



Eötvös Loránd Tudományegyetem

Informatikai Kar

Informatikatudományi Intézet

Programozási Nyelvek és Fordítóprogramok Tanszék

Magyar népzene felismerésére szolgáló Blueprint fejlesztése

Szerző:

Fegyó Benedek PZ20TK

Programtervező informatikus BSc.

Témavezető:

Dr. Tejfel Máté

Egyetemi docens

Budapest 2024

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

SZAKDOLGOZAT TÉMABEJELENTŐ

Hallgató adatai:

Név: Fegyő Benedek

Neptun kód: PZ20TK

Képzési adatok:

Szak: programtervező informatikus, alapképzés (BA/BSc/BProf)

Tagozat : Nappali

Belső témavezetővel rendelkezem

Témavezető neve: Dr. Tejfel Máté

munkahelyének neve, tanszéke: ELTE IK, Programozási nyelvek és Fordítóprogramok Tanszék

munkahelyének címe: 1117, Budapest, Pázmány Péter sétány 1/C.

beosztás és iskolai végzettsége: Habilitált egyetemi docens

A szakdolgozat címe: Magyar népzene felismerésére szolgáló blueprint fejlesztése.

A szakdolgozat témája:

(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását) A hang és beszéd felismerés egy régóta kutatott szegmense a mesterséges intelligenciával foglalkozó informatikának. Számos olyan modell létezik, amely képes arra, hogy felismerjen egy adott frekvenciát és megállapítsa, hogy az mennyi ideig szólt. Ez a megközelítés nagyon alacsony pontossággal alkalmazható csak a magyar népzene esetében annak specialitásai miatt. Ezt a problémát egy olyan blueprint fejlesztésével szeretném áthidalni, amely képes különböző fizikai vagy virtuális node-okon mesterséges intelligencia alapú felismerő programokat párhuzamosan futtatni. A felismerőprogramot finomhangolni szükséges annak érdekében, hogy képes legyen a különböző hangszerek skáláit és a magyar népzene jellemző ritmusképleteit felismerni. A cél, hogy az edge-node-ok a felismert hangokat és ritmusokat valós időben, minimális késleltetéssel közvetítsék a központi node-nak (central node), amely összeállítja a beérkezett szövegeket MusicXML formátumba. A program blueprintként való megvalósítása számos előnnyel jár. A blueprintek olyan előre elkészített, moduláris architektúrákat jelentenek, amelyek szabványosítják és megkönnyítik az összetett technológiai megoldások tervezését, fejlesztését és implementálását. A blueprint ezen felül biztosítja, hogy a rendszer moduláris és könnyen bővíthető legyen, lehetővé téve a további fejlesztéseket és finomhangolásokat anélkül, hogy a teljes rendszert újra kellene tervezni. A felvetett edge kommunikációs feladatra már léteznek meglévő, nyílt forráskódú megoldások, például az Akraio blueprint család. Ezekkel a blueprintekkel érdemes összevetni a saját megoldást, hogy rávilágítsunk a két megközelítésben rejlő különbségekre, mint például implementálási komplexitás, skálázhatóság és a létrehozott edge kliens-szerver hálózaton az alkalmazás teljesítménye.

Budapest, 2024. 05. 15.

Tartalomjegyzék

1.	Bevezetés	1
2.	Felhasználói dokumentáció.....	2
2.1.	Probléma rövid megfogalmazása	2
2.2.	Felhasznált módszerek	2
2.3.	Program használata.....	2
2.3.1.	Előfeltételek	3
2.3.2.	Virtuális Környezet telepítése teszteléshez	4
2.3.3.	Blueprint telepítése.....	5
2.3.4.	Zenefelismerő használata	6
2.4.	Futás közben adódó tipikus hibák megoldása	9
2.4.1.	A Multipass service nem elérhető.....	9
2.4.2.	A tesztelő SSH hibával tér vissza	10
3.	Fejlesztői dokumentáció	11
3.1.	A probléma specifikálása.....	11
3.2.	MQTT	11
3.3.	Crepe	12
3.4.	Python használata a Blueprint és az alkalmazás megírásához.....	13
3.5.	Virtuális környezet (<i>Virtual_ENV/...</i>)	13
3.5.1.	Multipass	13
3.5.2.	SSH kulcsok generálása	14
3.5.3.	Cloud-Init.yaml fájl létrehozása	14
3.5.4.	A virtuális gépek létrehozása és konfigurálása (<i>setup_virtual_env.bat</i>)	15
3.6.	Blueprint (<i>Blueprint/...</i>)	17
3.6.1.	Control Node konfigurálása (<i>control_node_setup.sh</i>)	20
3.6.2.	Node-ok konfigurálása (<i>node_setup.sh</i>)	20

3.7.	Zenefelismerő alkalmazás (<i>Application/...</i>)	21
3.7.1.	Adatok bekérése a felhasználótól (<i>lampion.py</i>)	22
3.7.2.	Üzenet és fájl küldése MQTT-n Python szkripttel (<i>mqtt_publish_message.py</i> és <i>mqtt_publish_file.py</i>)	24
3.7.3.	Control Node működése az alkalmazás futása közben (<i>control_node_action.sh</i>) 24	
3.7.4.	A felismerő node működése az alkalmazás futása közben (<i>node_action.sh</i>)..	26
3.8.	A felhasznált modulok és könyvtárak listája és verziószáma	33
3.9.	Tesztelési terv és a tesztelés eredményei	34
3.9.1.	A virtuális környezet felállításának tesztelése (<i>test_virtual_env.py</i>)	34
3.9.2.	A Blueprint telepítés sikerességének ellenőrzése (<i>test_blueprint.py</i>)	36
3.9.3.	Az Alkalmazás működésének ellenőrzése (<i>test_application.py</i>)	38
3.9.4.	A tesztelés eredménye	42
4.	Összefoglalás és további fejlesztési lehetőségek	43
5.	Köszönetnyilvánítás	45
6.	Irodalomjegyzék	46

1. Bevezetés

A népzene egy olyan hagyomány, amelynek megőrzése és tovább örökítése egy kifejezetten fontos és nehéz kulturális feladat. Egy olyan folyamat, amely tradicionálisan hallás utáni tanulás és a dallamok ismerete által marad fent folyamatosan a kihalás veszélyének van kitéve, mivel, ha egy zenész dallamait senki sem tanulja meg, azok egyszerűen feledésbe merülnek. (1) Ennek az elkerülésének egy fontos eszköze minél több dallam lejegyzése és digitalizálása, amelynek egy lépésében kíván segítséget nyújtani az általam készített szoftver, hogy ez a nehéz és körülményes folyamat valamelyest egyszerűbbé váljon. A feladat egy olyan szoftver létrehozása volt, amely képes felvett hanganyag alapján mesterséges intelligencia segítségével felismerni és ember számára felismerhető módon lekottázni akár több szólamú dallamokat. Cél volt, hogy mindezt olyan módon tegye, hogy a felhasználónak a lehető legkevesebb erőforrással kelljen rendelkeznie, annak érdekében, hogy a program minél több ember számára elérhető és használható legyen.

2. Felhasználói dokumentáció

2.1. Probléma rövid megfogalmazása

A magyar népzeneben a dallamokat hallás alapján, kézzől kézre adják tovább generációk óta. (2) Ennek megvan a szépsége, hiszen aki szeretné és lehetősége van rá, az közvetlenül a forrástól tud tanulni és az ő értelmezésében meghallgatni akár egy egész rendet vagy dallamfűzést. Azonban annak érdekében, hogy ezek a dallamok fent tudjanak maradni olyan esetekben is, amikor valamilyen okból kifolyólag nincs, aki megtanulja vagy lekottázza az adott hanganyagot szükségessé válik egy olyan alkalmazás, amely ezt automatikusan megteszi anélkül, hogy nagyfokú emberi beavatkozást igényelne. Ezen kívül az is lényeges, hogy minimális erőforrású eszközökön is fusson. Ezeket a szempontokat azért kell figyelembe venni, mivel egy népzenevel foglalkozó embertől sem az informatika beható ismerete, sem egy mesterséges intelligencia alapú felismerő futtatására elégséges személyi számítógép birtoklása nem elvárható. Ennek következtében a cél egy olyan alkalmazás kifejlesztése volt, ami automatikusan telepíti a szükséges modulokat, felépíti a hálózatot a távoli gépekkel, amiken a felismerők futhatnak és egy viszonylag egyszerű és megbízható felhasználói élményt biztosít a probléma komplexitását figyelembe véve.

2.2. Felhasznált módszerek

A probléma megoldására egy Blueprintet fejlesztettem ki, ami egy ismeretlen, előre nem definiált környezetben, automatikusan felállítja az MQTT (Message Queueing Telemetry Transport) hálózatot a felhasználó személyi számítógépe és a felismerőket futtató virtuális gépek között, majd azokra telepíti az alkalmazásához szükséges modulokat és fájlokat. A felismerők egy TensorFlow alapú music21 Python könyvtárat használnak, amelynek az eredményét feldolgozzák majd musicxml formátumúvá konvertálják és továbbítják a megfelelő Node felé.

2.3. Program használata

A program használatának több fázisa van. Először teljesíteni kell bizonyos előfeltételeket. Ezek olyan hardveres és szoftveres kikötések, amelyeknek teljesülnie kell az alkalmazás futtatása előtt. A fizikai kikötések amiatt szükségesek, mivel bizonyos funkciók működése nagyban korlátozott, vagy lehetetlen megfelelő hardver nélkül. A szoftveres előfeltételek abból adódnak, hogy az alkalmazás célzott felhasználói operációs rendszere a Windows. Ennek

következtében bizonyos modulok automatikus telepítése több komplikációt és kompatibilitási problémát okozna a jövőben, mint amennyivel megnehezíti a felhasználó dolgát.

A második fázis opcionális. Erre akkor van szükség, amikor az alkalmazást teszt üzemmódban szeretnénk futtatni. Ebben az esetben feltelepülésre kerül egy virtuális környezet a felhasználó személyi számítógépére, amely képes arra, hogy az alkalmazást teszt üzemmódban futtassa, ideértve a különböző funkciókat betöltő virtuális gépeket. Amennyiben az alkalmazást fizikai szervereken szeretnénk futtatni, ez a fázis teljes mértékben kihagyható.

A harmadik fázis a Blueprint telepítése. Amennyiben mind a fizikai mind a szoftveres előfeltételek teljesülnek, a *nodelist.txt* helyes kitöltésével és a *Blueprint/setup.bat* batch szkript futtatásával, a Blueprint egy kattintásra feltelepül.

A negyedik fázis az alkalmazás használata. Ez a *Application/full_lampion.bat* batch szkript futtatásával lehetséges, ami bekéri a felhasználótól a megfelelő adatokat, hangfájlokat, továbbítja őket a felismerőknek MQTT-n keresztül, majd a feldolgozott hanganyagot a Control Node egy kottává állítja össze és eltárolja.

2.3.1. Előfeltételek

2.3.1.1. Fizikai előfeltételek

Amennyiben az alkalmazást a felhasználó a személyi számítógépén szeretné teszt üzemmódban futtatni szüksége lesz egy olyan eszközre, amely képes létrehozni a megfelelő virtuális környezetet. Mivel a felismerők TensorFlow alapúak, amihez az ajánlott minimum RAM (Random-Access Memory) 8 gigabájt, az ajánlott minimum háttértár pedig 10 gigabájt, így a felhasználó személyi számítógépének képesnek kell lennie ezeknek a forrásoknak a felismerőnkénti allokálására az operációs rendszer futtatásán felül. (3) Amennyiben a felismerők optimális működése nem elvárt akkor ez az igény két gigabájtig csökkenthető. (Lásd 3.5.4 fejezet) Továbbá a Control Node és a Mosquitto Bróker további 2 – 2 gigabájt RAM-ot igényel. Így a tesztüzemmódban történő futtatás minimum RAM igényét a következő képlettel lehet meghatározni.

$$RAM\ igény = (Felismerők\ száma + 2) * 2Gb + Rendszer\ RAM\ igény$$

Ellenkező esetben, ha a felhasználó személyi számítógépén nem szeretnénk virtuális felismerőket futtatni, akkor a RAM és háttértár igény megegyezik az operációs rendszer megfelelő működéséhez szükségessel.

2.3.1.2. Szoftveres előfeltételek

A Blueprint telepítéséhez szükséges a GitHub. Ennek az automatikus telepítése Windowson bonyolultabbá és kevésbé megbízhatóvá tenné a telepítési folyamatot így ennek a telepítése átkerült a szoftveres előfeltételekbe. Ezt a hivatalos GitHub honlapon a telepítés („Download”) gomb megnyomásával a <https://gitforwindows.org/> elérési címen lehet megtenni.

Amennyiben az alkalmazást teszt üzemmódban szeretnénk futtatni, szükséges egy Virtuális Gép koordinátor. Ez teszi lehetővé, hogy a virtuális környezet létrehozása és használata folyékonyan történjen, ugyanakkor emuláljon egy valós környezetet, hogy a tesztelés annyira megközelítse a valóságot, amennyire lehetséges. Az alkalmazás fejlesztése és tesztelése közben a Multipass, Ubuntu alapú virtuális gép szervezőt használtam. Ezt a <https://multipass.run/docs/install-multipass> elérési címen, a leírt lépések pontos követésével lehet megtenni.

2.3.1.3. A GitHub könyvtár klónozása a felhasználó személyi számítógépére

A *git* sikeres telepítése után klónozni kell a forráskódot tartalmazó könyvtárat arra a számítógépre, ahonnan az alkalmazást futtatni szeretnénk. Ezt olyan módon lehet megtenni, hogy nyitunk egy parancssort, elnavigálunk arra a helyre, ahol tárolni szeretnénk az applikációt, majd begépeljük a *git clone* https://github.com/feqyobeno/Szakdolgozat_v2.git parancsot és az enter leütésével végrehajtjuk. Ha a parancs sikeresen végrehajtása került, akkor látnunk kell egy *Szakdolgozat_v2* mappát, amely tartalmazza a virtuális környezethez és a Blueprint telepítéséhez szükséges fájlokat, valamint az alkalmazás futtatásához szükségeseket.

