Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Automatyki i Informatyki Stosowanej

Praca dyplomowa magisterska

na kierunku Informatyka
w specjalności Systemy Internetowe Wspomagania Zarządzania

Uczenie federacyjne sztucznych sieci neuronowych w chmurze obliczeniowej

mgr inż. Bartosz Bok
Numer albumu 285709

inż. Przemysław Mazur
Numer albumu 289719

promotor
dr hab. inż. Mariusz Kamola

WARSZAWA 2024

Uczenie federacyjne sztucznych sieci neuronowych w chmurze obliczeniowej

Streszczenie.

Praca magisterska dotyczy analizy i oceny metod uczenia federacyjnego, z uwzględnieniem różnych technik, scenariuszy i konfiguracji sprzętowych. Przeprowadzono przegląd literatury dotyczącej metod uczenia federacyjnego oraz opisano techniki stosowane w tej dziedzinie. Celem pracy jest eksperymentalne sprawdzenie wpływu wybranych czynników na metryki uczenia federacyjnego, zarówno pod kątem modeli uczenia maszynowego, jak i wydajności różnych konfiguracji sprzętowych i sieciowych, przedstawienie zalet i wyzwań związanych z uczeniem federacyjnym oraz potencjalnych obszarów jego zastosowań.

W ramach badań przeprowadzono eksperymenty dla różnych konfiguracji uczenia synchronicznego i asynchronicznego. Uwzględniono scenariusze takie jak różna liczba modeli lokalnych, zmienne wartości współczynnika uczenia, grupowanie etykiet oraz nierównomierne rozłożenie etykiet z różnymi ważnościami modeli. Analizowano także różne konfiguracje sprzętowe, w tym liczbę wirtualnych rdzeni (vCPU), typy dysków, opóźnienia w komunikacji i awarie klastrów. Implementacja uczenia federacyjnego została przeprowadzona na rzeczywistych serwerach w chmurze obliczeniowej, z oddzielnymi klastrami dla każdej instytucji, co umożliwiło realistyczne odwzorowanie środowiska produkcyjnego. W badaniach zastosowano nowoczesne narzędzia, takie jak PyTorch do implementacji modeli, Kubernetes do zarządzania klastrami, Docker do konteneryzacji aplikacji oraz Linkerd, Minio i Argo Workflows do zarządzania komunikacją i przepływami pracy.

Na podstawie przeprowadzonych badań wyciągnięto wnioski na temat efektywności różnych podejść do uczenia federacyjnego oraz oszacowano koszty związane z poprawą wartości metryk modeli. Ponadto, opisano algorytmy i metody predykcji zasobów w ramach automatycznego skalowania klastrów, co pozwala na optymalizację kosztów operacyjnych. Na koniec zaproponowano kierunki dalszych badań, które obejmują rozwój bardziej zaawansowanych metod komunikacji, czy optymalizację procesów związanych z uczeniem federacyjnym.

Słowa kluczowe: uczenie federacyjne, agregacja modeli, chmura obliczeniowa, konteneryzacja.

Federated Learning of Artificial Neural Networks in Cloud Computing

Abstract.

The master's thesis concerns the analysis and evaluation of federated learning methods, taking into account various techniques, scenarios and hardware configurations. A literature review on federated learning methods was conducted and techniques used in this field were described. The aim of the work is to experimentally verify the impact of selected factors on federated learning metrics, both in terms of machine learning models and the performance of various hardware and network configurations, to present the advantages and challenges of federated learning and potential areas of its application. As part of the research, experiments were carried out for various synchronous and asynchronous learning configurations. Scenarios such as different numbers of local models, variable learning rates, label grouping and uneven label distribution with different model importance were considered. Various hardware configurations were also analyzed, including the number of virtual cores (vCPU), disk types, communication delays and cluster failures. The implementation of federated learning was carried out on real servers in the cloud computing, with separate clusters for each institution, which enabled a realistic representation of the production environment. The research used modern tools such as PyTorch for model implementation, Kubernetes for cluster management, Docker for application containerization, and Linkerd, Minio, and Argo Workflows for communication and workflow management.

Based on the conducted research, conclusions were drawn on the effectiveness of various approaches to federated learning and the costs associated with improving the values of model metrics were estimated. In addition, algorithms and methods for resource prediction as part of automatic cluster scaling were described, which allows for the optimization of operating costs. Finally, directions for further research were proposed, which include the development of more advanced communication methods or optimization of processes related to federated learning.

Keywords: federated learning, model aggregation, cloud computing, containerization.

Spis treści

Rozdział 1

Wstęp

B. Bok, P. Mazur

1.1 Motywacja

W ostatnich latach można zauważyć dynamiczny rozwój technologii informacyjnych oraz coraz większą potrzebę analizy dużych zbiorów danych [1]. Postęp technologiczny umożliwia gromadzenie i przetwarzanie ogromnych ilości informacji, co prowadzi do rewolucji w różnych dziedzinach, takich jak medycyna, finanse, marketing, czy logistyka [2],[3],[4],[5]. Wraz z rosnącą ilością danych pojawia się również coraz większe zapotrzebowanie na odpowiednie rozwiązania infrastrukturalne, które pozwolą na ich skuteczne przetwarzanie i analizę.

Jednym z kluczowych wyzwań, które towarzyszą analizie dużych zbiorów danych, jest ochrona prywatności oraz bezpieczna analiza tychże danych [6]. Problem ten staje się szczególnie istotny, gdy dane przynależą do różnych instytucji, organizacji czy urządzeń, co jest coraz bardziej powszechne w dobie internetu rzeczy (IoT) oraz globalnych sieci informacyjnych. W tradycyjnym modelu analizy danych informacje są zazwyczaj gromadzone w jednym centralnym miejscu, co może stanowić zagrożenie dla prywatności oraz bezpieczeństwa danych.

Aby sprostać tym wyzwaniom, zaczęto rozwijać nowe podejścia i technologie, które umożliwiają analizę danych bez konieczności ich centralnego gromadzenia. Jednym z najbardziej obiecujących rozwiązań w tym zakresie jest uczenie federacyjne [7].

Uczenie federacyjne to paradygmat uczenia maszynowego, który umożliwia trenowanie modeli na zbiorach danych rozproszonych pomiędzy wieloma źródłami, zachowując jednocześnie kontrolę nad prywatnością i bezpieczeństwem informacji. W odróżnieniu od tradycyjnych metod, w których dane są agregowane w jednym miejscu przed procesem uczenia, uczenie federacyjne pozwala na trenowanie modeli bez konieczności przenoszenia danych, eliminując tym samym ryzyko wycieku poufnych informacji.

1.2 Cel i zakres pracy

Celem niniejszej pracy magisterskiej jest analiza i ocena metod uczenia federacyjnego ze szczególnym naciskiem na zrozumienie jego podstawowych mechanizmów, zalet i wyzwań.

Zakres pracy obejmuje:

- Przegląd literatury dotyczącej uczenia federacyjnego, w tym kluczowych artykułów naukowych i najnowszych badań w tej dziedzinie.
- Przegląd literatury dotyczącej wyzwań infrastrukturalnych w tematyce uczenia federacyjnego.
- Omówienie technicznych aspektów uczenia federacyjnego, w tym metody agregacji modeli, liczba modeli biorących udział w scenariuszu, czy rozłożenie danych trenujących.
- Przedstawienie i badanie różnych scenariuszy uczenia federacyjnego, zarówno synchronicznego, jak i asynchronicznego.
- Porównanie skuteczności i wydajności uczenia federacyjnego w porównaniu do tradycyjnych metod uczenia maszynowego.
- Identyfikację obszarów, w których uczenie federacyjne może znaleźć zastosowanie jako odpowiedź na wyzwania związane z ochroną prywatności i analizą rozproszonych zbiorów danych.
- Krytyczna analiza wyników oraz wnioski dotyczące przyszłości uczenia federacyjnego, jego potencjalnych zastosowań oraz możliwych kierunków dalszych badań w tej dziedzinie.

Rozdział 2

Aktualny stan wiedzy na temat uczenia federacyjnego

W dzisiejszym świecie, w którym ogromne ilości danych są generowane i przechowywane przez różne podmioty, problem zachowania prywatności danych oraz efektywnej współpracy w procesie uczenia maszynowego nabiera coraz większego znaczenia [8]. W odpowiedzi na te wyzwania, w ostatnich latach rozwija się paradygmat uczenia federacyjnego. W tym rozdziale dokonano przeglądu kluczowych artykułów naukowych z tego obszaru, które pomagają zrozumieć jego istotę, techniki oraz zastosowania.

