Kvaliteta usluge zaključivanja dubokom neuronskom mrežom u okolini računarstva na rubu

Banko, Leon

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 743

KVALITETA USLUGE ZAKLJUČIVANJA DUBOKOM NEURONSKOM MREŽOM U OKOLINI RAČUNARSTVA NA RUBU

Leon Banko

Zagreb, veljača 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 743

KVALITETA USLUGE ZAKLJUČIVANJA DUBOKOM NEURONSKOM MREŽOM U OKOLINI RAČUNARSTVA NA RUBU

Leon Banko

Zagreb, veljača 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 30. rujna 2024.

DIPLOMSKI ZADATAK br. 743

Pristupnik:	**Leon Banko (0036525224)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentorica:	prof. dr. sc. Ivana Podnar Žarko

Zadatak: **Kvaliteta usluge zaključivanja dubokom neuronskom mrežom u okolini računarstva na rubu**

Opis zadatka:

U raspodijeljenoj okolini Interneta stvari danas se sve češće za obradu podataka uz računalni oblak koriste uređaji na rubu mreže (engl. edge computing) koji se nalaze u lokalnoj mreži i u blizini izvora podataka, tj. IoT-uređaja na koje su spojeni senzori, npr. kamere. Uređaji na rubu su mrežno bliže izvoru podataka, čime se smanjuje cjelokupno kašnjenje obrade podataka pristiglih s IoT-uređaja zbog smanjenog vremena za prijenos podataka do obradnog čvora i usluge zaključivanja (engl. inference) na rubu. Ali uređaji na rubu imaju ograničena sredstva u odnosu na računalni oblak, što negativno utječe na preciznost i trajanje zaključivanja modelom duboke neuronske mreže, npr. za prepoznavanje broja i vrste vozila na temelju slika koje pristižu s kamera. Vaš je zadatak oblikovati i implementirati rješenje za usmjeravanja podataka do odgovarajuće usluge zaključivanja modelom duboke neuronske mreže u okolini računarstva na rubu ili u računalnom oblaku koja uzima u obzir različite zahtjeve na kvalitetu usluge zaključivanja (npr. kašnjenje, potrošnja energije, preciznost). Potrebno je ispitati performance zaključivanja odabrane duboke neuronske mreže na različitim uređajima s ograničenim sredstvima (npr. Raspberry Pi 4, Nvidia Jetson Nano). Proširite postojeće rješenje posrednika za usmjeravanje IoT-podataka do odgovarajuće usluge zaključivanja u okolini računarstva na rubu tako da uz postojeći cilj smanjenja mrežnog kašnjenja posrednik uzima u obzir različite zahtjeve na kvalitetu usluge zaključivanja. Ispitajte rad posrednika u emuliranom okruženju računarstva na rubu gdje na putu od ruba prema oblaku raste mrežno kašnjenje, ali se mijenjaju i dostupna sredstva obradnih čvorova.

Rok za predaju rada: 14. veljače 2025.

Sadržaj

Uvod

Sve veća uporaba umreženih uređaja i programskih rješenja iz područja Interneta stvari (eng. *Internet of Things*, IoT) u današnje vrijeme utječe na eksponencijalni rast količine generiranih podataka i mrežnog prometa prema IoT-platformama smještenim u računalnom oblaku. S obzirom na primjene IoT-rješenja u sektoru prometa i industrije javlja se potreba za obradom podataka u stvarnom vremenu i zahtjev niskog kašnjenja prilikom udaljenog upravljanja uređajima. Tradicionalne paradigme temeljene na računalnom oblaku, iako učinkovite i skalabilne, često ne mogu zadovoljiti stroge zahtjeve kašnjenja kod nekih primjena kao što su npr. proširena stvarnost, autonomna vozila i industrijska automatizacija. Ovaj je izazov doveo do pojave računarstva na rubu kao mogućeg rješenja, u kojem se obrada podataka približava izvoru podataka, smanjujući kašnjenje i generirani promet prema oblaku. Računarstvo na rubu omogućuje pokretanje modela strojnog učenja (ML) izravno na krajnjem uređaju, omogućujući donošenje odluka u stvarnom vremenu bez oslanjanja na udaljene poslužitelje u oblaku. Međutim, isti model ne može se izvoditi u oblaku i uređaju ograničenih resursa. Ova primjena uvodi dodatni izazov pronalaženja ravnoteže između kašnjenja i točnosti modela. Složeniji modeli pružaju veću točnost prilikom zaključivanja, ali također zahtijevaju značajna računalna sredstva te mogu povećati vrijeme zaključivanja. Nasuprot tome, jednostavniji modeli mogu postići niže kašnjenje, ali moraju žrtvovati točnost zaključivanja, potencijalno degradirajući učinkovitost aplikacije. Nalaženje kompromisa između kašnjenja i točnosti ključna je za maksimiziranje performanci i učinkovitosti prilikom primjene dubokih neuronskih mreža za zaključivanje na računalnim sustavima i uređajima na rubu mreže.

Cilj ovog rada bio je oblikovati rješenje za usmjeravanje podataka s IoT-uređaja do odgovarajuće usluge zaključivanja modelom duboke neuronske mreže u okolini računarstva na rubu. Prošireno je postojeće rješenje posrednika tako da uz postojeći cilj smanjenja mrežnog kašnjenja posrednik uzima u obzir točnost zaključivanja.

Sustav je ispitan u emuliranom okruženju, pri čemu se razmatraju praktični izazovi povezani s postavljanjem modela strojnog učenja na uređaje na rubu mreže koji su svjesni kašnjenja u svom računalnom okruženju, što uključuje ograničenja resursa, varijabilnost mreže i energetsku učinkovitost. Za uslugu zaključivanja iskorištena je neuronska mreža

PilotNet, koja računa potrebne kutove zakretanja volana na temelju slike ceste ispred automobila.

Prvo poglavlje uvodi prednosti i izazove povezane s računarstvom na rubu te sadrži pregled uređaja koji se mogu iskoristiti za obradu na rubu mreže. U drugom poglavlju su testom performanci uspoređena vremena zaključivanja dvaju inačica modela PilotNet na različitim uređajima. Treće poglavlje opisuje postojeće rješenje za usmjeravanje, kao i promjene kojima je ono prilagođeno radu s modelima strojnog učenja u heterogenim okolinama računarstva na rubu. U četvrtom poglavlju je opisana simulirana okolina za ispitivanje i predstavljeni su rezultati eksperimenata kojima je provjeren rad prilagođenog sustava za usmjeravanje.

1. Umjetna inteligencija na rubu mreže

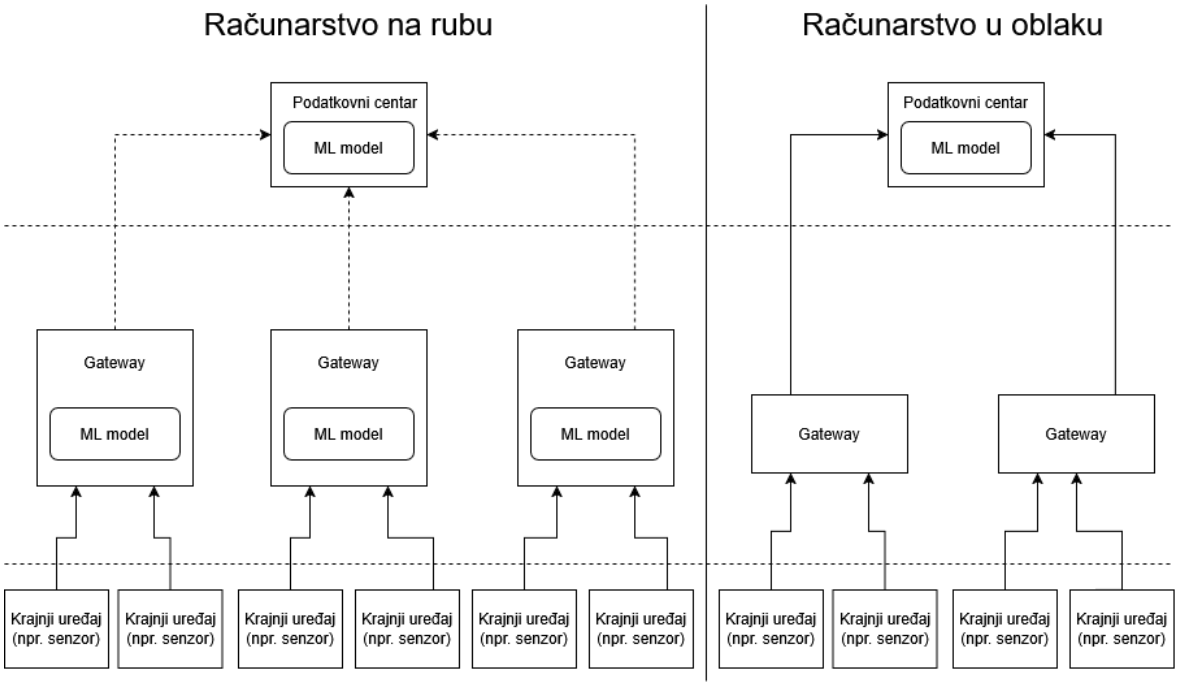
Mnoga IoT-rješenja temeljena na oblaku koriste središnji poslužitelj za obradu podataka koje generiraju krajnji uređaji. Sustavi poput uređaja u Internetu stvari i povezanih vozila često zahtijevaju obradu senzorskih podataka u stvarnom vremenu za donošenje odluka i izvođenje radnji. Pri tome se prijenos velikih količina mrežnog prometa prema podatkovnim centrima može pokazati kao usko grlo koje povećava kašnjenje i time degradira kvalitetu usluge (*Quality of Service*) i kvalitetu iskustva (*Quality of Experience*).

Računarstvo na rubu je arhitektura u kojoj se proces obrade i pohranjivanja podataka odvija blizu izvora podataka [11]. Iskorištava kapacitete pohrane i obrade velikog broja uređaja povezanih s Internetom postavljenih u svrhu pružanja međusloja između krajnjih uređaja i oblaka. Takva arhitektura pruža nekoliko prednosti nad arhitekturom računarstva u oblaku. Kašnjenje je smanjeno jer se izvorni podaci obrađuju blizu izvora podataka, umjesto da se prenose na udaljeni podatkovni centar, što smanjuje troškove komunikacije u odnosu na arhitekturu računarstva u oblaku. Računarstvo na rubu oslanja se na lokalnu obradu, a ne na centralizirane podatkovne centre u oblaku, no mnogi uređaji na rubu mreže kao što su IoT senzori, ugradbeni sustavi i mobilni uređaji ograničeni su resursima. Čvorovi na rubu, poput baznih stanica, često nemaju adekvatne računalne mogućnosti, a postojeća softverska rješenja često su specifična za hardver, što otežava prenosivost. Nadogradnja čvora na rubu je skupa i složena. Ova ograničenja stvaraju značajne izazove za performancama, energetskom učinkovitosti i pouzdanosti sustava. Stoga pokretanje modela umjetne inteligencije na uređajima s ograničenim resursima zahtijeva tehnike optimizacije modela poput kvantizacije [5].

Ostali izazovi u računarstvu na rubu uključuju učinkovito otkrivanje i upravljanje heterogenim čvorovima na rubu, optimalno dijeljenje i raspoređivanje zadataka po čvorovima decentralizirane mreže, održavanje performanci uz sprječavanje preopterećenosti čvorova te osiguranje sigurne i pouzdane komunikacije među čvorovima [3].

Orkestracija kontejnera je proces automatizacije postavljanja, upravljanja, skaliranja i umrežavanja kontejnera. U heterogenim i raspodijeljenim sustavima, orkestracija pruža poboljšanu skalabilnost, upravljanje računalnim resursima, toleranciju na greške i

otpornost. Kubernetes je platforma otvorenog koda za orkestraciju kontejnera [19]. K3s je Kubernetes distribucija osmišljena za sustave s ograničenim resursima kao što su IoT, poslužitelji na rubu mreže i sustavi temeljeni na ARM-u [20]. Zadržava osnovne mogućnosti Kubernetesa uz smanjenje složenosti i upotrebe resursa. Ima maleni otisak, s minimalnim zahtjevima za RAM i CPU, što ga čini prikladnim čak i za uređaje sa samo 512 MB radne memorije. Instalacija je brza i jednostavna budući da dolazi kao jedna binarna datoteka s minimalnim ovisnostima. Optimizirana je za računarstvo na rubu i IoT okruženja. Može učinkovito upravljati obradom podataka u stvarnom vremenu. Energetski je učinkovita, troši manje energije, što ju čini prikladnom za uređaje koji koriste baterije. Osim toga, K3s je siguran i skalabilan, s automatskim ažuriranjima i mogućnošću integracije s Kubernetesovim klasterima temeljenima na oblaku.



Slika 1.1 Usporedba računarstva na rubu i računarstva u oblaku

Za razliku od računarstva u oblaku, gdje se računalno intenzivni modeli strojnog učenja pohranjuju i pokreću u podatkovnim centrima, kod računarstva na rubu računanje je decentralizirano, tj. modeli strojnog učenja postavljaju se i pokreću blizu izvora podataka (Slika 1.1).

1.1. Metode kompresije modela strojnog učenja

Na modelima strojnog učenja se može primijeniti nekoliko tehnika kompresije. Ove tehnike naročito su korisne kod neuronskih mreža, gdje zbog prekomjernog broja parametara često dolazi do visokog vremena zaključivanja. Uporabom tehnika kompresije moguće je smanjiti vrijeme zaključivanja pod cijenu smanjena točnosti mreže. Osim rješavanja problema prevelikog broja parametara ove tehnike se mogu iskoristiti kako bi se učinkovitije izvodile na uređajima s ograničenim sredstvima. Ovo poglavlje opisuje nekoliko često korištenih tehnika kompresije.

