3 documentation
https://docs.python.org/3/library/tkinter.html

38 10 Powerful Python GUI Frameworks for 2024 - Full Scale
https://fullscale.io/blog/python-gui-frameworks/

42 43 44 45 Using Spec Files — PyInstaller 6.13.0 documentation
https://pyinstaller.org/en/stable/spec-files.html