Distributed Scalable Face Recognition System

Dokumentacija

Kolegij: Distribuirani sustavi
Fakultet informatike u Puli

Autor: Antonio Labinjan
Mentor: doc. dr. sc. Nikola Tanković

December 18, 2025

Contents

Sažetak

Ova dokumentacija opisuje dizajn i implementaciju distribuiranog, skalabilnog sustava za prepoznavanje lica u realnom vremenu temeljenog na modernim metodama računalnog vida i umjetne inteligencije. Sustav se sastoji od više udaljenih nodeova (edge kamera) koji lokalno izvode ekstrakciju značajki lica koristeći CLIP model, te centralnog servera koji koristi FAISS za vektorsko pretraživanje i klasifikaciju osoba. Arhitektura omogućuje horizontalno skaliranje, load balancing i failover mehanizme između nodeova, čime se postiže visoka pouzdanost i rad u stvarnom vremenu.

Komunikacija između nodeova i servera odvija se putem Redis posrednika, koji služi kao message broker i load balancing sloj. Redis omogućuje asinkronu razmjenu embeddinga, redoslijedno procesiranje zahtjeva i stabilan prijenos podataka čak i u slučajevima privremene mrežne nestabilnosti.

Svaki Node sadrži vlastiti token u JSON formatu, koji služi za autentikaciju prema Serveru. Server učitava sve poznate i dozvoljene tokene te pri svakom zahtjevu od Node-a provjerava podudarnost. Osim autentikacije, token definira i vremensku zonu u kojoj se Node nalazi, što omogućuje točno evidentiranje vremena detekcija i logova u slučaju rada u više vremenskih zona.

Na strani nodea implementiran je mehanizam shouldClassify() koji optimizira prijenos podataka - embedding se šalje prema serveru samo ako je detekcija dovoljno različita ili ako je prošlo određeno vrijeme od zadnjeg slanja. Time se značajno smanjuje mrežno opterećenje bez gubitka performansi sustava.

Sustav također uključuje unknown alert system koji prepoznaje višestruke pokušaje neuspjele identifikacije s različitih nodeova u kratkom vremenskom periodu, uz potpuno poštivanje GDPR smjernica i bez otkrivanja identiteta korisnika.

Rješenje pokazuje kako se principi distribuiranih sustava mogu učinkovito primijeniti na prepoznavanje lica u stvarnom vremenu, uz naglasak na skalabilnost, efikasnost i privatnost u edge-to-server arhitekturi.

Opis aplikacije

Razvijeni sustav predstavlja distribuirano rješenje za prepoznavanje lica u stvarnom vremenu koje kombinira snagu compter visiona i distribuiranih sustava. Sastoji se od više **nodeova** (edge kamera) koji lokalno obrađuju video u stvarnom vremenu, te centralnog **servera** koji objedinjuje rezultate, vrši klasifikaciju i vodi evidenciju detekcija. Također, vrši se dvostruka segmentacija i uklanjanje pozadine i na server-side djelu i na nodevima kako bi se smanjio utjecaj pozadine na klasifikaciju. U trenutnoj verziji, server je implementiran u FastAPI-ju, dok su nodevi implementirani u čistom Pythonu te koriste ugrađene kamere računala i dodatne vanjske kamere (trenutno USB priključak). Nodeovi koriste **CLIP** model za ekstrakciju značajki lica (embeddinga), dok server koristi **FAISS** za vektorsko pretraživanje i usporedbu embeddinga s postojećom bazom poznatih osoba. Komunikacija između nodeova i servera odvija se putem **Redis** posrednika koji služi kao message broker i load balancing sloj, čime se omogućuje skalabilna i asinkrona razmjena podataka.

Sustav podržava horizontalno skaliranje - dodavanjem novih nodeova automatski se povećava kapacitet obrade bez promjene konfiguracije servera. Osim toga, uključeni su **failover mehanizmi** koji omogućuju nastavak rada i u slučaju kvara pojedinog nodea ili pada servera. Nodesi su lako zamjenjivi i lako ih je dodati (potrebno je samo spojiti kameru i pokrenuti Python script). Ukoliko server padne, embeddinzi se svejedno šalju u Redis te mogu biti dohvaćeni kad server opet proradi.

Jedna od prednosti sustava je mehanizam `should_classify()`, koji značajno smanjuje promet u mreži jer osigurava da se embedding šalje samo kad je stvarno potreban (promjena lica ili protek vremena). Time se postiže optimalna ravnoteža između performansi i učinkovitosti.

Ciljano tržište i korisnici

Primarna ciljna skupina su **organizacije, poduzeća i institucije** kojima je potreban automatizirani, brzi i pouzdani sustav za identifikaciju i evidenciju osoba. Sustav se može koristiti u različitim scenarijima:

- **Evidencija prisutnosti zaposlenika** u tvrtkama, laboratorijima ili fakultetima.
- **Sigurnosni nadzor** u poslovnim i javnim prostorima, s mogućnošću detekcije nepoznatih osoba (unknown alert system).
- **Autonomni ulazni sustavi** koji reagiraju na prepoznato lice i odobravaju pristup bez potrebe za fizičkim kontaktom.

Za razliku od centraliziranih rješenja, ovaj sustav pruža **distribuiranost, otpornost i privatnost** – obrada se odvija lokalno na nodeovima, a prema serveru se šalju samo embeddingi, ne i slike lica. To znači da je sustav u skladu s **GDPR** smjernicama i može se primijeniti u okruženjima s visokim sigurnosnim i etičkim zahtjevima. Time se postiže moderna kombinacija umjetne inteligencije i distribuiranih principa koja otvara put prema **lokalnim edge-to-server** rješenjima spremnima za buduće proširenje u cloud okruženje.

Analiza tržišta i konkurencija

Motivacija za razvoj ovog projekta proizašla je iz prethodnog rada na jednostavnim web aplikacijama baziranim na arhitekturi **1-client–1-server**. Takav pristup bio je dovoljan za manje projekte, ali pokazao je svoja ograničenja u situacijama gdje je potrebno obraditi veći broj ulaza

(kamera) i postići real-time sinkronizaciju između više uređaja. Cilj je bio unaprijediti postojeću aplikaciju pretvaranjem je u **distribuirani sustav**, sposoban za obradu podataka s više kamera u stvarnom vremenu uz istovremeno održavanje stabilnosti, točnosti i brzine.

Glavni motiv je bio **automatizirati brojne svakodnevne procese bilježenja prisutnosti** — eliminirati ručni unos, liste i kartice te omogućiti sustavu da sam prepoznaje osobu pri ulasku ili izlasku. Takav pristup ne samo da štedi vrijeme, već i uklanja ljudske pogreške te omogućuje analitiku u stvarnom vremenu.

Na tržištu već postoje sustavi za prepoznavanje lica, poput **Azure Face API-ja**, **AWS Rekognitiona** ili **OpenCV/DeepFace** rješenja, ali svi oni imaju svoja ograničenja:

- **Cloud ovisnost:** većina rješenja zahtijeva konstantnu internetsku vezu i vanjsku infrastrukturu, što stvara ovisnost o trećim servisima.
- **Privatnost:** podaci o licima šalju se na vanjske servere, što je neprihvatljivo u GDPR osjetljivim okruženjima.
- **Centraliziranost:** tradicionalna rješenja koriste jedan server, što otežava skaliranje i povećava rizik od kvara.

