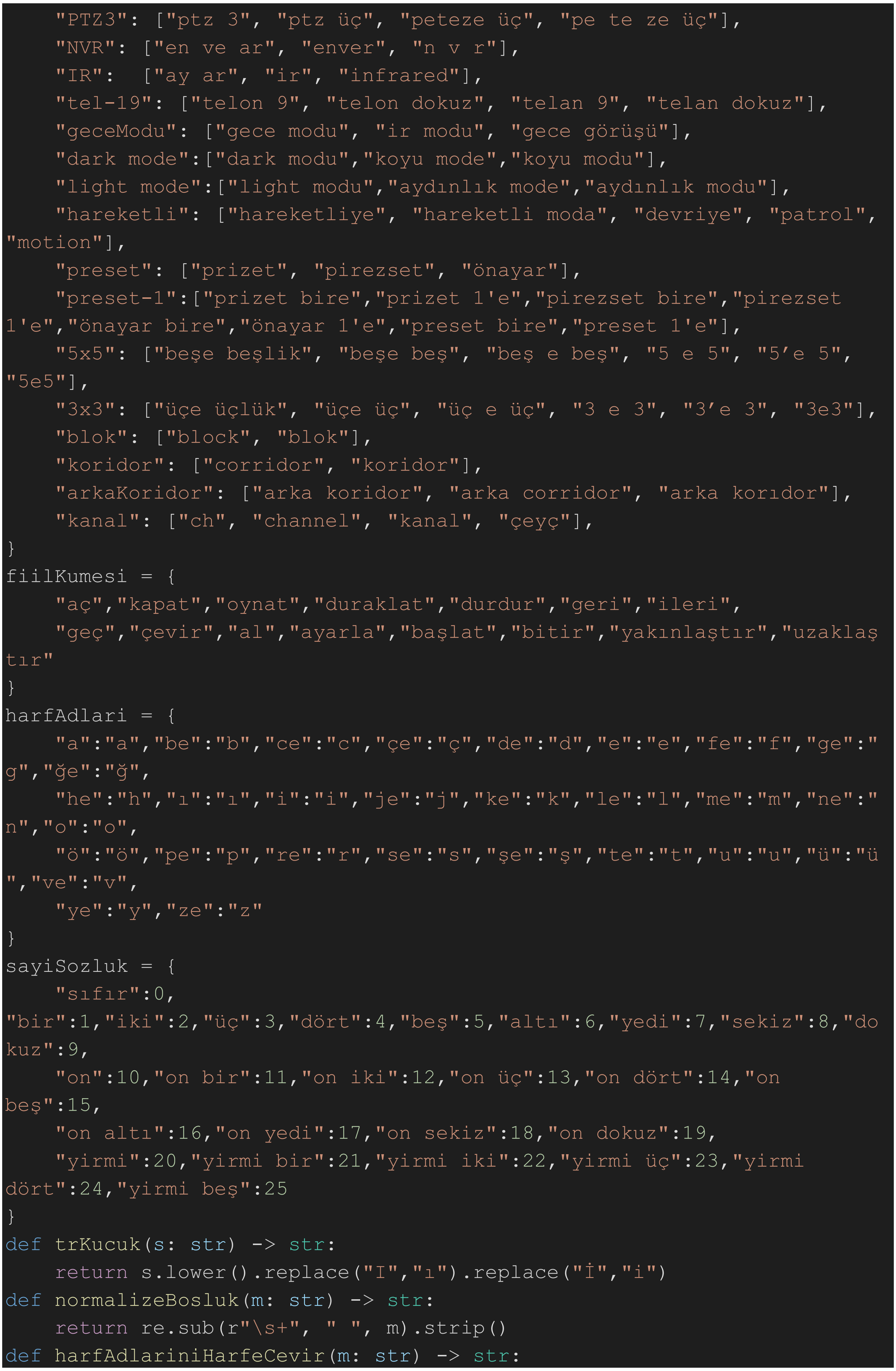
Kurulum ve Yetkilendirme

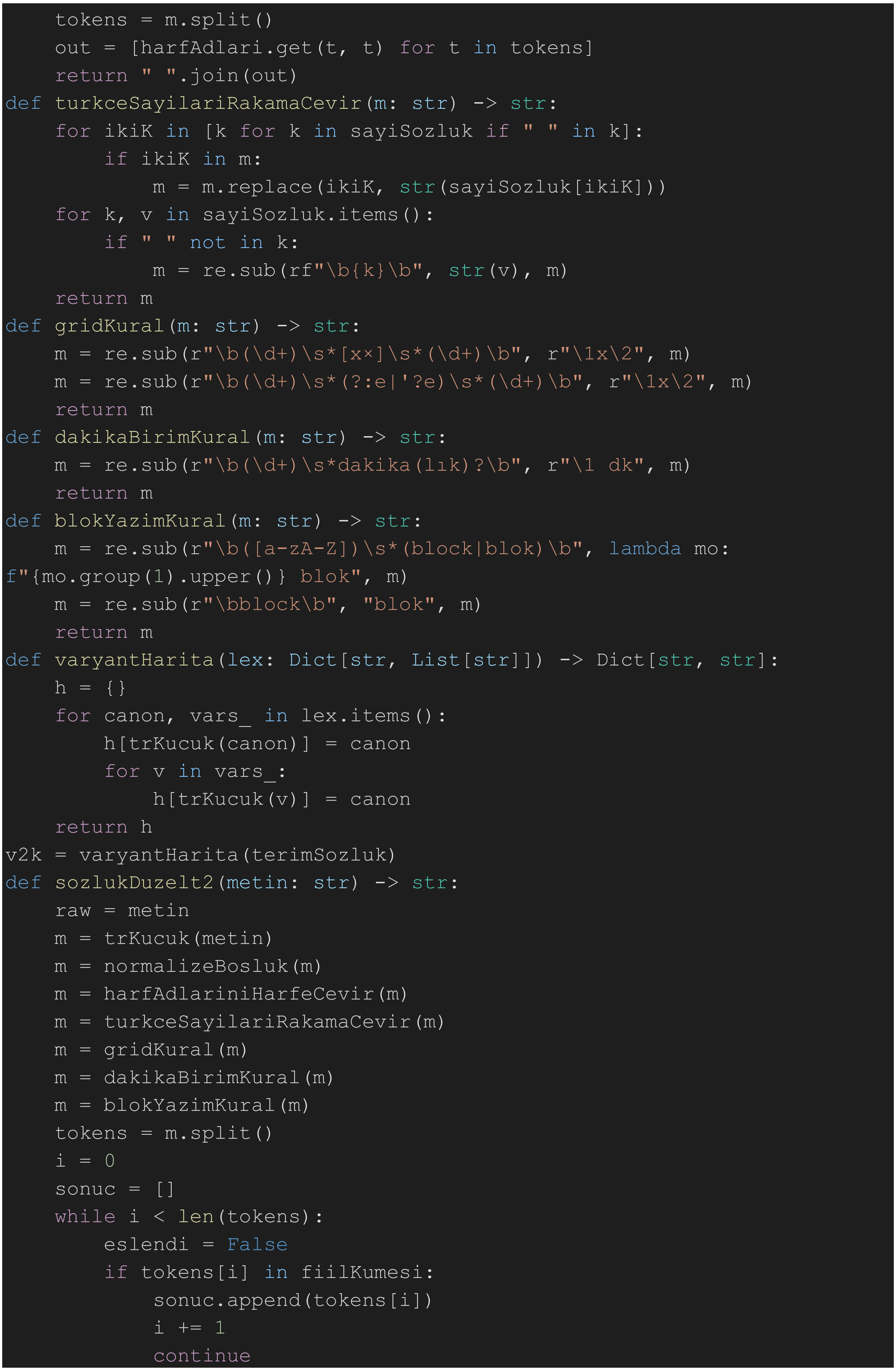
```
# ==============================================================================
# HÜCRE 1: KURULUM VE YETKİLENDİRME
# ==============================================================================

# Gerekli Python kütüphanelerini yükle
!pip install -q flask pyngrok "faster-whisper @ git+https://github.com/SYSTRAN/faster-whisper.git"
!pip install -q soundfile numpy librosa noisereduce psutil waitress

# FFmpeg kurulumu (ses formatı dönüşümleri için kritik)
print("\n--> FFmpeg kurulumu yapılıyor...")