1.1.1. Pruning

Pruning je široko korištena metoda za smanjenje parametara u unaprijed treniranim dubokim neuronskim mrežama (DNN), s ciljem smanjenja veličine pohrane i vremena izvođenja uz održavanje točnosti kroz ponovnu obuku. *Pruning* je metoda kompresije koja uključuje uklanjanje težina treniranog modela [1]. Iteracije uključuju iterativno smanjenje parametra i ponovno treniranje dok se ne postigne željena veličina mreže i ravnoteža točnosti. Ovaj proces usporediv je s ljudskim učenjem, gdje se irelevantne informacije iterativno izdvajaju iz prošlih iskustava. Obično je potrebno ponovno treniranje modela kako bi se vratila točnost nakon kompresije. Broj iteracija ponovne obuke i njihova računalna cijena mogu biti znatni. Najjednostavnija strategija koristi prag za odlučivanje koje težine ili jedinice ukloniti. Alternativno, unaprijed definirani postotak parametra modela može se smanjiti, često ciljajući one najbliže nuli.

Tehnike se dijele na temelju toga je li potrebno ponovno treniranje modela i kriterija koji se koriste za odabir parametara za uklanjanje [1]. Ovi kriteriji uključuju metode temeljene na veličini (uklanjanje najmanjih pondera), na temelju regularizacije (dodavanje kazni za poticanje malih pondera), na temelju osjetljivosti na gubitak (uklanjanje pondera s minimalnim učinkom na gubitak) i metode temeljene na pretraživanju (npr. upotrebom genetskih algoritama za pronalaženje optimalne podmreže). Također postoji razlika između strukturiranih i nestrukturiranih tehnika.

Strukturirani *pruning* uklanja čitave skupine težina, poput slojeva, filtera u konvolucijskom sloju ili čitavih neurona. Uklanjanjem cijelih strukturiranih grupa težina, metoda smanjuje opseg izračuna koji bi se morali napraviti u prolasku kroz graf težina modela. Time se čuva struktura mreže, omogućujući korištenje učinkovitog množenja guste matrice (*dense*

matrix) i dovodeći do bržeg računanja. Međutim, manje je fleksibilan od nestrukturiranog *pruninga* i potencijalno propušta neke pojedinačno važne težine.

S druge strane, nestrukturirani *pruning* je jednostavniji pristup. Podrazumijeva uklanjanje pojedinačnih neurona ili težina u cijeloj mreži, gdje god se utvrdi da imaju nisku istaknutost (osjetljivost). To znači da uklanjanje ovih elemenata minimalno utječe na izlaz ili funkciju gubitka modela. Nestrukturirani *pruning* može biti agresivan, uklanjajući velik dio parametara, često s malim utjecajem na izvedbu generalizacije. Međutim, to dovodi do provedbe operacija nad rijetkom matricom (*sparse matrix*), koje su računalno skupe i zahtijevaju specijalizirane biblioteke za množenje rijetkih matrica. Opći pristupi nestrukturiranom *pruningu* koriste minimalne pragove ovisno o samim težinama ili njihovim aktivacijama kako bi se odredilo treba li se pojedinačni parametar smanjiti ili ne. Ako parametar ne dosegne definirani prag, poništava se na nulu.

1.1.2. Kvantizacija

Kvantizacija je sažimanje vrijednosti korištenjem manje bitova. U slučaju neuronskih mreža primjenjuje se na težine, aktivacije i gradijente [2]. Kvantizacija uključuje preslikavanje kontinuiranog raspona vrijednosti (u slučaju neuronskih mreža, raspona težina, pristranosti ili aktivacija) u manji diskretni skup. Na primjer, 32-bitna vrijednost s pomičnim zarezom može se mapirati u 8-bitni cijeli broj.

Glavne vrste kvantizacije neuronskih mreža su:

- Uniformna kvantizacija: Vrijednosti se preslikavaju na ravnomjerno raspoređene razine. Dijeli se na simetričnu (simetrični raspon oko nule) i asimetričnu (proizvoljan raspon).
- Neuniformna kvantizacija: Razine i koraci kvantizacije nisu ravnomjerno raspoređeni (npr. logaritamska distribucija). To omogućuje bolji prikaz neuniformnih distribucija podataka, ali je općenito složenija za implementaciju od uniformne.
- Statička kvantizacija: Raspon vrijednosti koje treba kvantizirati unaprijed je izračunat i fiksiran tijekom zaključivanja.
- Dinamička kvantizacija: Raspon se izračunava dinamički za svaku aktivacijsku mapu tijekom izvođenja.

- Kvantizacija po slojevima, grupama, kanalima ili podkanalima: odnosi se na granuliranost pri kojoj se određuje raspon odsijecanja za težine.
- Kvantizacija mješovite preciznosti: Različiti slojevi ili operacije kvantizirani su s različitim preciznostima bita (npr. neki slojevi koriste INT8, drugi INT4) kako bi se uravnotežila točnost i učinkovitost korištenjem veće preciznosti za kritične dijelove mreže.
- Simulirana kvantizacija (lažna kvantizacija): pohranjuju se vrijednosti niske preciznosti, ali se izračuni i dalje izvode pomoću pokretnog zareza.
- Kvantizacija samo za cijeli broj: Sve se operacije izvode pomoću aritmetike cjelobrojnih brojeva niske preciznosti, u potpunosti iskorištavajući prednosti hardvera niske preciznosti.
- Ekstremna kvantizacija (binarizacija/ternarizacija): težine i aktivacije predstavljene su samo uz pomoć 1 odnosno 2 bita. Ovo značajno smanjuje memorijski otisak i troškove računanja, ali zahtijeva metode za ublažavanje gubitka točnosti.
- Vektorska kvantizacija: Grupira težine u klastere i koristi težište svakog klastera kao kvantiziranu vrijednost.

Također postoje različiti pristupi za uključivanje kvantizacije u proces treniranja, a to su *Quantizaiton aware training* (QAT) i *Post-training quantization* (PTQ).

1.1.3. Destilacija znanja

Destilacija znanja je učenje manje mreže pomoću veće mreže korištenjem nadzora veće mreže i uključuje minimiziranje entropije, udaljenosti ili odstupanja između njihovih procjena vjerojatnosti. Performance manje mreže su degradirane kada je jaz između manje i veće prevelik da manja može učiti [1].

1.2. Uređaji za računarstvo na rubu

Uređaji NVIDIA Jetson su napredne, kompaktne računalne platforme dizajnirane za pokretanje aplikacija koje koriste umjetnu inteligenciju na rubu. Sa svojim moćnim GPU-ima, energetski učinkovitom arhitekturom i podrškom za duboko učenje i računalni vid, ovi uređaji idealni su za računarstvo na rubu mreže, gdje se obrada podataka odvija blizu izvora podataka. Ovi uređaji široko su primjenjivi u područjima kao što su robotika,

autonomna vozila, pametni gradovi i industrijski IoT, budući da mogu izvoditi složene algoritme umjetne inteligencije u stvarnom vremenu s niskim kašnjenjem. Jetson moduli, kao što su Jetson Nano, Jetson Xavier NX i Jetson Orin, pružaju skalabilna rješenja i podržavaju niz aplikacija. Uz pomoć programske podrške kroz NVIDIA-in JetPack SDK i okvire za strojno učenje kao što su TensorFlow i PyTorch, Jetson uređaji omogućuju izgradnju i korištenje modela na rubu mreže, poboljšavajući učinkovitost, skalabilnost i inovacije u sustavima vođenim umjetnom inteligencijom.

	Jetson Nano 4GB	Jetson AGX Xavier	Jetson AGX Orin	Jetson TX2
CPU	Quad-core ARM Cortex-A57 MPCore	8-core NVIDIA Carmel Arm	12-core NVIDIA Arm Cortex A78AE	Dual-Core NVIDIA Denver 2 i Quad-Core Arm Cortex-A57 MPCore
GPU	Maxwell arhitektura, 128 NVIDIA CUDA jezgri	512-jezgreni, NVIDIA Volta, 64 Tensor jezgri	1792-jezgreni, NVIDIA Ampere, 56 Tensor jezgri	256-jezgreni, Pascal
Radna memorija	4 GB 64-bit LPDDR4, 25.6 GB/s	32GB 256-bit LPDDR4x 136.5GB/s	32GB 256-bit LPDDR5 204.8GB/s	8GB 128-bit LPDDR4 59.7GB/s
AI Performance	472 GFLOPs	32 TOPS	200 TOPS	1.33 TFLOPS

Tablica 1.1 Tehničke specifikacije različitih Nvidia uređaja

Različiti NVIDIA uređaji posjeduju širok raspon komponenti i stoga podržavaju širok spektar slučajeva uporabe (Tablica 1.1). Za ovaj rad su korišteni uređaji NVIDIA Jetson Nano 4GB i NVIDIA Jetson AGX Orin. Jetson Nano je jeftin, energetski učinkovit uređaj prikladan za jednostavne zadatke umjetne inteligencije kao što su detekcija objekata i robotika. Međutim, njegova ograničena računalna snaga čini ga neprikladnim za složene zadatke umjetne inteligencije kao što je obrada videozapisa u stvarnom vremenu. S druge strane, zahvaljujući snažnijim komponentama Orin AGX pruža mogućnost obavljanja zahtjevnijih zadataka kao što su upravljanje autonomnim vozilima, industrijska robotika i obrada videozapisa u stvarnom vremenu. Njegova visoka cijena i potrošnja energije mogu biti previsoki za male projekte i projekte s ograničenim resursima.

Sustavi autonomne vožnje temeljeni na računarstvu na rubu su pokretni. Često imaju vrlo stroga ograničenja potrošnje energije, kašnjenja i otpornosti na ispade. Kako bi jamčili sigurnost autonomnih vozila, velike količine podataka potrebno je obraditi u što kraćem vremenu [17]. Pored problema pronalaženja kompromisa točnosti i brzine izvođenja, čest problem koji se javlja prilikom autonomne vožnje je raspodjela zadataka obrade između mobilnog uređaja i poslužitelja u oblaku ili na rubu mreže. Jedno rješenje za rasterećenje mobilnog uređaja pomoću poslužitelja na rubu mreže je Edgent. Adaptivnim particioniranjem neuronskih mreža dio izračuna delegira se poslužitelju na rubu, dok se konfiguracijom ranih izlaza (*early exit*) neuronske mreže djelomično žrtvuje točnost i smanjuje kašnjenje kako bi se postigla ravnoteža između ta dva parametra [4].

Ostali radovi o analizi videozapisa na rubu mreže često se bave obradom slika primljenih od nadzornih kamera u svrhu praćenja prometa u pametnom gradu. Za nadzor prometa obično se koriste neuronske mreže za detekciju i klasifikaciju objekta poput YOLOv3 [14], budući da su brze i pružaju visoku točnost. Pristupi temeljeni na kamerama i računalnom vidu pružaju širok raspon slučajeva upotrebe i fleksibilniji su od ostalih pristupa (npr. postavljanje magnetometra, ultrazvučnih ili infracrvenih senzora), no za obradu slika na rubu zahtijevaju računala visokih performanci opremljena GPU-ovima i velikom količinom memorije [13]. Za ovaj slučaj upotrebe mogu se iskoristiti NVIDIA Jetson Nano i TX2. Osim detekcije objekta postoje i algoritmi za praćenje kretnji objekta na slici. Algoritam SORT uparuje otkrivene objekte u trenutnom okviru s onima iz prethodnog okvira [14]. Brzinu obrade slike na rubu moguće je poboljšati prethodnom obradom podataka u svrhu eliminacije redundantnih okvira, što smanjuje zahtjeve obrade, pohrane i mrežne propusnosti i povećava učinkovitost analize videozapisa [15].

Osim odabira poslužitelja, za rasterećenje sustava i postizanje kompromisa između točnosti i vremena izvođenja može se regulirati rezolucija ulazne slike. Slika s nižom rezolucijom ima manju veličinu i niže troškove računanja, a samim tim i manje kašnjenje prilikom prijenosa i obrade. Pristupi temeljeni na potpomognutom učenju (*reinforcement learning*) mogu se iskoristiti za zadatke rasterećenja i pronalaženja balansa između točnosti i vremena izvođenja, bez *a priori* informacija o mreži i poslužiteljima, isključivo na temelju povijesnih podataka o resursima. Takvi pristupi definiraju funkciju korisnosti, koja se sastoji od QoE parametara, npr. može biti težinski zbroj brzine i točnosti obrade slike [16], te može uključivati ostale parametre poput potrošnje energije [6]. Primjerice, ako se problem odabira rezolucije i poslužitelja formulira kao Multi-armed bandit (MAB)

problem, u kojemu se agentu nudi više fiksnih opcija od kojih svaka daje nagradu izvučenu iz nepoznate distribucije vjerojatnosti, otvara se mogućnost korištenja online učenja kako bi se kombinirali trenutni i povijesni podaci o mreži i resursima [16]. Agent ima za cilj maksimizirati kumulativnu nagradu tijekom niza pokušaja. Izazov leži u odabiru najbolje opcije za povlačenje, balansiranju potrebe za istraživanjem različitih opcija kako bi se saznalo o njihovoj distribuciji nagrada i iskorištavanju poznatih opcija koji su osigurali visoke nagrade.