Razvijeni sustav uklanja te probleme uvođenjem **distribuirane arhitekture** s lokalnom obradom podataka na **edge nodeovima** i centralnim serverom koji upravlja klasifikacijom i sinkronizacijom putem **Redis** brokera. Time je postignut balans između brzine, sigurnosti i fleksibilnosti, bez ovisnosti o vanjskim servisima.

Table 1: SWOT analiza distribuiranog sustava za prepoznavanje lica

Snage (S)	**Slabosti (W)**
• Distribuiran i skalabilan sustav • Lokalna obrada podataka (GDPR-friendly) • Real-time prepoznavanje i logging • Modularnost i lako dodavanje novih nodeova	• Zahtjevnija implementacija i konfiguracija • Potrebno održavanje više uređaja
Prilike (O)	**Prijetnje (T)**
• Primjena u pametnim zgradama i industriji 4.0 • Integracija s postojećim sustavima kontrole pristupa • Potencijal za komercijalizaciju i SaaS model	• Brz razvoj konkurentskih AI rješenja • Moguće regulatorne promjene (GDPR, AI Act) • Tehnički problemi s pouzdanošću mreže

Projekt time ne samo da rješava konkretan problem evidencije prisutnosti, već otvara vrata prema **budućim edge-to-server sustavima** koji kombiniraju računalni vid, distribuirano računalstvo i automatizaciju poslovnih procesa. Ovo rješenje demonstrira kako se napredni AI modeli mogu učinkovito raspodijeliti između rubnih uređaja i centralnog sustava, što predstavlja važan korak prema modernim, skalabilnim i etičkim sustavima prepoznavanja lica.

Distribuirani sustav za prepoznavanje lica sastoji se od centralnog servera, Redis queuea i više udaljenih čvorova (nodeova), pri čemu svaki čvor predstavlja kameru ili modul za snimanje lica. Čvorovi ekstraktiraju embeddinge lica pomoću CLIP modela i odlučuju, kroz funkciju `should_classify()`, kada će poslati embedding serveru, čime se optimizira mrežni promet. Svi embeddingi se šalju preko Redis queuea na centralni server, koji upravlja FAISS indeksom i provodi klasifikaciju te algoritam glasanja kako bi odredio identitet osobe. Server zatim vraća rezultate natrag čvorovima po potrebi, a cjelokupan sustav omogućava dinamičko dodavanje

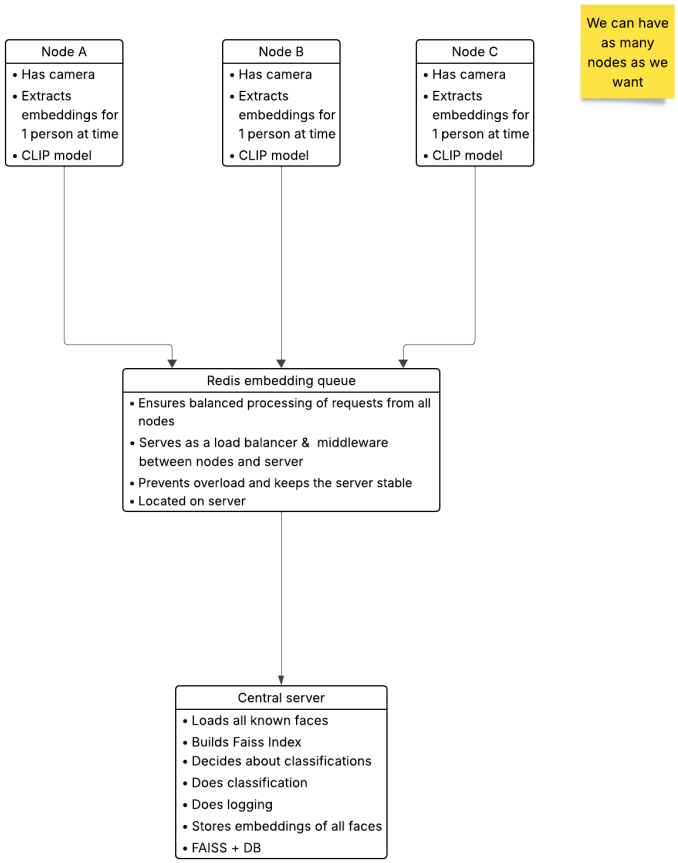

Figure 1: Shema arhitekture sustava

novih nodeova i podešavanje parametara u realnom vremenu, dok shema vizualno prikazuje tok podataka i veze između nodeova, Redis-a i servera.

Razrada funkcionalnosti

1. Funkcionalnosti servera

- Pokretanje FastAPI servera s REST API endpointima
- Učitavanje dataset-a lica iz foldera
- Dodavanje pojedinačnih lica u dataset i ekstrakcija CLIP embeddinga
- Normalizacija i spremanje embeddinga u memoriju
- Gradnja FAISS indeksa za brzu pretragu najbližih susjeda
- Klasifikacija lica pomoću FAISS indeksa i CLIP embeddinga
- Testiranje različitih threshold vrijednosti za klasifikaciju
- Praćenje neprepoznatih pokušaja (Unknown) i intruder alert sustav
- Logiranje detekcija u memoriju
- Globalno logiranje preko Redis streama
- Redis queue za slanje embeddinga u worker thread
- Worker thread koji čita embeddinge iz Redis queue-a i klasificira ih
- Dead-letter queue u Redisu za neispravne ili višestruko neuspjele poruke
- Pametno odlučivanje kada klasificirati embedding po node-u (`should_classify`)
- Praćenje posljednjeg embeddinga i timestamp-a po node-u
- API endpointi za pregled loga detekcija u JSON i HTML formatu
- API endpointi za pregled queue-a, threshold statistike i aktivnih node-ova
- Vizualizacija threshold statistike i aktivnih node-ova u HTML tablicama
- API endpointi za intruder alert log u JSON i HTML formatu
- Endpoint za ponovno učitavanje dataset-a i rebuild FAISS indeksa
- Ping endpoint za provjeru zdravlja servera
- Logging svih važnih operacija s vremenskim oznakama

Detaljnije

- Pokretanje FastAPI servera s REST API endpointima

  FastAPI omogućuje izgradnju brzog i skalabilnog backend sustava s jasnom podjelom odgovornosti. REST API endpointi osiguravaju standardiziranu i proširivu komunikaciju između servera i distribuiranih node-ova.

- Učitavanje dataset-a lica iz foldera

  Učitavanje dataset-a s diska omogućuje jednostavno inicijalno postavljanje sustava bez potrebe za dodatnim servisima. Time se osigurava reproducibilnost eksperimenata i konzistentnost ulaznih podataka.

- Dodavanje pojedinačnih lica u dataset i ekstrakcija CLIP embeddinga

  Dinamičko dodavanje novih identiteta omogućuje sustavu prilagodbu u realnom vremenu. Ekstrakcija CLIP embeddinga osigurava robusnu i semantički bogatu reprezentaciju lica.