!apt-get update && apt-get install -y ffmpeg
print("--> FFmpeg kurulumu tamamlandı.")

# Ngrok için yetkilendirme
from pyngrok import ngrok, conf
import getpass

# Lütfen tırnak işaretleri arasına kendi ngrok authtoken'ınızı yapıştırın
# https://dashboard.ngrok.com/get-started/your-authtoken adresinden alabilirsiniz.
NGROK_AUTHTOKEN = "32Mr2VfQZtgVuTL1IlcPjYGN4RN_3nwufMvrDuej6V4PTw3bC" # @param {type:"string"}

if not NGROK_AUTHTOKEN:
    print("Lütfen ngrok Authtoken'ınızı yukarıdaki alana girin.")
else:
    conf.get_default().auth_token = NGROK_AUTHTOKEN
    print("Ngrok Authtoken başarıyla ayarlandı.")
```

ngrok authtoken= 32Mr2VfQZtgVuTL1IlcPjYGN4RN_3nwufMvrDuej6V4PTw3bC

```
%%writefile sozluk_duzeltici.py
import re
from typing import List, Tuple, Dict

# ========== 1) SÖZLÜK ==========
terimSozluk: Dict[str, List[str]] = {
    # Kısaltmalar / cihazlar
    "PTZ": ["peteze", "pe te ze", "p t z", "pteze", "pte z"],
    "PTZ1": ["ptz 1", "ptz bir", "peteze bir"],
    "PTZ2": ["ptz 2", "ptz iki", "peteze iki", "ptz iki"],
```

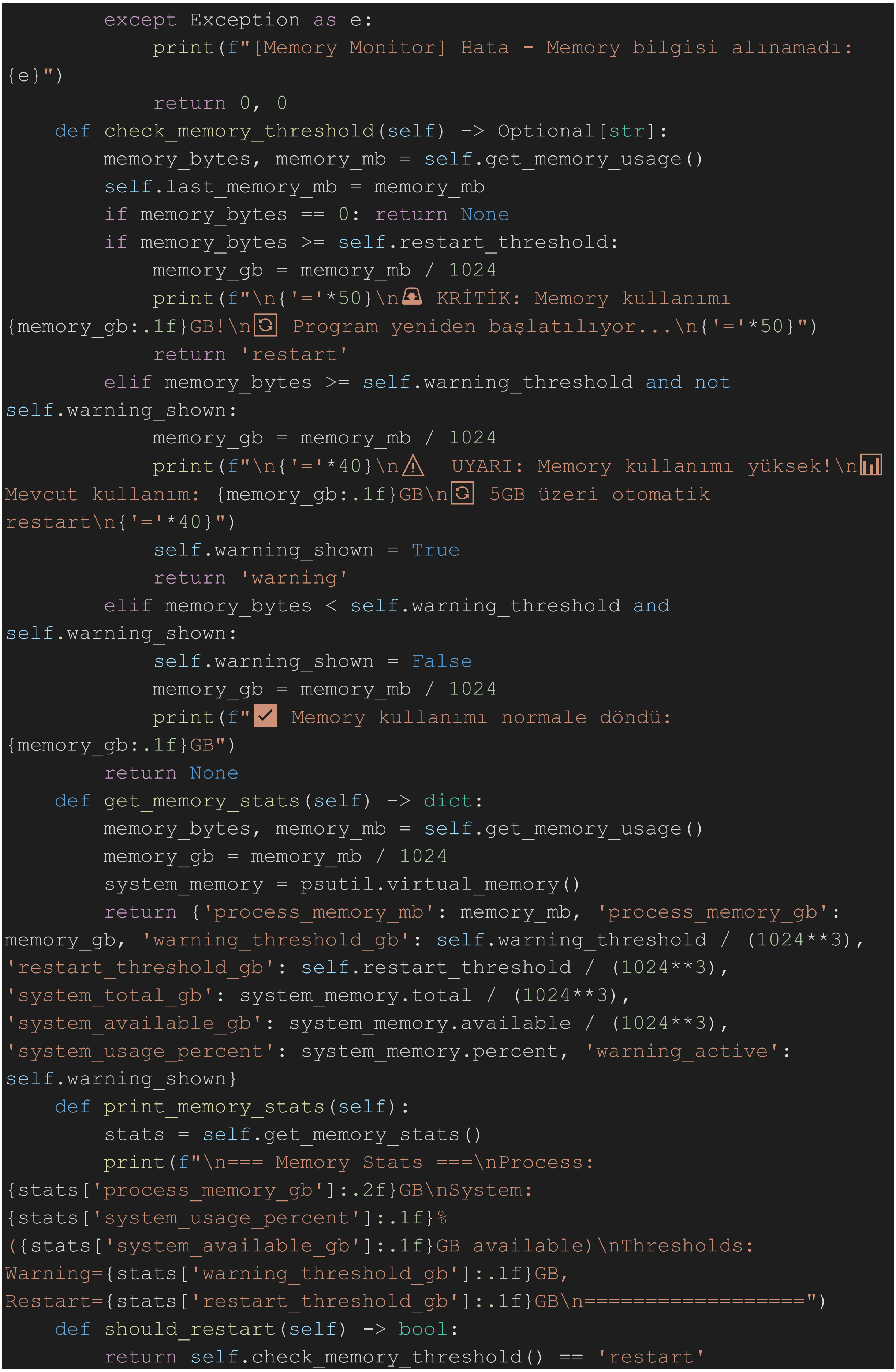


```
    "PTZ3": ["ptz 3", "ptz üç", "peteze üç", "pe te ze üç"],
    "NVR": ["en ve ar", "enver", "n v r"],
    "IR":  ["ay ar", "ir", "infrared"],
    "tel-19": ["telon 9", "telon dokuz", "telan 9", "telan dokuz"],
    "geceModu": ["gece modu", "ir modu", "gece görüşü"],
    "dark mode":["dark modu","koyu mode","koyu modu"],
    "light mode":["light modu","aydınlık mode","aydınlık modu"],
    "hareketli": ["hareketliye", "hareketli moda", "devriye", "patrol",
"motion"],
    "preset": ["prizet", "pirezset", "önayar"],
    "preset-1":["prizet bire","prizet 1'e","pirezset bire","pirezset
1'e","önayar bire","önayar 1'e","preset bire","preset 1'e"],
    "5x5": ["beşe beşlik", "beşe beş", "beş e beş", "5 e 5", "5’e 5",
"5e5"],
    "3x3": ["üçe üçlük", "üçe üç", "üç e üç", "3 e 3", "3’e 3", "3e3"],
    "blok": ["block", "blok"],
    "koridor": ["corridor", "koridor"],
    "arkaKoridor": ["arka koridor", "arka corridor", "arka korıdor"],
    "kanal": ["ch", "channel", "kanal", "çeyç"],
}
fiilKumesi = {
    "aç","kapat","oynat","duraklat","durdur","geri","ileri",
    "geç","çevir","al","ayarla","başlat","bitir","yakınlaştır","uzaklaş
tır"
}
harfAdlari = {
    "a":"a","be":"b","ce":"c","çe":"ç","de":"d","e":"e","fe":"f","ge":"
g","ğe":"ğ",
    "he":"h","ı":"ı","i":"i","je":"j","ke":"k","le":"l","me":"m","ne":"
n","o":"o",
    "ö":"ö","pe":"p","re":"r","se":"s","şe":"ş","te":"t","u":"u","ü":"ü
","ve":"v",
    "ye":"y","ze":"z"
}
sayiSozluk = {
    "sıfır":0,
"bir":1,"iki":2,"üç":3,"dört":4,"beş":5,"altı":6,"yedi":7,"sekiz":8,"do
kuz":9,
    "on":10,"on bir":11,"on iki":12,"on üç":13,"on dört":14,"on
beş":15,
    "on altı":16,"on yedi":17,"on sekiz":18,"on dokuz":19,
    "yirmi":20,"yirmi bir":21,"yirmi iki":22,"yirmi üç":23,"yirmi
dört":24,"yirmi beş":25
}
def trKucuk(s: str) -> str:
    return s.lower().replace("I","ı").replace("İ","i")
def normalizeBosluk(m: str) -> str:
    return re.sub(r"\s+", " ", m).strip()
def harfAdlariniHarfeCevir(m: str) -> str:
```



```python
    tokens = m.split()
    out = [harfAdlari.get(t, t) for t in tokens]
    return " ".join(out)
def turkceSayilariRakamaCevir(m: str) -> str:
    for ikiK in [k for k in sayiSozluk if " " in k]:
        if ikiK in m:
            m = m.replace(ikiK, str(sayiSozluk[ikiK]))
    for k, v in sayiSozluk.items():
        if " " not in k:
            m = re.sub(rf"\b{k}\b", str(v), m)
    return m
def gridKural(m: str) -> str:
    m = re.sub(r"\b(\d+)\s*[x×]\s*(\d+)\b", r"\1x\2", m)
    m = re.sub(r"\b(\d+)\s*(?:e|'?e)\s*(\d+)\b", r"\1x\2", m)
    return m
def dakikaBirimKural(m: str) -> str:
    m = re.sub(r"\b(\d+)\s*dakika(lık)?\b", r"\1 dk", m)
    return m
def blokYazimKural(m: str) -> str:
    m = re.sub(r"\b([a-zA-Z])\s*(block|blok)\b", lambda mo:
f"{mo.group(1).upper()} blok", m)
    m = re.sub(r"\bblock\b", "blok", m)
    return m
def varyantHarita(lex: Dict[str, List[str]]) -> Dict[str, str]:
    h = {}
    for canon, vars_ in lex.items():
        h[trKucuk(canon)] = canon
        for v in vars_:
            h[trKucuk(v)] = canon
    return h
v2k = varyantHarita(terimSozluk)
def sozlukDuzelt2(metin: str) -> str:
    raw = metin
    m = trKucuk(metin)
    m = normalizeBosluk(m)
    m = harfAdlariniHarfeCevir(m)
    m = turkceSayilariRakamaCevir(m)
    m = gridKural(m)
    m = dakikaBirimKural(m)
    m = blokYazimKural(m)
    tokens = m.split()
    i = 0
    sonuc = []
    while i < len(tokens):
        eslendi = False
        if tokens[i] in fiilKumesi:
            sonuc.append(tokens[i])
            i += 1
            continue
        for n in [3, 2, 1]:
            if i + n <= len(tokens):
                parca = " ".join(tokens[i:i+n])
                p = trKucuk(parca)
                if p in v2k:
                    sonuc.append(v2k[p])
                    i += n
                    eslendi = True
                    break
        if not eslendi:
            sonuc.append(tokens[i])
            i += 1
    out = " ".join(sonuc)
    out = re.sub(r"\bptz\s*(\d+)\b", lambda mo: f"PTZ{mo.group(1)}",
out, flags=re.IGNORECASE)
    out = re.sub(r"\bptz\b", "PTZ", out, flags=re.IGNORECASE)
    out = re.sub(r"\b(ch|channel|kanal)\s*(\d+)\b", r"kanal\2", out)
    return normalizeBosluk(out)
def sozlugeEkle(kanonik: str, *varyantlar: str):
    if kanonik not in terimSozluk:
        terimSozluk[kanonik] = []
    for v in varyantlar:
        if v not in terimSozluk[kanonik]:
            terimSozluk[kanonik].append(v)
    global v2k
    v2k = varyantHarita(terimSozluk)
```

```
%%writefile memory_monitor.py
import os, sys, psutil, time
from typing import Optional
class MemoryMonitor:
    def __init__(self, warning_threshold_gb: float = 3.0,
restart_threshold_gb: float = 5.0):
        self.warning_threshold = warning_threshold_gb * 1024 * 1024 *
1024
        self.restart_threshold = restart_threshold_gb * 1024 * 1024 *