Iako brojne aplikacije mogu imati koristi od smanjenog kašnjenja u odnosu na računarstvo u oblaku, glavni problem računarstva na rubu je što mobilni uređaji ne posjeduju dovoljno računalne snage za složene zadatke umjetne inteligencije. Zbog toga je važno odabrati algoritam strojnog učenja koji odgovara specifikacijama uređaja s ograničenim resursima. Neuronske mreže zahtijevaju značajnu količinu memorije i računalne snage za predviđanje i treniranje, no mogu se optimizirati za brže predikcije. Tree-based algoritmi su posebno prikladni za IoT uređaje s ograničenim resursima jer zahtijevaju vrlo malo memorije i računalne snage. Glavni izazovi korištenja algoritma K-NN za računanje na rubu su veličina podataka za obuku, vrijeme predikcije i izbor metrike udaljenosti. Međutim, postoje jednostavnije inačice algoritma koje ga čine prikladnim za izvođene na uređajima s ograničenim resursima. Algoritam SVM je uobičajen izbor za izvođenje na rubu. Izbor komunikacijskog protokola za strojno učenje na rubu mreže uključuje kompromis između potrošnje energije, dometa, brzine prijenosa podataka, kašnjenja i cijene. LPWAN tehnologije prikladne su za aplikacije niske potrošnje, dugog dometa s rijetkim prijenosom podataka, dok su mobilne i WiFi tehnologije prikladnije za aplikacije koje zahtijevaju veće brzine prijenosa podataka i manje kašnjenje. Integracija algoritma strojnog učenja s komunikacijskim protokolom pruža priliku za dodatnu optimizaciju performanci i energetske učinkovitosti [5].

Radovi koji proučavaju odnos između potrošnje energije i različitih vrsta mrežnih arhitektura kao što su centralizirane arhitekture temeljene na oblaku, *fog computing* i računarstvo na rubu koriste različite energetske modele za usporedbu. Primjerice, analiza slučaja koja proučava utjecaj računarstva na rubu na potrošnju energije IoT uređaja pokazuje da računarstvo na rubu značajno smanjuje potrošnju energije, posebice s češćim prijenosom podataka [9].

Također, predložen je pristup temeljen na taksonomiji za kategorizaciju arhitektura u oblaku i energetskom modelu. Četiri različite arhitekture u rasponu od potpuno

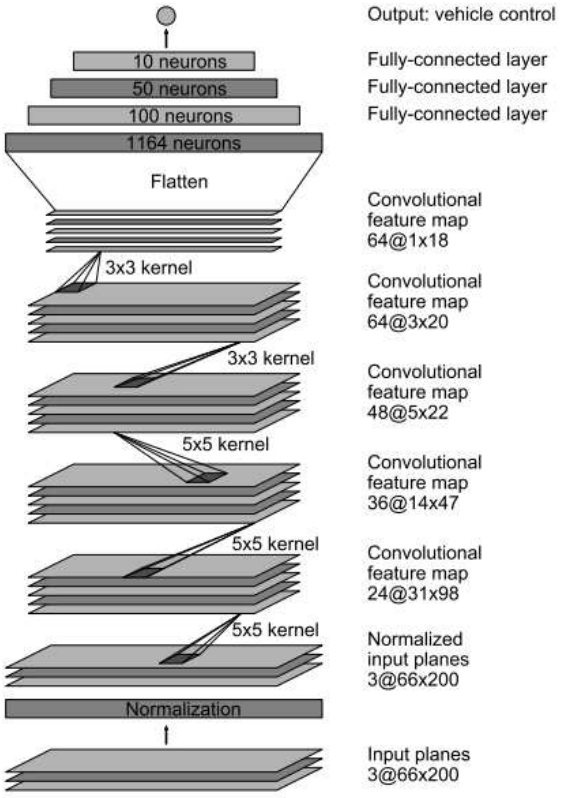
centralizirane do potpuno decentralizirane analizirane su pomoću energetskog modela. Rezultati eksperimenta su pokazali da potpuno distribuirana arhitektura troši između 14% i 25% manje energije od potpuno centralizirane odnosno djelomično distribuirane arhitekture. Promatranjem energetske učinkovitosti, predloženi model proučava utjecaj rashladnih sustava koji trenutno nisu zanemarivi u smislu potrošnje energije [10].

2. Mjerenje vremena zaključivanja i potrošnje energije prilikom zaključivanja

Autonomna vožnja automobila jedna je od primjena strojnog učenja koja osim točnosti, zahtjeva iznimno kratko vrijeme zaključivanja. Kako bi se omogućila kombinacija tih dvaju parametra kvalitete usluge, mogu se iskoristiti metode kompresije opisane u prošlom poglavlju. Korisnik može regulirati željeni omjer točnosti i vremena zaključivanja, ako istovremeno ima na raspolaganju model visoke točnosti i njegovu smanjenu inačicu, koja pruža rezultate manje točnosti i kraćeg kašnjenja. U ovom poglavlju je opisano nekoliko inačica duboke neuronske mreže PilotNet. Također, ispitane su performance dvaju modela kako bi se utvrdilo jesu li prikladne za korištenje u testnoj okolini za potrebe ovog rada.

2.1. Opis korištenih modela

NVIDIA je razvila neuronsku mrežu koja može upravljati automobilom. PilotNet [8] koristi regresijski model čiji je ulaz slika primljena od prednje kamere na automobilu, a izlaz broj stupnjeva zakretanja volana. Model koristi arhitekturu koja se sastoji od 9 slojeva, uključujući normalizacijski sloj, 5 konvolucijskih slojeva i 3 potpuno povezana sloja. Sustav donosi odluke na temelju istaknutih objekata na slici (npr. rub ceste ili crta na cesti) koje neuronska mreža izdvaja od pozadine. Trenirana je promatranjem vozača automobila s kamerama [8].



Slika 2.1 Slojevi originalne neuronske mreže PilotNet [8]

Radi optimizacije PilotNet-a za manje uređaje s ograničenim resursima, razvijena je modificirana verzija s jedinstvenim dizajnom [7]. Prilagodba je značajno smanjila parametre modela i korištenje memorije, poboljšavajući njegovu prikladnost za uređaje s ograničenim resursima. Rad predstavlja rješenje modela dubokog učenja za autonomnu vožnju PilotNet za upravljanje. Budući da izvorni PilotNet zahtijeva velike resurse, autori su ga modificirali korištenjem dubinski odvojivih konvolucija i slojeva uskog grla, smanjujući parametre za 62% i kašnjenje zaključivanja. Modificirani model postigao je točnost usporedivu s originalom. Sustav je ispitan u video igri Grand Theft Auto V pod različitim uvjetima (vrijeme, doba dana, vrsta ceste). Uz to, smanjeno je vrijeme zaključivanja, čime je poboljšana operativna učinkovitost. Unatoč ovim izmjenama, modificirani PilotNet održava usporedive gubitke i točnost s izvornim modelom. I izvorna i modificirana PilotNet verzija pokazale su slične performance autonomne vožnje, pri čemu su vožnja autocestom i dnevni uvjeti dali bolje rezultate od gradske vožnje odnosno noćnih uvjeta. Modificirani PilotNet nudi značajna poboljšanja u veličini i brzini, što ga čini prikladnim za uređaje s ograničenim resursima.

Budući da prema provedenim eksperimentima, modificirana verzija nije značajno smanjila vrijeme izvođenja predikcije na odabranim uređajima prilikom slanja pojedinačnih slika,

originalni i modificirani modeli PilotNeta su kvantizirani korištenjem int8 kvantizacije. Kvantizacija int8 se često preferira na uređajima s ograničenim resursima jer još uvijek pruža dovoljnu preciznost za mnoge zadatke dubokog učenja, uz veliko smanjenje veličine modela i troškova računanja. Modeli su kvantizirani pomoću okvira LiteRT (TfLite) biblioteke, uporabom kvantizacije dinamičkog raspona. Za obje kvantizirane PilotNet instance format korišten za ulazni sloj je float32, konvolucijski slojevi koriste int8, *flatten* sloj ostaje u float32, a potpuno povezani slojevi su u int8.

2.2. Poslužitelj PilotNet

U sklopu ovog rada implementiran je poslužitelj PilotNet za rukovanje unosom slike i izradu predviđanja pomoću unaprijed obučenog modela. Svrha ovog poslužitelja je umrežavanje modela PilotNet s uređajima i procesima koji šalju slike, tj. s klijentima. Iskorišten je kao usluga prema kojoj se usmjeravaju korisnički zahtjevi prilikom ispitivanja sustava u poglavlju 4. Nakon što je slikovna datoteka poslana putem HTTP zahtjeva, skripta je obrađuje pomoću biblioteke OpenCV. Slika se pretvara u crno-bijelu sliku, mijenja joj se veličina na fiksnu razlučivost od 160x120 piksela, a njezine dimenzije se proširuju kako bi odgovarale formatu unosa koji zahtijeva model. Obrađena slika zatim se prosljeđuje učitanom modelu biblioteke TensorFlow Keras, koji generira predikcije. Rezultat predikcije šalje se u HTTP odgovoru. Ovaj je program dizajniran za rad u Docker kontejneru, što mu omogućuje jednostavno i dosljedno pokretanje u različitim sustavima i klasterima.

2.3. Prvi skup mjerenja - mjerenja bez varijable batch_size

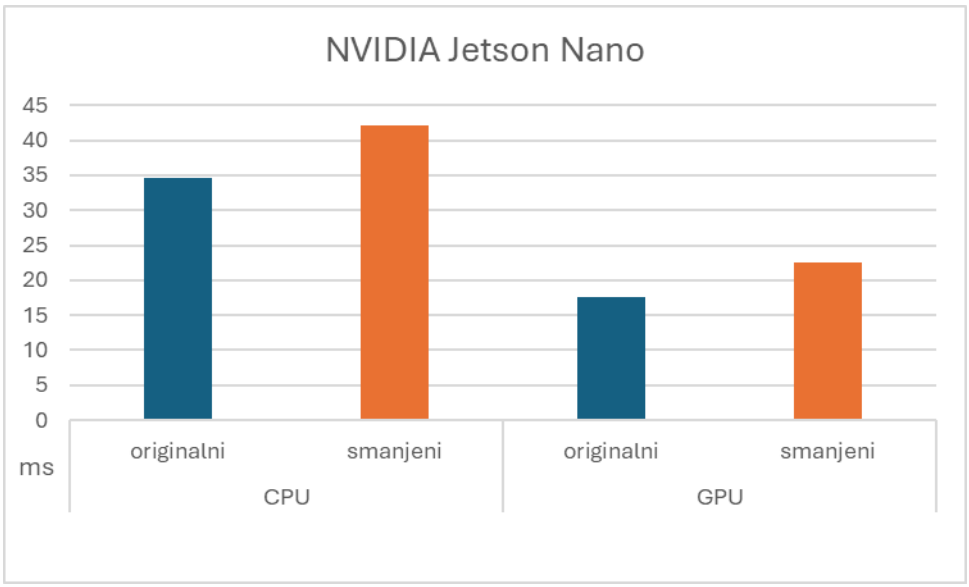
Radi usporedbe performanci originalnog modela PilotNet i optimizirane inačice s manje parametara, provedena su mjerenja na uređajima Nvidia Jetson Nano i Nvidia Orin AGX. Provedeno je ukupno 8 mjerenja s 3 parametra:

- uređaj (Jetson Nano, Orin AGX)
- model (originalni, smanjeni)
- komponenta uređaja (CPU, GPU)

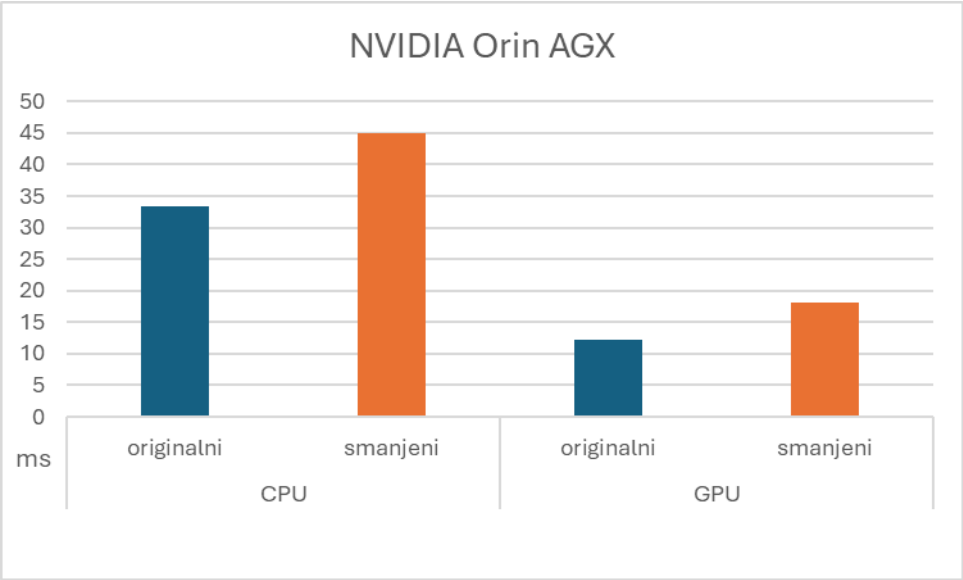
Prilikom svakog mjerenja kao ulaz je iskorišten isti uzorak od 50 fotografija iz PilotNetovog skupa podataka za testiranje. Slike korištene za mjerenje nisu bile dio batcha, nego je za svaku sliku pokrenuta i izmjerena zasebna predikcija. Osim vremena zaključivanja, izmjerena je i potrošnja energije pomoću WLAN utičnice. Za svaku kombinaciju uređaja, modela i komponente je izračunata srednja vrijednost (prosjek i medijan).