- Normalizacija i spremanje embeddinga u memoriju

  Normalizacija embeddinga smanjuje varijacije uzrokovane osvjetljenjem i skalom vektora. Spremanje u memoriju omogućuje brzu obradu bez dodatnih I/O operacija.

- Gradnja FAISS indeksa za brzu pretragu najbližih susjeda

  FAISS indeks omogućuje učinkovitu pretragu u visokodimenzionalnom prostoru čak i za velike skupove podataka. Time se postiže niska latencija potrebna za rad u stvarnom vremenu.

- Klasifikacija lica pomoću FAISS indeksa i CLIP embeddinga

  Kombinacija CLIP embeddinga i FAISS pretrage omogućuje preciznu identifikaciju osoba. Ovakav pristup nadilazi klasične klasifikatore i bolje se nosi s varijacijama u izgledu lica.

- Testiranje različitih threshold vrijednosti za klasifikaciju

  Eksperimentalno podešavanje threshold vrijednosti ključno je za balansiranje između false positive i false negative rezultata. Sustav se time može prilagoditi različitim sigurnosnim zahtjevima.

- Praćenje neprepoznatih pokušaja (Unknown) i intruder alert sustav

  Evidentiranje neprepoznatih lica omogućuje detekciju potencijalnih sigurnosnih prijetnji. Intruder alert sustav predstavlja osnovu za daljnju integraciju s alarmnim ili notifikacijskim mehanizmima.

- Logiranje detekcija u memoriju

  Privremeno logiranje u memoriji omogućuje brz pristup podacima za analizu i vizualizaciju. Ovo je posebno korisno za debugiranje i praćenje ponašanja sustava u realnom vremenu.

- Globalno logiranje preko Redis streama

  Redis stream omogućuje centralizirano i distribuirano prikupljanje logova iz više izvora. Time se osigurava konzistentan uvid u ponašanje cijelog sustava.

- Redis queue za slanje embeddinga u worker thread

  Queue mehanizam omogućuje asinkronu obradu embeddinga i razdvajanje prijema podataka od klasifikacije. Ovo značajno povećava skalabilnost sustava.

- Worker thread koji čita embeddinge iz Redis queue-a i klasificira ih

  Korištenjem worker thread-a postiže se paralelizacija obrade bez blokiranja glavnog server procesa. Time se povećava propusnost i stabilnost sustava.

- Dead-letter queue u Redisu za neispravne ili višestruko neuspjele poruke

  Dead-letter queue omogućuje izolaciju problematičnih poruka bez gubitka podataka. Ovakav mehanizam povećava robusnost i olakšava dijagnostiku grešaka.

- Pametno odlučivanje kada klasificirati embedding po node-u (`should_classify`)

Mehanizam `should_classify` smanjuje redundantne klasifikacije i nepotrebno opterećenje servera. Time se optimizira korištenje resursa u distribuiranom okruženju.

- Praćenje posljednjeg embeddinga i timestamp-a po node-u

  Praćenjem vremenskog konteksta embeddinga omogućuje se donošenje informiranih odluka o slanju novih podataka. Ovo je ključno za stabilan rad pri visokim frekvencijama frame-ova.

- API endpointi za pregled loga detekcija u JSON i HTML formatu

  Različiti formati izlaza omogućuju jednostavnu integraciju s drugim sustavima i pregled podataka kroz web sučelje. Time se povećava uporabljivost sustava.

- API endpointi za pregled queue-a, threshold statistike i aktivnih node-ova

  Ovi endpointi omogućuju uvid u stanje sustava u realnom vremenu. Administratori mogu brzo identificirati uska grla ili neaktivne node-ove.

- Vizualizacija threshold statistike i aktivnih node-ova u HTML tablicama

  Vizualni prikaz podataka olakšava interpretaciju performansi sustava. Time se ubrzava donošenje odluka tijekom testiranja i optimizacije.

- API endpointi za intruder alert log u JSON i HTML formatu

  Omogućuje pregled sigurnosnih incidenata i njihovu daljnju analizu. Ovakav zapis je ključan za forenzičke i audit potrebe.

- Endpoint za ponovno učitavanje dataset-a i rebuild FAISS indeksa

  Dinamičko ponovno učitavanje omogućuje ažuriranje sustava bez restarta. Time se smanjuje downtime i povećava fleksibilnost sustava.

- Ping endpoint za provjeru zdravlja servera

  Health check endpoint omogućuje automatsko praćenje dostupnosti servera. Ovo je standardna praksa u produkcijskim distribuiranim sustavima.

- Logging svih važnih operacija s vremenskim oznakama

  Detaljno logiranje omogućuje preciznu analizu ponašanja sustava kroz vrijeme. Vremenske oznake olakšavaju korelaciju događaja između servera i node-ova.

Funkcionalnosti Node-a

- Učitavanje tokena iz JSON datoteke
- Povezivanje s Redis serverom i provjera konekcije
- Postavljanje osnovnih parametara Node-a (NODE_ID, threshold distance i vrijeme)
- Praćenje posljednjeg embeddinga i vremena po node-u
- Inicijalizacija CLIP modela i procesora za ekstrakciju embeddinga lica
- Inicijalizacija Haar Cascade detektora lica
- Inicijalizacija Mediapipe Face Mesh-a za preciznu segmentaciju lica
- Segmentacija lica iz frame-a koristeći Face Mesh

- Odlučivanje treba li klasificirati novi embedding (`should_classify`)
- Praćenje frame-ova koji su previše tamni i ignoriranje u slučaju prevelike tamnosti
- Otvaranje kamere i provjera da je dostupna i radi (health check)
- Postavljanje rezolucije kamere (640x480)
- Nepravilno dohvaćeni frame-ovi se preskaču
- Detekcija lica u svakom frame-u koristeći Haar Cascade
- Segmentacija i ekstrakcija embeddinga iz svakog lica
- Normalizacija embeddinga
- Slanje embeddinga u Redis queue za daljnju obradu
- Logiranje svih važnih događaja i grešaka
- Vizualno označavanje detektiranih lica i status poruka na frame-u
- Praćenje performansi: FPS i prosječna latencija svakih 30 frame-ova
- Spremanje latencije u tekstualnu datoteku
- Omogućavanje prekida programa tipkom 'q' i čišćenje resursa pri zatvaranju

Detaljnije

- Učitavanje tokena iz JSON datoteke

  Svaki `Node` sadrži vlastiti token u JSON formatu, koji služi za autentikaciju prema `Serveru`. Server učitava sve poznate i dozvoljene tokene te pri svakom zahtjevu od `Node-a` provjerava podudarnost (*match*). Osim autentikacije, token definira i vremensku zonu u kojoj se `Node` nalazi, što omogućuje točno evidentiranje vremena detekcija i logova.

```
{
    "node_id": 0,
    "token": "94bddfac0ab2587c9a278bfe3ebc036fa3025bfda61acc2eba35bc0f778e270b",
    "timezone": "Europe/Zagreb"
}
```

Figure 2: Primjer tokena za Node0

- Povezivanje s Redis serverom i provjera konekcije

  Provjera konekcije pri pokretanju osigurava da je Node sposoban komunicirati s centralnim serverom. Rano otkrivanje problema sprječava gubitak podataka i nevidljive greške tijekom rada.