1024
        self.process = psutil.Process(os.getpid())
        self.warning_shown = False
        self.last_memory_mb = 0
        print(f"[Memory Monitor] Başlatıldı - Warning:
{warning_threshold_gb}GB, Restart: {restart_threshold_gb}GB")
    def get_memory_usage(self) -> tuple[float, float]:
        try:
            memory_info = self.process.memory_info()
            memory_bytes = memory_info.rss
            memory_mb = memory_bytes / (1024 * 1024)
            return memory_bytes, memory_mb
```

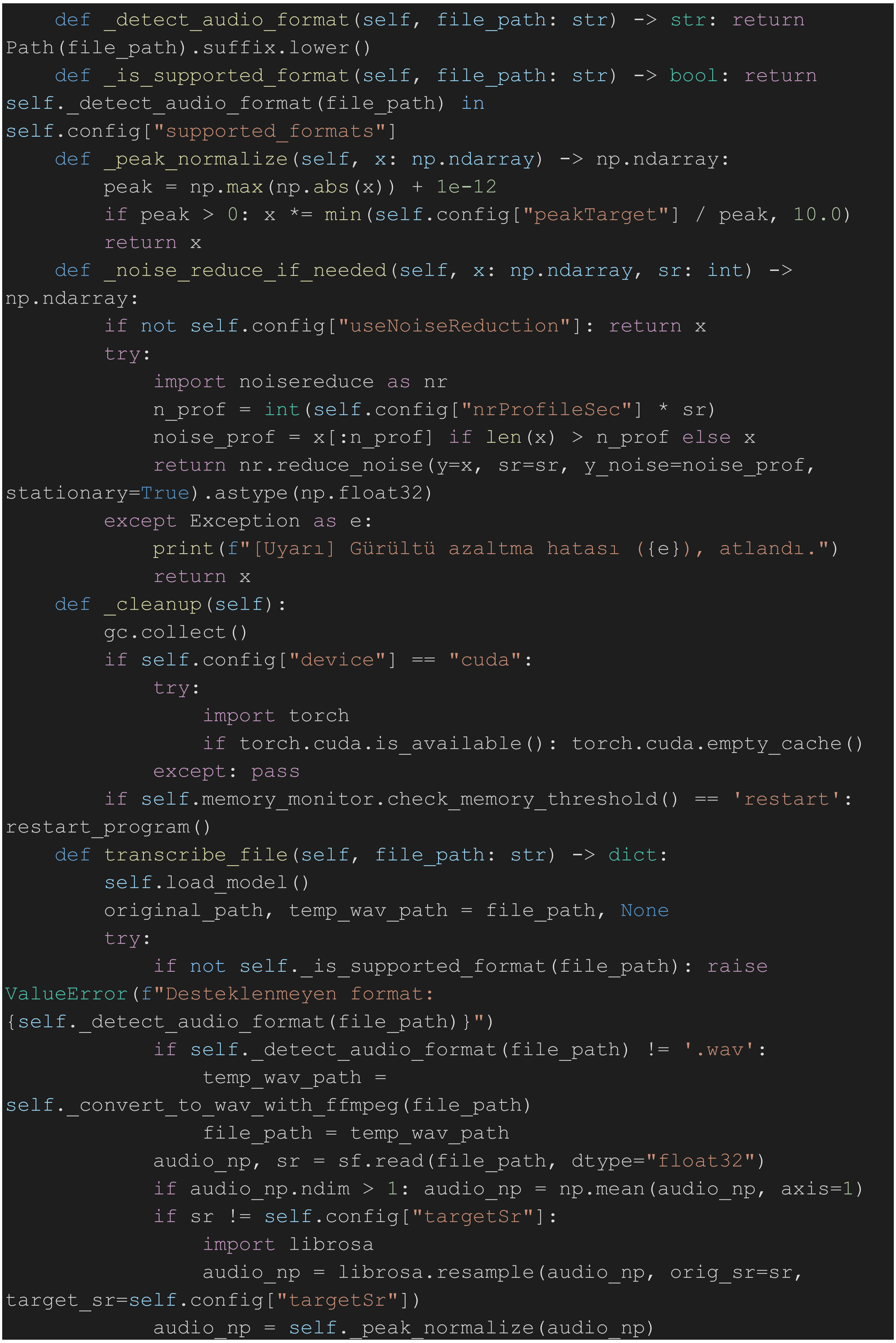


```
        except Exception as e:
            print(f"[Memory Monitor] Hata - Memory bilgisi alınamadı: {e}")
            return 0, 0
    def check_memory_threshold(self) -> Optional[str]:
        memory_bytes, memory_mb = self.get_memory_usage()
        self.last_memory_mb = memory_mb
        if memory_bytes == 0: return None
        if memory_bytes >= self.restart_threshold:
            memory_gb = memory_mb / 1024
            print(f"\n{'='*50}\n🚨 KRİTİK: Memory kullanımı {memory_gb:.1f}GB!\n🔄 Program yeniden başlatılıyor...\n{'='*50}")
            return 'restart'
        elif memory_bytes >= self.warning_threshold and not self.warning_shown:
            memory_gb = memory_mb / 1024
            print(f"\n{'='*40}\n⚠️  UYARI: Memory kullanımı yüksek!\n📊 Mevcut kullanım: {memory_gb:.1f}GB\n🔄 5GB üzeri otomatik restart\n{'='*40}")
            self.warning_shown = True
            return 'warning'
        elif memory_bytes < self.warning_threshold and self.warning_shown:
            self.warning_shown = False
            memory_gb = memory_mb / 1024
            print(f"✅ Memory kullanımı normale döndü: {memory_gb:.1f}GB")
        return None
    def get_memory_stats(self) -> dict:
        memory_bytes, memory_mb = self.get_memory_usage()
        memory_gb = memory_mb / 1024
        system_memory = psutil.virtual_memory()
        return {'process_memory_mb': memory_mb, 'process_memory_gb': memory_gb, 'warning_threshold_gb': self.warning_threshold / (1024**3), 'restart_threshold_gb': self.restart_threshold / (1024**3), 'system_total_gb': system_memory.total / (1024**3), 'system_available_gb': system_memory.available / (1024**3), 'system_usage_percent': system_memory.percent, 'warning_active': self.warning_shown}
    def print_memory_stats(self):
        stats = self.get_memory_stats()
        print(f"\n=== Memory Stats ===\nProcess: {stats['process_memory_gb']:.2f}GB\nSystem: {stats['system_usage_percent']:.1f}% ({stats['system_available_gb']:.1f}GB available)\nThresholds: Warning={stats['warning_threshold_gb']:.1f}GB, Restart={stats['restart_threshold_gb']:.1f}GB\n==================")
    def should_restart(self) -> bool:
        return self.check_memory_threshold() == 'restart'
    def reset_warning_flag(self):
        self.warning_shown = False
        print("[Memory Monitor] Warning flag resetlendi")
def restart_program():
    print("[Restart] Program yeniden başlatılıyor...")