2.3.2. Virtuális Környezet telepítése teszteléshez

A virtuális környezetet, az előfeltételek teljesítése után lehet telepíteni. A *Virtual_ENV* mappában megtalálható minden, amire ehhez szükség lesz. A mappa tartalmaz egy inicializációs konfigurációt, egy előre generált SSH kulcsot a virtuális gépek jelszó nélküli eléréséhez és a telepítő batch szkriptet. A *setup_virtual_environment.bat* batch fájl futtatásával felépül a virtuális környezet. Az adott batch fájl kétféleképpen is futtatható.

Megkeressük a *Virtual_ENV* mappában és jobb klikk után kiválasztjuk a „Futtatás adminisztrátorként” lehetőséget, majd ezt engedélyezzük a felugró ablakban. A második lehetőség, hogy indítunk egy terminált (command prompt) és a mappába navigálás után a *./setup_virtual_environment.bat* parancs kiadásával elindítjuk. Miután terminált a folyamat találunk négy virtuális gépet a rendszerünkön. Egy Mosquitto Brókert, ez fog a kommunikációért felelni, egy Control Node-ot, ami a folyamatot fogja irányítani és két felismerőt, Node1 és Node2, akik a felismerési folyamatért lesznek felelősek. Ezt tudjuk ellenőrizni a *multipass list* parancs parancssorból történő kiadásával.

```
C:\Users\Benedek>multipass list
Name                State      IPv4          Image
ControlNode         Running    172.22.222.155 Ubuntu 22.04 LTS
MosquittoBroker     Running    172.22.215.149 Ubuntu Mosquitto Appliance
Node1                Running    172.22.221.136 Ubuntu 22.04 LTS
Node2                Running    172.22.209.218 Ubuntu 22.04 LTS
```

1. ábra A *multipass list* parancs eredménye a virtuális környezet telepítése után

Az ábrán látszik, hogy létrehoztunk négy virtuális gépet. (1. ábra A *multipass list* parancs eredménye a virtuális környezet telepítése után) Az első oszlop tartalmazza a nevüket, a második, hogy aktívak és működnek, a harmadik az IP címüket és a negyedik, hogy milyen szoftver verzió futnak.

2.3.3. Blueprint telepítése

A Blueprint telepítéséhez szükségünk van egy fájlra, ami függőleges vonalakkal (pipe szeparátor; |) elválasztva tárolja az összes Node felhasználónevét, IP címét és ha van, jelszavát. Ennek a fájlnak a *Blueprint/nodelist.txt* útvonalon kell elhelyezkednie. Ez olyan funkciót tölt be, hogy a Blueprint-et telepítő szkript innen tudja az SSH-hoz szükséges adatokat automatikusan beolvasni, majd később, az alkalmazás futtatásakor újra felhasználni megkímélve ezzel a felhasználót attól, hogy ezeket minden futáskor manuálisan meg kelljen adnia. Ezt a szöveges fájlt ki lehet tölteni manuálisan az 1. ábra A *multipass list* parancs eredménye a virtuális környezet telepítése után alapján, vagy automatikusan a *Testing/test_virtual_env.py* Python szkript futtatásával.

```
ubuntu|172.22.215.149|pwd
vmuser|172.22.222.155|
vmuser|172.22.221.136|
vmuser|172.22.209.218|
```

2. ábra *nodelist.txt* tartalma virtuális környezet esetén

Az első sorba kerülnek a Mosquitto Bróker adatai, a másodikba a Control Node adatai és ez után következik annyi felismerő Node adata, ahány a rendelkezésünkre áll. Ez a 2. ábra *nodelist.txt* tartalma virtuális környezet esetén látható konfiguráció esetén kettő.

Miután a *nodelist.txt* szöveges fájlt kitöltöttük, a *Blueprint* mappában megtalálható a *setup.bat* batch fájl, ami felelős a Blueprint telepítéséért a megadott virtuális vagy fizikai eszközeinken. Amikor ezt futtatjuk, feltelepíti a megfelelő modulokat és Python könyvtárakat a megfelelő számítógépekre, majd terminál. Ennek a kódnak a futási ideje hosszúra nyúlhat, mivel több különböző nagyméretű telepítést hajt végre minden egyes felismerőn annak érdekében, hogy az alkalmazás akadály nélkül fusson.

2.3.4. Zenefelismerő használata

A zenefelismerő alkalmazást a Blueprint telepítése után lehet használni. Ezt olyan módon tehetjük meg, hogy futtatjuk az *Applications/full_lampion.bat* batch szkriptet. Ekkor elindul a felhasználói felület, amely különböző kérdések alapján összegyűjti, hogy milyen hanganyagot, hogyan szeretnénk felismerni. Az első kérdés, hogy hány hangszerből áll a felismerni kívánt dallam. Ez látható a 3. ábra Hangszerek számának bekérése a felhasználtól.

```
WELCOME TO LAMPION - A MULTI PITCH SOUND RECOGNITION SOFTWARE
-----
IN THE FIRST STEP YOU CAN SET UP EACH INSTRUMENT INDIVIDUALLY
or
YOU CAN USE A GENERAL INSTRUMENT FOR A QUICKER SETUP BUT GENERALLY WORSE RESULTS
PLEASE ENTER THE NUMBER OF INSTRUMENTS [Integer]:
█
```

3. ábra Hangszerek számának bekérése a felhasználtól

A hangszerek száma nem lehet több a felhasználó számára elérhető virtuális gépek számánál. Ennek az az oka, hogy egy felismerő egy szólamot tud egyszerre feldolgozni, így lényegében minden felismerőhöz hozzárendelünk egy dallamot. Ezután a felhasználónak meg kell adnia,

az adott hangszerek paramétereit. Ezek a következők: a hangszer neve, a hangszer által játszott dallam skálája és a hangfájl útvonala a személyi számítógépen. A hangszer neve bármi lehet, viszont a skálája ennél specifikusabb. Van két alapértelmezett opció, viszont lehetőség van egyedi skála megadására is. Ha a skála kromatikus („chromatic”) akkor az ['A', 'A#', 'B', 'C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#'] hangokat, ha pentatonikus („pentatonic”) akkor a ['A', 'C', 'D', 'E', 'G'] hangokat és ha egyedi akkor a felhasználó által megadott hangokat fogja felismerni az alkalmazás. Miután egy hangszer minden paraméterét megadta a felhasználó, az alkalmazás visszakérdez meg egyszer, hogy erősítse meg, vagy vesse el a konfigurációt, mivel ezt nem lesz lehetőség a későbbiekben módosítani. Az alkalmazás addig ismételi egy kérdést, amíg arra helyes paramétert vagy olyan fájlt nem kap, ami létezik. Egy kromatikus skálájú cselló paraméterezése látható a 4. ábra Hangszer paramétereinek beolvasása a felhasználotól.

```
INSTRUMENT 1 SETUP
-----
PLEASE ENTER THE NAME OF THE INSTRUMENT
Cello
PLEASE ENTER THE SCALE OF THE INSTRUMENT[pentatonic,chromatic,custom]
chromatic
PLEASE ENTER THE LOCATION OF THE WAV FILE (relative to the pitches folder):
../../pitches/beres
The file does not exist. Please enter a valid file location.
PLEASE ENTER THE LOCATION OF THE WAV FILE (relative to the pitches folder):
../../pitches/beres
The instrument defined is Instrument(name=CELLO, id=0, scale=CHROMATIC, pitch_file=../../pitches/beres.wav)
CONFIRM THE INSTRUMENT SETUP! THIS CANNOT BE CHANGED LATER [Y/N]
```

4. ábra Hangszer paramétereinek beolvasása a felhasználotól

Miután a felhasználó minden hangszert megadott és elfogadott, a hanganyagok kiküldésre kerülnek a felismerőkhöz, amelyek feldolgozzák majd továbbítják a kész hanganyagot a Control Node-nak, aki összeállítja a végső kottát. A kész musicxml formátumú kotta megtekinthető a Control Node `/home/vmuser/Szakdolgozat_v2/Application/musescore` mappájában.

A következőkben egy futási eredmény látható különböző paraméterekkel. Minden esetben megegyezett a népdal és a hangszer viszont változott a skála paraméter. Az 5. ábra "chromatic" paraméterrel futtatott felismerőán látható, hogy amikor kromatikus paraméterrel ismerjük fel a dallamot, akkor a skála összes hangja elérhető a felismerő számára.



5. ábra "chromatic" paraméterrel futtatott felismerő

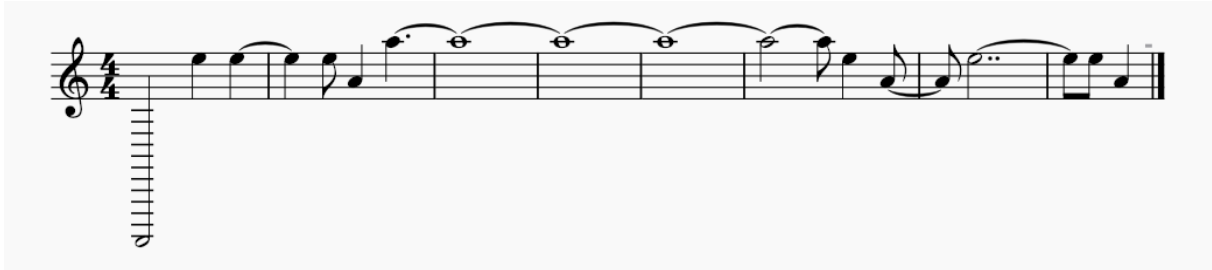
A 6. ábra "pentatonic" paraméterrel futtatott felismerő megfigyelhető, hogy amennyiben pentatonikus skálát adunk meg bemeneti paraméterként, a felismert hanganyagban kizárólag a skála hangjai jelennek meg. Nevezetesen az A, C, D, E és a G.



6. ábra "pentatonic" paraméterrel futtatott felismerő

A „custom” kulcsszó használatával ez tovább finomítható. Ennek abban az esetben van értelme, ha a felismerés előtt pontosan tudjuk, hogy milyen hangokból áll a dallam. Ekkor tudjuk biztosítani, hogy a kottában valóban csak azok a hangok jelenjenek meg amelyeket elvárunk. A 7. ábra ['A', 'E'] paraméterrel futtatott felismerőn látható, hogy mi történik akkor, ha az előre definiált skálák helyett egy személyre szabottat használunk amely csak az „A” és

„E” zenei hangokat tartalmazza. Megfigyelhető, hogy az így keletkezett kottában kizárólag az adott hangok találhatóak meg.




7. ábra ['A', 'E'] paraméterrel futtatott felismerő

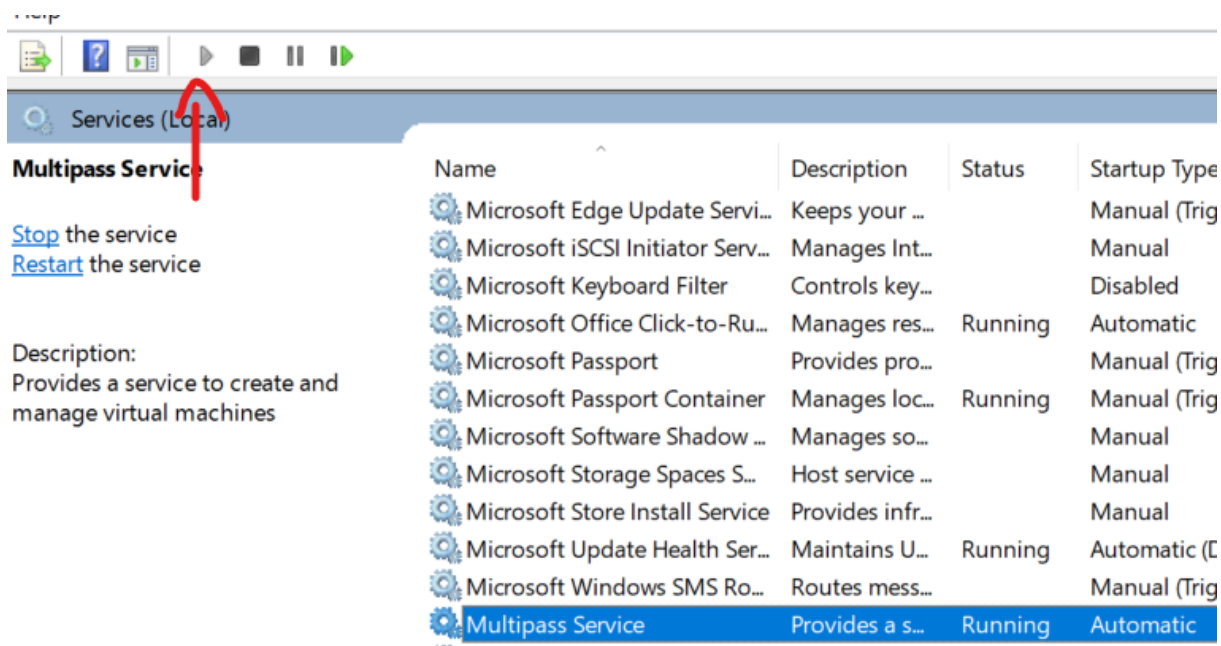
2.4. Futás közben adódó tipikus hibák megoldása

A következő szekcióban található a futás közben adódó tipikus hibák leírása és azok megoldása.

2.4.1. A Multipass service nem elérhető

Amennyiben a Multipass service nem elérhető a felhasználó személyi számítógépén, nem fog elindulni a virtuális környezet. Ennek a leggyakoribb oka, hogy a Multipass service nem indult el, vagy valamilyen okból kifolyólag megállt. Ezt a következő képen lehet újraindítani.

-  + r felhossa a Run ablakot
- services.msc + Enter
- Multipass Service kiválasztása és a bal felső sarokban lévő nyíl megnyomása (8. ábra Multipass Service elindítása)



8. ábra Multipass Service elindítása

2.4.2. A tesztelő SSH hibával tér vissza

Amennyiben az automatikus tesztek futtatása közben SSH hibával találkozunk, az valószínűleg annak a következménye, hogy a személyi számítógép még nem tárolta el az ismert IP címek között a virtuális vagy fizikai gép IP címét. Ha SSH kapcsolatot létesítünk manuálisan az adott géppel, akkor ez a cím automatikusan hozzáadásra kerül az ismert címekhez és megszűnik a probléma.