- Postavljanje osnovnih parametara Node-a (NODE_ID, threshold distance i vrijeme)

  Jedinstveni identifikator Node-a omogućuje praćenje i upravljanje distribuiranim izvorima podataka. Threshold parametri omogućuju fino podešavanje osjetljivosti klasifikacije po pojedinom Node-u.

- Praćenje posljednjeg embeddinga i vremena po node-u

  Praćenje prethodnih embeddinga omogućuje usporedbu s novim podacima u vremenskom kontekstu. Ovo je ključno za smanjenje redundantnih slanja i stabilan rad sustava.

- Inicijalizacija CLIP modela i procesora za ekstrakciju embeddinga lica

  CLIP model omogućuje robusnu ekstrakciju semantičkih značajki lica. Njegova generalizacijska sposobnost čini sustav otpornijim na varijacije u osvjetljenju i izrazu lica.

- Inicijalizacija Haar Cascade detektora lica

  Haar Cascade predstavlja lagano i brzo rješenje za detekciju lica u realnom vremenu. Pogodan je za rad na edge uređajima s ograničenim resursima.

- Inicijalizacija Mediapipe Face Mesh-a za preciznu segmentaciju lica

  Face Mesh omogućuje precizno određivanje granica i orijentacije lica. Time se poboljšava kvaliteta segmentacije i posljedično embeddinga.

- Segmentacija lica iz frame-a koristeći Face Mesh

  Precizna segmentacija smanjuje utjecaj pozadine i šuma iz okoline. Ovo rezultira stabilnijim i konzistentnijim embedding vektorima.

- Odlučivanje treba li klasificirati novi embedding (`should_classify`)

  Ovaj mehanizam omogućuje inteligentno filtriranje podataka prije slanja na server. Time se značajno smanjuje opterećenje mreže i centralnog klasifikatora.

- Praćenje frame-ova koji su previše tamni i ignoriranje u slučaju prevelike tamnosti

  Filtriranje tamnih frame-ova sprječava generiranje nekvalitetnih embeddinga. Time se smanjuje broj lažnih detekcija i poboljšava pouzdanost sustava.

- Otvaranje kamere i provjera da je dostupna i radi (health check)

  Provjera dostupnosti kamere osigurava da Node može pouzdano prikupljati vizualne podatke. Ovo je važno za autonomni rad bez stalnog nadzora.

- Postavljanje rezolucije kamere (640x480)

  Fiksna rezolucija omogućuje konzistentnu obradu frame-ova i predvidive performanse. Odabrana rezolucija predstavlja kompromis između brzine i kvalitete slike.

- Nepravilno dohvaćeni frame-ovi se preskaču

  Preskakanje oštećenih ili nepotpunih frame-ova sprječava pad aplikacije. Time se povećava stabilnost dugotrajnog rada Node-a.

- Detekcija lica u svakom frame-u koristeći Haar Cascade

  Kontinuirana detekcija omogućuje brzo reagiranje sustava na pojavu lica u kadru. Ovo je ključno za aplikacije u stvarnom vremenu.

- Segmentacija i ekstrakcija embeddinga iz svakog lica

  Ekstrakcija embeddinga iz segmentiranog lica osigurava da model koristi samo relevantne informacije. Time se povećava točnost klasifikacije.

- Normalizacija embeddinga

  Normalizacija smanjuje varijacije u magnitudi vektora i omogućuje stabilniju usporedbu. Ovo je posebno važno pri korištenju distance-based metoda.

- Slanje embeddinga u Redis queue za daljnju obradu

  Asinkrono slanje embeddinga omogućuje neblokirajući rad Node-a. Redis queue služi kao pouzdan međusloj za distribuiranu obradu.

- Logiranje svih važnih događaja i grešaka

  Detaljno logiranje omogućuje praćenje ponašanja Node-a kroz vrijeme. Ovo olakšava debugiranje i analizu performansi.

- Vizualno označavanje detektiranih lica i status poruka na frame-u

  Vizualni overlay pruža trenutni uvid u stanje sustava tijekom rada. Ovo je korisno za testiranje i demonstraciju funkcionalnosti.

- Praćenje performansi: FPS i prosječna latencija svakih 30 frame-ova

  Praćenje performansi omogućuje kvantitativnu evaluaciju rada sustava. FPS i latencija su ključni pokazatelji real-time sposobnosti.

- Spremanje latencije u tekstualnu datoteku

  Pohrana latencije omogućuje kasniju analizu i usporedbu različitih konfiguracija. Ovi podaci su korisni za eksperimentalnu evaluaciju sustava.

- Omogućavanje prekida programa tipkom 'q' i čišćenje resursa pri zatvaranju

  Kontrolirano gašenje osigurava pravilno oslobađanje resursa poput kamere i memorije. Time se sprječavaju curenja resursa i nestabilnosti sustava.

Implementacija

Za potrebe jednostavnog i pouzdanog pokretanja distribuiranog sustava za prepoznavanje lica korišten je `docker-compose`, koji omogućuje definiranje i orkestraciju svih komponenti sustava unutar jedinstvene konfiguracijske datoteke. Ovakav pristup osigurava da se FastAPI poslužitelj, Redis posrednik i svi pomoćni servisi pokreću u potpuno konzistentnim i reproducibilnim okruženjima, neovisno o softveru instaliranom na lokalnoj mašini. Svaki servis se izolira u vlastiti kontejner s točno određenim verzijama biblioteka i ovisnosti, čime se eliminiraju problemi vezani uz konfiguraciju sustava, verzijske konflikte i ručne instalacije. Pokretanje cjelokupne infrastrukture svodi se na jednu naredbu, što drastično ubrzava razvoj, testiranje i implementaciju, a istovremeno jamči da će sustav jednako raditi na svakom računalu ili poslužitelju.

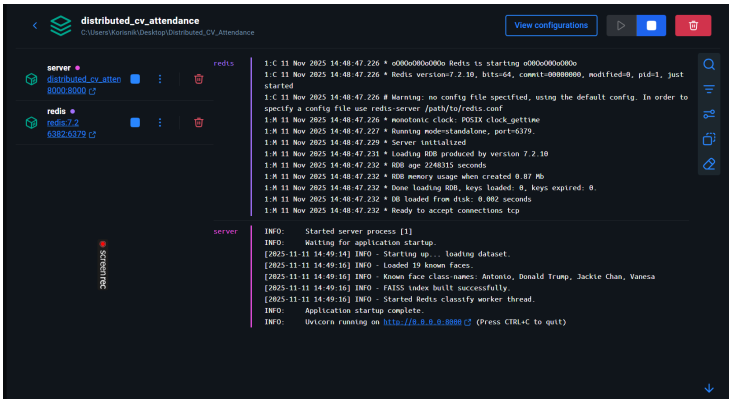

Figure 3: Docker compose okruženje pokrenutog sustava koje se sastoji od FastApi servera i Redisa

0.1 Dockerfile

```
FROM python:3.12

WORKDIR /app

RUN apt-get update && apt-get install -y \
    libgl1 \
    libglib2.0-0 \
 && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "server_FASTAPI:app", "--host", "0.0.0.0", "--port", "8000"]
```