    try:
        os.execv(sys.executable, [sys.executable] + sys.argv)
    except Exception as e:
        print(f"[Restart] HATA: Program yeniden başlatılamadı: {e}")
        sys.exit(1)
```

```
%%writefile intent_parser.py
import re, json, os, shutil
patterns=[
    {"intent":"camera_open", "pattern":re.compile(r"^(.*?)
kamera(?:yı|sı|sını|nın)?(
(görüntü|görüntüsü|görüntüsünü|ekranını|yayınını))?
(aç|getir|başlat)\.?$"), "slots":["camera_name"]},
    {"intent":"camera_rotate", "pattern":re.compile(r"^(.*?)
kamera(?:sı|sını|yı|yi|sının|nın)? (.+?)
(çevir|döndür)\.?$",re.IGNORECASE), "slots":["camera_name","value"]},
    {"intent":"review_open", "pattern":re.compile(r"^(.*?)
kamera(?:sı|sının)? (.+?) (geçmişini|geçmiş görüntüsünü|geçmiş
görüntüyü) (aç|getir)\.?$",re.IGNORECASE),
"slots":["camera_name","value"]},
    {"intent":"screen_layout", "pattern":re.compile(r"^(.*?) (ekran
düzenine|ekran düzenini) (dön|geç|aç)\.?$",re.IGNORECASE),
"slots":["template_layout"]},
    {"intent":"full_screen", "pattern":re.compile(r"^(.*?)
kamera(?:sı|sını)? (görüntüsünü|ekranını)? (tam ekran yap|tam ekrana
al|tam ekrana getir|ekranı kapla)\.?$",re.IGNORECASE),
"slots":["camera_name"]},
    {"intent":"screen_layout_full_screen",
"pattern":re.compile(r"^(.*?) (ekran|düzen)(i|ini)? (tam ekran yap|tam
ekrana al|büyüt)\.?$",re.IGNORECASE),
"slots":["last_template_layout"]},
    {"intent":"screenshot", "pattern":re.compile(r"^(.*?) ekran
görüntüsü(nü|n)? (al|çek|yakala|kaydet)\.?$",re.IGNORECASE),
"slots":["camera_name"]},
    {"intent":"dark_mode", "pattern":re.compile(r"^(dark mode
aç|karanlık modu aç|gece modunu aç|koyu mod aç|ekranı karanlık yap|gece
temasına geç|light mode kapat)\.?$",re.IGNORECASE), "slots":[]},
    {"intent":"light_mode", "pattern":re.compile(r"^(light mode
aç|aydınlık modu aç|gündüz modunu aç|açık modu aç|ekranı aydınlık
yap|gündüz temasına geç|dark mode kapat)\.?$",re.IGNORECASE),
"slots":[]},
]
def parse_intent(user_input:str):
    user_input=user_input.strip()
    for p in patterns:
        match=p["pattern"].match(user_input)
        if match:
            groups=match.groups()
            slots={slot:groups[i].strip() for i, slot in
enumerate(p["slots"])}
            result={"Intent":p["intent"]}
            result.update(slots)
            return result
    return {"Intent": "unknown", "raw_input":user_input}
def prepare_output_folder(folder="output"):
    if os.path.exists(folder): shutil.rmtree(folder)
    os.makedirs(folder)
def save_to_file(data, filename="output/intent_1.json"):
    with open(filename, "w",encoding="utf-8") as f:
        json.dump(data,f,ensure_ascii=False,indent=4)
    print(f"JSON kaydedildi: {filename}")
```

```
%%writefile main.py
import numpy as np, gc, soundfile as sf, subprocess, tempfile, os
from faster_whisper import WhisperModel
from pathlib import Path
from sozluk_duzeltici import sozlukDuzelt2
from memory_monitor import MemoryMonitor, restart_program
DEFAULT_CONFIG = {"modelSize": "medium", "device": "cpu",
"computeType": "float32", "targetSr": 16_000, "channels": 1, "dtype":
"float32", "useNoiseReduction": False, "nrProfileSec": 0.30,
"peakTarget": 0.99, "memoryWarningGB": 8.0, "memoryRestartGB": 10.0,
"ffmpeg_path": "ffmpeg", "supported_formats": [".wav", ".mp3", ".mp4",
".m4a", ".flac", ".aac", ".ogg", ".webm"], "temp_dir": None}
class TranscriptionEngine:
    def __init__(self, config=None):
        self.config = {**DEFAULT_CONFIG, **(config or {})}
        self.model = None
        self.memory_monitor =
MemoryMonitor(warning_threshold_gb=self.config["memoryWarningGB"],
restart_threshold_gb=self.config["memoryRestartGB"])
    def _check_cuda_available(self):
        try:
            import torch
            return torch.cuda.is_available()
        except: return False
    def _resolve_device_and_compute(self, device: str, compute_type:
str):
        if device == "cuda" and not self._check_cuda_available():
            print("[Model] Uyarı: CUDA bulunamadı - CPU'ya fallback
yapılıyor.")