3. Fejlesztői dokumentáció

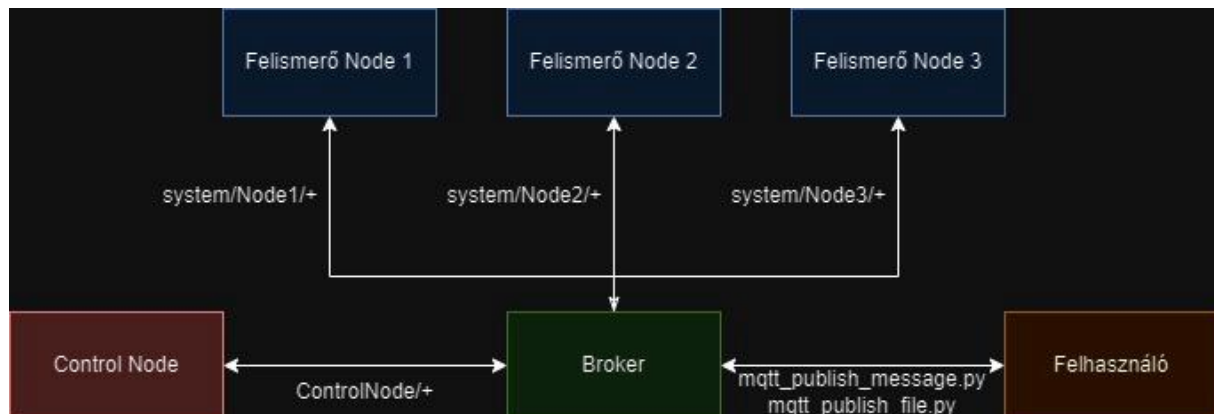
3.1. A probléma specifikálása

A mesterséges intelligencia alapú felvételtől történő zenefelismerés egy kifejezetten körülményes, lassú és legfőképpen nehéz feladat. Telepíteni kell hozzá egy megfelelő környezetet, le kell tölteni a hozzá szükséges modulokat, választani kell egy programozási nyelvet, abban implementálni az alkalmazást, tesztelni és kompatibilitás ellenőrizni majd futtatni. Ez a ma, forgalomban lévő személyi számítógépeken legjobb esetben is körülményes, mivel egy Virtuális Géphez, amin TensorFlow fut, a minimum ajánlott RAM 8Gb. (3) A Trend Force piackutató cég szerint a szakdolgozat írásakor (2024 Q2 - Q4) az átlagos, személyi számítógépekbe helyezett RAM 11.8 Gb, ami nem elegendő két párhuzamosan futó felismerő megfelelő működéséhez. (4) A szakdolgozatom ezekre a problémákra ajánl megoldást, egy olyan Blueprint (Lásd 3.6. fejezet) formájában, ami bizonyos előfeltételek teljesítése után képes egy kattintásra felépíteni egy olyan MQTT hálózatot, ahol a személyi számítógép helyett virtuális, vagy fizikai gépeken, párhuzamosan történik a felismerés. Ezek a gépek rendelkezhetnek olyan erőforrásokkal melyek nagyban meghaladják a személyi számítógép teljesítményét, ezáltal felgyorsítva a felismerési folyamatot. Jelen esetben egy olyan Blueprint-re van szükségünk, amely a számára elérhető fizikai vagy virtuális gépeken létrehoz egy Brókert, egy Control node-ot és legalább egy felismerőt. (9. ábra: Az alkalmazás MQTT hálózatának felépítése) Ez mindegyikre telepíti az MQTT-hez és az alkalmazás futtatásához szükséges modulokat, majd konfigurálja azokat és terminál. Ez után már futtatható lesz az alkalmazás.

3.2. MQTT

Az MQTT (Message Queuing Telemetry Transport) egy üzenet küldő protokoll, ami a dolgok internetéhez (IOT) lett kifejlesztve. A fizikai architektúrája kliensekből áll, amelyek feliratkoznak vagy publikálnak egy témára és egy brókerből, aki megkapja az üzeneteket, témákba rendezi őket és továbbítja azokat a megfelelő helyre. A témák mappa struktúra szerűen viselkednek, ami azt jelenti, hogy egy fő témának lehetnek altémái. Például a *messageing/Node1/wavfile* témánál egy kétszintű beágyazódás figyelhető meg. Az üzenetek mérete kicsi, aminek következtében távoli gépeket tud összekötni minimális erőforrás és sávszélesség felhasználásával. Az alkalmazásban egy bróker köt össze több klienst. Ezek a

kliensek a felhasználó, aki elindítja a felismerési folyamatot, a felismerő node-ok, akik felismerik a hanganyagokat és egy Control node, aki összegyűjti a beérkező szövegeket, kódba rendezi őket majd eltárolja a végeredményt. A 9. ábra: Az alkalmazás MQTT hálózatának felépítéseán megfigyelhető, hogy a hálózat elemei milyen témákon kommunikálnak egymással.



9. ábra: Az alkalmazás MQTT hálózatának felépítése

Mivel téma alapú a publikációs séma és egy témára bárhány kliens publikálhat, így a feldolgozott szövegeket eljuttatni a control node-nak egy kis sávszélességet igénylő, alacsony rendszerterhelésű feladat. Ezáltal a felismerő node-oknak, amelyek gyakran limitált erőforrású virtuális gépeken futnak, több erőforrása marad az alkalmazás futtatására, mintha például egy tradicionálisabb HTTP protokollon történne a kommunikáció. (5) A hálózathoz csatlakozó eszközök száma nincs meghatározva, ami előnyös egy olyan alkalmazás esetében, ami a feladatokat több felismerő között osztja szét, mivel a kommunikációs protokoll nem szab fizikai korlátot a node-ok lehetséges számának. A beépített QoS (Quality of Service) biztosíthatja az üzenetek megbízható megérkezését. Ennek három szintje van. „0”, amikor egy üzenet maximum egyszer lesz elküldve, „1”, amikor egy üzenet legalább egyszer lesz elküldve és „2”, amikor az üzenet biztosan egyszer lesz elküldve. (6)

3.3. Crepe

A CREPE egy monofónikus hangmagasság meghatározó, amely mély konvolúciós neurális hálózaton alapul és közvetlenül idő-doménú hanghullám bemeneten dolgozik. A konvolúciós hálót többféle zenei hangon tanították, jól teljesít zajos környezetben, nyílt forráskódú és

képes valós időben hangot felismerni. Ezen tulajdonságok alkalmassá teszik egy olyan alkalmazáshoz, ahol ugyanannak a Python kódnak kell felismernie különböző hangszerekből származó, valószínűsíthetően zajos, koncertfelvételeket. (7)

3.4. Python használata a Blueprint és az alkalmazás megírásához

A Python egy magas szintű programozási nyelv, ami egyre populárisabb a fejlesztők körében. (8) Ezen kívül egy gazdag és jól karban tartott könyvtárral rendelkezik, amelyben szinte bármilyen feladathoz található egy annak elvégzésére való modul. Ennek következtében egy Blueprint-en belül a bróker, a felhasználói felület, a felsimerő és a kommunikációért felelős szkript mind használhatják ugyanazt a nyelvet. Ez megkönnyíti a telepítést mivel a különböző virtuális gépeken egyedül Pythonra van szükség, ami helytakarékos. A fejlesztés szempontjából is előnyös a keresztplatform támogatása miatt. A Windows operációs rendszeren megírt Python kód futtatható Linuxon és fordítva. (9)

3.5. Virtuális környezet (*Virtual_ENV/...*)

Az alkalmazás fejlesztése és tesztelése virtuális környezetben történt. Ennek több oka is volt. Az első és legfontosabb, hogy a valóságban távoli gépeken, szervereken futó alkalmazások leggyakrabban virtuális gépeken futnak, annak érdekében, hogy a rendszer erőforrásai megfelelően legyenek megosztva, illetve, hogy amennyiben egy alkalmazás futása közben probléma lép fel, az ne befolyásolja a többi folyamatot. A Facebook példának okáért az *Amazon Web Services* virtuális gépein futtat alkalmazásokat, többek között weboldalakot és felhasználói adatok tárolására szolgáló adatbázisokat. (10) Mivel az alkalmazásom alapból virtuális gépekre készül, így valódi szerverekre történő telepítéskor nem igényel további konfigurációt.

A második, hogy a teljes MQTT hálózat és a rajta futó alkalmazás futtatható és tesztelhető legyen egy személyi számítógépen. Ez mind a két folyamatot megkönnyíti, mivel nem kell fizikai hardvert vásárolni, a létrehozott virtuális gépeket gyorsan lehet módosítani, optimalizálni mind RAM, mind háttértár szempontjából. További előnye a módszernek, hogy megtartja a fizikai szeparációt a gépek között, így a környezet hasonló marad egy valódi fizikai gépeken létrehozott MQTT hálózathoz.

3.5.1. Multipass

A Multipass egy platform független eszköz, amivel felhő stílusú virtuális gépeket lehet létrehozni. (11) Azon kívül, hogy támogatja a személyes konfigurációt *cloud-init* interfészen keresztül, egy nagy kép könyvtárral érkezik, amiből szabadon lehet különböző célokra virtuális gépet választani. Erre egy kiváló példa a Mosquitto Appliance, ami egy Mosquitto brókerként működő, előre elkészített virtuális gép. Olyan VM-eket hoz létre, amelyek nem befolyásolják a fizikai gépet, ennek következtében kiválóan lehet velük modellezni egy valós alkalmazás fizikai szerveren történő működését.

3.5.2. SSH kulcsok generálása

A virtuális környezet létrehozásának első lépése egy SSH kulcs generálása. Ennek az a funkciója, hogy jelszó nélkül be lehessen lépni távolról a Node-okba. Ez azért fontos, mivel a Blueprint telepítése közben a fizikai gépnek be kell tudnia lépni a virtuális gépekbe, annak érdekében, hogy különböző folyamatokat elindíthasson, mint például a megfelelő fájlok letöltése GitHub-ról, vagy az MQTT működéséhez szükséges hálózati engedélyek beállítása. Az SSH kulcs generálása a következő kódrészlettel történik.

```
ssh-keygen -C vmuser -f multipass-ssh-key
```

A *-C* kapcsolóval hozzáadunk egy *vmuser* kommentet. Ez lesz a felhasználónév, amit az SSH-hoz használni fogunk, a *-f* kapcsolóval pedig megadjuk, hogy a kulcs milyen nevű fájlban legyen tárolva.

3.5.3. Cloud-Init.yaml fájl létrehozása

A Cloud-init egy standardizált megközelítés a cross-platform virtuális gépek konfigurációjának megadására. A Multipass, képes .yaml fájlból beolvasott konfiguráció alapján létrehozni egy virtuális gépet, így amennyiben átadjuk az általunk generált SSH kulcsot, azzal lehetővé tesszük, hogy jelszó azonosítás nélkül beléphessünk a kívánt virtuális gépbe.

```
users:  
  - default  
  - name: vmuser  
  sudo: ALL=(ALL) NOPASSWD:ALL  
  ssh_authorized_keys:  
    - <ssh key>
```

1. Forráskód Részlet A Cloud-Init.yaml fájl tartalma

A *.yaml* fájl az *users* kulcsszóval definiálja azokat a felhasználókat, amelyeket létre kell hozni a gépen. A *-default* azt jelzi, hogy alapértelmezett felhasználót kell beállítani, ami a *multipass*-al létrehozott virtuális gép esetében Ubuntu. A *-name* adja meg az új felhasználó nevét. Ennek a névnek meg kell egyeznie az *ssh* kulcs generálásakor megadott névvel, vagyis amennyiben a „*vmuser*” nevet adtuk meg kommentben a *-C* kapcsolóval az *ssh-keygen*-nek, akkor most is a „*vmuser*”-t kell használni. A *sudo: ALL=(ALL) NOPASSWD:ALL* jelszó nélküli *sudo* jogot ad a felhasználó számára, ami azt jelenti, hogy bármelyik felhasználó nevében futtathat parancsokat a rendszeren, jelszó megadása nélkül. Az *ssh_authorized_keys* tartalmazza a publikus kulcsot, amely megadásával a felhasználó SSH hozzáférést nyerhet a virtuális géphez. Amennyiben az SSH kulcsot a *-f multipass-ssh-key* kapcsolóval definiáltuk, akkor a *multipass-ssh-key.pub* fájl tartalmát kell beilleszteni ide.

3.5.4. A virtuális gépek létrehozása és konfigurálása (*setup_virtual_env.bat*)

A virtuális gépek létrehozása és konfigurálása a *Virtual_ENV/setup_virtual_env.bat* batch fájl futtatásával történik.