Važno je napomenuti da prva predikcija izvršena na GPU-u traje dugo, nakon čega se vrijeme zaključivanja za sve sljedeće zahtjeve značajno smanjuje. Zbog toga je u ovom slučaju bolje koristiti medijan kao srednju vrijednost.

Dijagrami prikazuju medijane svakog provedenog mjerenja na dva različita uređaja u milisekundama.



Slika 2.2 Medijani vremena zaključivanja za Jetson Nano



Slika 2.3 Medijani vremena zaključivanja za Jetson Orin AGX

Rezultati pokazuju da kompresija modela nije skratila vrijeme izvođenja predikcije, čak i na uređaju sa slabijom snagom CPU-a i GPU-a. Sukladno očekivanjima, AGX Orin izvodi predikciju brže od uređaja Jetson Nano, a izvođenje predikcije na CPU-u je sporije nego izvođenje na GPU-u istoga uređaja. Zanimljivo je uočiti da uređaj AGX Orin nudi bolje performance samo u slučaju kada se koristi GPU, dok su rezultati na oba uređaja usporedivi prilikom primjene CPU-a. Iz ovoga se može zaključiti da je uređaj Orin AGX uz primjenu GPU-a najbrži za izvođenje zaključivanja primjenom ispitanih modela.

2.4. Rezultati mjerenja potrošnje energije

Prilikom pokretanja predikcija na uređaju Jetson Nano potrošnja je iznosila 4 W, a prilikom predikcija na uređaju Orin AGX potrošnja je iznosila 10 W. Na oba uređaja potrošnja izmjerena prilikom pokretanja predikcija na CPU-u je bila jednaka potrošnji prilikom pokretanja na GPU-u.

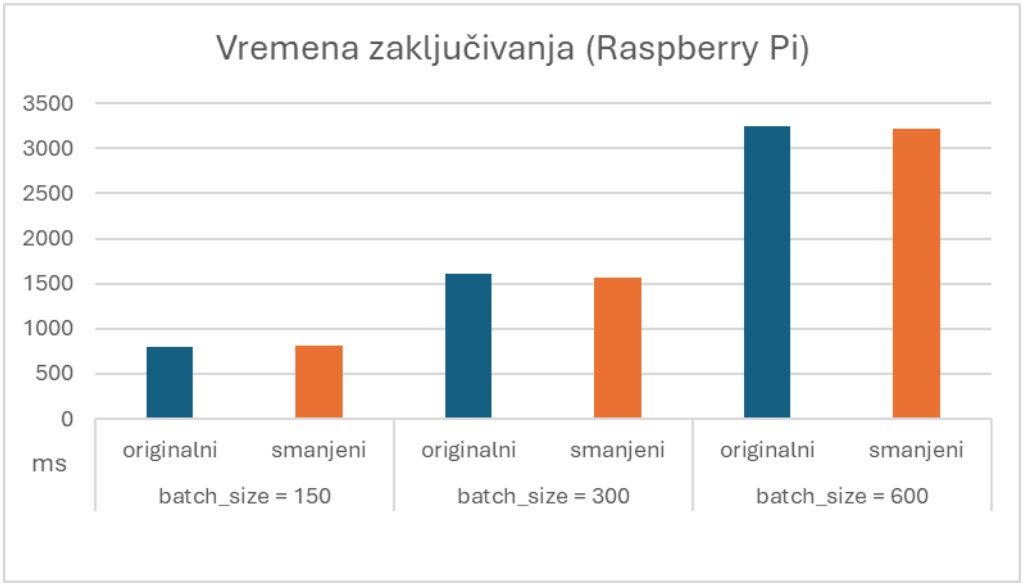
2.5. Drugi skup mjerenja - mjerenja s varijablom batch_size

Kako bi se utvrdio utjecaj varijable batch_size na vrijeme zaključivanja, ponovljena su mjerenja uz različite vrijednosti varijable batch_size. Također su provedena ista mjerenja

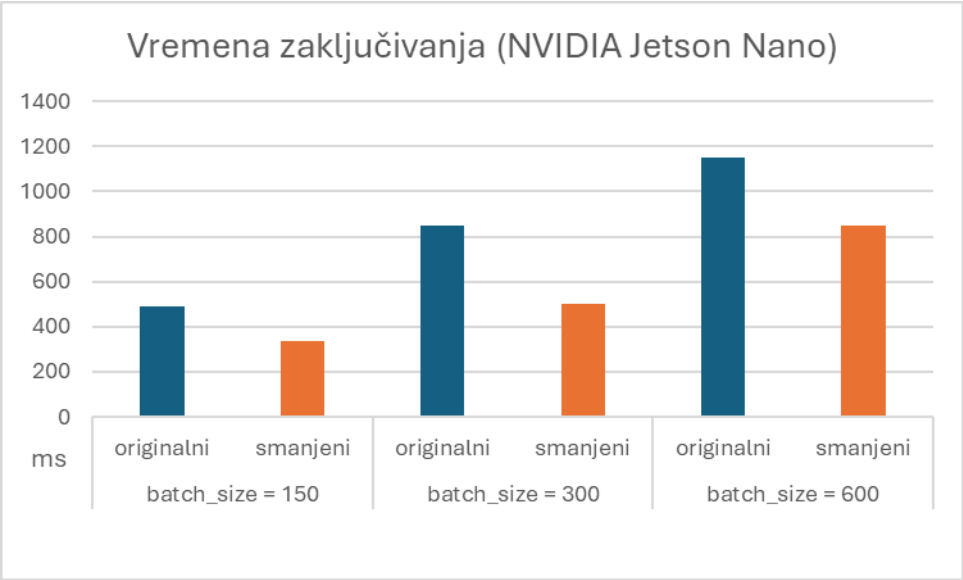
na računalu Raspberry Pi. Pokrenuta je skripta koja učitava uzorke fotografija veličine batch_size i koristi ih kao ulaz u neuronsku mrežu. Budući da Raspberry Pi nije opremljen GPU-om, zaključivanje se pokretalo na procesoru, dok je na ostalim uređajima korišten GPU. Ukupno je provedeno 18 mjerenja s 3 ulazna parametra:

- uređaj (Raspberry Pi, NVIDIA Jetson Nano, NVIDIA Orin AGX)
- model (originalni, smanjeni)
- batch_size (150, 300, 600)

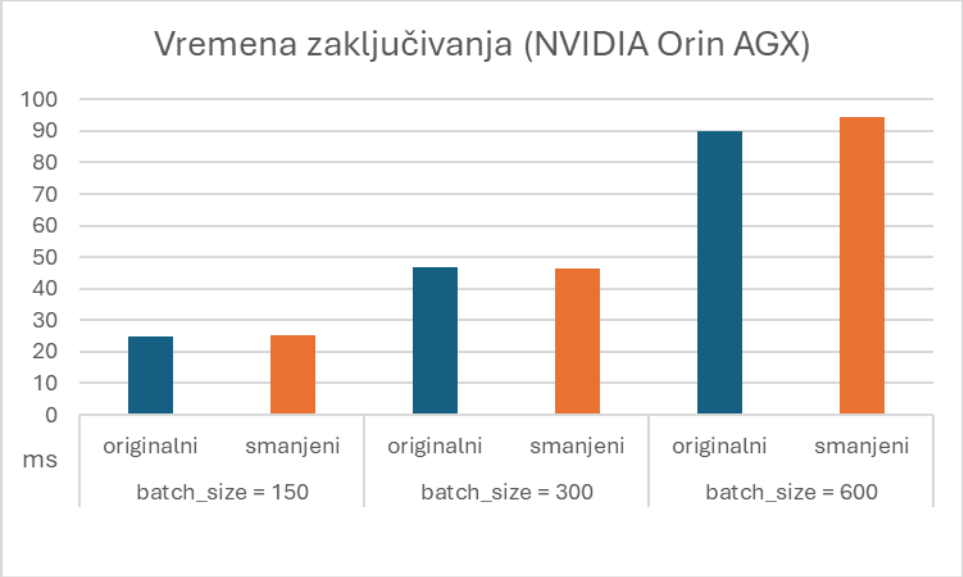
Za svaku kombinaciju parametara je izmjereno 50 vremena zaključivanja i njihov medijan. Dijagrami prikazuju medijane vrijednosti svakog mjerenja u milisekundama.



Slika 2.4 Medijani vremena zaključivanja za Raspberry Pi



Slika 2.5 Medijani vremena zaključivanja za Jetson Nano



Slika 2.6 Medijani vremena zaključivanja za Orin AGX

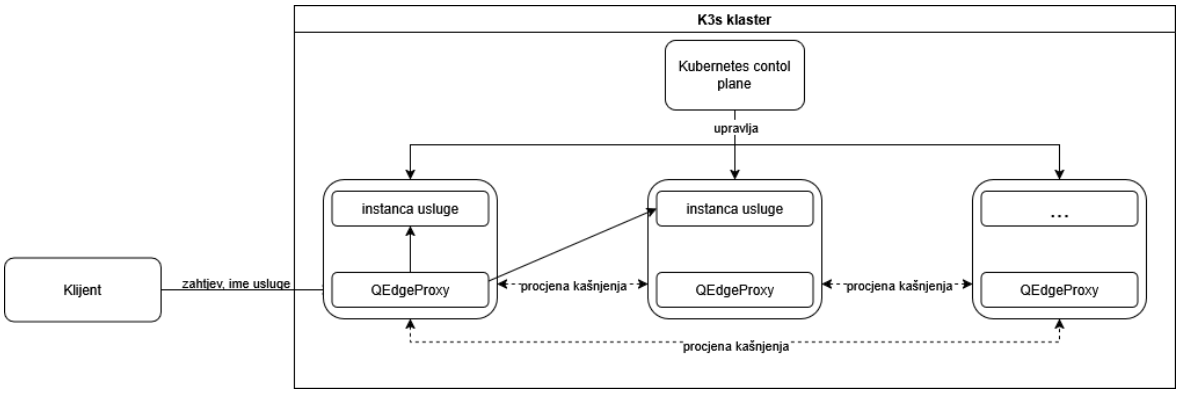
Rezultati mjerenja pokazuju da na uređajima Raspberry Pi i NVIDIA Jetson Orin kompresijom modela nisu dobivene bolje performance prilikom obrade skupine slika. Međutim, na uređaju NVIDIA Jetson Nano zapaženo je kraće vrijeme zaključivanja smanjenog modela u odnosu na originalni. Očekivano, s povećanjem vrijednosti batch_size se povećava vrijeme zaključivanja. Zbog snažnijih komponenti, uređaj Orin AGX ima značajno kraće vrijeme zaključivanja od uređaja Jetson Nano, a Jetson Nano ima značajno kraće vrijeme zaključivanja od uređaja Raspberry Pi.

3. Sustav za usluge zaključivanja dubokom neuronskom mrežom u okolini računarstva na rubu

U ovom poglavlju je opisano rješenje za adaptivno usmjeravanje u Kubernetesovom klasteru. Kako bi se prilagodilo radu s uslugama koje računanje obavljaju pomoću modela dubokih neuronskih mreža, implementirana je proširena inačica posrednika. Osim kašnjenja, prošireni posrednik uzima u obzir i točnost neuronske mreže, te nudi korisniku opciju konfiguracije željenih parametara vezanih uz točnost.

3.1. Arhitektura sustava

Slika 3.1**Pogreška! Izvor reference nije pronađen.** prikazuje primjer arhitekture sustava u Kubernetesovom klasteru. Prije pokretanja posrednika u grozdu, postavljaju se željene razine točnosti i kašnjenja instanci usluga na temelju koje se zahtjevi usmjeravaju prema njima. Na putu od ruba prema oblaku raste mrežno kašnjenje, ali se mijenjaju i dostupna sredstva čvorova za obradu. Prema paradigmi računarstva na rubu, uređaji opremljeni boljim hardverom se nalaze bliže izvoru podataka, tj. klijentu i dalje od oblaka. Osim mrežnog kašnjenja pojavljuje se i različito vrijeme zaključivanja neuronskih mreža na različitim čvorovima, kako je pokazano mjerenjima u drugom poglavlju.