```

```
            device, compute_type = "cpu", "float32"
        return device, compute_type
    def load_model(self):
        if self.model is not None: return self.model
        ms = self.config["modelSize"]
        dev, ct = self._resolve_device_and_compute(self.config["device"], self.config["computeType"])
        print(f"[Model] faster-whisper yükleniyor -> model={ms}, device={dev}, compute_type={ct}")
        try: self.model = WhisperModel(ms, device=dev, compute_type=ct)
        except Exception as e:
            print(f"[Model] Hata: {e}, CPU fallback deneniyor...")
            self.model = WhisperModel(ms, device="cpu", compute_type="float32")
        print("[Model] ✅ Model yüklendi ve hazır.")
        if self.memory_monitor.check_memory_threshold() == 'restart': restart_program()
        return self.model
    def _check_ffmpeg_available(self) -> bool:
        try:
            result = subprocess.run([self.config["ffmpeg_path"], "-version"], capture_output=True, text=True, timeout=10)
            return result.returncode == 0
        except: return False
    def _convert_to_wav_with_ffmpeg(self, input_path: str) -> str:
        if not self._check_ffmpeg_available(): raise RuntimeError("FFmpeg bulunamadı!")
        temp_dir = self.config["temp_dir"] or tempfile.gettempdir()
        with tempfile.NamedTemporaryFile(suffix='.wav', dir=temp_dir, delete=False) as temp_wav:
            temp_wav_path = temp_wav.name
        ffmpeg_cmd = [self.config["ffmpeg_path"], "-i", input_path, "-ar", str(self.config["targetSr"]), "-ac", "1", "-c:a", "pcm_s16le", "-y", temp_wav_path]
        try:
            print(f"[FFmpeg] Dönüştürme başlıyor: {Path(input_path).name} -> WAV")
            result = subprocess.run(ffmpeg_cmd, capture_output=True, text=True, timeout=300)
            if result.returncode != 0: raise subprocess.SubprocessError(f"FFmpeg hatası: {result.stderr or result.stdout}")
            print("[FFmpeg] ✅ Dönüştürme başarılı")
            return temp_wav_path
        except Exception as e:
            if os.path.exists(temp_wav_path): os.unlink(temp_wav_path)
            raise RuntimeError(f"FFmpeg dönüştürme hatası: {e}")
```



```
    def _detect_audio_format(self, file_path: str) -> str: return
Path(file_path).suffix.lower()
    def _is_supported_format(self, file_path: str) -> bool: return
self._detect_audio_format(file_path) in
self.config["supported_formats"]
    def _peak_normalize(self, x: np.ndarray) -> np.ndarray:
        peak = np.max(np.abs(x)) + 1e-12
        if peak > 0: x *= min(self.config["peakTarget"] / peak, 10.0)
        return x
    def _noise_reduce_if_needed(self, x: np.ndarray, sr: int) ->
np.ndarray:
        if not self.config["useNoiseReduction"]: return x
        try:
            import noisereduce as nr
            n_prof = int(self.config["nrProfileSec"] * sr)
            noise_prof = x[:n_prof] if len(x) > n_prof else x
            return nr.reduce_noise(y=x, sr=sr, y_noise=noise_prof,
stationary=True).astype(np.float32)
        except Exception as e:
            print(f"[Uyarı] Gürültü azaltma hatası ({e}), atlandı.")
            return x
    def _cleanup(self):
        gc.collect()
        if self.config["device"] == "cuda":
            try:
                import torch
                if torch.cuda.is_available(): torch.cuda.empty_cache()
            except: pass
        if self.memory_monitor.check_memory_threshold() == 'restart':
restart_program()
    def transcribe_file(self, file_path: str) -> dict:
        self.load_model()
        original_path, temp_wav_path = file_path, None
        try:
            if not self._is_supported_format(file_path): raise
ValueError(f"Desteklenmeyen format:
{self._detect_audio_format(file_path)}")
            if self._detect_audio_format(file_path) != '.wav':
                temp_wav_path =
self._convert_to_wav_with_ffmpeg(file_path)
                file_path = temp_wav_path
            audio_np, sr = sf.read(file_path, dtype="float32")
            if audio_np.ndim > 1: audio_np = np.mean(audio_np, axis=1)
            if sr != self.config["targetSr"]:
                import librosa
                audio_np = librosa.resample(audio_np, orig_sr=sr,
target_sr=self.config["targetSr"])
            audio_np = self._peak_normalize(audio_np)
            audio_np = self._noise_reduce_if_needed(audio_np,
self.config["targetSr"])
            if self.memory_monitor.check_memory_threshold() ==
'restart': restart_program()
            print(f"[STT] ⚡ Çözümleme başlıyor:
{Path(original_path).name}")
            segments, _ = self.model.transcribe(audio_np,
language="tr", task="transcribe")
            raw_text = " ".join(seg.text.strip() for seg in
segments).strip()
            corrected_text = sozlukDuzelt2(raw_text)
            self._cleanup()
            return {"raw_text": raw_text, "corrected_text":
corrected_text, "original_file": original_path, "processed_file":
file_path}
        finally:
            if temp_wav_path and os.path.exists(temp_wav_path):
                try: os.unlink(temp_wav_path)
                except Exception as e: print(f"[Cleanup] Geçici dosya
silinemedi: {e}")
    def transcribe_from_bytes(self, audio_bytes: bytes,
original_filename: str = "audio.webm") -> dict:
        if self.memory_monitor.check_memory_threshold() == 'restart':
restart_program()
        file_ext = Path(original_filename).suffix.lower() or ".webm"
        # Use a more robust way to create and write to the temp file
        with tempfile.NamedTemporaryFile(suffix=file_ext, delete=False)
as temp_input:
            temp_input_path = temp_input.name
            temp_input.write(audio_bytes)
        try:
            result = self.transcribe_file(temp_input_path)
            result["original_filename"] = original_filename
            return result
        finally:
            if os.path.exists(temp_input_path):
                try: os.unlink(temp_input_path)
                except Exception as e: print(f"[Cleanup] Geçici input
dosyası silinemedi: {e}")
    def get_memory_stats(self) -> dict: return
self.memory_monitor.get_memory_stats()
    def print_memory_stats(self):
self.memory_monitor.print_memory_stats()
_engine = None
def get_engine():
    global _engine
    if _engine is None: _engine = TranscriptionEngine()
    return _engine
```

```
def transcribe_wav(file_path: str) -> str: return
get_engine().transcribe_file(file_path)["corrected_text"]
def load_whisper_model(): return get_engine().load_model()
```

```
%%writefile run_pipeline.py
import os
from main import TranscriptionEngine
from intent_parser import parse_intent, prepare_output_folder,
save_to_file
class AudioProcessingPipeline:
    def __init__(self, model_config=None):
        print("="*50 + "\nEtkileşimli Transkripsiyon ve Niyet Analizi
Sistemi\n" + "="*50)
        print("\n[Sistem] Whisper modeli belleğe yükleniyor...")