3.5.4.1. A bróker node létrehozása (*mosquitto_broker_setup.sh*)

A *multipass*-ban egy előre elérhető appliance a Mosquitto Ubuntu alapú virtuális gép, amely indítás után képes üzeneteket küldeni és fogadni az MQTT protokollon keresztül. Először ellenőrizzük, hogy már létezik-e a Mosquitto virtuális gép, és amennyiben nem, létrehozzuk. A *-n* kapcsolóval lehet nevet adni a VM-nek. Ellenkező esetben elindítjuk az előre létrehozott példányt.

```
multipass list | findstr /C:"MosquittoBroker" >nul
if %errorlevel% neq 0 (
    multipass launch appliance:mosquitto -n MosquittoBroker
    ...
) else (
    multipass start MosquittoBroker
)
```

2. Forráskód Részlet A Broker Node létrehozása

A bróker bár fut és pingelhető, a feladatát még nem képes ellátni, mivel nincs alapértelmezetten definiálva a hallgató port, nem biztos, hogy az összes IP címre hallgat.

```
#!/bin/bash

sudo cp /var/snap/mosquitto/common/mosquitto_example.conf
/var/snap/mosquitto/common/mosquitto.conf

echo "listener 1883 0.0.0.0" | sudo tee -a
/var/snap/mosquitto/common/mosquitto.conf > /dev/null

echo "allow_anonymous true" | sudo tee -a
/var/snap/mosquitto/common/mosquitto.conf > /dev/null

sudo systemctl restart snap.mosquitto.mosquitto.service

echo "ubuntu:pwd" | sudo chpasswd
```

3. Forráskód Részlet A Mosquitto Broker konfigurálása

Ezeknek a problémáknak a kijavítására bemásoljuk a `mosquitto_example.conf` fájlt a `mosquitto.conf` helyére, amivel alapértelmezett konfigurációt hozunk létre a Mosquitto számára. Ezután hozzáfűzzük a `listener 1883 0.0.0.0` sort a `mosquitto.conf`-hoz. Ez megadja, hogy a bróker az összes IP címre hallgasson a 1883-as porton. Engedélyezzük az anonim hozzáférést az `allow_anonymous true` konfigurációs fájlba másolásával, hogy a brókerhez való csatlakozás során ne kelljen a node-oknak azonosítani magukat. Ez olyan megfontolásból történt, hogy semmilyen nagy kockázatú adat nem megy keresztül a hálózaton, így a biztonsági kockázat elhanyagolható. Ahhoz, hogy a módosítások érvénybe lépjenek, újra kell indítani a Mosquitto szolgáltatást, ami a `sudo systemctl restart snap.mosquitto.mosquitto.service` parancs futtatásával lehetséges. A Mosquitto kommunikációhoz, bár nem kapcsolódik, viszont hasznos itt beállítani, az alapértelmezett felhasználó jelszavát. Ez arra szolgál, hogy a későbbiekben, amennyiben szükséges, be lehessen lépni SSH kapcsolattal a virtuális gépre. Az előbb említett konfigurációs lépések a `mosquitto_broker_setup.sh` shell szkriptben találhatóak, amelyet a bróker VM indítása után rámásolunk, engedélyt adunk a futására, majd futtatunk.

```

multipass transfer mosquitto_broker_setup.sh
MosquittoBroker:/home/ubuntu/mosquitto_broker_setup.sh

multipass exec MosquittoBroker -- chmod +x
/home/ubuntu/mosquitto_broker_setup.sh

multipass exec MosquittoBroker -- /home/ubuntu/mosquitto_broker_setup.sh

```

4. Forráskód Részlet A *mosquitto_broker_setup.sh* futtatása

3.5.4.2. A Control Node és felismerő node-ok létrehozása

A virtuális környezet létrehozásakor a különböző node-oknak nincs semmiféle alapvető feladata azon kívül, hogy az előfeltételeknek megfelelő konfigurációval induljanak el. Ez ebben az esetben azt jelenti, hogy megfelelő fizikai paraméterekkel rendelkezzen, lehessen rá telepíteni a megfelelő modulokat és lehessen vele SSH kapcsolatot létesíteni. Bizonyos Python könyvtárak használata esetében ez különösen fontos, mivel például a TensorFlow-hoz, ami kritikus része a zenefelismerő alkalmazás működésének, az ajánlott minimális memória 8 GB.

(3) A személyi számítógépen erre nincsen lehetőség, mivel ez már két hangszer esetén is több memóriát igényelne, mint amennyi rendelkezésemre áll. Az előbb említett limitáció következtében kénytelen voltam kikísérletezni a „*trial and error*” módszer alapján, a minimálisan szükséges RAM-ot, ami bár nem elég az optimális működéshez, de megfelelő tesztelésre. Ez a szám a lapozási mechanizmus engedélyezését követően két gigabájt lett. Az előbbieken felsorolt kritériumokat figyelembe véve a node-ot a következőképpen kell elindítani:

```

multipass launch jammy -n %node1% --cloud-init cloud-init.yaml --disk 10G
--memory 2G

```

5. Forráskód Részlet A *%node1%* virtuális gép indítása a megfelelő paraméterekkel

Ahol a *-n* kapcsoló után kell megadni a virtuális gép nevét, a *--cloud-init* után a 3.5.3 szekcióban említett konfigurációs fájlt, a *--disk* után a háttértár méretét és a *--memory* után a memória méretét.

3.6. Blueprint (*Blueprint/...*)

A „Blueprint” azaz tervrajz általában egy olyan vázlatot jelent az informatikában, amely egy rendszer, szoftver vagy alkalmazás struktúráját és működését írja le. Ez a gyakorlatban annyit jelent, hogy egy ismeretlen, előre nem definiált környezetben, automatikusan felállítja egy

adott alkalmazás működéséhez szükséges kommunikációs hálót és telepíti a rendszer főbb komponensein a megfelelő modulokat és eszközöket, majd konfigurálja azokat.

Léteznek ugyan nyílt forráskódú, mindenki számára elérhető Blueprintek, mint például az Akarino projekt, viszont ezek a szoftverek nagyon ritkán vannak frissítve, aminek következtében, a használatuk legjobb esetben is körülményes. (12) Bizonyos bennük használt modulok mára már elérhetetlenek, vagy teljesen máshogy működnek, mint ahogyan eredetileg. Ezt mutatja be a második ábra, ahol különböző Akarino Blueprintek kerültek összehasonlításra adott szempontok alapján. Észrevehető, hogy bizonyos Blueprintekből a legutolsó kiadás is legalább kétéves a szakdolgozat írásának pillanatában, ezen felül az adott verzió dokumentációja csak egy referencia egy régebbire, ami akár 6 – 8 éves is lehet és javarészt elavult. (10. ábra: Már létező Akarino Blueprintek összegzése különböző paraméterek szerint) Ennek következtében amennyiben egy kommunikációs hálót, vagy egyedi igényű futási környezetet szeretnénk Blueprint segítségével implementálni, jobban járunk, mind idő, mind energiabefektetésileg, ha megírjuk a sajátunkat.

	uCPE R6	5g MEC/Slice System	Enterprise Applications on Lightweight 5G Telco Edge (EALTEdge)
Releasek Száma	6	6	6
Utolsó release dátuma(Legutolsó installation guide)	2020.jun.5	2020.aug.14	2022.apr.28
Architektúra	ELIOT Manager bárhány uCPE Edge Node	Edge Connector, Edge gateway	Centre Node (Telco Cloud), Edge node (Telco Edge)
Mi kell a kapcsolathoz	Internet és network kapcsolat az ELIOT Manager és uCPE node-ok között (Installációhoz sshpass) Nodelist file-ba elvárás a node ip címe valamint az ssh-hoz szükséges belépési adatok	Az Edge Controller és a Edge Node-ok hosztjainak megfelelő és egyedi hostnévvel kell rendelkezniük. Ezt az állománsnevet az /etc/hosts állományban kell megadni. Az Ansible leltárt be kell állítani. Proxy-t kell konfigurálni, ha szükséges. Ha privát tárolót használunk Github tokenet kell beállítani.”	„virtuális gépek előtelepített Ubuntu 18.04-MECM csomóponthoz. Ubuntu 18.04-gyel előtelepített virtuális gépek a MEC Host Node-ok számára root felhasználó létrehozása a telepítési csomópontban, a MEC-csomópontban és a MEC host-csomópontban. SSH-kiszolgáló fut az összes csomóponton. Ansible > 2.10.7 telepítve a One Click Deployment Node-ban (Jump Host). git telepítve a Jump Hostban.”
Fizikai architektúra	Hivatalosan Huawei Public Cloud Virtual Machines gyakorlatban Baremetal-on is	Virtuális és Fizikai gépek	Hivatalosan Huawei Public Cloud Virtual Machines gyakorlatban Baremetal-on is
Skálázhatóság	ELIOT Manager 1 Eliot uCPE node 1 - x (nincs elméleti felső korlátja) Az ELIOT architektúra egy ELIOT-menedzserből és több ELIOT-csomópontból áll. Az ELIOT (edge) csomópontok skálája 1 egyetlen csomóponttól 10, 100, 10K vagy több csomópontig terjedhet.	Csak egy Control node támogatott Nyilvános repo, üzemeltetői hálózat (Vw Mobilhálózat Exposed Capabilities) Az edge connector ehhez csatlakozik. UPF/SWG <-Edge forgalom-> Edge GW a 4G/5G hálózat széléhez közel telepítve, hogy lehetővé tegye a forgalom tehermentesítését az adatsík szempontjaitól a helyi forgalomirányítással,	Egy központi nóde, Elméletben végtelen Edge Node
Alkalmazott software (kliens - szerver architektúra)			CENTER Node --> MECM , AppStore and Developer Platform, Edge Node --> aPaaS, PaaS, MEP Server
Alkalmazott szoftverek (dependenciák)	Docker, k8s(kubeadm, kubelet, kubectl), Prometheus, Cadvisor	Golang, Ginko, Docker, Docker-compose	Docker, k8s, Edge Gallery
Technikai korlátok	A Master és Slave nodeoknak azonos hálózaton kell lenniük	controller_group csak 1 hosztot tartalmazhat. Az edgenode_group tartalmazza a Kubernetes munkásokként / Edge Gatewayként beállítandó hostokat.”	Amennyiben az előfeltételek teljesülnek, nincs elméleti limitáció „A tervezet egy kattintással telepíti az EALTEdge tervezet összetevőit. EALTEdge telepítési mód: EALTEdge 2 telepítési módot támogat: Multi Node és All-In-One (AIO) Node telepítés. AIO (All in One) üzemmód: Ebben az üzemmódban mind a 3 csomópont (OCD, amely a telepítési csomópont, a központ csomópont és az élcsoomópont)
Telepítés elméletben	nodelist fájl kitöltése a nodeok fizikai adataival majd setup shellscript futtatása	Egy kattintásos telepítés, abban az esetben, ha minden előfeltétel teljesül	
Telepítés gyakorlatban	A setup shellscript deprecated, mind a docker mind a k8s installáció hibás és újrairandó	-	Egy kattintásos telepítés
Security	Nincs extra biztonsági lépés az alapvető hálózati kommunikáció és k8s biztonsági intézkedésein kívül	Nincs extra biztonsági lépés, a hálózatra alapvetően beépítetten kívül	szolgáltatások és a gyártó alkalmazások számára. Vault: Vault: A Vault a titkok biztonságos elérésének eszköze. A titok bármely, amihez a hozzáférést szigorúan ellenőrizni szeretné, például API-kulcsok, jelszavak, tanúsítványok és más. A Vault egységes felületet biztosít bármely titokhoz, miközben szigorú hozzáférés-ellenőrzést és részletes ellenőrzési naplót rögzít.
Dokumentáltság	Az R2 jól dokumentált mind architektúra, mind telepítés szempontjából, viszont azóta az újabb releasek csak visszareferálnak erre	Az R3 nagyon jól dokumentált, de a következő kiadások mind erre a kiadásra utalnak vissza, azaz nem modernizálták őket.	Jól dokumentált az utolsó kiadás is

10. ábra: Már létező Akraino Blueprintek összegzése különböző paraméterek szerint

Bizonyos technikák azonban elleshetők és használhatóak az előbb említett konstrukciókból. Ilyen például a *nodelist.txt* ami egy pipe („|”) szimbólummal elválasztva tartja számon az összes komponens felhasználónevét, IP címét és amennyiben van, jelszavát. Ennek a fájlnek a létezése nagyban elősegíti a skálázhatóságot mivel azzal szemben, hogy minden node-ra

egyesével telepítenénk a kívánt modulokat, csak egy ciklus segítségével végig iterálunk a fájlon és végrehajtjuk a kívánt műveleteket minden elemen.

3.6.1. Control Node konfigurálása (*control_node_setup.sh*)

A Control Node-nak az a feladata, hogy fogadja a felismerők által feldolgozott adatokat, majd azokat feldolgozza és tárolja. Mivel az üzeneteket MQTT-n keresztül kapja, így szükség van olyan kliensprogramokra, illetve Python könyvtárakra, amelyek képessé teszik ezek küldésére és fogadására. Ilyen a *mosquitto-clients* és a *paho-mqtt*. A feldolgozni kapott adatokat *musicxml* formátumban kapja, és a Python programozási nyelv segítségével dolgozza fel, így szükség van a *music21* könyvtárra is, ami egy Python alapú eszköztár számítógépes zeneelmélethez. (13)

Annak érdekében, hogy a telepítési folyamat skálázható legyen akár távoli, vagy fizikai szerverekre a Multipass beépített shell parancsa helyett, ami csak lokálisan működik, SSH-val lépünk be a node-okba. A *nodelist* szöveges fájlból beolvassuk a Control Node IP címét és a hozzá tartozó felhasználónevet, majd a Windows-ba beépített SSH parancs segítségével belépünk távolról. (14) Ezután, ha már létezik, töröljük, majd GitHub-ról letöltjük a Blueprint legújabb változatát, ami tartalmazza mind a telepítőket, mind a futtatható alkalmazásokat. Ezután belépünk a Blueprint mappába, engedélyt adunk a *control_node_setup.sh*-nak a futásra és futtatjuk. Ez telepíti az előző bekezdésben említett szükséges kliensprogramokat és Python könyvtárakat.

```
ssh %a@%b -i ../Virtual_ENV/multipass-ssh-key "if [ -d 'Szakdolgozat_v2' ];  
then rm -rf Szakdolgozat_v2; fi && git clone %REPO_URL% && cd Szakdolgozat_v2  
&& chmod +x Blueprint/control_node_setup.sh &&  
./Blueprint/control_node_setup.sh"
```