2.1 Literatura dotycząca tematyki uczenia federacyjnego

B.Bok

Koncepcja rozproszonego uczenia maszynowego może obejmować m.in. uczenie rozproszone (*ang. distributed training*) [9], gdzie jedna instytucja rozprasza proces uczenia na wiele maszyn oraz uczenie federacyjne (*ang. federated learning*), gdzie różne instytucje pilnując prywatności swoich danych, wysyłają swoje modele, które następnie są agregowane. Uczenie federacyjne jest szczególnie przydatne w sytuacjach wymagających ochrony prywatności, np. w sektorze zdrowia czy finansów, ponieważ pozwala trenować globalny model bez ujawniania wrażliwych informacji.

Jednym z ważniejszych artykułów wprowadzających w tematykę uczenia federacyjnego jest praca McMahan et al. [10], która przedstawia algorytm uczenia głębokich sieci neuronowych na zdecentralizowanych danych, eliminując konieczność przesyłania danych do centralnego serwera. Artykuł ten stanowi fundament dla dalszych badań w tej dziedzinie.

Autorzy zaprezentowali równanie 1, które opisuje agregację wag modeli lokalnych (klientów) na serwerze.

Autorzy artykułu [11] skupiają się na strategiach poprawy wydajności komunikacji w uczeniu federacyjnym. Przedstawiają różne metody minimalizacji przesyłanych danych między klientami a serwerem, co ma kluczowe znaczenie dla efektywnego funkcjonowania systemów federacyjnego uczenia.

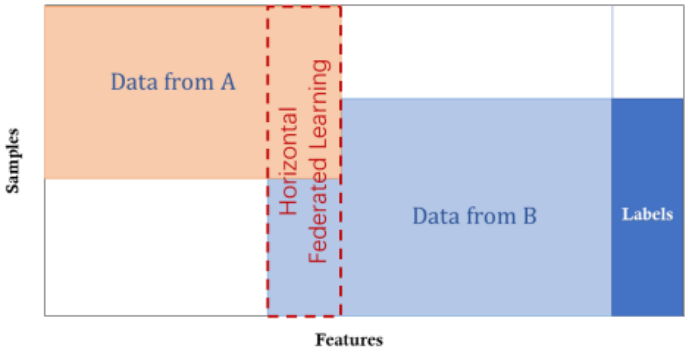
W artykule Yang et al. [12] omówiono zarówno koncepcję, jak i praktyczne zastosowania uczenia federacyjnego. Autorzy przedstawiają różnorodne techniki, w tym bezpieczeństwo, optymalizację i skalowalność, które są istotne w kontekście federacyjnego uczenia maszynowego.

Algorithm 1 FederatedAveraging

Input: The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

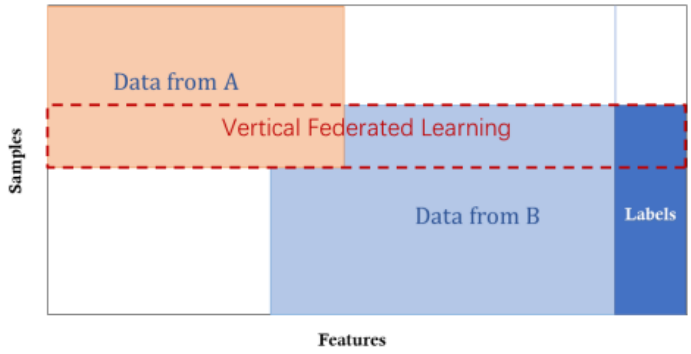
Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$ (random set of m clients)
  for each client $k \in S_t$ **in parallel do**
    $w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
  end for
  $m_t \leftarrow \sum_{k \in S_t} n_k$
  $w_{t+1} \leftarrow \sum_{k \in S_t} \frac{n_k}{m_t} w_{t+1}^k$
end for

Na rysunku 2.1 przedstawiono horyzontalne uczenie federacyjne (*ang. horizontal federated learning*), gdzie każdy klient posiada różne próbki (*ang. samples*), ale z tymi samymi cehami (*ang. features*). Jest to przydatne, gdy różne jednostki zbierają podobne dane, np. szpitale.



Rysunek 2.1: Horyzontalne uczenie federacyjne [12].

Rysunek 2.2 pokazuje wertykalne uczenie federacyjne (*ang. vertical federated learning*), gdzie każdy klient ma różne cechy tych samych próbek. Jest to użyteczne, gdy różne organizacje posiadają różne informacje o tych samych jednostkach, np. banki i firmy ubezpieczeniowe.

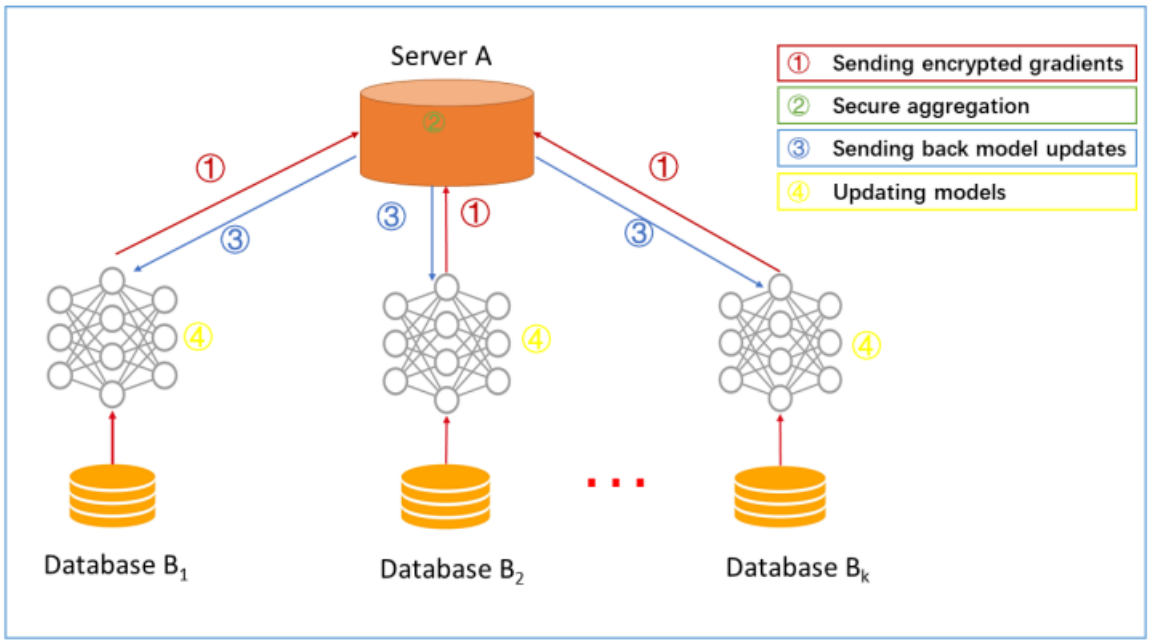


Rysunek 2.2: Wertykalne uczenie federacyjne [12].

Rysunek 2.3: Architektura horyzontalnego uczenia federacyjnego Rysunek 2.3

przedstawia proces horyzontalnego uczenia federacyjnego, który obejmuje:

1. Wysyłanie zaszyfrowanych wag/gradientów.
2. Bezpieczną agregację.
3. Wysyłanie zaktualizowanych modeli.
4. Aktualizację modeli na urządzeniach lokalnych.



Rysunek 2.3: Architektura horyzontalnego uczenia federacyjnego [12].

Istnieją również prace, które skupiają się na sposobie przekazywania modeli lokalnych między instytucjami, a serwerem centralnym. W jednej z takich prac [13] autorzy przedstawili koncepcję synchronicznego i asynchronicznego uczenia federacyjnego. W podejściu synchronicznym modele lokalne po wytrenowaniu jednej epoki przesyłają swoją wersję modelu na model centralny, który otrzymując wszystkie modele lokalne agreguje je, następnie rozsyłając z powrotem do klientów. Cały proces powtarza się daną liczbę kroków lub po osiągnięciu zadowalających wyników. Proces ten został przedstawiono poglądowo w pseudokodzie 2.