Dockerfile prikazan u sustavu služi za kontejnerizaciju FastAPI servisa unutar distribuiranog sustava. Kao osnovna slika koristi se `python:3.12`, čime se osigurava konzistentno i reproducibilno izvršno okruženje neovisno o krajnjem sustavu. Radni direktorij unutar kontejnera postavljen je na `/app`, što omogućuje jasno strukturiran raspored aplikacijskog koda. Radi podrške computer

visionu i grafičkim ovisnostima, instaliraju se sistemske biblioteke `libgl1` i `libglib2.0-0`, koje su nužne za ispravno izvođenje OpenCV i srodnih paketa u headless okruženju. Python ovisnosti definirane su u datoteci `requirements.txt` te se instaliraju korištenjem `pip`-a bez zadržavanja lokalnog cachea, čime se smanjuje konačna veličina Docker slike. Cjelokupni izvorni kod aplikacije zatim se kopira u kontejner, a mrežni port `8000` izlaže se kako bi servis bio dostupan ostalim komponentama distribuiranog sustava. Pokretanje aplikacije ostvaruje se putem ASGI poslužitelja `Uvicorn`, koji omogućuje asinkrono rukovanje zahtjevima i skalabilnu komunikaciju između distribuiranih čvorova.

Za razliku od serverskih komponenti, kamere odnosno *node* procesi pokreću se lokalno na fizičkim uređajima, izvan Docker okruženja. Glavni razlog za takav pristup je činjenica da pristup fizičkoj kameri unutar kontejnera često nije pouzdan niti podržan na svim operacijskim sustavima, što bi moglo otežati inicijalizaciju video streama. Osim toga, lokalno izvršavanje omogućuje da se obrada slike i ekstrakcija značajki obave s minimalnom latencijom, bez dodatnih mrežnih slojeva između kamere i procesorske logike. Time se postiže veća responzivnost sustava i smanjuje opterećenje glavnog poslužitelja, jer se prema Redis posredniku šalju samo već obrađeni embeddingi, a ne sirovi video podaci. Ovakva arhitektura također omogućuje fleksibilno skaliranje - broj lokalnih kamera može se povećavati bez potrebe za mijenjanjem centralne infrastrukture, što sustav čini iznimno prilagodljivim i skalabilnim u realnim uvjetima implementacije.

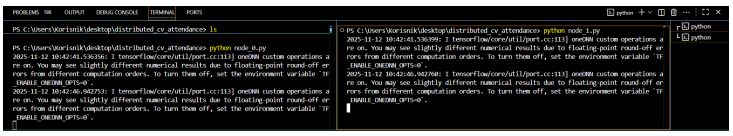

Figure 4: Pokretanje nodesa kroz terminal

Kao što je vidljivo na slici, za dodavanje novog nodea u sustav, dovoljno je fizički spojiti kameru i pokrenuti kod. Jedina manja komplikacija očituje se u tome što je svakoj novoj kameri potrebno podesiti indeks i ID u kodu. Koristi se pravilo po kojem je ID jednak indeksu kamere i nazivu node fajla što olakšava snalaženje i brzo i efikasno upravljanje kamerama.

Sam server ima izrazito jednostavnan grafički interface koji se sastoji od tablice logova u formatu: timestamp, name, score, node ID, threshold i timezone. Timestamp - predstavlja točan datum i vrijeme detekcije (s preciznošu od sekudne) Name - predstavlja ime osobe koja je detektirana Node ID - predstavlja unique identifikator svakog nodea Threshold - predstavlja optimalni threshold koji je iskorišten za detekciju Timezone - predstavlja vremensku zonu u kojoj se node vrti i time omogućava rad na širokom području (kad bi se kamere teoretski spojile bežično)

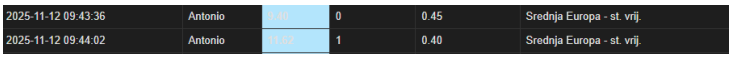

Figure 5: Primjer logginga

Interface na nodesima je također minimalan. Pri pokretanju nodea otvara se cv2 window koji prikazuje feed trenutne kamere te stvara bounding box oko detektiranih lica. Pošto se klasifikacija odvija na serveru te zbog privatnosti, na node feedu se ne vidi identitet osobe (ne zapisuje se rezultat klasifikacije koji je poznat samo serveru).

Za asinkronu komunikaciju između distribuiranih čvorova i centralnog poslužitelja implementiran je `Redis` koji u ovom sustavu djeluje kao middleware i load balancer. Svaki čvor, nakon što generira *embedding* lica pomoću CLIP modela, šalje podatke u zajednički `Redis` queue u kojem se privremeno pohranjuju svi zahtjevi za klasifikaciju. Centralni poslužitelj istodobno čita te zapise i obrađuje ih pomoću FAISS baze, čime se postiže dinamičko balansiranje opterećenja

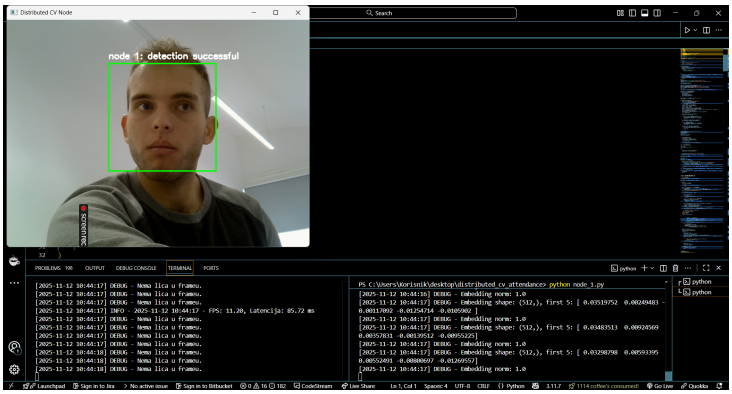

Figure 6: Primjer detekcije

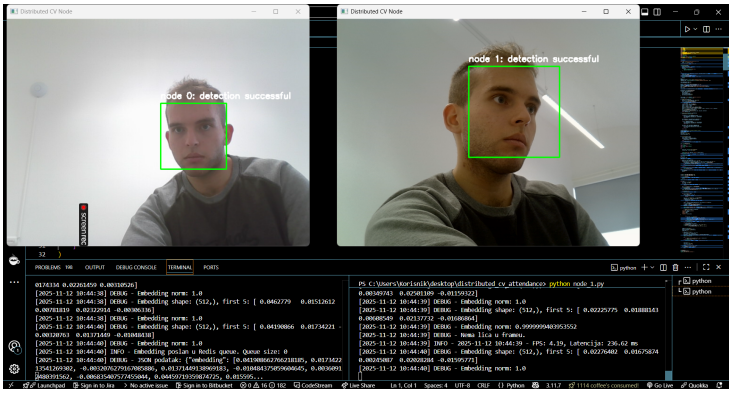

Figure 7: Primjer detekcije s više kamera

između više aktivnih čvorova i poslužiteljskih procesa. Ovakva arhitektura omogućuje skalabilan i otporan sustav: broj čvorova i poslužitelja može se slobodno povećavati bez potrebe za promjenom osnovne logike komunikacije. Redis time učinkovito eliminira bottleneckove i omogućuje paralelnu, distribuiranu obradu podataka u stvarnom vremenu.