6. Forráskód Részlet a *control_node_setup.sh* futtatása

3.6.2. Node-ok konfigurálása (*node_setup.sh*)

A felismerő node-ok konfigurálása hasonló a control node-hoz abból a szempontból, hogy a *nodelist* szöveges fájlból kiolvassuk a node IP címét és felhasználónevét majd SSH kapcsolattal belépünk rá, letöltjük GitHub-ról a Blueprintet majd végrehajtjuk a *node_setup.sh*-t. Azonban node-okból bármennyi lehet, aminek következtében az előbb említett folyamatot beágyazzuk egy ciklusba, ami végig iterál a hátralévő sorokon és mindegyikre végrehajtja a konfigurációs folyamatot. Ezen kívül a node-okon történik a zenék felismerése, Python programozási nyelvet

használva, aminek következtében a TensorFlow-t és a Crepe-t is telepíteni kell. Továbbá, annak érdekében, hogy a node ne fogyjon ki a memóriából a TensorFlow telepítése közben, a 3.5.4.2 szekcióban említettek alapján be kell állítani a lapozási mechanizmust.

```
#!/bin/bash
sudo fallocate -l 1G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile

sudo apt install mosquitto-clients -y

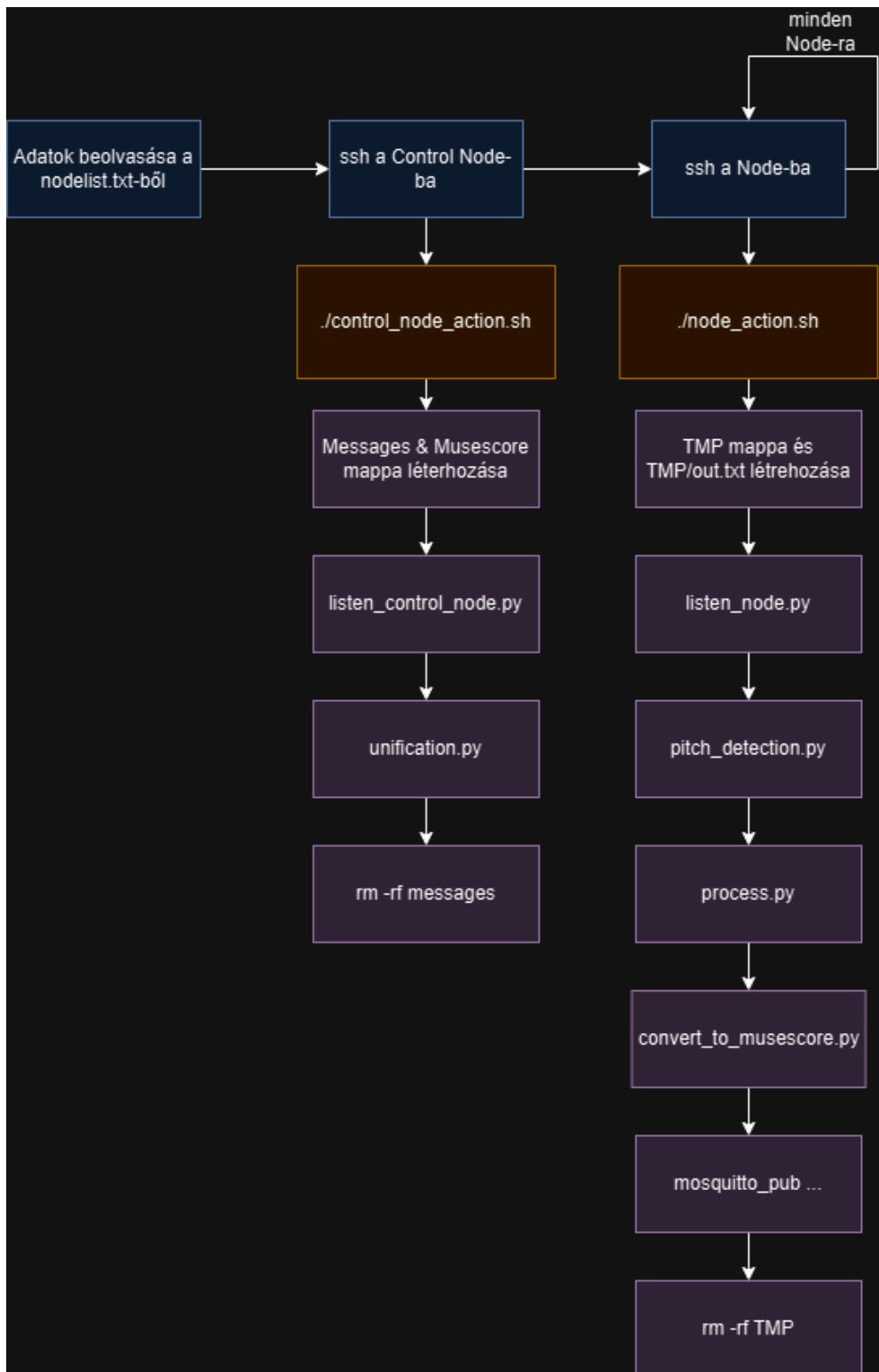
sudo apt install python3-pip -y

sudo pip install --upgrade tensorflow
pip install crepe
pip install music21
pip install paho-mqtt
```

7. Forráskód Részlet a *node_setup.sh* tartalma

3.7. Zenefelismerő alkalmazás (*Application/...*)

A zenefelismerő alkalmazás szétosztja a hangszereket és azok szólamát tartalmazó wav fájlokat a felismerő node-ok között, amelyek feldolgozzák és továbbítják az így keletkezett musicxml dokumentumot a Control node felé, aki ezeket egy végleges kottává olvasztja össze és tárolja. Ez után az alkalmazás terminál. Ezt a viselkedést olyan módon lehet elérni, hogy futtatjuk a *full_lampion.bat* batch szkriptet. Ekkor automatikusan elindul a *lampion.py* amely bekéri a felhasználótól a hangszerek adatait, majd tovább küldi őket feldolgozásra. A 11. ábra: Az alkalmazás híváslistája és a hívott szkriptek híváslistája szemlélteti, hogy milyen folyamatok játszódnak le a *full_lampion.bat* batch szkript futtatása közben. Először beolvasásra kerülnek a belépési adatok a *nodelist.txt*-ből, majd mind a Control Node-on, mind a felismerőkön végrehajtásra kerülnek az alkalmazás futásához szükséges szkriptek és utasítások. A mappák dinamikus létrehozásának és törlésének az a szerepe, hogy egy Node többszörösen is felhasználható legyen különböző bemenetekkel. Az ábrán említett Python szkriptek tartalma és feladata a következő fejezetekben kerül kifejtésre.



11. ábra: Az alkalmazás híváslistája és a hívott szkriptek híváslistája

3.7.1. Adatok bekérése a felhasználótól (*lampion.py*)

Az adatok bekérése a felhasználótól a *lampion.py* Python szkript futtatásával történik. A *lampion.py* fájl egy Python szkript, amely különböző eszközök (*instruments*) és csomópontok (*nodes*) közötti kommunikációt valósít meg MQTT protokoll segítségével. A szkript két fő

funkciót tartalmaz: az egyik a vezérlő csomópont (*Control Node*) inicializálása, a másik pedig az eszközök csomópontjainak indítása és a szükséges fájlok küldése. Ezeket a fájlokat a felhasználótól kéri be ellenőrzött módon. Először a hangszerek számát, majd az adott hangszerek nevét, skáláját majd azt, hogy hol található a felismerni kívánt .wav fájl. Ezt követően a felhasználónak meg kell erősítenie a hangszer paramétereit, mivel ezt visszamenőlegesen nem lehet módosítani. Az így bekért adatokat egy *Instrument* típusú objektumban tárolja minden hangszerhez.

```
temp_instrument = Instrument(instrument_name, i, instrument_scale,
wav_file_location)
```

8. Forráskód Részlet *Instrument* objektum létrehozása

Az *initiate_controll_node* függvény felelős a vezérlő csomópont inicializálásáért. Ez a függvény egy alprocesszt indít, amely a *mqtt_publish_message.py* szkriptet futtatja, és egy üzenetet küld a vezérlő csomópontnak arról, hogy hány eszköz van csatlakoztatva. (Lásd 3.7.2 fejezet) A függvényt egy külön szálban (*control_thread*) futtatjuk, hogy ne blokkolja a fő program futását.

```
def initiate_controll_node():
    subprocess.run(["python3", "mqtt_publish_message.py", broker_ip,
"ControlNode/initiate_control_node", str(len(instruments))])
```

9. Forráskód Részlet az *initiate_control_node()* függvény tartalma

A *start_instrument_node* függvény minden egyes hangszerhez node párhoz külön-külön fut le. Először egy üzenetet kerül küldésre az adott node-nak, hogy indítsa el az eszközt az előzőekben bemutatott módon. Ezután létrehoz egy ideiglenes fájlt, amely tartalmazza az hangszer nevét és skáláját, majd ezt a fájlt és az hangszerhez tartozó WAV fájlt is elküldi az adott node-nak. Végül egy üzenetet küld, hogy jelezze a felismerési folyamat inicializálásának végét.

A szkript több szálát használ a párhuzamos végrehajtáshoz. Minden felismerési folyamathoz létrehoz egy új szálát, amely a *start_instrument_node* függvényt futtatja. Ezeket a szálakat egy listában (*threads*) tárolja, és mindegyiket elindítja. A program minden szálát megvár (*join*), hogy biztosítsa, hogy minden felismerő inicializálva lett a Control Node inicializálásunk befejezése előtt.

```

for thread in threads:
    thread.join()

control_thread.join()

```

10. Forráskód Részlet Alfolyamatok bevárása

3.7.2. Üzenet és fájl küldése MQTT-n Python szkripttel (*mqtt_publish_message.py* és *mqtt_publish_file.py*)

Annak érdekében, hogy az alkalmazás minél könnyebben futtatható legyen különböző operációs rendszereken az MQTT-n történő üzenetek és fájlok küldése a felhasználó személyi számítógépén Python szkripttel lett megvalósítva. Az MQTT kliens kapcsolódik a szerverhez, majd közzéteszi a paraméterként kapott témán az üzenetet és lecsatlakozik.

```

client.connect(broker, port)

# Start the network loop
client.loop_start()

# Publish the message
result = client.publish(topic, message)

# Wait for the message to be published
result.wait_for_publish()

# Stop the network loop and disconnect
client.loop_stop()
client.disconnect()

```

11. Forráskód Részlet MQTT üzenetküldési logika

Fájl küldése esetén ez a folyamat annyiban módosul, hogy a fájl bináris módon kerül megnyitásra (*rb – read binary*) majd publikálásra.

```

with open(file_path, 'rb') as file: # Open the file in binary mode
    file_content = file.read()
    client.publish(topic, file_content)

```

12. Forráskód Részlet Fájl bináris módon történő olvasása és publikálása

3.7.3. Control Node működése az alkalmazás futása közben (*control_node_action.sh*)

A Control Node az alkalmazás indulásakor nem tudja, hogy hány darab feldolgozott musicxml fájlt vár, amelyek az egyesítésre váró szövegeket tartalmazzák. Ezt a számot, amit a továbbiakban hívunk „n”-nek ($n \in \mathbb{N}^+$) akkor fogja megkapni, amikor a felhasználó már

megadta és megerősítette a felismerni kívánt szölamokat. Ez után „n” darab musicxml fájlt vár, amiket ideiglenesen tárol, majd, amikor megérkezett az összes egyesít. Utolsó lépésben a kottát elmenti és törli a temporális fájlokat, majd terminál.

```
python3 listen_control_node.py  
  
python3 unification.py  
  
rm -rf messages
```

13. Forráskód Részlet Control Node hívási verme

3.7.3.1. Hallgatódzó mód (listen_control_node.py)

A hallgatódzási folyamat a *listen_control_node.py* Python szkript futtatásával valósul meg. Először ellenőrizzük, hogy a könyvtár ahová menteni szeretnénk az ideiglenes fájljainkat létezik-e. Amennyiben nem, létrehozuk. Ezután létrehozuk a *callback* függvényeket azokra az esetekre, amikor csatlakozunk, vagy üzenetet kapunk. A csatlakozás *callback* függvénye kiírja, hogy milyen visszatérési értékkel sikerült csatlakozni a szerverhez, majd feliratkozik a *ControlNode/+* témára. A „+” jel, jelen esetben egy *wildcard*, ami azt jelenti, hogy a *ControlNode* téma minden altémájára érkező üzenetet olvasson be.

```
TOPIC = 'ControlNode/+'  
...  
def on_connect(client, userdata, flags, rc):  
    print(f"Connected with result code {rc}")  
    client.subscribe(TOPIC)
```

14. Forráskód Részlet Kapcsolódás a „ControlNode” téma összes altémájára

A beérkezett üzenetek *callback* függvénye kétféle módon működik, amelyek között a „first” logikai változó igazra, vagy hamisra változtatásával tudunk kapcsolni. Az első üzenet előtt ez a logikai változó igaz, ami azt jelenti, hogy a *ControlNode/+* témáról egy számot szeretnénk beolvasni. Ez a szám lesz az „n”, amely meghatározza, hogy hány darab feldolgozott szölamot kell a továbbiakban várni. Ez után a „first” változó értékét hamisra állítjuk. Ez után „n” darab fájlt olvasunk ki a *ControlNode/+* témából és a tartalmukat a messages mappába mentjük. Annak érdekében, hogy minden szölamnak a neve egyedi legyen, a beérkezett musicxml fájloknak a téma nevét adjuk, amelyen érkeztek. Ennek az egyediségét a felismerő node-ok biztosítják azáltal, hogy a saját felhasználónevüket használják publikálásnál altémának. (Lásd 3.7.4.5 fejezet) A kívánt számú beérkezett fájl után a hallgatódzó módért felelős Python szkript

lecsatlakozik az MQTT hálózatról és terminál. A *number_of_messages* változó értéke megegyezik a még várt fájlok számával.

```
if number_of_messages == 0:
    print(f"Received all the message(s). Disconnecting...")
    client.disconnect()
```

15. Forráskód Részlet A minden üzenet megérkezett eset kezelése

3.7.3.2. A beérkezett feldolgozott anyagok egyesítése, tárolása és az ideiglenes fájlok törlése (*unification.py*)

A beérkezett musicxml fájlokat a *listen_control_node.py* a *messages* mappába írja, olyan módon, hogy a nevük egy egyedi azonosító, amely magába foglalja, hogy melyik node-ról érkeztek. Ezeket a fájlokat az *unification.py* segítségével beolvassuk, a music21 könyvtár segítségével ellenőrizzük, hogy csak egy szólamot tartalmaznak és amennyiben igen, akkor hozzáadja a *combined_score stream.Score()* típusú változóhoz. Amikor a *messages* mappa összes musicxml fájlján végigért a ciklus, a *combined_score* változóban tárolt teljes kotta tartalmát elmenti a *musescore* mappába és terminál. Ezután töröljük a *messages* mappa tartalmát, hogy felkészüljünk a következő kombinálási feladatra.

```
for file_path in file_list:
    # Read the MusicXML file into a Stream object
    try:
        part = converter.parse(file_path)

        # Ensure we're dealing with a single part, not a full score
        if isinstance(part, stream.Score):
            part = part.parts[0]

        # Append the part to the combined score
        combined_score.append(part)
```

16. Forráskód Részlet Szólamok kombinálása

3.7.4. A felismerő node működése az alkalmazás futása közben (*node_action.sh*)

A felismerő node feladata, hogy a beérkezett WAV formátumú hanganyagot a megadott paraméterek alapján, mint a hangszer és annak skálája, a Crepe mély neuronháló segítségével felismerje, feldolgozza, musicxml formátummá konvertálja, majd továbbítsa a Control Node felé az MQTT hálózaton keresztül. Mivel nem hozunk létre új node-okat minden feladathoz, így a node-oknak univerzális felismerőként kell funkcionálniuk, amelyek megfelelő

paraméterezés mellett képesek feldolgozni különböző hangszerekből és stílusokból származó hanganyagokat.