Algorithm 2 Synchronous Federated Learning

initialize w_0
for $t = 1, 2, \ldots, T$ **do**
  server sends w_t to all clients
  for $k = 1, 2, \ldots, K$ **do**
    $w_t^k \leftarrow w_t$
    $w_t^k \leftarrow \text{train}(w_t^k, \mathcal{D}k)$
    client k sends w_t^k to the server
  end for
  $w_{t+1} \leftarrow \text{aggregate}(w_t^k : k = 1, 2, \ldots, K)$
end for

gdzie:

- K - liczba klientów, indeksowanych przez k,
- T - liczba globalnych epok,
- w_t - wektory wag modelu na serwerze centralnym w iteracji t,
- w_t^k - wektory wag modelu lokalnego na kliencie k w iteracji t,
- $\mathcal{D}_k$ - zbiór danych dostępnych lokalnie dla klienta k,
- train$(w_t^k, \mathcal{D}_k)$ - funkcja trenowania lokalnego modelu na kliencie k,
- aggregate$(w_t^k : k = 1, 2, \ldots, K)$ - funkcja agregacji wag lokalnych modeli.

Jednak ze względu na trudność utrzymania synchronizacji wielu instytucji, częściej spotykanym przypadkiem w praktyce jest uczenie asynchroniczne, co również zostało opisane w pracy. To podejście zostało przedstawione poglądowo w pseudokodzie 3.

Algorithm 3 Asynchronous Federated Learning with Aggregation of Local and Previous Global Models

```
initialize w_0
server sends w_0 to all clients
while True do
    if receiving w_t^k from client k then
        aggregate global model: w_{t+1} ← aggregate(w_t, w_t^k)
        server sends updated w_{t+1} back to client k
    end if
end while
```

gdzie:

- w_t - wektor wag globalnego modelu na serwerze w iteracji t,
- w_t^k - wektor wag lokalnego modelu na kliencie k w iteracji t,
- aggregate(w_t, w_t^k) - funkcja agregacji, która łączy bieżący globalny model w_t z modelem lokalnym w_t^k otrzymanym od klienta k, aby utworzyć nowy zaktualizowany globalny model w_{t+1},
- k - indeks klienta, który aktualnie wysyła swoje lokalne wagi w_t^k do serwera.

Autorzy założyli, że serwer, otrzymując nową wersję modelu lokalnego, uśrednia go z poprzednią wersją modelu centralnego. Ma to na celu złagodzenie wpływu nieaktualnych modeli. Należy jednak zwrócić uwagę, że w sytuacji, gdy dana instytucja przez dłuższy czas nie wysyła swoich zaktualizowanych modeli lokalnych, jej wkład w wiedzę zawartą w modelu centralnym może ulec znacznemu zmniejszeniu. Wynika to z faktu, że model centralny nie uwzględnia najnowszych danych i wyników treningu tej instytucji, co prowadzi do jego potencjalnej dezaktualizacji względem jej specyficznych danych.

Rozwiązaniem tego problemu może być podejście, w którym serwer centralny przy każdej agregacji uwzględnia najnowsze wersje wszystkich dostępnych modeli lokalnych. Dzięki temu każdy klient, nawet jeśli przez pewien czas nie dostarcza nowych danych, nadal ma wpływ na aktualizacje modelu centralnego, co pozwala na bardziej równomierne i reprezentatywne uaktualnianie modelu globalnego. Zostało ono przedstawiono poglądowo za pomocą pseudokodu 4.

Algorithm 4 Asynchronous Federated Learning with Averaging of Latest Models

initialize w_0
t=0
server sends w_0 to all clients
while True **do**
  if receiving w_t^k from client k **then**
    update $\mathcal{W}_t$ with the newest local models
    update global model $w_{t+1} \leftarrow \text{aggregate}(\mathcal{W}_t)$
    server sends updated w_{t+1} back to client k
    t+=1
  end if
end while

gdzie:

- w_t - wektor wag globalnego modelu na serwerze centralnym w iteracji t,
- w_t^k - wektor wag lokalnego modelu na kliencie k w iteracji t,
- $\mathcal{W}_t$ - zbiór najnowszych lokalnych modeli otrzymanych od różnych klientów do momentu iteracji t,
- aggregate($\mathcal{W}_t$) - funkcja agregacji, która wykonuje uśrednianie najnowszych modeli lokalnych zawartych w zbiorze $\mathcal{W}_t$ w celu uzyskania nowego globalnego modelu w_{t+1},
- k - indeks klienta, który aktualnie wysyła swoje lokalne wagi w_t^k do serwera.

Postanowiono więc w niniejszej pracy skupić się na problemie zanikającej ważności modeli lokalnych, ponieważ nie jest to problem często omawiany w dotychczasowej literaturze. W szczególności, w niniejszej pracy przeanalizowano wpływ opóźnień w aktualizacjach modeli lokalnych na ich znaczenie w kontekście budowy modelu agregującego, co może prowadzić do utraty dokładności i aktualności tego ostatniego.

2.2 Literatura dotycząca tematyki wyzwań infrastrukturalnych

P. Mazur

Federacyjne uczenie się wprowadza szereg unikalnych wyzwań w porównaniu do tradycyjnych, scentralizowanych metod. Jednym z głównych problemów jest efektywna komunikacja między urządzeniami a serwerem centralnym. Ponieważ uczenie federacyjne wymaga wielu rund komunikacji w celu agregacji aktualizacji modelu, koszt przesyłania tych aktualizacji może być zbyt wysoki, zwłaszcza w sytuacjach z ograniczoną przepustowością sieci. McMahan et al. [10] zajmują się tym problemem, proponując algorytmy efektywne komunikacyjnie, takie jak Federated Averaging, które zmniejszają częstotliwość i rozmiar przesyłanych aktualizacji, niezbędnych do skutecznego trenowania modelu.

Skalowanie uczenia federacyjnego wprowadza dodatkowe komplikacje. Bonawitz et al. [14] omawiają wyzwania związane z projektowaniem systemów uczenia federacyjnego na dużą skalę. Obejmują one równoważenie obciążenia, bezpieczeństwo oraz zapewnienie odporności systemu na awarie i ataki. Autorzy podkreślają również znaczenie technik kryptograficznych w zabezpieczaniu aktualizacji modelu i utrzymaniu prywatności danych w systemach o dużej skali.

Pomimo postępów w dziedzinie uczenia federacyjnego, pozostaje wiele otwartych problemów, na które zwracają uwagę Kairouz er al. [15]. Wśród nich znajdują się heterogeniczność danych i urządzeń, efektywność protokołów komunikacyjnych oraz potrzeba udoskonalenia mechanizmów ochrony prywatności. Autorzy podkreślają konieczność opracowania nowych algorytmów, które będą w stanie radzić sobie z różnorodnymi rozkładami danych oraz zróżnicowanymi możliwościami obliczeniowymi urządzeń uczestniczących w uczeniu federacyjnym.

Proces treningu modeli w środowisku chmurowym może być realizowany na wiele sposobów, od wykorzystania maszyn wirtualnych po obliczenia w klastrach. W artykule [16] Zhou et al. proponują orkiestrację zadań uczenia maszynowego w oparciu o platformę Kubeflow i komponent Kubeflow Pipelines, na klastrze Kubernetes. Z kolei w swojej pracy [17] Wang et al. wykorzystują bardziej rozbudowane narzędzie zapewniające orkiestrację zadań uczenia maszynowego - Argo Workflows, dającego większą swobodę konfiguracji użytkownikowi.

2.3 Plan eksperymentów

B.Bok, P.Mazur

W niniejszej pracy magisterskiej postanowiono zbadać kilka istotnych kwestii związanych z uczeniem federacyjnym, które nie zostały dostatecznie omówione w dotychczasowej literaturze.

Po pierwsze, celem pracy jest zidentyfikowanie, jakie czynniki i w jaki sposób wpływają na wyniki w uczeniu federacyjnym. Zdecydowano się przeanalizować wpływ różnych parametrów trenowania, takich jak nierównomierność rozłożenia klas w danych lokalnych, grupowanie etykiet, wielkość danych treningowych, liczba klientów uczestniczących w procesie uczenia oraz wartość współczynnika uczenia. Zbadane zostaną również różne ważności modeli lokalnych, aby ocenić ich wpływ na jakość końcowego modelu agregującego. Tego typu analiza pozwoli na lepsze zrozumienie, jakie elementy procesu trenowania są kluczowe dla uzyskania optymalnych wyników w uczeniu federacyjnym.