Uloga Pydantic-a U sustavimu, Pydantic model služi kao ključni mehanizam za validaciju ulaznih podataka. Na primjer, model `EmbeddingRequest` osigurava da svaki primljeni embedding bude lista realnih brojeva (`list[float]`), dok `node_id` bude cijeli broj s defaultnom vrijednošću 0. Time se sprječava slanje neispravnih ili nekompatibilnih podataka u FAISS pretragu ili mehanizam glasanja, što povećava robusnost i pouzdanost sustava. Osim toga, Pydantic automatski parsira JSON requeste u Python objekte, smanjujući potrebu za ručnom konverzijom tipova, a u kombinaciji s FastAPI-em olakšava generiranje precizne dokumentacije API-ja, omogućujući frontend aplikacijama da točno znaju što poslati serveru.

Sam hardverski setup je prilično jednostavan. Sastoji se od centralnog računala na kojem se pokreće server i kamera (nodesa):

Pošto se u trenutnoj implementaciji koriste usb kamere, ograničenje takvog sustava je svakako limitiran broj kamera.

Arhitektura, klase i komponente

Ovaj distribuirani sustav sastoji se od dva glavna dijela: *node-side* (radni čvorovi s kamerama) i *server-side* (centralni sustav). Svaki dio sadrži specifične module i klase koje zajedno omogućuju

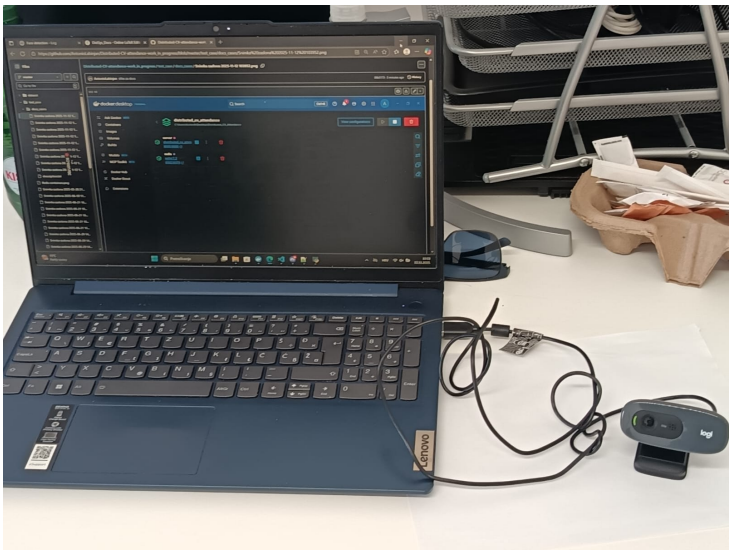

Figure 8: Primjer detekcije s više kamera

ekstrakciju značajki, slanje embeddinga, klasifikaciju, logiranje i nadzor sustava.

0.2 Node-side Komponente

0.2.1 Camera / Capture Modul

- **CameraReader** – odgovoran za dohvat frameova ili event-based podataka.
- **FrameGrabber** – obavlja manipulaciju frameova i rezanje lica.

0.2.2 Embedding Modul (CLIP Encoder)

- **ClipEncoder**
  - `encode_face(image)` – generira embedding lica.

0.2.3 Should-Classify Mehanizam

- **ShouldClassify**
  - uloga: sprečava nepotrebno slanje embeddinga.
  - pamti: zadnji poslani embedding po čvoru, vrijeme slanja.

0.2.4 Queue Sending Modul

- **RedisPublisher**
  - `publish_embedding(embedding, node_id)`

0.3 Server-side Komponente

0.3.1 Redis Queue Consumer / Worker

- **RedisConsumer**
  - sluša queue i prosljeđuje embeddinge klasifikatoru.

0.3.2 Feature Matcher i Klasifikator (FAISS + Voting)

- **FaissIndexManager**
  - upravlja FAISS indexom.
  - `search(embedding, k)` – vraća najbližih k susjeda.
- **VotingClassifier**
  - implementira k1/k2 glasanje.
  - vraća: ID osobe, confidence, distance.
- **ThresholdManager**
  - dinamičko prilagođavanje decision thresholda.

0.3.3 Logging Modul

- **DetectionLogger**
  - zapisuje sve detekcije (node, osoba, vrijeme, distance

0.3.4 Monitoring i Dashboard

- **StatsCollector** – prikuplja metrike sustava.
- **DashboardAPI** – služi za prikaz statistika, logova i stanja nodeova.

0.4 Infrastructure i Background Komponente

0.4.1 Config Modul

- **Config / Settings**
  - centralizirana konfiguracija sustava: putanje modela, Redis URL, pragovi, parametri votinga itd.

0.4.2 Healthcheck i Failover

- **NodeHealthMonitor**
  - provjerava aktivnost nodeova.
- **FailoverManager**
  - upravlja fallback scenarijima i prebacivanjem opterećenja.

0.5 Feature Store / Dataset Komponente

- **EmbeddingStore** – pohranjuje embeddinge korisnika.
- **FaceCropper** – odgovoran za detekciju i segmentaciju lica.

0.6 API Sloj

- Implementiran preko FastAPI-a.
  - `/register_user`
  - `/update_embedding`
  - `/get_logs`
  - `/stats`
  - `/threshold/update`

1 REST API Endpoints

Sustav koristi REST API implementiran pomoću *FastAPI* frameworka. U nastavku je pregled dostupnih endpointa, njihove namjene i povratnih podataka.

Figure 9: Pregled API ruta

1.1 /redis-test

Method: GET
Provjera ispravnosti Redis veze. Endpoint služi kao health-check za Redis instancu.

1.2 /log

Method: GET
Vraća sistemski log u sirovom (tekstualnom ili JSON) obliku, namijenjen za programatsku analizu i debugiranje.

1.3 /log/html

Method: GET
HTML prikaz sistemskog loga, optimiziran za pregled u pregledniku.

1.4 /

Method: GET
Početni endpoint aplikacije. Koristi se kao osnovni health-check ili landing endpoint.

1.5 /queue_contents

Method: GET
Dohvaća trenutno stanje Redis queuea, uključujući sve zaprimljene, ali još neobrađene stavke.

1.6 /threshold_stats

Method: GET
Prikazuje statistiku vezanu uz threshold mehanizme klasifikacije (npr. aktivni pragovi, povijesne promjene i frekvenciju prilagodbi).

1.7 /ping

Method: GET
Minimalni endpoint za provjeru dostupnosti servera. Vraća jednostavan odgovor bez dodatne obrade.

1.8 /active_nodes/html

Method: GET
HTML prikaz trenutno aktivnih čvorova u distribuiranom sustavu, uključujući njihove statuse i zadnje vrijeme aktivnosti.

1.9 /intruder_alerts

Method: GET
Vraća popis detektiranih intruder događaja u strukturiranom formatu pogodnom za daljnju obradu.