3.7.4.1. Hallgatódzó mód (*listen_node.py \$(username)*)

A node univerzális mivolta következtében szüksége van bizonyos paraméterekre a felismerési folyamat elvárt végrehajtásához. Ezek a paraméterek a hangszer neve, skálája és a felismerni kívánt WAV hangfájl. Ezeket az adatokat MQTT-n keresztül a felhasználótól kapja meg. A *node_action.sh* amit az alkalmazás indításánál futtatunk a felismerő node-on (3.7) először létrehoz egy ideiglenes mappát, azon belül az *out.txt* temporális kimeneti fájlt, majd elindítja a *listen_node.py*-t a *\$(hostname)* bemeneti paraméterrel. Ez azért fontos, mivel ezen a témán fog hallgatni a felismerő a bejövő üzenetekre, és mivel a node neve egy egyedi azonosító, ezzel tudjuk biztosítani, hogy ne keveredjenek össze az üzenetek. A node elkezd hallgatni a „*system/\$(username)/+*” témán és addig vár, amíg nem kap egy „start” üzenetet. Amikor ezt megkapta, az ezután következő üzeneteket fájlként tárolja az ideiglenesen létrehozott mappába (TMP) addig amíg nem érkezik egy „end”, ami után terminál. Annak érdekében, hogy a beérkezett fájlok helyes formátumban legyenek tárolva, az eredeti nevén létrehozott témában publikáljuk a tartalmát. Ezután a témát adjuk a fájl nevének a felismerő node-on. Például a „beres.wav” hangfájlt a „*system/Node1/beres.wav*” témán publikáljuk, amit aztán a node beolvas és a „*TMP/system_Node1_beres.wav*” néven ment.

```
topic = msg.topic
    payload = msg.payload
    file_path = os.path.join(SAVE_DIR, topic.replace('/', '_'))
    print(f"{topic} {file_path}")
    with open(file_path, 'wb') as f:
        f.write(payload)
```

17. Forráskód Részlet Az üzenetként kapott fájl feldolgozása

3.7.4.2. Beérkezett hanganyag felismerése a paraméterek alapján (*pitch_detection.py TMP/*.wav*)

A mentett WAV fájl felismerése a *pitch_detecion.py TMP/*.wav* szkript végrehajtásával történik. Mivel az ideiglenes mappát a felismerési folyamat végén töröljük és csak egy audió fájlt kapunk node-onként, így biztosak lehetünk benne, hogy a *TMP* mappa egy .wav kiterjesztésű elemet tartalmaz. A *scipy.io* könyvtárat használva beolvassuk a mintavételezési rátát és a hanganyagot a *sr* és *audio* változókba.

```
filename = sys.argv[1]
sr, audio = wavfile.read(f"{filename}")
```

18. Forráskód Részlet A .wav fájl betöltése

Ez után meghívjuk a *crepe.predict(...)* függvényt, paraméterül adjuk neki a hanganyagot, a mintavételezési rátát és bekapcsoljuk a Viterbi simítást. A függvény egy rendezett négyessel tér vissza, ami tartalmazza az időt, a jósolt frekvenciát, a konfidenciát és a nyers aktivációs mátrixot. A *numpy* könyvtár használatával egy kétdimenziós mátrixba rendezzük a rendezett négyes első három elemét, mivel az aktivációs mátrixra nem lesz szükség a feldolgozáshoz. Ez után az időt kicseréljük mintavételezési rátára, olyan módon, hogy a létrehozott *numpy.column_stack* első oszlopát egyenlővé tesszük két egymást követő idő különbségével.

```
time, frequency, confidence, activation = crepe.predict(audio, sr,
viterbi=True)

data = numpy.column_stack((time, frequency, confidence))

rate = data[1][0] - data[0][0]

for i in range(len(data)):
    data[i][0] = rate
```

19. Forráskód Részlet A .wav fájl feldolgozása

Ez után a nyers felismerési adatot mentjük a *TMP/out.txt* fájlba tabulátorokkal elválasztva, 3 tizedesjegyre kerekítve, majd a program terminál.

```
numpy.savetxt('TMP/out.txt', data, fmt='%.3f', delimiter='\t')
```

20. Forráskód Részlet A feldolgozás eredményének mentése

3.7.4.3. Felismert hanganyag feldolgozása (*process.py TMP/system*.txt*)

Miután a *pitch_detection.py* terminál a *client_action.sh* meghívja a *process.py* Python szkriptet, aminek a feladata, hogy feldolgozza a nyers adatokat. Először létrehoz egy tömböt a lehetséges frekvenciákkal egy oktávon. Ez olyan módon történik, hogy alapul vesz egy kromatikus skálát, ahol az „A” hang felel meg a 440Hz-nek majd kiszámolja, hogy a bizonyos hangok hol helyezkednek el a 0Hz-től számított első 8 oktávban. Ez a lista lesz az, ami az elérhető hangokat tárolja. Ennél azért nincs szükség szélesebb frekvencia halmazra, mivel az emberi fül ennél magasabb frekvenciát már nem hall. (15)


```

notes = [ 'A', 'A#', 'B', 'C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#']
octaves = np.arange(0, 9)
frequencies = []

for octave in octaves:
    for note in notes:
        frequency = 440 * (2 ** (octave - 4 + (notes.index(note) / 12)))
        frequencies.append(frequency)

note_freq_array = np.array(frequencies)
note_freq_array_2 = list(zip(9*notes,note_freq_array))

note_freq_array = np.append(note_freq_array,0)

```

21. Forráskód Részlet Skála generálása

Ez így készített skála kromatikus, ez viszont nem minden esetben optimális. Ez annak a következménye, hogy vannak olyan darabok, ahol ennek csak egy részhalmazát használjuk. A 3.7.1 fejezetben láttuk, hogy a felhasználótól bekérjük a hangszer skáláját. Ez az információ itt kerül felhasználásra. Amennyiben a felhasználó a „chromatic” kulcsszót használta, nem történik semmi. A „pentatonic” kulcsszó használatával a skála leszűkül az „A,C,D,E,G” ötfokú, azaz pentatónikus, részhalmazra. Amennyiben a felhasználó a „custom” kulcsszót használta és egyedi skálát adott meg, abban az esetben csak azok a hangok fognak az eredményben megjelenni, amelyeket megadott.

```

if content[1].upper() == "chromatic".upper():
    pass
elif content[1].upper() == "pentatonic".upper():
    t_notes = [ 'A', 'C', 'D', 'E', 'G']
    note_freq_array_2 = [item for item in note_freq_array_2 if item[0][0:] in t_notes]
    note_freq_array = np.array([freq for note, freq in note_freq_array_2])
else:
    t_notes = content[1].strip('[]').replace("'", "").split(',')
    note_freq_array_2 = [item for item in note_freq_array_2 if item[0][0:] in t_notes]
    note_freq_array = np.array([freq for _, freq in note_freq_array_2])

```

22. Forráskód Részlet Skála szűkítése a megadott paraméter alapján

Ez után beolvassuk a felismerés végén tárolt fájlt és elkezdjük a feldolgozást. Az adatokat egy *numpy* tömbben tároljuk el. Amennyiben egy felismert frekvenciának a konfidenciája 0.5 alatt van akkor egyenlővé tesszük a hozzá időben legközelebbi értékkel. Ezután megkeressük a

zeneileg érvényes frekvenciák közül, hogy melyik van a legközelebb az adott méréshez, amivel aztán egyenlővé tesszük. Például, ha a felismerő 439.83 Hz-t mért, akkor ezt lecseréljük a hozzá abszolút értékben legközelebb lévő érvényes frekvenciára, a 440 Hz-re.

```
def find_closest_frequency(number):
    absolute_diff = np.abs(note_freq_array - number)
    closest_index = np.argmin(absolute_diff)
    closest_frequency = note_freq_array[closest_index]
    return closest_frequency
```

23. Forráskód Részlet A legközelebbi érvényes frekvencia megkeresése

Annak érdekében, hogy megkapjuk, mennyi ideig tartott egy hang az időben, összegezzük az egymás után következő azonos frekvenciájú mérések mintavételezési idejét, és átlagoljuk a konfidenciájukat.

```
count=0
sum = 0
confidence = 0
last_value = -1
data_2 = np.array([])

for i in range(0, len(data)):
    value = data[i,1]
    time = data[i,0]
    confidence_tmp = data[i,2]
    if value == last_value or last_value == -1:
        sum+=time
        confidence += confidence_tmp
        count += 1
        last_value = value
    else:
        data_2 = np.append(data_2,[last_value, sum, (confidence / count)],
                           axis = 0)

        count=0
        sum = 0
        confidence = 0
        last_value = -1
```

24. Forráskód Részlet Azonos hangok kombinálása

A jelenlegi adatban még előfordulhatnak felismerési hibák, olyan hangok, amelyek valamilyen külső vagy belső tényező következtében tévesen lettek mérve. Ezeknek a soroknak két jellemzője van, irreálisan rövid ideig tartanak (három századmásodpercnél rövidebbek), vagy nagyon alacsony átlagos konfidenciaszinttel rendelkeznek. Annak érdekében, hogy ne

rövidítsük meg a sorokat, mivel így elveszne a szólamok közötti összhang, az ilyen hibás adatok helyére szünetet illesztünk be. A feldolgozott adatot, ami a kezdeti mintavételezési idő, frekvencia, konfidencia helyett most már a hang hossza, hangmagasság, átlagos konfidencia sorokat tartalmazza elmentjük a *TMP/processed.txt* szöveges fájlba az utolsó oszlopot elhagyva, mivel arra a továbbiakban nem lesz szükségünk.

```
to_delete = []
for i in range(1,len(data_2)-1):
    if((data_2[i, 1] < 0.1) and (data_2[i-1, 0] == data_2[i+1, 0])):
        ...
    elif((data_2[i, 1] < 0.05)): ... #0.1 > s
    elif(data_2[i, 2] < 0.70 and round(data_2[i,0]) != 0):
        ...
```

25. Forráskód Részlet Felismert hangok konfidencia alapú tisztítása

3.7.4.4. A feldolgozott hanganyag konvertálása musicxml formátummá (*convert_to_musescore.py*)

A feldolgozott fájlt, a *processed.txt*-t átadjuk a *convert_to_musescore.py* Python szkriptnek, ami a music21 könyvtár segítségével musicxml formátumúvá konvertálja a felismert adatokat. A hanghosszokat, a *processed.txt* első oszlopát, zeneileg értelmezhető hosszokra konvertálja. Ezek közül a legkisebb a másodperc egy-harmincketted része (0.03125s), ennek következtében törlésre került minden mérés, ami három századmásodpercnél rövidebb. (Lásd 3.7.4.3 fejezet)

```
durations = [1/32, 1/16, 1/8, 1/4, 1/2, 1, 2, 4, 8, 16] # Available durations
closest_duration = min(durations, key=lambda x: abs(x - timestamp))
```

26. Forráskód Részlet Intervallumok létrehozása és legközelebbi intervallum keresése

Ez után létrehozunk egy music21 *stream.Score()* objektumot, amelybe beleírjuk a hangokat vagy amennyiben 0 volt a frekvencia a szüneteket és a hozzájuk tartozó időintervallumokat, majd az eredményt musicxml formátumban mentjük.

```

def generate_musicxml(durations, frequencies, output_file):
    # Create a music21 Stream object to store the musical elements
    score = stream.Score()

    # Create a Part object to hold the notes
    part = stream.Part()
    score.append(part)

    # Iterate over the durations and frequencies arrays
    for dur, freq in zip(durations, frequencies):
        if freq == 0:
            # Create a Rest object for the pause
            r = note.Rest()

            # Create a Duration object with the corresponding duration value
            d = duration.Duration(dur)

            # Set the quarter length of the Rest object
            r.quarterLength = d.quarterLength

            # Add the Rest object to the Part object
            part.append(r)
        else:
            # Create a Note object with the corresponding frequency
            n = note.Note()
            n.pitch.frequency = freq

            # Create a Duration object with the corresponding duration value
            d = duration.Duration(dur)

            # Set the quarter length of the Note object
            n.quarterLength = d.quarterLength

            # Add the Note object to the Part object
            part.append(n)

    # Convert the score to MusicXML format and save to file
    score.write('musicxml', fp=output_file)

```

27. Forráskód Részlet A felismert és letisztított eredmény .musicxml formátumúvá konvertálása

3.7.4.5. Az eredmény publikálása a Control Node felé (mosquitto_pub ...)

A felismerő node miután felismerte a beérkező hangfájlt, feldolgozta és tárolta musicxml formátumban közzé teszi ezt a saját témáján az MQTT hálózaton annak érdekében, hogy a Control Node beolvashassa és feldolgozhassa. (Lásd 3.7.3.1 fejezet) Ezt a *mosquitto_pub*

parancssori eszköz segítségével hajtjuk végre. A `-h` kapcsoló meghatározza a bróker elérési címét, a `-t` a téma nevét és a `-f` a küldésre szánt fájl helyét.

```
mosquitto_pub -h MosquittoBroker -t "ControlNode/$hostname" -f TMP/*.musicxml
```

28. Forráskód Részlet Az eredmény publikálása a "ControlNode/\$hostname" témára

A „`*`” helyettesítő karakter a `*.musicxml` fájlnevében olyan megfontolásból lett használva, hogy a shell szkriptet ne kelljen node-specifikussá tenni. Minden node egy `musicxml` fájlt hoz létre, viszont ezek nem azonos néven kerülnek mentésre. A „`*`” használatával kiküszöbölhető minden lehetséges ebből adódó probléma.

3.7.4.6. Az ideiglenes fájlok törlése

Annak érdekében, hogy az alkalmazás következő futásakor ne lépjen fel konfliktus a régi és az új fájlok között, a `node_action.sh` shell szkript végén az ideiglenes mappát töröljük.

```
rm -rf TMP
```

29. Forráskód Részlet Ideiglenes mappa törlése

Ezzel véget ér a felismerési és publikálási folyamat és a `node_action.sh` terminál.

3.8. A felhasznált modulok és könyvtárak listája és verziószáma

Modulok és Könyvtárak	Verziószám
Git	2.47.1
Multipass	1.13.1+win
Python	3.13.0
python3-pip	24.0
music21	9.1.0
Crepe	0.0.16
TensorFlow	2.17.0
paho-mqtt	2.1.0

mosquitto-clients	2.0.20
-------------------	--------

1. táblázat Az alkalmazásban felhasznált modulok és könyvtárak listája és verziószáma

3.9. Tesztelési terv és a tesztelés eredményei

A tesztelési tervet, három fő részre lehet tagolni, a fejlesztési folyamat főbb szekciói alapján. Ezek a virtuális környezet felállítása, a Blueprint telepítése és az alkalmazás helyes működése. Ezek különböző aspektusait tesztelve biztosíthatjuk, hogy az alkalmazás megfelelő környezetben, helyesen működik.