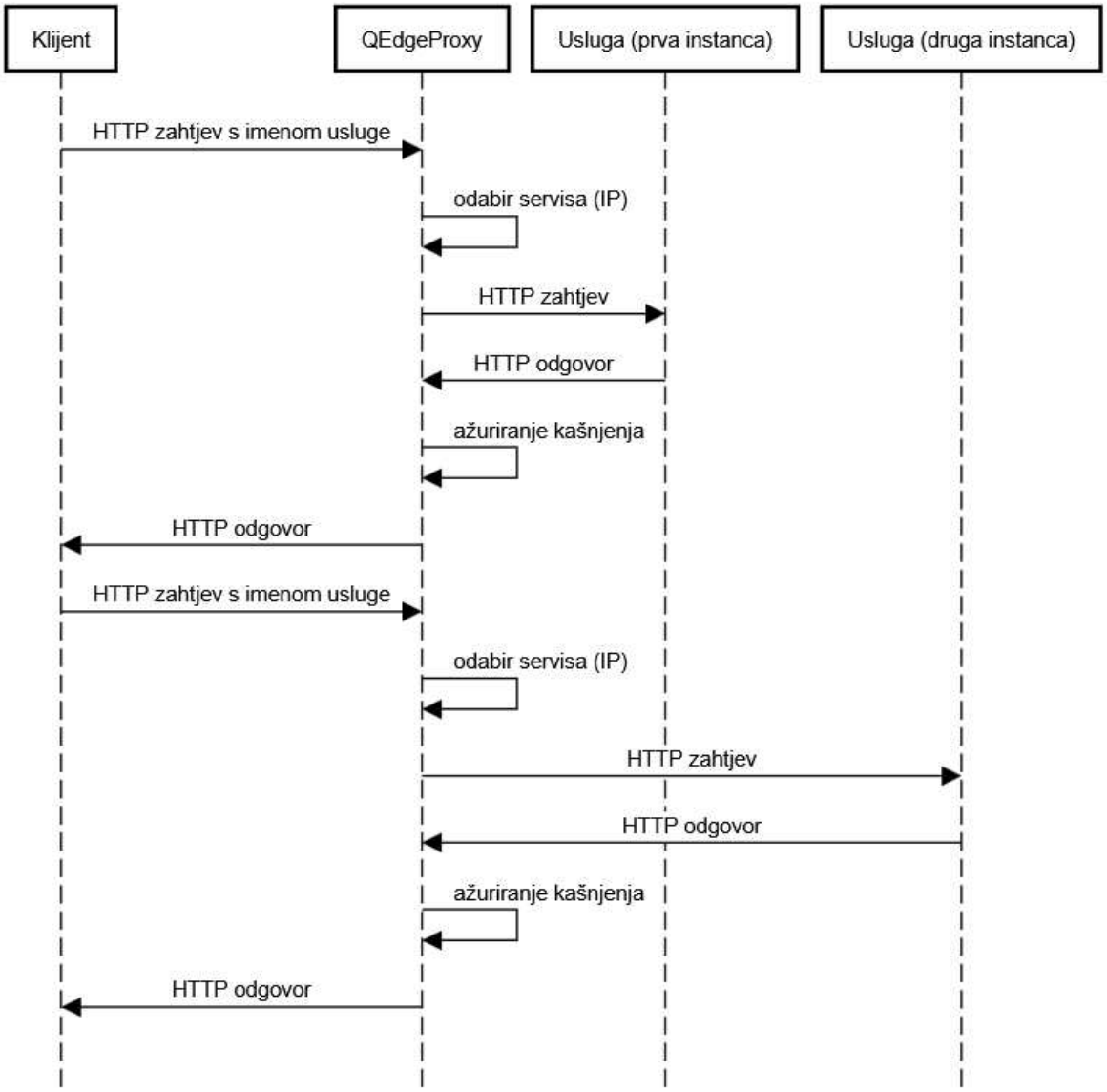


Slika 3.1 Skica arhitekture sustava za usmjeravanje zahtjeva prema servisima

3.2. Opis postojećeg rješenja za prilagodljivo usmjeravanje (QEdgeProxy)

QEdgeProxy je *load-balancer* i posrednički poslužitelj za okruženje Kubernetes i pogodno za postavljanje u računalnom kontinuumu razvijeno na Zavodu za telekomunikacije FER-a [12]. QEdgeProxy to rješava tako što djeluje kao balanser opterećenja. To pomaže u održavanju QoS-a unatoč kvarovima čvorova i degradaciji mreže. Usmjerava zahtjev na jednu od instanci usluge koja zadovoljava parametre kvalitete usluge koje je definirao korisnik. Napisan je u programskom jeziku Go i pokreće se u Docker kontejneru. Za svaku uslugu procjenjuje i pohranjuje kašnjenje kako bi se kasnije mogli ispuniti zahtjevi korisnika. Također može pratiti opterećenje čvorova kako bi odredio koji čvorovi su preopterećeni. Kriteriji koji se koriste za praćenje opterećenja su iskorištenost radne memorije i procesora. QEdgeProxy komponenta je pokrenuta na svakom čvoru računalnog kontinuuma i djeluje kao posrednik između klijenata i IoT usluga. Temeljna funkcija je dinamičko održavanje skupa instanci usluga, tzv. *QoS pool*, instanci usluga koje zadovoljavaju specifične QoS zahtjeve za određenu uslugu. Zahtjevi klijenata prosljeđuju se odgovarajućim instancama usluga unutar *QoS poola*. Sustav koristi dva glavna algoritma: algoritam za usmjeravanje zahtjeva koji odabire instance i pohranjuje QoS mjerenja te algoritam za praćenje stanja kontinuuma koji ažurira *QoS pool* [12].

QoS pool definiran je kao podskup instanci usluge koje će vjerojatno zadovoljiti minimalne QoS zahtjeve usluge. Sustav omogućuje prilagodljivo ispunjenje QoS-a, na temelju aproksimacija i njegovog ponašanja u prošlosti, npr. procijenjeno mrežno kašnjenje na temelju prošlih interakcija [12].



Slika 3.2 Primjer tijeka komunikacije putem posrednika QEdgeProxy

QEdgeProxy se sastoji od 3 komponente: K3s client, balancer i reverse HTTP proxy.

K3s client olakšava upravljanje Kubernetesovim resursima u K3s klasteru. Nudi funkcionalnost za učinkovito rukovanje informacijama o podovima i čvorovima, predmemoriranje podataka i održavanje statusa usluga. Klijent se spaja na Kubernetes klaster i postavlja potrebnih o uključuje stvaranje klijenata za pristup Kubernetes resursima i metrikama te konfiguriranje trajanja vrijednosti u *cacheu* za podove i čvorove.

K3s client koristi sinkronizirani *cache* za pohranjivanje podataka o podu koji sadrže usluge. Dohvaća informacije o podu iz *cachea*, ako je dostupna, ili upitom Kubernetes API-ju. Ako se podaci dohvaćaju iz API-ja, metoda također postavlja slušatelja za praćenje događaja povezanih s pod-om kao što su dodavanja, brisanja i ažuriranja.

Za praćenje ispravnosti i korištenja resursa čvorova klastera, klijent povremeno osvježava metriku čvora. Podaci o korištenju CPU-a i radne memorije dohvaćaju se i pohranjuju za svaki čvor s Kubernetesovog poslužitelja za metrike. To omogućuje praćenje performanci čvora u stvarnom vremenu.

Load-balancer služi za dinamičko uravnoteženje opterećenja za Kubernetes usluge, fokusirajući se na odabir najprikladnijeg modula za obradu zahtjeva na temelju kašnjenja, dostupnosti i opterećenja resursa. Dizajniran je da osigura učinkovito usmjeravanje zahtjeva uz održavanje performanci sustava i ispunjavanje zahtjeva za kvalitetu usluge (QoS).

Njegova primarna funkcija je dohvaćanje svih podova povezanih s određenom uslugom u Kubernetesovom namespaceu, filtriranje na temelju dostupnosti i približna procjena njihovog mrežnog kašnjenje. Implementacija daje prioritet podovima s nižim kašnjenjem. Dodatno, preopterećeni čvorovi identificiraju se na temelju metrike korištenja resursa, a njihovi podovi se privremeno ne koriste osim ako ne postoji druga mogućnost. Ako nema dostupnih i ne preopterećenih podova, funkcija koristi lokalni servis (ako postoji) ili nasumično odabire pod.

Nakon mjerenja mrežnog kašnjenja, stare vrijednosti se ažuriraju pomoću težina. Ako neki od pozadinskih sustava vrati pogrešku ili ne odgovori, pod je označen kao nedostupan, te stavljen u razdoblje mirovanja prije nego što se može ponovno upotrijebiti za usmjeravanje. Kada nedostaju podaci o kašnjenju za pod ili su zastarjeli, skripta ih izračunava ili približno procjenjuje slanjem HTTP zahtjeva poslužitelju. Rezultati se pohranjuju u *cache* radi poboljšanja učinkovitosti i izbjegavanja suvišnih zahtjeva. Skripta također potvrđuje da usluga zadovoljava minimalni QoS prag, koji je definiran kao omjer broja dostupnih podova i svih podova.

Reverse proxy dinamički usmjerava dolazne HTTP zahtjeve na pozadinske poslužitelje (izvorne usluge) u Kubernetesovom okruženju. Odgovoran je za određivanje odgovarajućeg pozadinskog poslužitelja za određenu uslugu. Program koristi *load balancer* za dinamičko otkrivanje i usmjeravanje prometa na pozadinske module. Informacije o izmjerenom kašnjenju ili kvaru šalju se *load balanceru* radi ažuriranja. Odgovor pozadinskog poslužitelja na kraju se prosljeđuje natrag klijentu.

Ovaj dizajn čini program robusnim rješenjem za rukovanje dinamičkim usmjeravanjem zahtjeva u Kubernetesu kako bi se osiguralo pouzdano i učinkovito upravljanje prometom.

Algoritam 3.1 Odabir instance usluge prema postojećem posredniku.

```
1: procedure choosePod(service)
2:     pods ← filterHealthyPods(service)
3:     for pod in pods
4:           if pod.latency < maxLatency
5:                 bestPods ← append(bestPodIPs, pod)
6:           end if
7:     end for
8:     sort(bestPods)
9:     return bestPods[0]
10: end procedure
```

U algoritmu 3.1 procedura *choosePod* odabire najbolju instancu dane usluge na temelju kašnjenja. Algoritam počinje filtriranjem nedostupnih podova. Tako osigurava da se uzmu u obzir samo operativne instance. Zatim iterira kroz listu dostupnih podova, provjeravajući je li kašnjenje svake grupe ispod unaprijed definiranog praga *maxLatency*. Ako pod ispunjava ovaj uvjet, dodaje se na popis potencijalnih kandidata, tj. *bestPods*. Nakon odabira najboljih podova, lista se sortira na temelju kašnjenja. Na kraju, algoritam odabire i vraća prvi pod u sortiranoj listi, za koji se smatra da će biti najbolji izbor za obradu zahtjeva.

3.3. Opis prilagođenog rješenja za prilagodljivo usmjeravanje

Osnovna funkcija posrednika proširena je tako da se jednu vrstu usluge može rasporediti, kontrolirati i koristiti više različitih poslužitelja grupiranih u različite razine kvalitete, odnosno točnosti. Ovo proširenje je osmišljeno radi usmjeravanja podataka do odgovarajuće instance usluge zaključivanja modelom duboke neuronske mreže. Kako bi se razlikovali različiti modeli i razine točnosti (npr. visoka i niska) upotrijebljene su oznake (*labels*) platforme Kubernetes. Klijent prilikom slanja zahtjeva ne šalje razinu točnosti, nego samo ime usluge. Posrednik odabire potrebnu razinu točnosti na temelju pragova koji su navedeni u konfiguraciji.

U svrhu raspoređivanja zahtjeva između različitih modela i razina preciznosti korišten je algoritam *Weighted Round Robin* (WRR) [18]. WRR je algoritam za uravnoteženje opterećenja koji proširuje osnovnu metodu *Round Robin* (RR) dodjeljivanjem različitih

težina poslužiteljima ili resursima na temelju njihovog kapaciteta ili važnosti. Umjesto ravnomjerne distribucije zahtjeva, WRR osigurava da poslužitelji većeg kapaciteta prime više zahtjeva od onih manjeg kapaciteta. Svakom poslužitelju (ili resursu) dodjeljuje se težina na temelju njegove procesorske snage, propusnosti ili drugih relevantnih metrika. Zahtjevi se distribuiraju kružno, ali poslužitelji s većim težinama primaju proporcionalno više zahtjeva. Algoritam kruži kroz popis poslužitelja, dodjeljuje broj zahtjeva jednak njihovoj težini prije prelaska na sljedeći poslužitelj.

Prilikom konfiguracije posrednika, za svaku razinu se postavlja željeni minimalni postotak zahtjeva koji se trebaju usmjeriti prema razini (npr. barem 50% zahtjeva je potrebno slati na model s višom točnošću). Sve dok postoje nezadovoljeni zahtjevi za korištenjem određene razine točnosti posrednik koristi WRR kako bi raspodijelio zahtjeve po razinama, a razine čiji je minimalni prag zadovoljen se zanemaruju dok nisu zadovoljeni ostali zahtjevi. U trenutku kada su svi takvi zahtjevi zadovoljeni, posrednik se vraća na uobičajeni način rada, to jest šalje zahtjev poslužitelju koji zadovoljava ostale kriterije i ima najmanje kašnjenje.

Algoritam 3.2 Odabir instance usluge prema prilagođenom posredniku.