        self.transcription_engine = TranscriptionEngine(model_config)
        self.transcription_engine.load_model()
        print("[Sistem] ✅ Model başarıyla yüklendi ve kullanıma
hazır.")
        prepare_output_folder()
        self.file_counter = 1
    def process_audio_file(self, audio_path: str, save_to_json: bool =
True) -> dict:
        print(f"\n[Pipeline] Ses dosyası işleniyor: {audio_path}")
        if not os.path.exists(audio_path): raise
FileNotFoundError(f"Dosya bulunamadı: {audio_path}")
        transcription_result =
self.transcription_engine.transcribe_file(audio_path)
        if not transcription_result["corrected_text"].strip(): raise
ValueError("Transkripsiyon başarısız - konuşma tespit edilemedi")
        print(f"[Pipeline] Transkript:
{transcription_result['corrected_text']}")
        intent_result =
parse_intent(transcription_result["corrected_text"])
        if save_to_json:
            output_filename = f"output/intent_{self.file_counter}.json"
            save_to_file(intent_result, filename=output_filename)
            print(f"[Pipeline] Sonuç kaydedildi: {output_filename}")
            self.file_counter += 1
        return intent_result
def process_audio_from_file_upload(file_content: bytes, filename: str)
-> dict:
    pipeline = get_pipeline()
    transcription_result =
pipeline.transcription_engine.transcribe_from_bytes(file_content,
filename)
    print(f"[Pipeline] Transkript:
{transcription_result['corrected_text']}")
    # Sadece transkripti değil, tüm niyet analizi sonucunu döndürelim.
    # Bu, ön yüzde hem metni hem de niyeti gösterebilmemizi sağlar.
    intent_result =
parse_intent(transcription_result['corrected_text'])

    # Final JSON'a ham ve düzeltilmiş metni de ekleyelim, daha
bilgilendirici olur.
    final_result = {
        "transcription_raw": transcription_result.get("raw_text", ""),
        "transcription_corrected":
transcription_result.get("corrected_text", ""),
        "intent_analysis": intent_result
    }
    return final_result
_pipeline = None
def get_pipeline():
    global _pipeline
    if _pipeline is None:
        # Colab'da varsayılan olarak CUDA ve float16 kullanalım
        # torch.cuda.is_available() kontrolü main.py içinde zaten
yapılıyor
        config = {"device": "cuda", "computeType": "float16",
"modelSize": "medium"}
        _pipeline = AudioProcessingPipeline(model_config=config)
    return _pipeline
```

```
#
========================================================================
=======
# HÜCRE 3 (SON REVİZYON - YENİ SEKMEDE AÇILAN VERSİYON)
#
========================================================================
=======
import torch
import logging
import base64
from flask import Flask, request, jsonify, render_template_string
from pyngrok import ngrok

# ----------------------------------------------------------------------
-----
# 1. Kendi Pipeline Modüllerimizi Import Etme
# ----------------------------------------------------------------------
-----
from run_pipeline import get_pipeline, process_audio_from_file_upload

# Log ayarları
logging.basicConfig(level=logging.INFO, format='%(asctime)s -
%(levelname)s - %(message)s')
```

```
# ----------------------------------------------------------------------
-----
# 2. Modeli ve Pipeline'ı Yükleme
# ----------------------------------------------------------------------
-----
try:
    print("="*60)
    logging.info("Whisper modeli ve işlem pipeline'ı belleğe
yükleniyor...")
    get_pipeline()
    logging.info("✅ Model başarıyla yüklendi ve sunucu hazır!")
    print("="*60)
except Exception as e:
    logging.error(f"Model yüklenirken kritik bir hata oluştu: {e}",
exc_info=True)
    if torch.cuda.is_available():
        torch.cuda.empty_cache()

# ----------------------------------------------------------------------
-----
# 3. Flask Uygulaması (HTML Sayfası Sunacak Şekilde Güncellendi)
# ----------------------------------------------------------------------
-----
app = Flask(__name__)
PORT = 5000

# API ENDPOINT'LERİ (DEĞİŞİKLİK YOK)
def process_request(request_data, endpoint_name):
    if not request_data or 'audio_base64' not in request_data:
        return jsonify({"error": "JSON formatında 'audio_base64' verisi
gönderilmedi."}), 400
    try:
        audio_bytes = base64.b64decode(request_data['audio_base64'])
        filename = request_data.get('filename', 'audio.webm')
        result = process_audio_from_file_upload(audio_bytes, filename)
        return jsonify(result)
    except Exception as e:
        logging.error(f"[{endpoint_name}] İşlem sırasında hata: {e}",
exc_info=True)
        return jsonify({"error": "Sunucuda bir hata oluştu.",
"details": str(e)}), 500

@app.route('/preview', methods=['POST'])
def preview_endpoint():
    return process_request(request.json, "Preview")

@app.route('/final', methods=['POST'])
def final_endpoint():
    return process_request(request.json, "Final")

# YENİ: ARAYÜZÜ SUNAN ANA SAYFA ENDPOINT'İ
@app.route('/')
def index():