3.9.1. A virtuális környezet felállításának tesztelése (*test_virtual_env.py*)

Arról, hogy a virtuális környezet helyesen lett felállítva, különböző tesztek segítségével tudunk meggyőződni. Ellenőrizni kell, hogy minden virtuális gép fut-e, azokon helyesek-e az Ubuntu verziók, a Mosquitto Bróker helyesen lett-e létrehozva és fut-e rajta a „Mosquitto Service” és hogy minden Node esetében működik-e az SSH. Ezeket a lépéseket a saját függvényükben definiáltuk, olyan módon, hogy azok igazzal vagy hamissal térjenek vissza. Amennyiben az összes igazzal tért vissza, a tesztelés sikeresnek tekinthető.

3.9.1.1. A virtuális gépek futásának ellenőrzése

A *get_multipass_list()* függvény segítségével elindítunk egy alfolyamatot, ami a *multipass list* parancs futtatásának eredményét eltárolja egy változóban. Ez a változó, a helyes felbontás után a virtuális gépek számával megegyező sorból fog állni, amelyek a *név*, *IP cím*, *státusz*, *Ubuntu verzió* adatokat tartalmazzák. A *check_node_status()* -> *bool* függvény segítségével végigiterálunk a változó sorain és amennyiben minden sor tartalmazza a „Running” azaz fut adatot, akkor visszatérünk igazzal. Ellenkező esetben hibát adunk és visszatérünk hamissal.

```
def get_multipass_list():
    try:
        result = subprocess.run(['multipass', 'list'], capture_output=True,
text=True, check=True)
        return result.stdout
    except subprocess.CalledProcessError as e:
        print(colored(f"An error occurred: {e}", 'red'))
        return None

def check_node_statuses() -> bool:
    try:
        assert True == all([True if "Running" in x else False for x in
multipass_list_output])
        print(colored('All instances are running', 'green'))
        return True
    except AssertionError:
        print(colored("Not all instances are running", 'red'))
        return False
```

30. Forráskód Részlet A virtuális gépek adatainak beolvasása és státuszuk ellenőrzése

3.9.1.2. A virtuális gépek operációs rendszerének ellenőrzése

A 3.9.1.1-es fejezetben bemutatott módon, minimális módosítások elvégzésével, le tudjuk tesztelni az operációs rendszerek helyességét is. Arra kell szűrnünk, hogy amennyiben a sor nem tartalmazza a „Mosquitto” szót, azaz az adott virtuális gép nem a Bróker, akkor az adott virtuális gép Ubuntu verziószáma egyezzen meg a 22.04-el. Ezt a következő sor segítségével tudjuk megtenni.

```
assert True == all([True if "22.04" in x else False for x in
multipass_list_output if "Mosquitto" not in x])
```

31. Forráskód Részlet Feltételezés, hogy minden virtuális Gépen Ubuntu 22.4 van

3.9.1.3. A Mosquitto Bróker operációs rendszerének ellenőrzése

A 3.9.1.2 fejezetben említetthez hasonlóan le lehet ellenőrizni a Mosquitto Bróker operációs rendszerét is, csupán a szűrési feltételen kell változtatni. Amennyiben a virtuális gép neve megegyezik a „MosquittoBroker” szöveggel, akkor a verziójának meg kell egyeznie az „Ubuntu Mosquitto Appliance” szöveggel. Amennyiben ez teljesül, a tesztelés eredménye igaz.

```
assert True == all([True if all([True if y in x else False for y in "Ubuntu
Mosquitto Appliance".split()]) else False for x in multipass_list_output if
x[0] == "MosquittoBroker"]])
```

32. Forráskód Részlet Feltételezés, hogy amennyiben a virtuális gép neve „MosquittoBroker” akkor a típusa „Ubuntu Mosquitto Appliance”

3.9.1.4. Az SSH kapcsolat ellenőrzése a Node-ok és a felhasználó személyi számítógépe között

Az SSH kapcsolatot olyan módon tudjuk ellenőrizni a felhasználó személyi számítógépe és a felismerő között, hogy egy alfolyamatban megpróbálunk csatlakozni hozzá, egyszer, batch módban. Amennyiben a folyamat végrehajtása sikeres, azaz az SSH kapcsolat létrejött, visszatérünk igazzal. Ezt az ellenőrzést csak a Control Node-on és a felismerőkön kell végrehajtani, mivel a Blueprint telepítése során csak ezekhez szeretnénk SSH-val kapcsolódni.

```
subprocess.run(
    ['ssh', '-v', f'{username}@{ip}', '-i',
    '../Virtual_ENV/multipass-ssh-key', '-o', 'BatchMode=yes', '-o',
    'ConnectionAttempts=1', 'true'],
    check=True,
    timeout=10,
    stdout=subprocess.DEVNULL,
    stderr=subprocess.DEVNULL
)
```

33. Forráskód Részlet SSH kapcsolat tesztelése

3.9.1.5. A Mosquitto Service futásának ellenőrzése

Azt, hogy a „Mosquitto Service” működik-e a státuszának lekérdezésével tudjuk ellenőrizni. Mivel a virtuális környezetünk felépítéséhez Multipass-t használtunk, így a „multipass exec” függvény segítségével le tudjuk kérdezni a „snap.mosquitto.mosquitto.service” státuszát. Amennyiben a státusz aktív, visszatérünk igazzal, ellenkező esetben pedig hamissal.

```
result = subprocess.run(['multipass', 'exec', 'MosquittoBroker', '--', 'sudo',
'systemctl', 'is-active', '--quiet', 'snap.mosquitto.mosquitto.service'],
check=True, timeout=10)
```

34. Forráskód Részlet A mosquitto szolgáltatás állapotának lekérdezése

3.9.2. A Blueprint telepítés sikerességének ellenőrzése (test_blueprint.py)

A Blueprint telepítése közben több olyan folyamat is lejátszódik, aminek a sikeres végbemenetele nem biztosított. Ennek következtében annak ellenőrzése, hogy minden sikeresen települt-e egy kritikus lépés annak érdekében, hogy megbizonyosodhassunk arról, hogy az Alkalmazás használatához minden készen áll. Ehhez ellenőrizni kell, hogy a GitHub könyvtár, amit letöltöttünk a virtuális gépekre megfelelő, az elvárt Python könyvtárak hiánytalanul települtek mind a felismerő, mind a Control Node-okon és hogy az MQTT kommunikáció helyesen működik.

3.9.2.1. A GitHub könyvtár ellenőrzése a Node-okon

Ahhoz, hogy ellenőrizni tudjuk, hogy a megfelelő GitHub könyvtár helyesen lett lemásolva a Node-okra, szükségünk van ezeknek az SSH belépési adataira. Ezeket a 3.9.1.1-es fejezetben látott módon beolvassuk a `get_multipass_list()` függvényt meghívva. Minden Node-ra belépünk SSH kapcsolattal és ellenőrizzük, hogy létezik a Szakdolgozat_v2. Ezen felül a kimenethez hozzacsatoljuk a `git-remote url origin` eredményét. Amennyiben a kimenet enterenként tagolva megegyezik a „`[True, Expected_Git_URL]`” listával, ahol az „`expected_git_url`” annak a GitHub könyvtárnak a webcíme, ahol a szakdolgozat tárolva van, akkor visszatérünk Igazzal.

```
ssh -i "%SSH_KEY%" "%USERNAME%@%IP_ADDRESS%" "if [ -d %FOLDER_PATH% ]; then  
echo True; else echo False; fi && cd %FOLDER_PATH% && git remote get-url  
origin"
```

35. Forráskód Részlet Az alkalmazást tároló könyvtár meglétének ellenőrzése

```
result = result.stdout.split('\n')  
    if result[0] and result[1] == expected_url:  
        print(colored(f"Git repo is correct on {host}", 'green'))  
    return True
```

36. Forráskód Részlet A 35. Forráskód Részlet eredményének feldolgozása

3.9.2.2. Megfelelő eszközök és Python könyvtárak meglétének ellenőrzése a Control Node-on

A Control Node-on ellenőrizni kell, hogy megtalálhatóak-e a „`paho-mqtt`” és „`music21`” Python könyvtárak ezen felül a „`mosquitto-clients`” eszköz. Ezt többek között olyan módon lehet ellenőrizni, hogy megjelenítjük az adott könyvtárakat a pip modul segítségével és lekérdezzük az eszköz verziószámát. Amennyiben ezeket a Shell hívásokat logikai és kapcsolatba hozzuk egymással és egy „`echo True | echo False`” paranccsal akkor csak abban az esetben fogunk igazat kapni, ha minden Python könyvtár telepítve van, úgy, mint ahogy a Mosquitto kommunikációhoz szükséges eszköz. Ennek a hívásnak az értékével térünk vissza.

```
ssh -i "%SSH_KEY%" "%USERNAME%@%IP_ADDRESS%" "pip show paho-mqtt && pip show  
music21 && (command -v mosquitto_sub --version >nul 2>&1 && echo True || echo  
False)"
```

37. Forráskód Részlet Eszközök és Python könyvtárak ellenőrzése

3.9.2.3. Megfelelő eszközök és Python könyvtárak meglétének ellenőrzése a Felismerőkön

Ahhoz, hogy le tudjuk ellenőrizni azt, hogy a felismerőkön megtalálható-e minden szükséges Python könyvtár és Shell eszköz, hasonlóan kell eljárunk, mint az előző szekcióban (Lásd 3.9.2.2 fejezet). A Felismerőkön a következő moduloknak kell szerepelnie: „*paho-mqtt*”, „*music21*”, „*crepe*”, „*tesnorflow*” és a „*mosquitto-clients*” eszköz. Ezeknek a meglétét az előbbiekben látott módon ellenőrizzük.

```
ssh -i "%SSH_KEY%" "%USERNAME%@%IP_ADDRESS%" "pip show paho-mqtt && pip show music21 && pip show crepe && pip show tensorflow && (command -v mosquitto_sub --version >nul 2>&1 && echo True || echo False)"
```

38. Forráskód Részlet Eszközök és Python könyvtárak ellenőrzése a felismerőkön

3.9.2.4. Az MQTT kommunikáció ellenőrzése

Az MQTT kommunikációt azért kell ellenőrizni, mivel a Felismerők MQTT-n keresztül kommunikálnak a felhasználó személyi számítógépével és a Control Node-al. Ezt olyan módon tudjuk ellenőrizni, hogy megnézzük a Node képes-e csatlakozni a Mosquitto Brókerhez. Amennyiben igen, abból feltételezhető, hogy képes üzenetek küldésére és fogadására. Ezt a csatlakozást olyan módon ellenőrizzük, hogy futtatunk egy Python szkriptet ami az adott MQTT Brókerhez csatlakozik és a csatlakozás *callback* függvénye igazzal tér vissza, ha a kapcsolat státusza „0”, azaz sikeres. Ezzel az RC értékből következtetett logikai változóval térünk vissza.

```
def on_connect(client, userdata, flags, rc):  
    if rc == 0:  
        print("True")  
    else:  
        print("False", rc)
```

39. Forráskód Részlet A csatlakozáskor automatikusan meghívódó függvény

```
result = subprocess.run([".\check_mqtt.bat", user, host, ssh_key_path],  
capture_output=True, text=True, check=True)
```

40. Forráskód Részlet A check_mqtt.bat futásának eredményének ellenőrzése

3.9.3. Az Alkalmazás működésének ellenőrzése (*test_application.py*)

Az alkalmazás helyes működését az őt felépítő komponensek egységenként történő tesztelésével lehet megállapítani. Meg kell állapítani, hogy a felhasználótól helyesen olvassa-e be az adatokat és ezeket helyesen továbbítja majd dolgozza fel.

3.9.3.1. A felhasználótól történő adat beolvasásának tesztelése

Az alkalmazás indításakor bizonyos adatokat a felhasználótól kérdezzük be. Ezeknek a feldolgozása hasonló olyan szempontból, hogy minden bemeneti adaton típust ellenőrzünk és esetleges hibák esetén visszatérünk egy hibával és újra kérdezzük az adott paramétert. Ezt a viselkedést utánozza a `get_number_of_instruments()` -> `int` függvény, amely kap egy bemeneti paramétert és eldönti, hogy annak helyes-e a típusa és az értéke egy meghatározott tartományon belül megtalálható-e. Amennyiben igen, visszatér az értékkel, ellenkező esetben hibát ad. Ezt a függvényt hívjuk meg különböző hibás és hibátlan inputokkal annak érdekében, hogy megbizonyosodhassunk róla, hogy a konstrukció működik általános esetben is.

```
while True:
    try:
        t = int(t)
        if t > 0:
            return t
        else:
            return 0
    except ValueError:
        return -1
```

41. Forráskód Részlet Általános típus ellenőrzés

3.9.3.2. Üzenet küldés ellenőrzése a személyi számítógépről a Node-okra

Ahhoz, hogy az üzenetküldést le tudjuk ellenőrizni a felismerőkön, párhuzamosan kell futtatnunk két folyamatot. Az egyik folyamat figyel és kiírja a beérkező üzeneteket, míg a másik küldi azokat. Ezeket párhuzamosan indított parancssorokban futtatjuk és ellenőrizzük, hogy véghez megy-e a folyamat, azaz elküldésre kerül az üzenet. Amennyiben ez minden esetben megtörténik visszatérünk igazzal.

```
start cmd /c ssh %vmuser%@%ipaddress% -i ../Virtual_ENV/multipass-ssh-key "cd Szakdolgozat_v2 && cd Application && python3 listen_node.py testingTopic "
```

```
start cmd /c python3
C:\Users\Benedek\Documents\ELTE\Szakdolgozat_v2\Application\mqtt_publish_message.py %brokeraddress% system/testingTopic/start start
if errorlevel 1 (
    set success=false
)
```