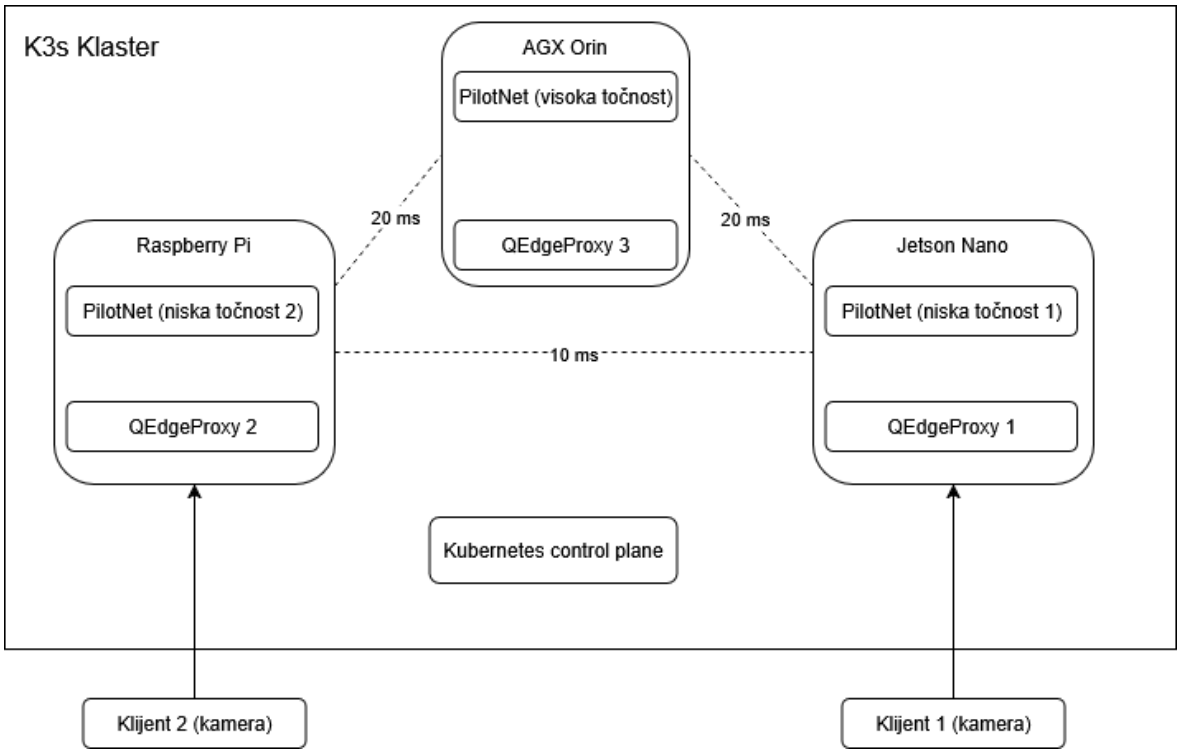
```
1: procedure choosePod(service)
2:     pods ← filterHealthyPods(service)
3:     for pod in pods
4:           if pod.latency < maxLatency
5:                 bestPods ← append(bestPodIPs, pod)
6:           end if
7:     end for
8:     sort(bestPods)
9:     level ← weightedRoundRobin(service)
10:    for pod in bestPodIPs
11:          if pod.accuracy = level
12:                return pod
13:          end if
14:    end for
15:    return bestPods[0]
16: end procedure
```

Algoritam 3.2 je proširenje algoritma 3.1. Nakon selekcije najboljih podova, određuje željenu razinu točnosti usluge pomoću metodom WRR. Zatim u sortiranoj listi najboljih podova traži pod najmanjeg kašnjenja koji pruža željenu razinu točnosti. Ako pod željene

razine točnosti nije pronađen, procedura vraća pod najmanjeg kašnjenja, kao kod algoritma 3.1.

4. Ispitivanje rada posrednika

Posrednik je ispitan unutar K3s klastera koji se sastoji od master poda i radnih podova koji sadrže PilotNet usluge za izračun predikcije. Na uređajima Raspberry Pi i Jetson Nano su pokrenute kvantizirane inačice modela duboke neuronske mreže PilotNet, dok je na uređaju AGX Orin pokrenuta modificirana inačica bez kvantizacije. Cilj ovog eksperimenta je bio usporediti izmjerena vremena kašnjenja s postavljenim pragom za kašnjenje te utvrditi opterećenje procesora čvorova prilikom rada posrednika i poslužitelja za zaključivanje. Također, promatrana je potrošnja energije uređaja Orin AGX. Za ispitivanje istodobno su korištena dva HTTP klijenta. Slika 4.1 prikazuje okolinu za provedbu ovih eksperimenata. Klijenti su pokrenuti na master čvoru klastera i konfigurirani su da asinkrono šalju određen broj zahtjeva s fotografijama ceste, čime se simulira rad kamere u automobilu. Slike ceste su izdvojene iz videozapisa vožnje automobila. Tijekom eksperimenta interval je postavljen tako da se generira 60 zahtjeva sa slikom u sekundi. Prilikom eksperimenta klijenti su slali zahtjeve posrednicima na uređajima Jetson Nano i Raspberry Pi. Posrednici su zatim prosljeđivali zahtjeve na jedan od tri PilotNet poslužitelja (Slika 4.1). Kada posrednici šalju poruku, mrežnom kašnjenju i vremenu zaključivanja dodaje se simulirano kašnjenje, ovisno o čvoru. Za čvor AGX Orin je simulirano kašnjenje postavljeno na 20 ms, a za ostale čvorove 10 ms. Provedeno je ukupno tri eksperimenta s različitim QoS parametrima. Eksperimenti su označeni slovima od A do C prema strogosti QoS parametara, od najpopustljivijeg do najstrožeg (Tablica 2).



Slika 4.1 Okolina sa simuliranim kašnjenjima korištena prilikom ispitivanja posrednika

QoS parametar	Eksperiment A	Eksperiment B	Eksperiment C
maxLatency	230 ms	215 ms	200 ms
levelRequirements (formata [visoka točnost, niska točnost])	[20, 0]	[50, 0]	[80, 0]
coolDownBaseDurationS	10 s	10 s	10 s

Tablica 1 Odabrani QoS parametri

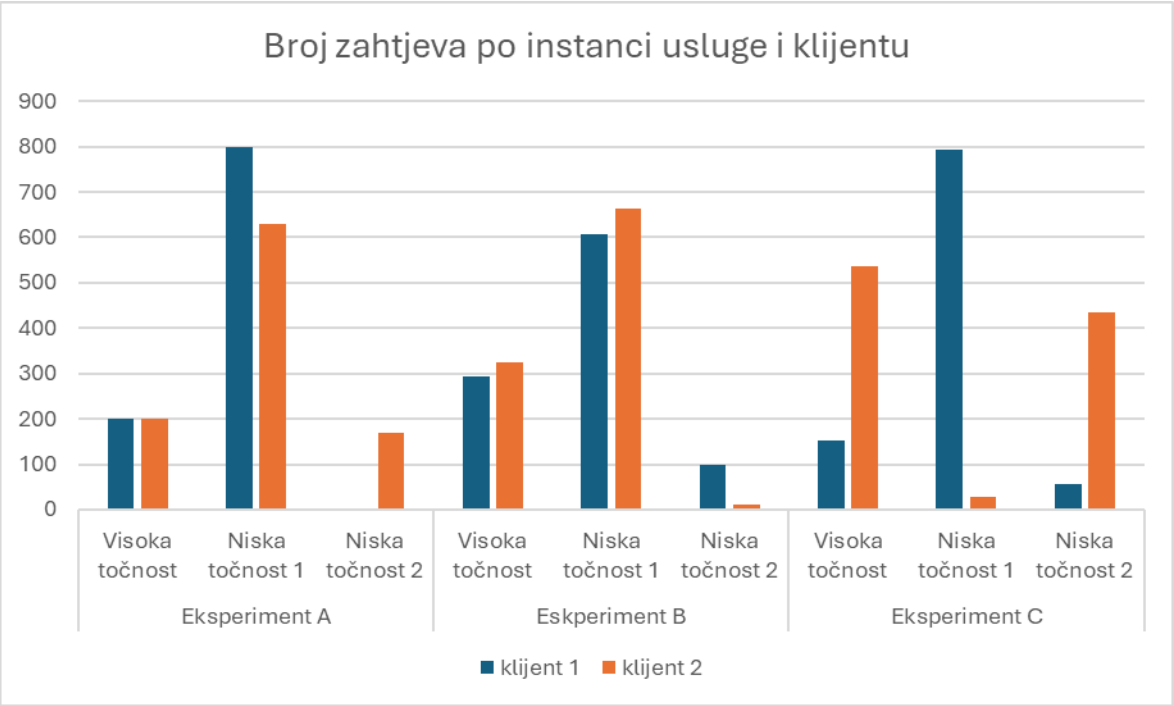
Tablica 1 prikazuje QoS parametre koji su korišteni u ovim eksperimentima. Parametar *maxLatency* je gornja dozvoljena granica za kašnjenje. Parametrom *levelRequirements* od posrednika se traži da određeni minimalni postotak zahtjeva usmjerava prema uslugama visoke točnosti. Za usluge visoke točnosti odabrane su različite vrijednosti, a za usluge niske točnosti ne postoji zahtjev. Parametar *coolDownBaseDurationS* definira period mirovanja za instance izbačene iz *QoS poola*. Za potrebe ovih eksperimenata, treba biti manji od trajanja eksperimenta kako bi se izbačeni podovi mogli vratiti natrag. Važno je napomenuti da *maxLatency* ima prioritet nad parametrom *levelRequirements*. Ako

primjerice usluga više točnosti nije dostupna zbog visokog kašnjenja, onda se *levelRequirements* privremeno zanemaruje. Tablica 2 i **Pogreška! Izvor reference nije pronađen.** prikazuju rezultate rada posrednika, odnosno broj poruka s obzirom na eksperiment i odredište usmjeravanja. Poslano je 1000 zahtjeva po klijentu. Za svaki eksperiment izračunat je postotak zahtjeva ispod granice *maxLatency* (Tablica 2). Instanca niske točnosti 1 primila je najviše zahtjeva jer je Jetson Nano opremljen snažnim hardverom, što uz jednostavniji model osigurava niska ukupna kašnjenja. U eksperimentima A i B, instanca niske točnosti 2 primila je malo zahtjeva jer je Jetson Nano snažniji od uređaja Raspberry Pi, a mrežno kašnjenje u ovom slučaju zanemarivo.

	Klijent	Visoka točnost	Niska točnost 1	Niska točnost 2	Postotak zahtjeva koji su zadovoljili granicu kašnjenja
Eksperiment A	Klijent 1	200	798	2	99.5 %
	Klijent 2	200	630	170	
Eksperiment B	Klijent 1	293	608	99	96.6 %
	Klijent 2	326	664	10	
Eksperiment C	Klijent 1	151	794	55	95.15 %
	Klijent 2	536	29	435	

Tablica 2 Brojevi zahtjeva koje su posrednici usmjerili prema instancama usluge

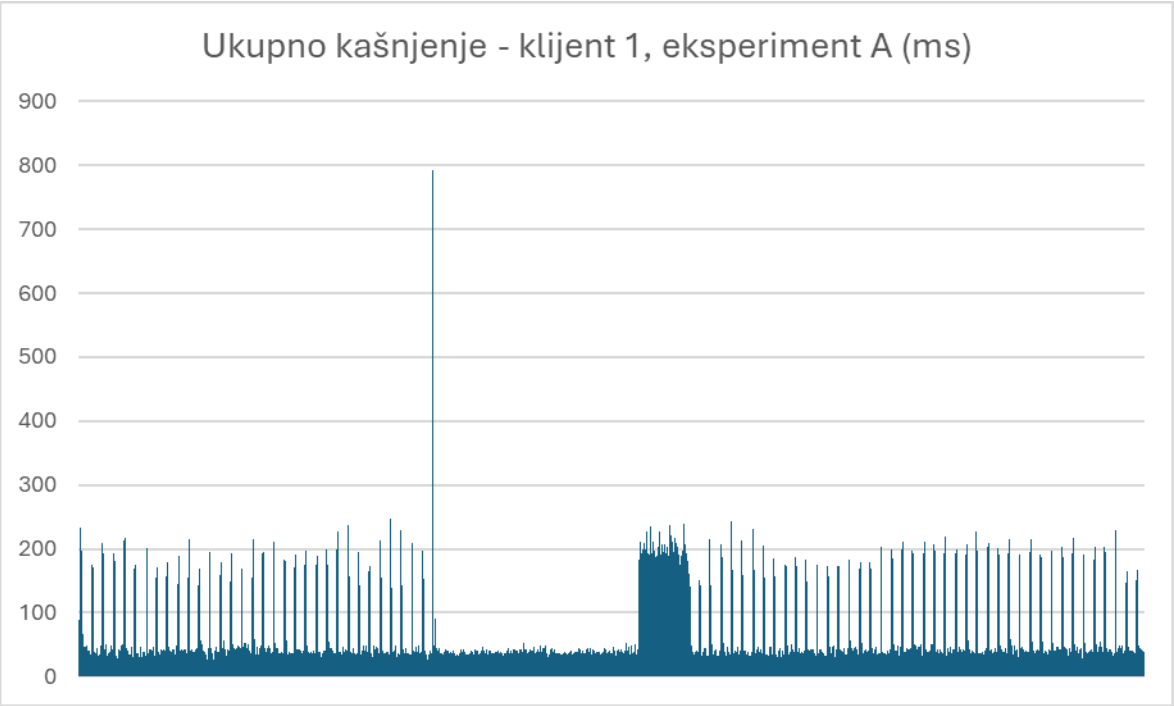
U eksperimentu A zadovoljeno je pravilo za slanjem 20 % zahtjeva prema instanci više točnosti, kao što je vidljivo u Tablica 1. Međutim, to pravilo nije ispunjeno na kraju eksperimenta B i C budući da je odabrana gornja granica kašnjenja nekoliko puta prouzročila izbacivanje instance visoke točnosti iz *QoS poola*.