Następnie postanowiono przeprowadzić badania nad wpływem parametrów konfiguracyjnych na czas synchronicznego uczenia federacyjnego. Eksperymenty koncentrują się na wpływie wielkości zasobów obliczeniowych maszyn wirtualnych oraz różnych konfiguracji dostępnej pamięci masowej. W literaturze dotyczącej uczenia federacyjnego zauważono braki w kontekście kosztów procesu, dlatego skupiono się również na rekomendacjach związanych z optymalizacją kosztową.

Kolejnym zagadnieniem podjętym w pracy jest uczenie asynchroniczne w kontekście federacyjnym. W tym przypadku postanowiono zbadać scenariusz, w którym na serwerze agregowane są zawsze najnowsze wersje modeli lokalnych, a nie - jak to bywa w podejściu standardowym - kombinacja modeli lokalnych i poprzedniego modelu globalnego. Analiza ta ma na celu zbadanie, jak taki sposób agregacji wpływa na stabilność i dokładność modelu agregującego, a także na potencjalne ryzyko związane z asymetrią aktualizacji (np. sytuacje, w których niektórzy klienci aktualizują modele lokalne znacznie rzadziej niż inni). Oczekuje się, że podejście to pozwoli na lepsze wyrównanie wpływu poszczególnych klientów na wynik końcowy oraz zmniejszenie ryzyka utraty ważnych informacji z mniej aktywnych klientów.

W dalszej części pracy skoncentrowano się na problemie zanikającej ważności modeli lokalnych, który nie jest szeroko omawiany w dotychczasowej literaturze. Przeanalizowano wpływ opóźnień w aktualizacjach modeli lokalnych na ich znaczenie w procesie budowy modelu agregującego. W praktycznych scenariuszach opóźnienia

mogą wynikać z wielu czynników, takich jak różne wielkości zasobów obliczeniowych, nierównomierne rozłożenie danych czy awarie. Na podstawie tych czynników przygotowano eksperymenty analizujące ich wpływ na wskaźniki jakości modeli.

Z uwagi na iteracyjny charakter uczenia federacyjnego, postanowiono również omówić możliwości automatycznego skalowania maszyn wirtualnych wykorzystywanych w procesie. Dodatkowym aspektem jest predykcja zasobów na podstawie historycznych przebiegów i danych treningowych, co stanowi kolejny obszar związany z optymalizacją kosztową procesu.

Rozdział 3

Architektura rozwiązania

P. Mazur

Z uwagi na charakterystykę procesu uczenia federacyjnego, uwzględniając podejście synchroniczne oraz asynchroniczne, należało zaprojektować i zaimplementować adekwatną architekturę. W niniejszym rozdziale wyznaczono najważniejsze cele stawiane opracowywanemu rozwiązaniu, a następnie zaproponowano spełniającą je architekturę. Opisano również wykorzystane narzędzia oraz najważniejsze szczegóły związane z implementacją.

3.1 Propozycja architektury z omówieniem wybranych narzędzi

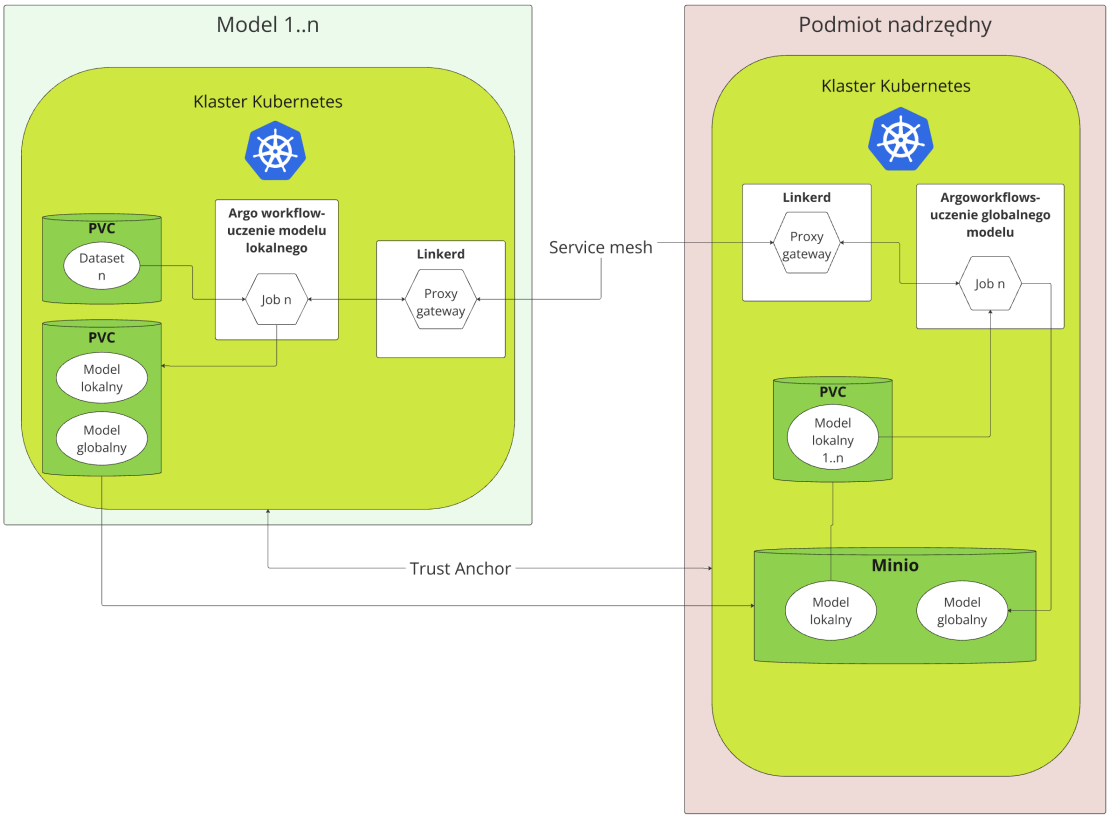
Biorąc pod uwagę specyfikę uczenia federacyjnego wyznaczono najważniejsze wymagania stawiane architekturze, przedstawiono ogólny schemat oraz opisano najważniejsze narzędzia niezbędne do implementacji rozwiązania.

3.1.1 Cele stawiane architekturze

Najważniejsze wymagania architektury zebrano w poniższych punktach.

- **Skalowalność** - rozwiązanie powinno zapewniać możliwość sprawnego zwiększania mocy obliczeniowej oraz być w stanie współpracować z różną ilością instytucji.
- **Trwałe przechowywanie danych** - dane oraz modele muszą być przechowywane w sposób trwały i bezpieczny. Powinny być również w łatwy sposób dostępne podczas treningu lub agregacji modeli.
- **Orkiestracja zadań uczenia maszynowego** - system musi umożliwiać efektywną orkiestrację zadań federacyjnego uczenia maszynowego, co obejmuje pobieranie danych, uczenie modeli lokalnych, wymianę modeli oraz ich agregację.
- **Bezpieczna komunikacja** - komunikacja między klastrami musi być bezpieczna, aby zapewnić poufność i integralność danych.