1.10 /intruder_alerts/html

Method: GET
HTML vizualizacija intruder alertova, namijenjena ljudskom nadzoru sustava.

1.11 /reload_dataset

Method: GET
Ručno ponovno učitavanje dataset-a za klasifikaciju bez potrebe za restartom servera.

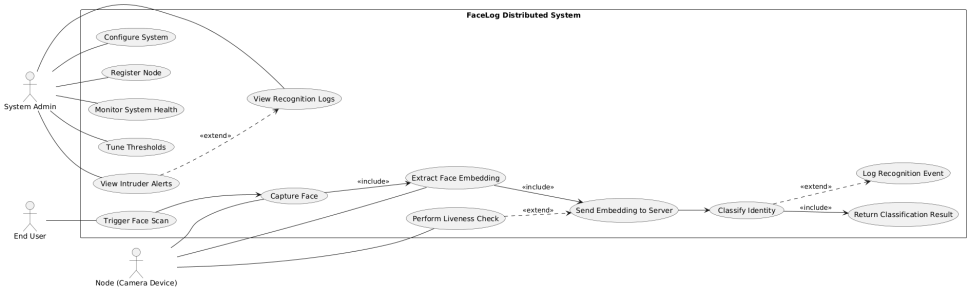

Figure 10: Use case dijagram

Use case dijagram prikazan na slici 10 opisuje funkcionalne interakcije unutar distribuiranog sustava. Sustav uključuje tri glavna aktera: *System Admin*, *End User* i *Node (Camera Device)*. Krajnji korisnik pokreće proces prepoznavanja iniciranjem skeniranja lica, čime se aktivira dohvat slike lica na razini čvora. Svaki čvor zadužen je za prikupljanje slike, ekstrakciju embeddinga te opcionalnu provjeru živosti prije slanja podataka prema centralnom poslužitelju.

Ekstrakcija ugrađenih vektora i njihovo slanje poslužitelju predstavljaju obavezne korake u postupku prepoznavanja, što je u dijagramu modelirano *include* relacijama. Nakon zaprimanja podataka, centralni poslužitelj provodi klasifikaciju identiteta i vraća rezultat prepoznavanja. Događaji prepoznavanja mogu se opcionalno zapisivati u sustav zapisa ovisno o konfiguraciji sustava i ishodu klasifikacije, što je prikazano korištenjem *extend* relacije.

Administratorske funkcionalnosti obuhvaćaju konfiguraciju sustava, registraciju čvorova, nadzor stanja sustava te podešavanje thresholda. Osim toga, sustav omogućuje pregled zapisa o prepoznavanju, pri čemu su upozorenja o neovlaštenim pokušajima pristupa (intruder alerts) dostupna kao proširenje osnovne funkcionalnosti pregleda zapisa. Ovakav pristup omogućuje jasnu podjelu odgovornosti između kamera i centralnog servera te podržava skalabilnu i prilagodljivu implementaciju sustava.

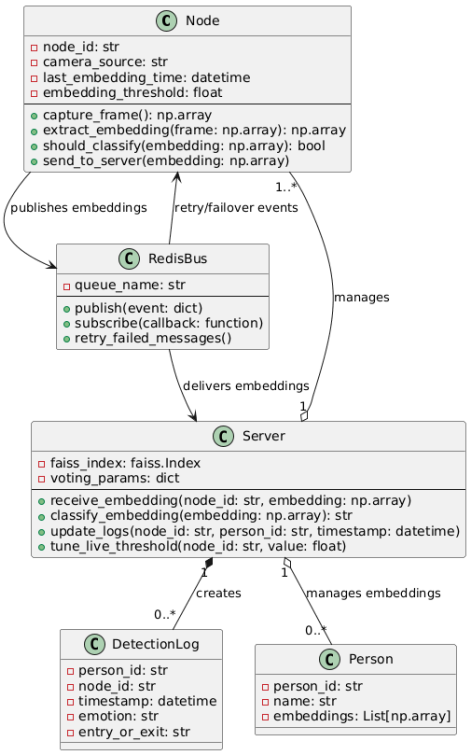

Figure 11: Class dijagram

Class dijagram distribuiranog sustava prikazuje glavne komponente i njihove međusobne odnose.

- **Node** predstavlja kameru koja bilježi frameove, ekstraktira embeddings i odlučuje kada ih poslati serveru. Svaki Node šalje podatke preko Redis busa.
- **Server** prima embeddinge od Node-ova, klasificira ih, ažurira `DetectionLog` i upravlja podacima o osobama (`Person`). Jedan Server može upravljati s više Node-ova (*1..**).
- **DetectionLog** bilježi informacije o detektiranim osobama, uključujući ime i vrijeme. Server može kreirati više logova (*0..**) za različite detekcije.
- **Person** sadrži informacije o osobi i pripadajuće embeddinge. Server upravlja s više osoba (*0..**).
- **Redis** omogućuje komunikaciju između Node-ova i Servera. Node objavljuje embeddinge (*publish*), a Server se pretplaćuje na njih (*subscribe*). Redis također omogućuje retry mehanizme za neuspjele poruke.

Veze na dijagramu:

- Server *agregira* Node-ove (1..*), što znači da Server upravlja životnim ciklusom Node-ova.
- Server *komponira* DetectionLog (0..*), jer logovi nastaju isključivo kroz Server i bez njega ne postoje.
- Server *agregira* Person objekte (0..*), jer Server upravlja njihovim embedding podacima, ali oni mogu postojati neovisno.
- Node šalje embeddinge putem Redis busa do Servera, a Redis osigurava retry/failover evente natrag Node-u.

Korisničke upute

Pokretanje sustava i načini izvođenja

Sustav je dizajniran tako da se može pokretati u više različitih konfiguracija, ovisno o razini tehničke složenosti, dostupnoj infrastrukturi i ciljanom scenariju korištenja. Time se omogućuje jednostavno testiranje u razvojnim uvjetima, ali i stabilno izvođenje u produkcijskom okruženju.

Opcija 1: Ručno pokretanje servera i node-ova

U ovoj konfiguraciji projekt se preuzima lokalno, a FastAPI server se pokreće izravno korištenjem Python interpretera, primjerice naredbom `python server_FASTAPI.py`. Redis servis mora biti prethodno pokrenut ili dostupan na lokalnoj mreži.

Node aplikacije se pokreću ručno, pri čemu se `node_0` koristi za rad s ugrađenom kamerom računala. Dodavanjem dodatnih kamera pokreću se novi node-ovi inkrementiranjem broja u nazivu skripte, identifikatora Node-a te indeksa kamere unutar koda.

Ovakav pristup je najjednostavniji za inicijalno testiranje i razvoj. Omogućuje potpunu kontrolu nad svakim dijelom sustava, ali zahtijeva ručnu konfiguraciju i veću pažnju pri upravljanju ovisnostima.

Opcija 2: Pokretanje servera putem Docker image-a

U drugoj konfiguraciji FastAPI server se pokreće unutar Docker container-a korištenjem unaprijed izgrađenog Docker image-a. Docker image sadrži aplikaciju zajedno sa svim potrebnim bibliotekama i ovisnostima, čime se eliminiraju problemi vezani uz različite verzije softvera.