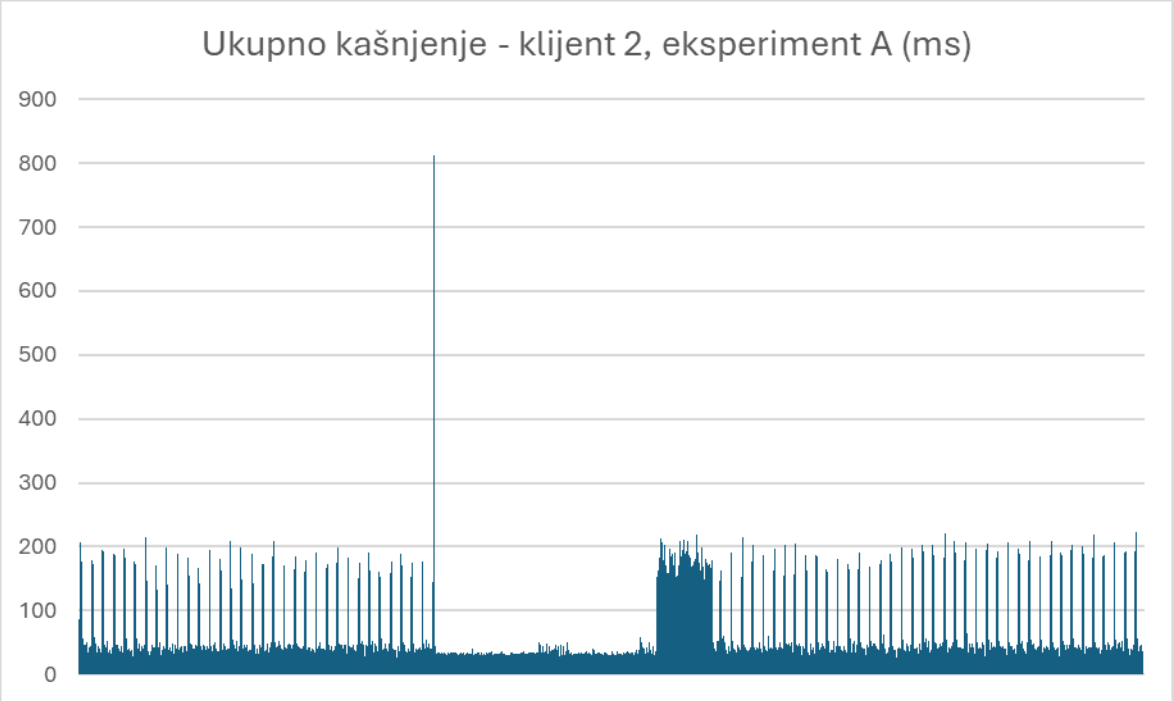
Slika 4.2 Brojevi zahtjeva po instanci usluge i klijentu

Tijekom eksperimenta C većina zahtjeva koji su prošli kroz posrednike su završili na lokalnoj usluzi ili na instanci visoke točnosti (**Pogreška! Izvor reference nije pronađen.**). Iako su instance niske točnosti tijekom cijelog trajanja eksperimenta ostali u *QoS poolu*, zbog preopterećenja lokalnog čvora povremeno je došlo do slanja zahtjeva na susjedni čvor s istim modelom. Do ove pojave dolazi jer je vrijeme zaključivanja lokalne usluge nadmašilo mrežno kašnjenje, stoga je u tom trenutku bilo učinkovitije slati zahtjev na susjednu uslugu.

QEdgeProxy može izračunati kašnjenje na dva načina: procjenom (slanje *ping* poruke drugom posredniku na određenom čvoru) ili mjerenjem (nakon primitka odgovora od usluge na određenom čvoru). Kod usluga koje izvršavaju složene i dugotrajne izračune, kao što je instanca visoke točnosti, između ove dvije vrste kašnjenja pojavljuje se velika razlika jer je vrijeme zaključivanja nekoliko puta veće od mrežnog kašnjenja. Kada čvor ispadne iz *QoS poola*, nakon perioda mirovanja se procjenjuje njegovo kašnjenje. To omogućuje spuštanje kašnjenja ispod granice i vraćanje čvora u *QoS pool* iz stanja mirovanja.



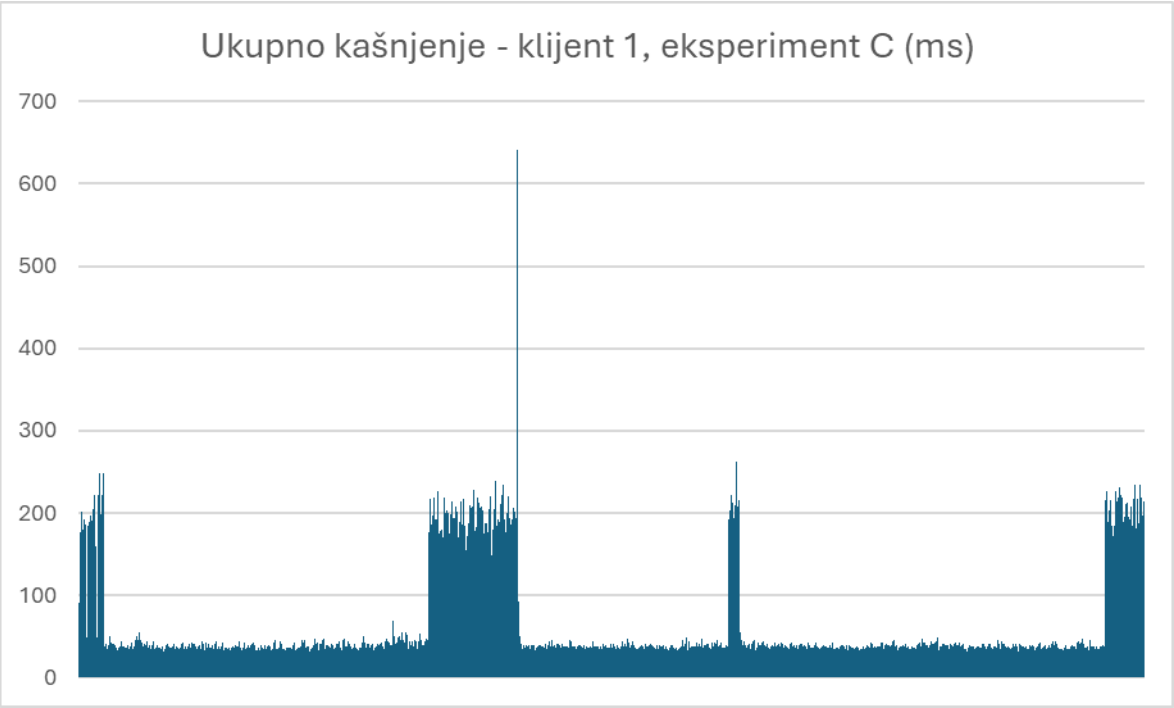
Slika 4.3 Kašnjenja izmjerena na klijentu 1 tijekom eksperimenta A



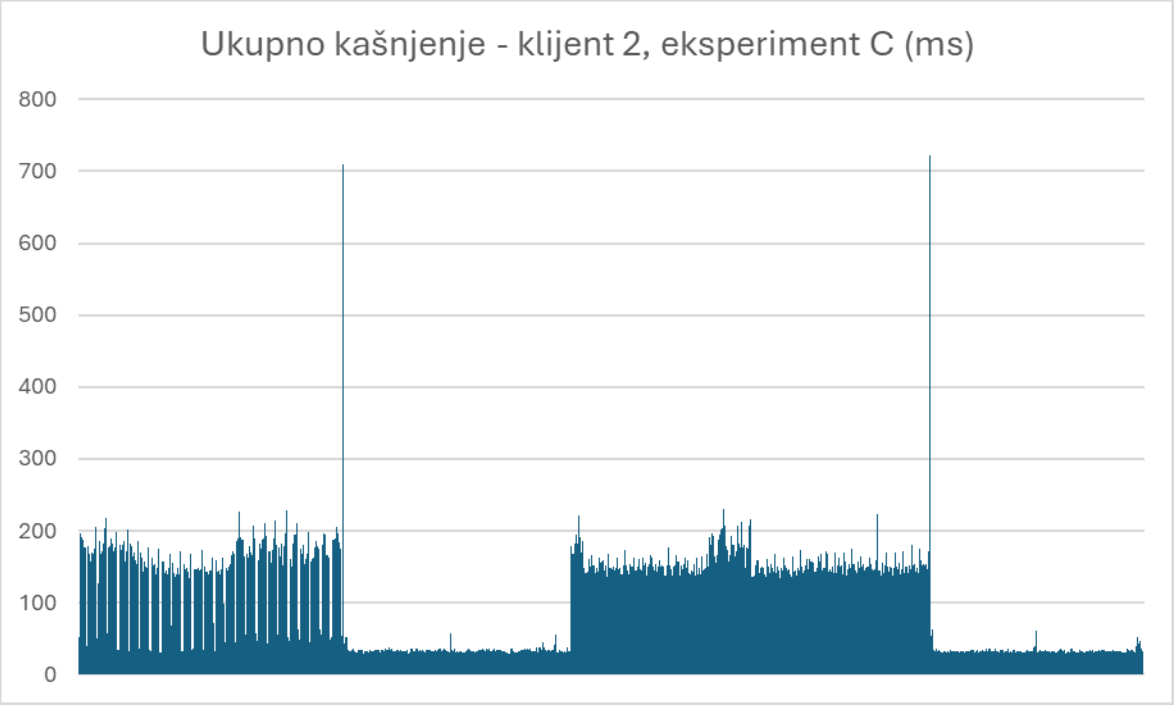
Slika 4.4 Kašnjenja izmjerena na klijentu 2 tijekom eksperimenta A

SlikeSlika 4.3Slika 4.4 prikazuju ukupna vremena kašnjenja poruka koje su klijenti slali tijekom eksperimenta A. Ukupno kašnjenje je zbroj mrežnog kašnjenja, simuliranog kašnjenja i vremena zaključivanja. Na slikama je jasno vidljiva faza mirovanja (*cooldown*)

instance visoke točnosti, kao i njen povratak u *QoS pool*. Iako je instanca visoke točnosti ispala iz *QoS poola*, zadovoljeno je pravilo da joj treba biti poslano 20% zahtjeva.



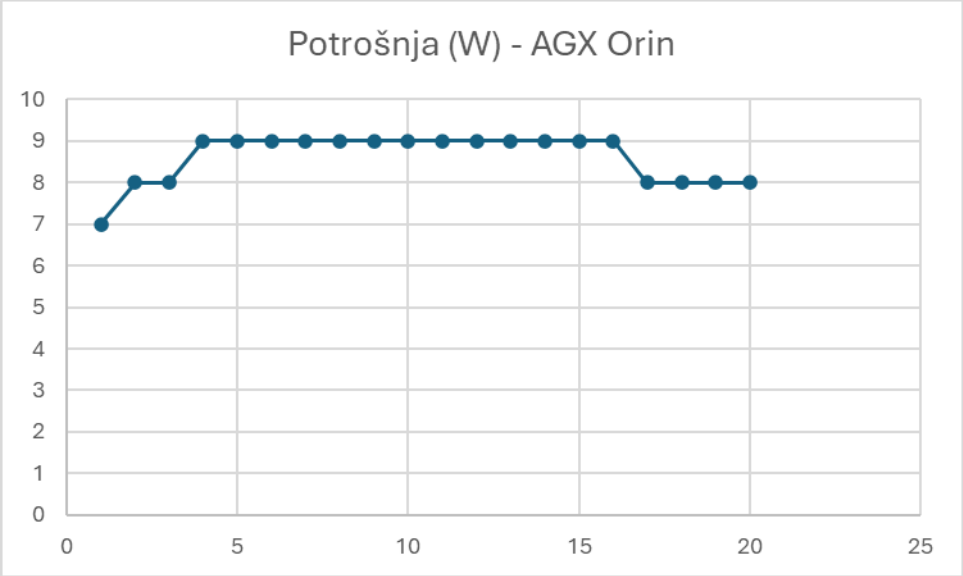
Slika 4.5 Kašnjenja izmjerena na klijentu 1 tijekom eksperimenta C



Slika 4.6 Kašnjenja izmjerena na klijentu 2 tijekom eksperimenta C

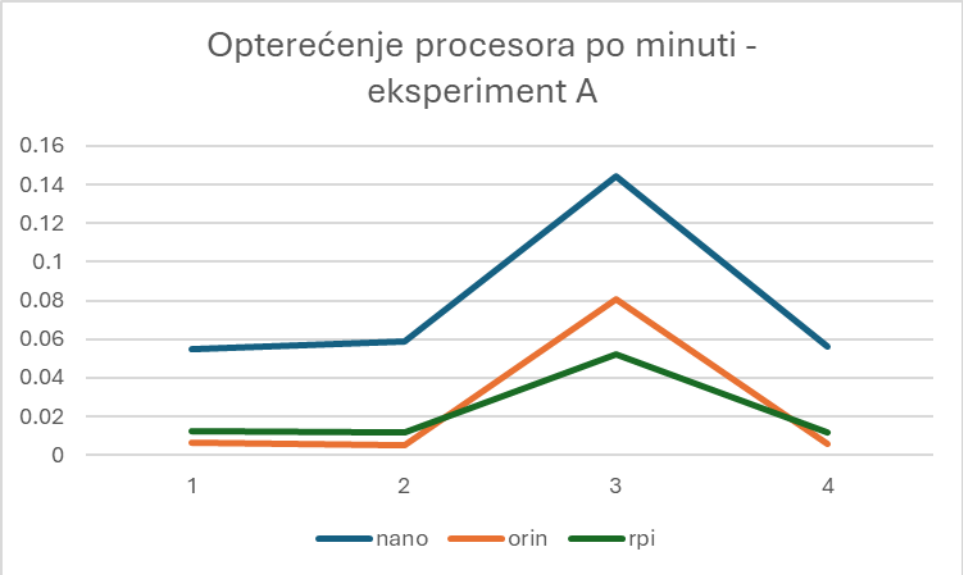
Slike 4.5 i 4.6 prikazuju ukupna vremena kašnjenja poruka koje su klijenti slali tijekom eksperimenta C. Zbog povećanog prometa i opterećenja, instanca visoke točnosti ispala je

iz *QoS poola* nekoliko puta. Također, prilikom ažuriranja kašnjenja čvora QEdgeProxy kombinira staru i novu vrijednost koristeći težine. Na dijagramima je vidljivo da to uzrokuje određenu razinu tolerancije na prelaženje gornje granice kašnjenja (*max latency*).