3.1.2 Schemat architektury



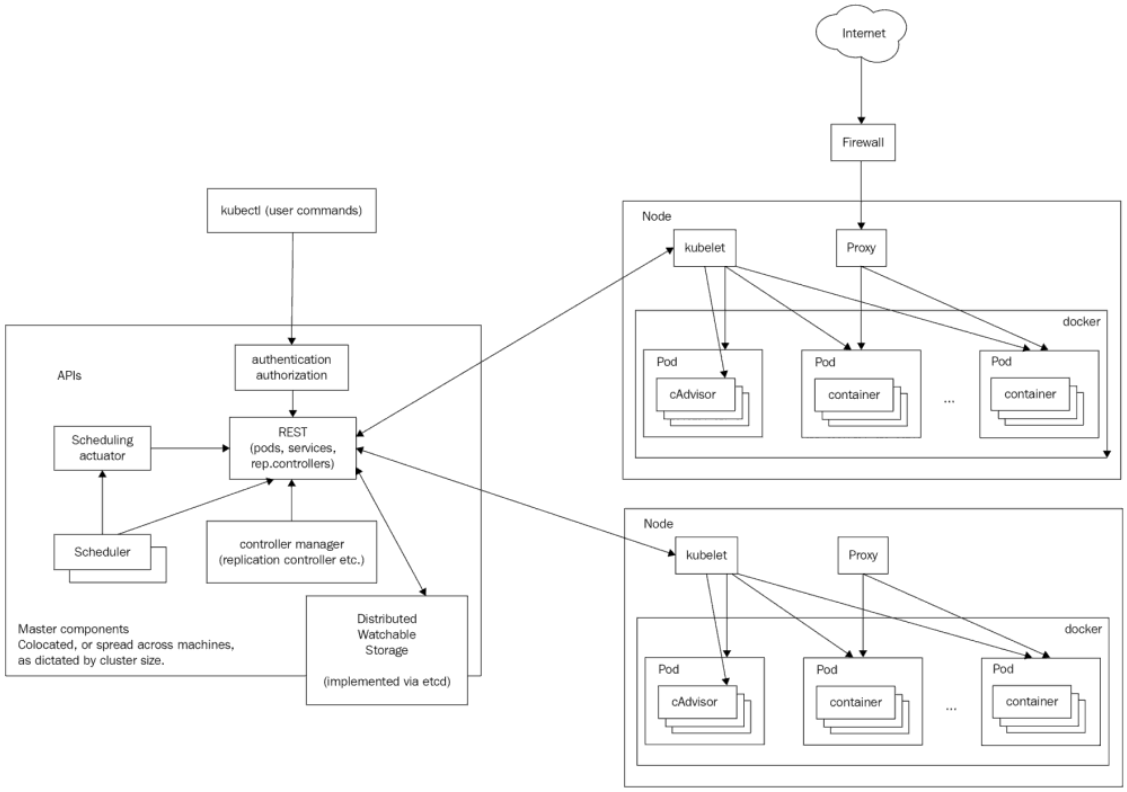
Rysunek 3.1: Ogólna architektura rozwiązania.

Powyższy rysunek 3.1 przedstawia architekturę uczenia federacyjnego opartą o klastry Kubernetes. Struktura jest podzielona na najważniejsze części:

- **Model 1..n** - reprezentuje poszczególne jednostki organizacyjne posiadające własne klastry Kubernetes. Każdy model posiada różne zestawy danych (*Dataset n*) i modele lokalne. Uczenie lokalnych modeli odbywa się za pomocą narzędzia `Argo Workflows`, które uruchamia zadania (*Job n*) w ramach klastrów Kubernetes.
- **Podmiot nadrzędny** - reprezentuje centralny klaster Kubernetes, który zarządza globalnymi modelami. Podmiot nadrzędny również wykorzystuje `Argo Workflows` do zarządzania i uczenia globalnych modeli. Wyniki uczenia lokalnych modeli są przesyłane za pośrednictwem `Linkerd`, który zapewnia komunikację między organizacjami a podmiotem nadrzędnym za pomocą *Service Mesh*.
- **PVC (Persistent Volume Claim)** - każdy klaster Kubernetes używa PVC do przechowywania lokalnych i globalnych modeli, co zapewnia trwałość danych.
- **Minio** - w centralnym klastrze Kubernetes znajduje się system plików `Minio`, który służy do przechowywania modeli lokalnych i globalnych.
- **Trust Anchor** - mechanizm bezpieczeństwa oparty o mTLS, który zapewnia integralność i bezpieczeństwo wymiany danych pomiędzy organizacjami a podmiotem nadrzędnym.

3.1.3 Klaster Kubernetes

Kubernetes to otwarte oprogramowanie do automatyzacji wdrażania, skalowania i zarządzania aplikacjami kontenerowymi. Poniżej przedstawiono poglądową architekturę klastra.



Rysunek 3.2: Kubernetes - architektura [18].

W swojej książce [18] Safyan dokładnie opisuje wszystkie najważniejsze człony klastra Kubernetes. Do najważniejszych elementów w klastrze należą:

- **API Server** - zgodnie z rysunkiem 3.2 to centralny punkt komunikacji dla wszystkich komponentów Kubernetes. Oferuje interfejs RESTful, umożliwiający operacje CRUD (Create, Read, Update, Delete) na zasobach. Autoryzuje i weryfikuje żądania oraz aktualizuje stan klastra.

- **Przestrzeń nazw** (namespace) - mechanizm logicznej izolacji zasobów. Umożliwia to podział klastrów na odrębne sekcje, w których mogą działać różne aplikacje i zespoły bez wzajemnego zakłócania się. Przestrzeń nazw pomaga zarządzać zasobami i uprawnieniami użytkowników w dużych klastrach.

- **Sekret** - obiekt, który służy do przechowywania i zarządzania wrażliwymi informacjami, takimi jak hasła, tokeny OAuth, klucze SSH itp. Zamiast umieszczać te informacje bezpośrednio w plikach konfiguracyjnych, sekret zapewnia bezpieczny sposób ich przechowywania i udostępniania aplikacjom działającym w klastrze. Umożliwia łatwy import wrażliwych danych przez zamontowanie pliku pod określoną ścieżką lub przekazanie ich za pomocą zmiennych środowiskowych.

- **Pod** - najmniejsza i najprostsza jednostka w Kubernetes. Zawiera jeden lub więcej kontenerów, które współdzielą zasoby, takie jak sieć i system plików. Pody są efemeryczne i mogą być replikowane dla skalowalności i niezawodności.
- **Service** - abstrakcjny obiekt definiujący logiczny zestaw podów i politykę dostępu do nich, umożliwiający odkrywanie usług i równoważenie obciążenia.
- **Ingress** - zarządza dostępem zewnętrznym do usług w klastrze, zwykle z wykorzystaniem HTTP. Definiuje reguły routingu, które mogą przekierowywać ruch do wybranych serwisów na podstawie ścieżek URL i nazw domen. Wymaga kontrolera Ingress do zarządzania regułami routingu.
- **Persistent Volume Claim (PVC)** - żądanie przez użytkownika na przestrzeń dyskową. Abstrakcja, która umożliwia dynamiczne zarządzanie przestrzenią dyskową. Wykorzystanie PVC pozwala na łatwe zarządzanie danymi oraz ich trwałe przechowywanie, nawet po zakończeniu pracy kontenerów.
- **Scheduler** - komponent odpowiedzialny za przydzielanie podów do węzłów w klastrze. Decyzje są podejmowane na podstawie dostępnych zasobów i wymagań podów. Optymalizuje rozmieszczenie podów w celu efektywnego wykorzystania zasobów.
- **Controller Manager** - uruchamia kontrolery, które zarządzają stanem zasobów w klastrze. Przykłady kontrolerów to Deployment Controller, ReplicaSet Controller, Node Controller. Odpowiada za utrzymanie pożądanego stanu systemu.
- **Job** - tworzy jeden lub więcej podów i zapewnia, że określona liczba zadań zakończy się pomyślnie. Używany do krótkotrwałych zadań, które muszą być wykonane do końca.

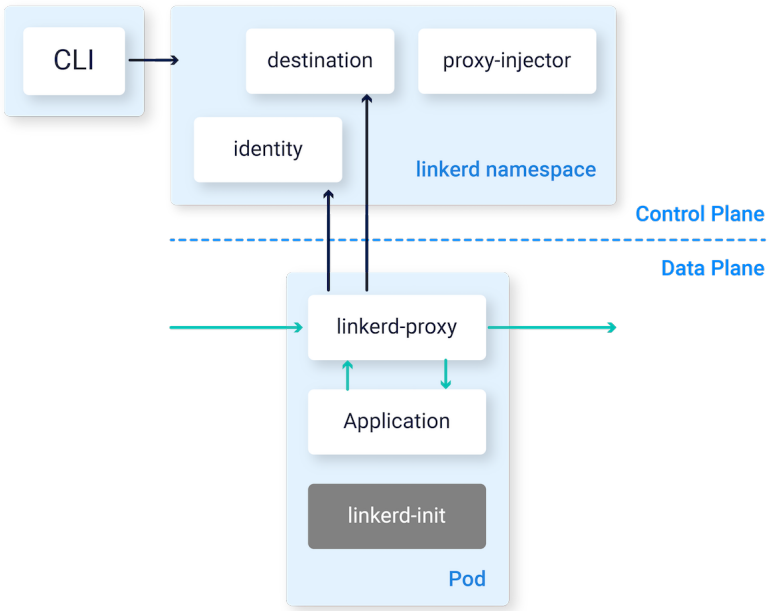
Każdy klaster w zaproponowanej architekturze zarządza kontenerami oraz zasobami niezbędnymi do uruchamiania zadań uczenia maszynowego, zapewniając skalowalność, wysoką dostępność oraz automatyczne zarządzanie zasobami.