    # Tüm HTML ve JavaScript kodunu bir Python string'i olarak tanımlıyoruz.
    html_template = """
    <!DOCTYPE html>
    <html lang="tr">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Canlı Transkripsiyon</title>
        <style>
            body { font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Helvetica, Arial, sans-serif; background-color: #f0f0f0; margin: 0; padding: 2em; display: flex; justify-content: center; align-items: flex-start; }
            .container { background-color: white; border-radius: 10px; box-shadow: 0 4px 15px rgba(0,0,0,0.1); padding: 25px; width: 100%; max-width: 800px; }
            h1, h2, h3 { color: #333; }
            h1 { margin-top: 0; }
            .status-bar { display: flex; align-items: center; gap: 15px; margin-bottom: 20px; }
            #status-light { width: 30px; height: 30px; border-radius: 50%; background-color: #bbb; border: 2px solid #888; transition: background-color 0.3s; }
            #status-text { font-size: 1.2em; font-weight: bold; color: #555; }
            .output-box { min-height: 100px; border: 1px solid #ddd; padding: 15px; border-radius: 5px; margin-top: 10px; }
            #preview-div { background: #fff; color: #555; font-style: italic; }
            #final-div { background: #e6ffed; color: #003300; white-space: pre-wrap; word-wrap: break-word; font-family: monospace, monospace; font-size: 1em; }
        </style>
    </head>
    <body>
        <div class="container">
            <h1>🎙️ Canlı Transkripsiyon Arayüzü</h1>
            <p>Kaydı başlatmak için <b>Shift + R</b>, durdurmak ve sonucu almak için <b>Shift + S</b> tuşlarına basın.</p>
            <div class="status-bar">
                <div id="status-light"></div>
                <div id="status-text">BEKLEMEDE</div>
            </div>

            <h3>Geçici Ön İzleme Metni</h3>
            <div id="preview-div" class="output-box"></div>

            <h3>✅ Gerçek Çıktı (Niyet Analizi)</h3>
            <pre id="final-div" class="output-box"></pre>
        </div>

        <script>
            const statusLight = document.getElementById('status-
light');
            const statusText = document.getElementById('status-text');
            const previewDiv = document.getElementById('preview-div');
            const finalDiv = document.getElementById('final-div');

            let isRecording = false;
            let mediaRecorder;
            let fullAudioChunks = [];

            async function sendAudioToApi(base64data, endpoint) {
                // Artık tam URL'e gerek yok, çünkü sayfa aynı
sunucudan geliyor.
                // Sadece /preview veya /final kullanmak yeterli.
                try {
                    const response = await fetch(`/${endpoint}`, {
                        method: 'POST',
                        headers: { 'Content-Type': 'application/json'
},
                        body: JSON.stringify({ audio_base64:
base64data, filename: 'audio.webm' })
                    });
                    if (!response.ok) {
                        console.error(`API Hatası
(${response.status}):`, await response.text());
                        return null;
                    }
                    return await response.json();
                } catch (error) {
                    console.error(`${endpoint} endpoint'ine istek
gönderilirken hata:`, error);
                    return null;
                }
            }

            // ... (startRecording ve stopRecording fonksiyonları
öncekiyle aynı) ...
            function startRecording() {
                if (isRecording) return;
                navigator.mediaDevices.getUserMedia({ audio: true })
                    .then(stream => {
                        isRecording = true;
                        statusLight.style.backgroundColor = '#f44336';
                        statusText.textContent = 'KAYDEDİLİYOR (Shift+S
ile durdur)';
                        previewDiv.innerHTML = '';
                        finalDiv.innerHTML = '';
                        fullAudioChunks = [];

                        mediaRecorder = new MediaRecorder(stream, {
mimeType: 'audio/webm' });

                        mediaRecorder.addEventListener('dataavailable',
async (event) => {
                            if (event.data.size > 0) {
                                fullAudioChunks.push(event.data);
                                const reader = new FileReader();
                                reader.readAsDataURL(event.data);
                                reader.onloadend = async () => {
                                    const base64AudioData =
reader.result.split(',')[1];
                                    const result = await
sendAudioToApi(base64AudioData, 'preview');
                                    if (result &&
result.transcription_corrected) {
                                        previewDiv.textContent +=
result.transcription_corrected + ' ';
                                    }
                                };
                            }
                        });

                        mediaRecorder.addEventListener('stop', async ()
=> {
                            const fullAudioBlob = new
Blob(fullAudioChunks, { type: 'audio/webm' });
                            const reader = new FileReader();
                            reader.readAsDataURL(fullAudioBlob);
                            reader.onloadend = async () => {
                                statusLight.style.backgroundColor =
'#ffeb3b';
                                statusText.textContent =
'İŞLENİYOR...';
                                const base64AudioData =
reader.result.split(',')[1];
                                const result = await
sendAudioToApi(base64AudioData, 'final');
                                if (result) {
                                    finalDiv.textContent =
JSON.stringify(result, null, 2);
                                    statusLight.style.backgroundColor =
'#4CAF50';
                                    statusText.textContent =
'TAMAMLANDI (Shift+R ile yeni kayıt)';
                                } else {
                                    finalDiv.textContent = "Hata: Sonuç
alınamadı.";
                                    statusLight.style.backgroundColor =
'#bbb';
                                    statusText.textContent = 'HATA
(Shift+R ile tekrar dene)';
                                }
                            };
                        });

                        mediaRecorder.start(2000);
                    })
                    .catch(err => {
                        statusText.textContent = 'HATA: Mikrofona
erişim izni gerekli.';
                        alert("Mikrofona erişim izni vermeniz
gerekiyor. Lütfen sayfa ayarlarından izin verip sayfayı yenileyin.");
                        console.error("Mikrofon hatası:", err);
                    });
            }

            function stopRecording() {
                if (!isRecording || !mediaRecorder) return;
                isRecording = false;
                mediaRecorder.stop();
                mediaRecorder.stream.getTracks().forEach(track =>
track.stop());
            }

            document.addEventListener('keydown', (event) => {
                if (event.shiftKey && event.key.toLowerCase() === 'r')
{
                    event.preventDefault();
                    startRecording();
                } else if (event.shiftKey && event.key.toLowerCase()
=== 's') {
                    event.preventDefault();
                    stopRecording();
                }
            });
        </script>
    </body>
    </html>
    """
    return render_template_string(html_template)

# ---------------------------------------------------------------------
-----
# 4. Ngrok Tünelini Başlatma ve Uygulamayı Çalıştırma
# ---------------------------------------------------------------------
-----
try:
    # ngrok tünelini aç ve genel URL'yi al
    public_url = ngrok.connect(PORT)
    print("="*60)
    print(f"✅ Sunucu başarıyla başlatıldı!")
    print(f"🌍 UYGULAMAYI AÇMAK İÇİN BU LİNKE TIKLAYIN: {public_url}")
    print("="*60)
    print("Uygulama yeni bir sekmede açılacak. Tüm işlemleri o sekmede 
yapın.")
    print("Bu Colab hücresini uygulama çalıştığı sürece durdurmayın.")

    # Flask uygulamasını çalıştır
    app.run(port=PORT)

except Exception as e:
    print(f"❌ Sunucu başlatılamadı. Hata: {e}")
```

https://colab.research.google.com/drive/1g524WYUw0Owkjb2FJQNhm8Fda6BmVAkw?usp=sharing