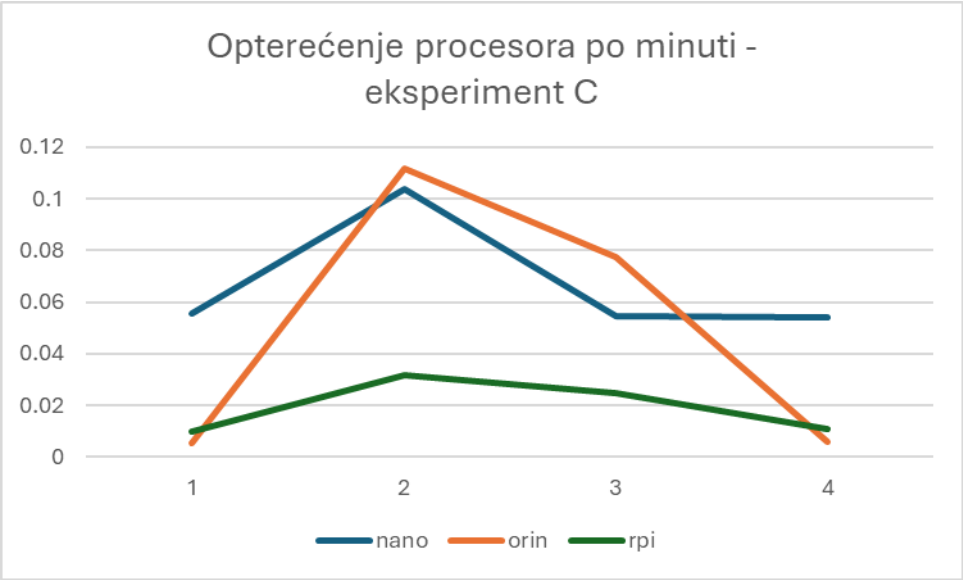


Slika 4.7 Potrošnja energije uređaja AGX Orin tijekom izvođenja eksperimenta C

Tijekom eksperimenta C svakih 10 sekundi zabilježena je potrošnja uređaja AGX Orin. Slično kao i kod testa opisanog u drugom poglavlju, potrošnja uređaja je porasla 9 W (Slika 4.7). Na Slika 4.8 i Slika 4.9 vidljiv je porast opterećenja procesora za vrijeme trajanja ispitivanja. Na čvorovima koji primaju više poruka u minuti izmjereno je veće opterećenje.



Slika 4.8 Opterećenje procesora tijekom eksperimenta A



Slika 4.9 Opterećenje procesora tijekom eksperimenta C

Ispitivanje demonstrira da je pravilan izbor QoS parametara naročito važan za postizanje optimalne ravnoteže između kašnjenja i točnosti u ovoj okolini. Glavni problem otkriven pomoću ovoga ispitivanja je da prilikom istodobnog slanja zahtjeva, posrednici uskraćuju jedni drugima pristup usluzi s visokom točnošću. Zbog manjeg mrežnog kašnjenja, posrednik čvora Raspberry Pi je poslao mnogo više zahtjeva prema čvoru AGX Orin. Posljedica toga je da ostali čvorovi generalno dobivaju predikcije manje točnosti. Budući rad mogao bi proučiti kako bi se ovaj učinak mogao ublažiti, npr. davanjem prioriteta čvorovima na temelju podataka o točnosti povijesnih zahtjeva u privremenoj memoriji. Također, usporedba rada posrednika QEdgeProxy s jednostavnijim algoritmima (npr. *Round robin* ili *IP hash*) usmjeravanja pružila bi dodatan uvid o učinkovitosti ovakve okoline.

Zaključak

Računarstvo na rubu obradom podataka bliže izvoru nudi prednost manjeg kašnjenja u usporedbi s tradicionalnim računarstvom u oblaku. Međutim, računarstvo na rubu također uključuje izazove povezane s ograničenim računalnim resursima, heterogenim hardverom i učinkovitom raspodjelom radnog opterećenja. Kompromis točnosti i kašnjenja u strojnom učenju i računarstvu na rubu predstavlja značajan izazov, osobito u decentraliziranim i heterogenim okruženjima. Učinkovita raspodjela radnog opterećenja ključna je za optimizaciju kvalitete iskustva (QoE) i kvalitete usluge (QoS). Ovaj rad pokazuje da primjena više inačica istog modela dobivenih kompresijom osnovnog modela omogućuje prilagodljivu dodjelu zadataka, osiguravajući da su brzina i točnost zaključivanja usklađeni s ograničenjima sustava. Nadalje, uključivanje posrednika koji prikuplja informacije o stanju mreže i poslužiteljima u stvarnom vremenu poboljšava donošenje odluka, što dovodi do optimalnog korištenja resursa i poboljšanih performanci.

Dok ovaj pristup uspješno uravnotežuje točnost i kašnjenje, potrebna su daljnja istraživanja kako bi se poboljšali mehanizmi dinamičke prilagodbe. Budući rad mogao bi istražiti strategije raspoređivanja zasnovane na potpomognutom učenju kako bi se poboljšala raspodjela zadataka. Dodatno, integracija naprednijih tehnika kompresije modela, kao što je destilacija znanja, mogla bi dodatno povećati učinkovitost. Uključivanje alternativnih mjera performanci, kao što je opterećenje GPU-a, moglo bi pružiti sveobuhvatniji pogled na performance sustava i dodatno poboljšati strategije optimizacije.

Literatura

[1] O'Neill, James. *An Overview of Neural Network Compression*. ArXiv abs/2006.03669 (2020)

[2] Gholami, Amir, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney i Kurt Keutzer. *A Survey of Quantization Methods for Efficient Neural Network Inference*. ArXiv abs/2103.13630 (2021)

[3] Saini, Kavita, Uttama Pandey i Pethuru Raj. *Chapter Nine - Edge computing challenges and concerns.* Adv. Comput. 127 (2022), str. 259-278.

[4] Li, En, Liekang Zeng, Zhi Zhou i Xu Chen. *Edge AI: On-Demand Accelerating Deep Neural Network Inference via Edge Computing.* IEEE Transactions on Wireless Communications 19 (2019), str. 447-457.

[5] Merenda, Massimo, Carlo Porcaro i Demetrio Iero. *Edge Machine Learning for AI-Enabled IoT Devices: A Review*. Sensors (Basel, Switzerland) 20 (2020)

[6] Ning, Zhaolong, Peiran Dong, Xiaojie Wang, Joel J.P.C. Rodrigues i Feng Xia. *Deep Reinforcement Learning for Vehicular Edge Computing*. ACM Transactions on Intelligent Systems and Technology (TIST) 10 (2019), str. 1-24.

[7] Jukić, Ana Petra, Ana Selek, Marija Seder i Ivana Podnar Žarko. *Autonomous Driving with a Deep Dual-Model Solution for Steering and Braking Control*. 2024 9th International Conference on Smart and Sustainable Technologies (SpliTech) (2024), str. 1-6.

[8] Bojarski, Mariusz, David W. del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao i Karol Zieba. *End to End Learning for Self-Driving Cars*. ArXiv abs/1604.07316 (2016)

[9] Mocnej, Jozef, Martin Miskuf, Peter Papcun i Iveta Zolotová. *Impact of Edge Computing Paradigm on Energy Consumption in IoT.* IFAC-PapersOnLine 51 (2018), str. 162-167.

[10] Ahvar, Ehsan, Anne-Cécile Orgerie i Adrien Lèbre. *Estimating Energy Consumption of Cloud, Fog, and Edge Computing Infrastructures.* IEEE Transactions on Sustainable Computing 7 (2019), str. 277-288.

[11] Varghese, Blesson, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick i Dimitrios S. Nikolopoulos. *Challenges and Opportunities in Edge Computing*. 2016 IEEE International Conference on Smart Cloud (SmartCloud) (2016): str. 20-26.

[12] Čilić, Ivan, Valentin Jukanović, Ivana Podnar Žarko, Pantelis A. Frangoudis i Schahram Dustdar. *QEdgeProxy: QoS-Aware Load Balancing for IoT Services in the Computing Continuum.* 2024 IEEE International Conference on Edge Computing and Communications (EDGE) (2024), str. 67-73.

[13] Nikodem, Maciej, Mariusz Słabicki, Tomasz R. Surmacz, Pawel Mrówka i Cezary Dolega. *Multi-Camera Vehicle Tracking Using Edge Computing and Low-Power Communication*. Sensors (Basel, Switzerland) 20 (2020)

[14] Barthélemy, Johan, Nicolas Verstaevel, Hugh I. Forehead i Pascal Perez. *Edge-Computing Video Analytics for Real-Time Traffic Monitoring in a Smart City.* Sensors (Basel, Switzerland) 19 (2019)

[15] Wan, Shaohua, Songtao Ding i Chen Chen. *Edge computing enabled video segmentation for real-time traffic monitoring in internet of vehicles.* Pattern Recognit. 121 (2021): 108146.

[16] Liu, Heting i Guohong Cao. *Deep Learning Video Analytics Through Online Learning Based Edge Computing.* IEEE Transactions on Wireless Communications 21 (2022): 8193-8204.

[17] Liu, Shaoshan, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang i Weisong Shi. *Edge Computing for Autonomous Driving: Opportunities and Challenges*. Proceedings of the IEEE 107 (2019): 1697-1716.

[18] Baker F. i Pan R. *On Queuing, Marking, and Dropping*. Internet Engineering Task Force (IETF). 2016. https://www.rfc-editor.org/rfc/rfc7806.html; pristupljeno 11.2.2025.

[19] The Kubernetes Authors. *Kubernetes Documentation.* 2025. https://kubernetes.io/docs/home/; pristupljeno 11.2.2025.

[20] K3s Project Authors. *K3s - Lightweight Kubernetes* 2025. https://docs.k3s.io/; pristupljeno 11.2.2025.

Sažetak

Kvaliteta usluge zaključivanja dubokom neuronskom mrežom u okolini računarstva na rubu

U ovom radu je oblikovano rješenje za usmjeravanje podataka s IoT-uređaja do odgovarajuće usluge zaključivanja modelom duboke neuronske mreže u okolini računarstva na rubu. Za uslugu zaključivanja iskorištena je neuronska mreža PilotNet i komprimirane inačice mreže PilotNet. Performance dubokih neuronskih mreža ispitane su na različitim uređajima s ograničenim sredstvima. Prošireno je postojeće rješenje posrednika tako da uz postojeći cilj smanjenja mrežnog kašnjenja posrednik uzima u obzir točnost zaključivanja. Rad posrednika ispitan je u emuliranom okruženju računarstva na rubu gdje na putu od ruba prema oblaku raste mrežno kašnjenje, ali se mijenjaju i dostupna sredstva obradnih čvorova.

Ključne riječi: računarstvo na rubu, usmjeravanje, duboke neuronske mreže, kvaliteta usluge

Summary

Quality of Service for Deep Neural Network Inference in an Edge Computing Environment

In this thesis, a solution is designed for routing data from IoT devices to the appropriate inference service using a deep neural network model in an edge computing environment. The PilotNet neural network and its compressed versions were used as inference services. The performance of deep neural networks was evaluated on various devices with limited resources. The existing routing solution has been enhanced so that, in addition to minimizing network latency, it also considers inference accuracy. This router was tested in an emulated edge computing environment, where network latency increases from edge to cloud, while the available resources of processing nodes vary.

Keywords: edge computing, routing, deep neural networks, quality of service

Privitak.

Skripte za mjerenje vremena zaključivanja

Za mjerenje vremena zaključivanja pokrenute su sljedeće skripte napisane u programskom jeziku python. Prilikom pokretanja potrebno je putem argumenata komandne linije zadati uređaj (“cpu” ili “gpu”) i batch_size. Također je potrebno instalirati pakete numpy, keras i tensorflow. Za svako mjerenje se ispisuje vrijeme u sekundama na standardnom izlazu.

Skripta za pokretanje mjerenja na originalnom modelu:

```
import tensorflow as tf
import numpy as np
import time, sys
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense, Lambda

#load data
test_images = np.load('test_images.npy', allow_pickle=True)
test_labels = np.load('test_labels.npy', allow_pickle=True)

#load original pilotnet
input_shape = (160, 120, 1)  # (height, width, channels)

model = Sequential([
    # Normalize pixel values to the range [-1, 1]
    Lambda(lambda x: x / 127.5 - 1., input_shape=input_shape),

    Conv2D(24, (5, 5), strides=(2, 2), activation='relu'),
    Conv2D(36, (5, 5), strides=(2, 2), activation='relu'),
    Conv2D(48, (5, 5), strides=(2, 2), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),
    Conv2D(64, (3, 3), activation='relu'),

    Flatten(),

    Dense(100, activation='relu'),
    Dense(50, activation='relu'),
```

```
    Dense(10, activation='relu'),
    Dense(1)
])

model.load_weights('pilotnet_original.h5')

#inference measuring
batch_size = int(sys.argv(2))

with tf.device('/' + sys.argv[1] + ':0'):
    for i in range(50):
        start_time = time.time()
        model.predict(test_images[:batch_size])#,
test_labels[:batch_size])
        end_time = time.time()

        print(f"{end_time-start_time}")
```

Skripta za pokretanje mjerenja na smanjenom modelu:

```
import tensorflow as tf
import numpy as np
import time, sys
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense, Lambda

#load data
test_images = np.load('test_images.npy', allow_pickle=True)
test_labels = np.load('test_labels.npy', allow_pickle=True)

#load modified pilotnet
model = tf.keras.models.load_model('pilotnet_modified.h5')

#inference measuring
batch_size = int(sys.argv(2))

with tf.device('/' + sys.argv[1] + ':0'):
    for i in range(50):
        start_time = time.time()
        model.predict(test_images[:batch_size])#,
test_labels[:batch_size])
```

```
end_time = time.time()

print(f"{end_time-start_time}")
```