3.1.4 Minio

Minio to serwer obiektowy zgodny z interfejsem API S3, który umożliwia przechowywanie danych w formie obiektów. W tej architekturze Minio jest używane do przechowywania zarówno lokalnych modeli, jak i głównego modelu globalnego. Minio zapewnia wysoką wydajność oraz skalowalność, co jest kluczowe dla przechowywania dużych ilości danych i modeli. Umożliwia również łatwą integrację z innymi narzędziami i serwisami, dzięki zgodności z API S3.

3.1.5 Linkerd

Linkerd to narzędzie typu service mesh, które zapewnia warstwę abstrakcji i zarządzania komunikacją między usługami w klastrze. Linkerd jest wdrażany na każdym z klastrów, gdzie pełni funkcję bramy, umożliwiającej odbieranie żądań z modeli lokalnych. To podejście działa w prawie każdej topologii sieci, wymagając jedynie, aby adres IP bramy był osiągalny przez klaster źródłowy. Linkerd zapewnia zarządzanie komunikacją między usługami, monitorowanie ruchu oraz automatyczne zarządzanie certyfikatami mTLS, co zwiększa bezpieczeństwo i niezawodność systemu.



Rysunek 3.3: Linkerd - architektura [19].

Na rysunku 3.3 przedstawiono uproszczoną architekturę Linkerd. Control plane w Linkerd to zbiór usług, które umożliwiają zarządzanie i kontrolę nad całą infrastrukturą Linkerd. Dzięki temu komponentowi możliwe jest centralne zarządzanie konfiguracją i monitorowanie stanu systemu. To tutaj dokonywana jest identyfikacja do jakiego serwisu należy przekierować dany ruch oraz potwierdzenie tożsamości nadawcy.

Z kolei data plane składa się z przezroczystych proxy, które działają jako dodatkowe kontenery w podach obok każdej instancji aplikacji. Proxy automatycznie przechwytują i obsługują cały ruch TCP do i z usługi, jednocześnie komunikując się z control plane w celu uzyskania odpowiednich konfiguracji i aktualizacji.

Linkerd oferuje również interfejs wiersza poleceń (CLI), który pozwala na interakcję zarówno z control plane, jak i data plane. CLI umożliwia administratorom i deweloperom wygodne zarządzanie konfiguracją, monitorowanie stanu systemu oraz diagnozowanie problemów w działaniu usług.

3.1.6 Argo Workflows

Argo Workflows to narzędzie do orkiestracji zadań w Kubernetes, umożliwiające definiowanie, planowanie i monitorowanie skomplikowanych przepływów pracy. Argo Workflows jest wykorzystywane do zarządzania procesem uczenia modeli maszynowych, zarówno lokalnych, jak i globalnych. Umożliwia automatyzację procesów oraz monitorowanie postępu zadań, co jest kluczowe dla efektywnego zarządzania procesem uczenia federacyjnego. Argo Workflows zostało wybrane ze względu na swoją elastyczność, skalowalność oraz integrację z Kubernetes.

W początkowych fazach implementacji wykorzystano Kubeflow Pipelines jako narzędzie orkiestrujące proces uczenie. Ze względu na ograniczenia związane z konfiguracją całego procesu, narzędzie zostało zmienione na Argo Workflows. Decyzja

została podjęta głównie ze względu na ograniczenia związane ze zmianą konfiguracji pamięci oraz samych podów, w których wykonywany jest trening modeli.

3.1.7 Terraform

Terraform to narzędzie do zarządzania infrastrukturą jako kodem, umożliwiające tworzenie, modyfikowanie i wersjonowanie infrastruktury w sposób deklaratywny. Terraform został wykorzystany do tworzenia klastrów Kubernetes i innych zasobów chmurowych. Umożliwia automatyzację wdrożeń, co zwiększa efektywność i redukuje błędy ludzkie. Terraform został wybrany ze względu na swoją elastyczność, wsparcie dla wielu dostawców chmurowych oraz możliwość zarządzania infrastrukturą w sposób zautomatyzowany. Dzięki wykorzystaniu tego narzędzia zmiany wielkości maszyn wirtualnych ograniczały się do zmiany poszczególnych zmiennych w kodzie.

3.2 Implementacja rozwiązania

W poniższym rozdziale omówiona zostanie implementacja rozwiązania związana z warstwą infrastruktury, komunikacji między modelami lokalnymi oraz orkiestacji procesu uczenia federacyjnego.

3.2.1 Klastry Kubernetes

Klastry Kubernetes zostały stworzone z wykorzystaniem narzędzia Terraform. Wykorzystano do tego chmurę Openstack z modułem orkiestrującym proces tworzenia klastrów - Magnum. Definicję klastra przedstawiono poniżej:

```
resource "openstack_containerinfra_cluster_v1" "central" {
  cluster_template_id = "e145fc40-b6f0-4f5c-93fd-ec65c0a5ae4c"
  create_timeout      = 60
  flavor              = "eo2a.large"
  keypair             = "pmazur"
  labels = {
    ingress_controller            = "nginx"
    master_lb_floating_ip_enabled = "true"
    etcd_volume_type = "ssd"
  }
  master_count  = 1
  master_flavor = "eo2a.medium"
  name          = "v2"
  node_count    = 1
  region        = "WAW3-2"
  merge_labels = true
}
```

W prosty sposób możliwe jest dodanie kolejnej maszyny wirtualnej do klastra, co ułatwiło przeprowadzanie eksperymentów:

```
resource "openstack_containerinfra_nodegroup_v1" "nodegroup_central" {
  cluster_id         = openstack_containerinfra_cluster_v1.central.id
  flavor_id          = "eo2a.xlarge"
  image_id           = "bea63892-1896-41bd-9c25-86b3a5d5bcb9"
  max_node_count = 1
  merge_labels   = null
  min_node_count = 1
  name           = "default-worker-3"
  node_count     = 1
  region         = "WAW3-2"
  role           = "worker"
}
```

Każdy z klastrów został utworzony w przedstawiony wyżej sposób, wraz z dodatkową maszyną wirtualną. Orkiestrator Magnum poza samą instalacją Kubernetes odpowiedzialny jest również za instalacją narzędzi potrzebnych do prawidłowego funkcjonowania klastra, takich jak CNI, CSI czy nginx-ingress controller.

3.2.2 Komunikacja między klastrami

Do zapewnienia prostej i bezpiecznej komunikacji między klastrami wykorzystano jedną z funkcji Linkerd, jaką jest service mirror. Wybrany serwis, po oznaczeniu go za pomocą adnotacji zostanie odwzorowany w połączonym klastrze. Dzięki temu możliwa jest komunikacja z wykorzystaniem serwisu, tak jakby aplikacja docelowa znajdowała się w tym samym klastrze:

```
curl -X POST http://v1-central:5000/set_minio_path
```

Klastry zostały wcześniej połączone z pomocą CLI Linkerd, gdzie między innymi zostały wymienione klucze służące do szyfrowania komunikacji między nimi oraz do identyfikacji połączeń.

3.2.3 Minio

W celu zapewnienia dostępu do danych oraz wymiany modeli między klastrem centralnym a modelami lokalnymi zaimplementowano object storage w postaci minio. Wykorzystano do tego klaster centralny. Każdy z modeli lokalnych ma dostęp za pomocą klucza oraz sekretu do dedykowanych kubełków, na które może umieścić model lokalny, oraz z którego może pobrać model centralny po agregacji. Każdy proces uczenia na danym klastrze generuje unikalny folder w kubełku na potrzeby zapisu. Bezpośrednio miedzy klastrami dochodzi do wymiany ścieżek do modeli w minio, natomiast modele przesyłane są za pośrednictwem object storage. Dzięki temu komunikacja między klastrami nie wymaga dużej przepustowości.