Redis instanca se u ovom slučaju postavlja zasebno, bilo lokalno ili na udaljenom serveru. Node aplikacije se i dalje pokreću izravno na host sustavu kako bi imale pristup fizičkim kamerama.

Ovakav pristup povećava reproducibilnost i stabilnost server dijela sustava. Docker image osigurava da se aplikacija uvijek pokreće u identičnom okruženju, neovisno o operacijskom sustavu ili lokalnoj konfiguraciji.

Opcija 3: Pokretanje sustava pomoću Docker Compose-a

Treća i preporučena opcija koristi Docker Compose za simultano pokretanje FastAPI servera i Redis servisa. Pomoću `docker compose up -d` naredbe cijela serverska infrastruktura pokreće se automatski, uz jasno definirane mrežne veze i portove.

Docker Compose omogućuje deklarativno definiranje više servisa unutar jedne konfiguracijske datoteke. Time se pojednostavljuje upravljanje sustavom, smanjuje mogućnost konfiguracijskih grešaka i omogućuje brzo ponovno pokretanje cijelog sustava.

Node aplikacije se u ovom scenariju pokreću redom, pri čemu svaki Node ima vlastiti identifikator i indeks kamere. Redis portovi su jasno definirani unutar Compose konfiguracije, što osigurava pouzdanu komunikaciju između svih komponenti.

Uloga Docker image-a i Docker Compose-a

Docker image predstavlja zapakiranu verziju aplikacije zajedno s njezinim ovisnostima i konfiguracijom. Njegova glavna svrha je osigurati konzistentno izvršavanje aplikacije u različitim okruženjima bez dodatne konfiguracije.

Docker Compose nadograđuje ovaj koncept omogućujući orkestraciju više container-a istovremeno. Umjesto ručnog pokretanja pojedinih servisa, cijela arhitektura sustava definirana je na jednom mjestu.

Prednosti Docker Compose pristupa

Docker Compose predstavlja optimalno rješenje za ovaj sustav jer omogućuje:

- centralizirano upravljanje serverom i Redis servisom,
- brzu inicijalizaciju i ponovno pokretanje sustava,
- jasnu separaciju odgovornosti između komponenti,
- jednostavno skaliranje i prilagodbu konfiguracije.

Zbog navedenih razloga, Docker Compose pristup smatra se najprikladnijim za stabilno i dugotrajno izvođenje distribuiranog sustava za prepoznavanje lica u stvarnom vremenu.

Korisničke upute i interakcija sa sustavom

Nakon uspješnog pokretanja servera, Redis servisa i distribuiranih Node aplikacija, sustav je spreman za korištenje. Interakcija korisnika sa sustavom odvija se indirektno, putem vizualnog sučelja Node-ova i administracijskih API endpointa servera.

Rad korisnika s Node aplikacijama

Svaki Node predstavlja autonomnu jedinicu povezanog kamerom koja kontinuirano obrađuje video tok u stvarnom vremenu. Korisnik ne mora ručno upravljati klasifikacijom, već je cjelokupni proces automatiziran.

Nakon pokretanja Node-a, korisnik:

- postavlja kameru tako da je lice jasno vidljivo u kadru,
- osigurava adekvatno osvjetljenje kako bi se izbjegli tamni frame-ovi,
- ulazi u vidno polje kamere i zadržava se nekoliko trenutaka.

Node automatski detektira lice, provodi segmentaciju i ekstrakciju embeddinga te odlučuje hoće li podatke poslati na klasifikaciju prema centralnom serveru.

Vizualna povratna informacija korisniku

Tijekom rada, Node aplikacija prikazuje video tok s vizualnim oznakama:

- pravokutnike oko detektiranih lica,
- status obrade (npr. slanje embeddinga, čekanje odgovora),
- osnovne performanse poput FPS-a i trenutne latencije.

Ovakav prikaz omogućuje korisniku trenutačni uvid u stanje sustava i potvrdu da Node ispravno funkcionira.

Automatska klasifikacija i povratna informacija

Kada embedding bude poslan na server, klasifikacija se izvršava asinkrono. Rezultat klasifikacije (prepoznata osoba ili oznaka *Unknown*) obrađuje se bez potrebe za dodatnom korisničkom interakcijom.

U slučaju da lice nije prepoznato ili da više uzastopnih pokušaja završi kao nepoznato, sustav automatski generira zapis u intruder alert logu.

Interakcija s više Node-ova

Sustav podržava istovremeni rad više Node-ova povezanih na različite kamere. Svaki Node radi neovisno, ali svi dijele centralni server i Redis infrastrukturu.

Korisnik može:

- pokretati i zaustavljati pojedine Node-ove prema potrebi,
- dodavati nove kamere bez prekida rada servera,
- pratiti aktivne Node-ove putem administracijskih endpointa.

Ovakav pristup omogućuje jednostavno proširenje sustava bez dodatne rekonfiguracije.

Administrativni nadzor putem servera

Server nudi niz API endpointa koji omogućuju administrativni nadzor sustava. Putem web preglednika korisnik može pregledavati:

- logove detekcija i klasifikacija,
- zapise o neprepoznatim pokušajima,
- stanje Redis queue-a i aktivnih Node-ova,
- statistiku korištenih threshold vrijednosti.

Ovi endpointi omogućuju transparentan uvid u rad sustava i olakšavaju analizu performansi.

Zaustavljanje sustava

Node aplikacija se može zaustaviti pritiskom tipke `q`, pri čemu se kamera i svi resursi pravilno oslobađaju. Server i Redis servisi se mogu zaustaviti standardnim Docker ili sistemskim naredbama, ovisno o odabranoj konfiguraciji pokretanja.

Ovakav način rada osigurava sigurno i kontrolirano gašenje sustava bez gubitka podataka ili oštećenja stanja.

2 Zaključak

U ovom radu prikazan je distribuirani sustav prepoznavanja lica temeljen na CLIP + FAISS, s više Nodeova koji bilježe i šalju embeddinge Serveru putem Redis komunikacijskog sloja. Sustav uključuje napredne mehanizme poput liveness detekcije, tuneable live thresholda, te integracije vremenskih zona kroz tokenizirane Nodeove.

Class dijagram jasno ilustrira odnose između glavnih komponenti: svaki Server upravlja više Nodeova i osobama (*Person*), bilježi detekcije u logove (*DetectionLog*) i koordinira embeddinge za prepoznavanje. Redis služi kao pouzdani posrednik za razmjenu podataka i podržava retry/-failover mehanizme, čime se osigurava robusnost i skalabilnost sustava. Sustav je spreman za implementaciju u školama, tvrtkama i sigurnosnim aplikacijama gdje je potrebna brza, pouzdana i skalabilna identifikacija lica. Njegova fleksibilna arhitektura omogućuje jednostavno dodavanje novih Nodeova, senzora ili AI modula, što ga čini idealnim kandidatom za komercijalnu eksploataciju i razvoj inovativnih proizvoda u području nadzora i automatizacije.

Zaključno, dokumentirani sustav demonstrira napredne tehnike računalnog vida i distribuiranog procesiranja, te pruža čvrstu osnovu za stvaranje komercijalno atraktivnih i skalabilnih rješenja.