42. Forráskód Részlet Üzenetküldési folyamat ellenőrzése

3.9.3.3. A felismerő tesztelése

A felismerő egy mesterséges intelligencia alapú felismerő, amely három értékkel tér vissza minden egyes intervallumban, amelyek eltárolásra kerülnek a *TMP/out.txt* szöveges fájlban. Így annak érdekében, hogy tesztelni tudjuk, működik-e a felismerő le kell ellenőriznünk, hogy amennyiben a kimeneti fájl létrejön a futás eredményeképpen, akkor pontosan három lebegőpontos értéket tartalmaz minden sorban.

```
with open('TMP/out.txt', 'r') as file:
    lines = file.readlines()
    lines = [x.strip().split() for x in lines]
    if len(lines[0]) == 3 and all(all(is_float(element) for element in
line) for line in lines):
        print(colored("The output of the recogniser is correct",
'green'))
    return True
```

43. Forráskód Részlet A felismerő futási eredményének ellenőrzése

3.9.3.4. A felismert anyag feldolgozásának tesztelése

A felhasználó megadhatja a kezdeti konfigurációban, hogy a felismert kotta milyen skálájú legyen, milyen hangokat tartalmazzon. A feldolgozónak többek között az a feladata, hogy ez a feltétel teljesüljön és csak olyan hangok kerüljenek a kottába, amelyeket a felhasználó meghatározott. Ezt olyan módon tudjuk ellenőrizni, hogy futtatjuk a feldolgozó szkriptet különböző bemenetekkel és megnézzük, hogy a kimenet megfelel-e az elvártnak. Például, ha a konfigurációs fájlban az szerepel, hogy a kotta álljon kizárólag „Á” hangokból, akkor a „processed.txt” csak az „Á” hanghoz tartozó frekvenciákat tartalmazza-e.

```

if lines[0][1].upper() == "chromatic".upper():
    return all([x[0] in note_freq_array for x in processed_lines])
elif lines[0][1].upper() == "pentatonic".upper():
    t_notes = [ 'A', 'C', 'D', 'E', 'G' ]
    note_freq_array_2 = [item for item in note_freq_array_2 if
item[0][0:] in t_notes]
    note_freq_array = np.array([freq for note, freq in
note_freq_array_2])
    return all([x[0] in note_freq_array for x in processed_lines])
else:
    t_notes = lines[0][1].strip('[]').replace("'", "").split(',')
    note_freq_array_2 = [item for item in note_freq_array_2 if
item[0][0:] in t_notes]
    note_freq_array = np.array([freq for _, freq in
note_freq_array_2])
    return all([x[0] in note_freq_array for x in processed_lines])

```

44. Forráskód Részlet Skálák helyességének ellenőrzése

3.9.3.5. Musicxml formátummá konvertálás tesztelése

Miután feldolgoztuk a felismert hanganyagot, musicxml formátumúvá alakítjuk. Ez azért fontos mert a legtöbb ingyenes kottázó alkalmazás felismeri és kezeli ezt a formátumot. Annak érdekében, hogy ellenőrizni tudjuk, hogy a konverter működik, be kell olvasnunk a konvertált fájlt és validálni a formátumát. Ezt a music21 könyvtár *parse()* metódusával tehetjük meg.

```

try:
    score = music21.converter.parse('TMP/output.musicxml')
    if score:
        print(colored("The generated MusicXML is valid", 'green'))
        return True

```

45. Forráskód Részlet Az output.musicxml validálása

3.9.3.6. A felismert szólamok kombinálásának tesztelése

Az applikáció futásának az egyik utolsó lépése a felismert szólamok kombinálása és egy kottává történő összefűzése. Azt, hogy ez helyesen történt-e konvencionális eszközökkel tesztelni körülményes, mivel az eredmény függ a felismerő kimenetétől. Azonban bizonyos aspektusokat úgy is tesztelhetünk, hogy bármit tudnánk a végleges kimenet tartalmáról. Le tudjuk ellenőrizni, hogy a fájl helyes formátumban van-e és hogy a végleges kotta szólam száma megegyezik-e a bemeneti hangfájlok számával.

```
musicxml_files = [f for f in os.listdir(messages_folder) if
f.endswith('.musicxml')]
    num_files = len(musicxml_files)

    combined_sheet = converter.parse('musescore/combined_sheet.musicxml')
    if len(combined_sheet.parts) == num_files:
        return True
```

46. Forráskód Részlet A szólamegyesítési folyamat tesztelése

3.9.4. A tesztelés eredménye

A tesztelési terv, virtuális környezet, Blueprint és alkalmazás tesztelése, megvalósításra került a *Testing* mappában. A szkriptek sorban kerültek futtatásra egy előtte letisztított környezetben, amelyet a *multipass delete -all* és a *multipass purge* parancs kiadásával állítottuk elő. A 12. ábra A *test_virtual_env.py* futásának eredménye13. ábra A *test_blueprint.py* futásának eredménye és 14. ábra A *test_application.py* futásának eredménye látszik, hogy amennyiben a Virtuális környezet és a Blueprint sikeresen lett telepítve, illetve az alkalmazás működik, akkor a program megfelel minden tesztesetnek.

```
C:\Users\Benedek\Documents\ELTE\Szakdolgozat_v2\Testing>python3 test_virtual_env.py
All instances are running
Test 1/6 passed
All nodes are running Ubuntu 22.04 LTS
Test 2/6 passed
Mosquitto Broker is running the correct multipass appliance
Test 3/6 passed
nodelist.txt file updated
Test 4/6 passed
SSH connection to vmuser@172.18.217.232 successful
SSH connection to vmuser@172.18.215.83 successful
SSH connection to vmuser@172.18.208.81 successful
Test 5/6 passed
Mosquitto service is running on MosquittoBroker
Test 6/6 passed
-----Testing The Virtual Environment : completed-----
All tests passed
```

12. ábra A *test_virtual_env.py* futásának eredménye

```

C:\Users\Benedek\Documents\ELTE\Szakdolgozat_v2\Testing>python3 test_blueprint.py
Checking git repo on 172.18.255.201
Git repo is correct on 172.18.255.201
Test 1/9 passed
Checking git repo on 172.18.250.29
Git repo is correct on 172.18.250.29
Test 2/9 passed
Checking git repo on 172.18.245.167
Git repo is correct on 172.18.245.167
Test 3/9 passed
Python libraries are installed correctly on 172.18.255.201
Test 4/9 passed
Python libraries are installed correctly on 172.18.250.29
Test 5/9 passed
Python libraries are installed correctly on 172.18.245.167
Test 6/9 passed
MQTT is working correctly on 172.18.255.201
Test 7/9 passed
MQTT is working correctly on 172.18.250.29
Test 8/9 passed
MQTT is working correctly on 172.18.245.167
Test 9/9 passed
-----Testing The Blueprint Setup : completed-----
All tests passed

```

13. ábra A test_blueprint.py futásának eredménye

```

C:\Users\Benedek\Documents\ELTE\Szakdolgozat_v2\Testing>python3 test_application.py
Messaging works correctly on ubuntu@172.29.178.201
Messaging works correctly on vmuser@172.29.189.254

C:\Users\Benedek\Documents\ELTE\Szakdolgozat_v2\Testing>python3 ../Application/pitch_detection.py
../pitches/beres.wav
2024-11-27 18:03:05.470642: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow
binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the
appropriate compiler flags.
53/53 ----- 13s 242ms/step
The output of the recogniser is correct
The processor works correctly
The combined sheet has the correct number of parts
The generated MusicXML is valid
-----Testing The Application : completed-----
All tests passed

```

14. ábra A test_application.py futásának eredménye

4. Összefoglalás és további fejlesztési lehetőségek

Az alkalmazásnak sikerült elérnie a célját olyan szempontból, hogy használata jelentősen meggyorsítja egy dallamnak a digitalizálását, lekottázását, úgy, hogy a felismert végeredmény egy jó kiindulási alapot ad, ami kézzel már könnyen finomítható, véglegesíthető. Ez azt jelenti, hogy sikerült egy olyan eszközt létrehozni, ami megfelelő fizikai infrastruktúrával (nagy teljesítményű felismerő szerverek) jelentősen hozzájárulhat a magyar népi kultúrának egy részének, a népzeneének a fennmaradásához. Jelenleg nem található hasonló, széles körben elterjedt eszköz a népzene kutatásban. A szoftver fejlesztésének a folyamata teret hagyott további fejlesztési lehetőségek kivitelezésére. Az alkalmazásnak nincs grafikus felhasználói

felülete (GUI), ami nehezíti a használatát mivel egy informatikában nem jártas embernek parancssoron alkalmazást futtatni nem egy megszokott folyamat. Ezen kívül a jelenlegi rendszerben egy MQTT Bróker van, ami azt jelenti, hogy ha annak a működése valamilyen okból kifolyólag korlátozott, az leállíthatja az egész felismerési folyamatot. Ezt a kritikus pontot lehetne ellensúlyozni több bróker és egy terhelés elosztó segítségével, amely feltérképezi, hogy melyek az aktív brókerek és arra tereli a forgalmat. Amikor az alkalmazás lokálisan, virtuális környezetben fut, az azon felül, hogy rendkívül hardver igényes, instabil is. Ez annak a következménye, hogy mesterséges intelligencia alapú felismerőket futtatni nagy teljesítmény igényű, aminek következtében egy eszköz könnyen elérheti a kapacitásának limitjét. Ezt a problémát olyan módon lehetne kiküszöbölni, hogy az alkalmazást magát is áthelyezzük egy távoli szerverre, amihez a felhasználó inentől kezdve, mint szolgáltatás férhetne hozzá. Legutolsó sorban a felismerési pontosság még nem érte el azt a szintet, hogy az alkalmazás által készített kotta hibátlanul legyen nevezhető. Ezt a feldolgozási és felismerési folyamat finomhangolásával, vagy egy teljesen erre a célra készített mély neuronháló készítésével lehetne fejleszteni olyan szintre, hogy az elkészült kottába ne kelljen kézzel belenyúlani egy hibátlan eredmény eléréséhez.

5. Köszönetnyilvánítás

Szeretném elsősorban megköszönni konzulensemnek Dr. Tejfel Máténak, aki szakmai tudásával nagyban hozzájárult a szakdolgozatom létrejöttéhez. Továbbá szeretném megköszönni Dr Alwahab Dhulfiqar Zoltánnak, aki segítséget nyújtott a Blueprint-ek és IoT hálózatok mélyebb megismerésében. Nagy segítség volt továbbá párom, akinek mindig számíthattam az építő jellegű hozzászólásaira és véleményére.

6. Irodalomjegyzék

1. A MAGYAR NÉPZENE TÖRTÉNETI RÉTEGEI, NEMZETKÖZI KAPCSOLATAI, FEJLŐDÉSTÖRTÉNETE. *arcanum.com*. [Online] [Hivatkozva: 2024. 07 26.] <https://www.arcanum.com/hu/online-kiadvanyok/MagyarNeprajz-magyar-neprajz-2/vi-nepzene-neptanc-nepi-jatek-8ADD/magyar-nepzene-8AED/a-magyar-nepzene-torteneti-retegei-nemzetkozi-kapcsolatai-fejlodestortenete-8C9E/>.
2. GYULA, ORTUTAY. *Magyar Néprajzi Lexikon*. Budapest : Akadémia Kiadó , 1979. ISBN 963 05 1285 8 sorozat.
3. EITCA. *What is the minimum amount of RAM recommended for allocating to the virtual machine running TensorFlow?* [Online] 2023. 08 08. [Hivatkozva: 2024. 10 03.] <https://eitca.org/artificial-intelligence/eitc-ai-dltf-deep-learning-with-tensorflow/tensorflow/installing-tensorflow/examination-review-installing-tensorflow/what-is-the-minimum-amount-of-ram-recommended-for-allocating-to-the-virtual-machine-running-tens>.
4. trendforce.com. *CSPs to Expand into Edge AI, Driving Average NB DRAM Capacity Growth by at Least 7% in 2025, Says TrendForce*. [Online] Press Center, 2024. [Hivatkozva: 2024. 11 02.] <https://www.trendforce.com/presscenter/news/20240625-12200.html>.
5. emqx. *MQTT vs HTTP: Ultimate IoT Protocol Comparison Guide*. [Online] 2024. [Hivatkozva: 2024. 11 10.] <https://www.emqx.com/en/blog/mqtt-vs-http>.
6. Team, HiveMQ. HiveMQ. *hivemq.com*. [Online] HiveMQ, 2024. 02 20. [Hivatkozva: 2024. 10 29.] <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/>.
7. Kim, Jong Wook, és mtsai. *arxiv.org*. *CREPE: A Convolutional Representation for Pitch Estimation*. [Online] Cornell University, 2018. [Hivatkozva: 2024. 10 29.] <https://arxiv.org/abs/1802.06182>.
8. Scarlett, Rizel. *Why Python keeps growing, explained*. *github.blog*. [Online] GitHub, 2023. 03 07. [Hivatkozva: 2024. 09 26.] <https://github.blog/developer-skills/programming-languages-and-frameworks/why-python-keeps-growing-explained/>.

9. Python 3.13.0 documentation. *docs.python*. [Online] The Python Software Foundation, 2024. [Hivatkozva: 2024. 09 02.] <https://docs.python.org/3/>.
10. Yadavpuneet. medium.com. *Why Does Facebook Use Amazon Web Services (AWS) for its Infrastructure?* [Online] Medium, 2023. 08 11. [Hivatkozva: 2024. 10 28.] <https://medium.com/@yadavpuneet2001/why-does-facebook-use-amazon-web-services-aws-for-its-infrastructure-dc9759d44134#:~:text=Scalability%20and%20Global%20Reach,to%20users%20across%20diverse%20regions..>
11. Canonical Multipass. *Multipass Documentation*. [Online] 2024. [Hivatkozva: 2024. 09 23.] <https://multipass.run/docs>.
12. Thorking. lf-akraino.atlassian.net. *Akraino Wiki*. [Online] Akraino, 2024. [Hivatkozva: 2024. 10 18.] <https://lf-akraino.atlassian.net/wiki/spaces/AK/overview?homepagelid=13664258>.
13. Cuthbert, Michael, és mtsai. music21.org. *music21 Documentation*. [Online] 2014. [Hivatkozva: 2023. 07 22.] <https://www.music21.org/music21docs/index.html>.
14. Dows, Nguqyen. learn.microsoft.com. *Tutorial: SSH in Windows Terminal*. [Online] Microsoft, 2024. 05 29. [Hivatkozva: 2024. 06 23.] <https://learn.microsoft.com/en-us/windows/terminal/tutorials/ssh>.
15. *Upper Limit of Frequency for Human Hearing*. Pumphrey, R. 1950., Nature, old.: 166.