Klucze dostępowe do Minio przechowywane są bezpośrednio na klastrze Kubernetes za pomocą obiektu sekret, dzięki czemu nie jest potrzebne przechowywanie wrażliwych danych w repozytorium kodu. Klucze przekazywane są za pomocą zmiennych środowiskowych bezpośrednio podczas tworzenia poda, co ułatwia również zmiany w konfiguracji.

3.2.4 Orkiestracja procesu uczenia federacyjnego

W celu orkiestracji uczenia federacyjnego na każdym z klastrów uruchomiono dedykowany workflow, w zależności od wykonywanego zadania, agregacji lub uczenia modelu lokalnego. Proces ten odbywa się w wielu krokach i obejmuje pobieranie danych, trenowanie modeli dla określonej ilości epok, a także zarządzanie wieloma wersjami modeli lokalnych. Każdy krok workflow jest reprezentowany przez szablon (*template*), który definiuje konkretne zadanie, takie jak uruchomienie kontenera, wykonanie skryptu, czy też sekwencję innych kroków.

Poniżej zamieszczono opis szablonów wykorzystywanych w przepływie związanym z agregacją modeli lokalnych.

- **main** - jest to główny szablon, od którego rozpoczyna się wykonanie workflow. Zawiera sekwencję kroków, które są wykonywane w ustalonej kolejności. To w nim definiowany jest przebieg całego procesu - pobranie danych a następnie wykonanie agregacji dla zadanej liczby epok.

- **read-images-from-minio** - szablon, który odpowiada za pobieranie obrazów z serwera MinIO. Używa kontenera z narzędziem *minio/mc* do połączenia z MinIO i pobrania danych do lokalnego woluminu.
- **epoch** - szablon reprezentujący pojedynczą epokę trenowania modelu. W jego ramach wykonywane są kroki takie jak pobieranie lokalnych modeli, agregacja modelu, przesyłanie wyników oraz uruchamianie dalszych zadań trenowania.
- **get-local-models-dag** - szablon definiujący DAG (Directed Acyclic Graph) z zadaniami równoległymi, które odpowiadają za pobieranie lokalnych wersji modeli.
- **generate-model** - szablon odpowiedzialny za agregację modeli lokalnych do centralnego modelu. Proces ten jest wykonywany w dedykowanym kontenerze, który uruchamia skrypt agregacyjny.
- **upload-model** - szablon przesyłający wygenerowany centralny model oraz wskaźniki jakości na serwer MinIO, aby umożliwić dalsze przetwarzanie i analizę wyników.
- **trigger-train-workflows-dag** - szablon, który uruchamia dalsze zadania trenowania dla każdej wersji lokalnego modelu. W ramach DAG-u wykonywane są zapytania HTTP do serwerów, aby wyzwolić nowe workflowy dla kolejnych epok trenowania.
- **trigger-train-workflow** - szablon wysyłający zapytania POST do wybranego serwera, aby uruchomić nowe zadanie trenowania dla konkretnej wersji modelu.

Szablony workflow związanego z uczeniem modeli lokalnych zawierają następujące różnice:

- `start-server` - szablon ten odpowiada za uruchomienie serwera, który odpowiada za interakcję z modelem centralnym. Serwer umożliwia komunikację z klastrem centralnym poprzez HTTP, co pozwala na dynamiczne zarządzanie procesem agregacji modeli.
- `get-central-model` - szablon ten jest odpowiedzialny za pobieranie modelu centralnego przed rozpoczęciem trenowania lokalnych modeli w każdej epoce (z wyjątkiem pierwszej). Model centralny jest następnie wykorzystywany do dalszego trenowania.
- `read-minio` - szablon ten obsługuje pobieranie danych z zewnętrznego systemu przechowywania MinIO po uruchomieniu serwera. Dane te są następnie wykorzystywane w procesie trenowania modelu.
- `upload-model` - szablon przesyłający wygenerowany lokalny model oraz wskaźniki jakości na serwer MinIO, aby umożliwić agregację modeli.
- `trigger-aggregete-workflow` - szablon, który odpowiada za wywoływanie zapytań HTTP do serwera, umożliwiając uruchamianie kolejnych procesów agregacji modeli. W ramach tego kroku przekazywana jest również ścieżka z lokalizacją modelu lokalnego w MinIO.

Wszystkie szablony wykorzystują zdefiniowane PVC do przechowywania danych, modeli oraz wyników. Montowanie PVC umożliwia współdzielenie danych pomiędzy różnymi krokami workflow.

Workflow związany z agregacją modeli lokalnych składa się z następujących kroków, które są wykonywane w poniższej kolejności:

1. **Pobranie obrazów z MinIO** - w tym kroku, szablon *read-images-from-minio* pobiera obrazy z serwera MinIO do lokalnego PVC.

2. **Trenowanie modeli przez określoną liczbę epok** - w kolejnych krokach workflow wykonuje sekwencję trenowania modeli:
   (a) *Pobranie lokalnych modeli* - każda epoka rozpoczyna się od kroku *get-local-models-dag*, który pobiera wszystkie wersje lokalnych modeli.
   (b) *Agregacja modeli* - w kroku *generate-model*, lokalne modele są agregowane do centralnego modelu na podstawie danych z bieżącej epoki.
   (c) *Przesyłanie wyników* - centralny model oraz wskaźniki jakości są przesyłane na serwer MinIO przy użyciu szablonu *upload-model*.
   (d) *Wyzwolenie kolejnych treningów* - w ostatnim kroku epoki, *trigger-train-workflows-dag* uruchamia trenowanie dla każdej wersji lokalnego modelu.
3. **Zakończenie Workflow** - Po wykonaniu wszystkich epok workflow kończy swoje działanie, pozostawiając wyniki na serwerze MinIO do dalszej analizy.

Proces trenowania modeli lokalnych składa się z kilku kluczowych kroków, które są realizowane w określonej sekwencji. Poniżej przedstawiono szczegółowy opis poszczególnych etapów:

1. **Pobranie obrazów z MinIO** - pierwszym krokiem jest pobranie obrazów z serwera MinIO do lokalnego woluminu. W tym celu używany jest szablon *read-images-from-minio*. Obrazy te stanowią dane treningowe niezbędne do trenowania modeli lokalnych.
2. **Trenowanie modeli przez określoną liczbę epok** - proces trenowania modeli składa się z 14 epok, z których każda realizuje określone kroki:
   (a) **Pobranie modelu centralnego** - w każdej epoce, z wyjątkiem pierwszej, proces rozpoczyna się od pobrania modelu centralnego. Realizowane jest to za pomocą szablonu *get-central-model*, który pobiera najnowszą wersję modelu centralnego z serwera MinIO.
   (b) **Trenowanie modelu lokalnego** - po pobraniu modelu centralnego, przystępuje się do trenowania modelu lokalnego. W kroku tym wykorzystywany jest szablon *generate-model* (w pierwszej epoce używany jest *generate-model-epoch-1*), który trenuje model lokalny na podstawie bieżących danych. Parametry procesu trenowania, takie jak (*learning-rate*) czy liczba wersji, są definiowane w argumentach szablonu.
   (c) **Przesyłanie wyników** - po zakończeniu trenowania, wyniki są przesyłane na serwer MinIO. Szablon *upload-model* odpowiada za przesłanie wytrenowanego modelu lokalnego oraz powiązanych wskaźników jakości. Ścieżka do modelu jest zapisywana w zmiennej *minio-path*, co umożliwia jej łatwe odnalezienie i wykorzystanie w kolejnych krokach.
   (d) **Wyzwolenie agregacji** - na zakończenie każdej epoki przesyłana jest ścieżka do MinIO wraz z informacją o zakończeniu traningu dla danej epoki. Szablon *trigger-aggregete-workflow* wysyła odpowiednie żądanie HTTP do serwera, co inicjuje kolejne procesy agregacji.
3. **Zakończenie Workflow** - po zakończeniu wszystkich 14 epok, workflow kończy swoje działanie. Wyniki, w tym wytrenowane modele i zebrane wskaźniki jakości, są pozostawiane na serwerze MinIO, gdzie mogą zostać poddane dalszej analizie i ocenie.

3.3 Asynchroniczne uczenie federacyjne

Z uwagi na ograniczenia związane z synchronicznym uczeniem federacyjnym zaproponowano podejście asynchroniczne. Każdy z modeli lokalnych uzyskuje możliwość generowania modelu centralnego na żądanie. Model zostanie wygenerowany w oparciu o najnowsze modele lokalne. Informacja o najnowszym modelu centralnym przechowywana jest w dedykowanym serwerze.

3.3.1 Serwer przechowujący stan klastra

W ramach asynchronicznego podejścia wymagana jest ciągła wiedza o najświeższym modelu dla każdej wersji. Zaimplementowano serwer z wykorzystaniem Python'a oraz biblioteki cherrypy. Ścieżka "set_minio_path"umożliwia zapisanie w pamięci ścieżki do modelu, natomiast "get_minio_path"zwraca zapisano wcześniej ścieżkę, lub "null"w przypadku braku w pamięci ścieżki. Komunikacja odbywa się za pomocą żądań HTTP. Każdy model lokalny posiada własny serwer przechowujący informację o ostatnim wygenerowanym modelu.

3.3.2 Agregacja modeli centralnych

Każdy z modeli lokalnych posiada własny workflow, czekający na żądanie wygenerowania modelu centralnego. Po zakończeniu uczenia i przesłaniu modelu lokalnego do minio, wyzwalana jest agregacja wraz z przesłaniem ścieżki do modelu. W klastrze centralnym po otrzymaniu żądania aktualizowana zostaje ścieżka do modelu lokalnego w serwerze. Następnie odpytywane są wszystkie serwery o pozostałe, aktualne modele lokalne. Przejście do następnego kroku wymaga obecności przynajmniej dwóch modeli lokalnych, w przeciwnym wypadku następuje oczekiwanie aż wymagana liczba modeli zostanie osiągnięta. Po spełnieniu warunku następuje pobranie wszystkich dostępnych modeli lokalnych, uśrednianie i przekazanie nowego modelu centralnego.

Workflow związany z asynchroniczną agregacją modeli składa się z następujących kroków, realizowanych w określonej sekwencji, przy czym poszczególne etapy są wykonywane równolegle dla różnych wersji modeli.

1. **Pobranie obrazów z MinIO** - pierwszym krokiem w procesie jest pobranie obrazów z serwera MinIO. W tym celu wykorzystuje się szablon *read-images-from-minio*, który pobiera obrazy z określonej ścieżki do lokalnego woluminu, co umożliwia ich dalsze przetwarzanie podczas trenowania modeli.
2. **Sekwencyjne trenowanie modeli przez określoną liczbę epok** - w głównej części procesu modele są trenowane przez określoną liczbę epok, przy czym każdy krok trenowania obejmuje:
   (a) **Oczekiwanie na model lokalny** - każdy z modeli lokalnych posiada własny workflow agregujący, który oczekuje na przesłanie nowego modelu lokalnego i zwraca najnowszy model centralny.
   (b) **Pobranie lokalnych modeli** - na początku każdej epoki lokalne modele są pobierane przy użyciu szablonu *get-local-model*. Proces ten polega na uruchomieniu serwera oraz odczytaniu modelów z serwera MinIO. W ramach tego kroku odpytywane są wszystkie serwery przechowujące ścieżki do najbardziej aktualnych modeli lokalnych z poszczególnych wersji.
   (c) **Agregacja lokalnych modeli** - Po pobraniu lokalnych modeli następuje ich agregacja w centralny model przy użyciu szablonu *generate-model*.

(d) **Przesyłanie wyników** - po zakończeniu procesu agregacji, centralny model oraz związane z nim wskaźniki jakości są przesyłane na serwer MinIO przy użyciu szablonu *upload-model*.

(e) **Wyzwolenie treningu kolejnych epok modeli lokalnych** - ostatnim krokiem w każdej epoce jest uruchomienie treningu kolejnej epoki dla modelu, który wyzwolił agregację, co realizowane jest przez szablon *trigger-train-workflow*. Proces ten polega na zaktualizowaniu ścieżki do centralnego modelu i ponownym rozpoczęciu trenowania modeli lokalnych.

3. **Zakończenie procesu** - po zakończeniu treningu wszystkich epok następuje zapis ostatecznej wersji modelu centralnego, wraz ze wskaźnikami jakości na serwerze MinIO.

3.4 Podsumowanie

Zaproponowana architektura spełnia najważniejsze cele, jakie powinny zostać spełnione w celu przeprowadzenia poprawnych eksperymentów uczenia federacyjnego w chmurze. Implementacja oraz wykorzystane narzędzia zapewniają również elastyczność w konfiguracji infrastruktury jak i logiki działania całego procesu. Umożliwia to testowanie różnych scenariuszy uczenia federacyjnego.

Rozdział 4

Implementacja wybranych scenariuszy

B. Bok

W niniejszym rozdziale omówiono wskaźniki jakości oceny skuteczności klasyfikatorów wieloklasowych, które były kluczowe w przeprowadzonych badaniach. Szczegółowo przedstawiono metody i wskaźniki, pozwalające na dokładną ocenę wyników modeli klasyfikacyjnych.

4.1 Wskaźniki jakości klasyfikatora wieloklasowego

Modele lokalne, jak i model agregujący, będą zaprezentowane jako klasyfikator wieloklasowy. Wskaźniki jakości dla klasyfikatora wieloklasowego pozwalają na ocenę skuteczności modeli w sytuacjach, gdzie występuje więcej niż dwie klasy. Każda z tych metryk dostarcza informacji na temat jakości klasyfikacji i pozwala zrozumieć, w jaki sposób model radzi sobie z różnymi klasami. Poniżej przedstawiono najważniejsze wskaźniki jakości stosowane w klasyfikacji wieloklasowej.

Macierz pomyłek (Confusion Matrix)

Macierz pomyłek to tabela, która pokazuje liczbę prawidłowych i nieprawidłowych klasyfikacji dla każdej z klas. Dla klasyfikacji wieloklasowej jest to macierz kwadratowa o wymiarach liczby klas.

Dokładność (Accuracy)

Dokładność jest definiowana jako stosunek liczby poprawnych klasyfikacji do całkowitej liczby klasyfikacji.

$$\text{Accuracy} = \frac{\text{Liczba poprawnych klasyfikacji}}{\text{Całkowita liczba klasyfikacji}}$$

Precyzja (Precision)

Precyzja dla klasy i to stosunek liczby prawdziwych pozytywnych predykcji dla tej klasy do sumy prawdziwych i fałszywych pozytywnych predykcji dla tej klasy.

$$\text{Precision}_i = \frac{TP_i}{TP_i + FP_i}$$

gdzie TP_i to liczba prawdziwych pozytywnych dla klasy i, a FP_i to liczba fałszywych pozytywnych dla klasy i.

Czułość (Recall)

Czułość dla klasy i to stosunek liczby prawdziwych pozytywnych predykcji do sumy prawdziwych pozytywnych i fałszywych negatywnych predykcji dla tej klasy.

$$\text{Recall}_i = \frac{TP_i}{TP_i + FN_i}$$

gdzie FN_i to liczba fałszywych negatywnych dla klasy i.

F1-Score

F1-Score jest średnią harmoniczną precyzji i czułości. F1-Score dla klasy i definiowany jest jako:

$$\text{F1-Score}_i = \frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i}$$

Średnia ważona (Weighted Average)

Średnia ważona powyższych metryk (precyzji, czułości, F1-Score) dla całego zestawu danych, biorąc pod uwagę liczbę próbek w każdej klasie.

4.2 Dane

W uczeniu federacyjnym często ma się do czynienia z danymi, które instytucje niechętnie wymieniają między sobą ze względu na ich poufność lub wartość. Przykładami takich danych mogą być informacje medyczne, finansowe lub drogowe. W tym badaniu wybrano znaki drogowe jako przedmiot analizy, zakładając scenariusz, w którym firmy opracowujące modele dla pojazdów autonomicznych nie chcą udostępniać swoich danych, ale dążą do poprawy jakości swoich modeli.

Do budowy i trenowania modeli wykorzystano dane przedstawiające znaki drogowe, które zostały pobrane z zestawu danych German Traffic Sign Recognition Benchmark (GTSRB) [20]. Dane te są dostępne publicznie na stronie internetowej: https://benchmark.ini.rub.de/gtsrb_news.html.

Zestaw danych GTSRB jest powszechnie używanym zbiorem testowym w zadaniach rozpoznawania obrazów, szczególnie w kontekście klasyfikacji znaków drogowych. Zawiera on obrazy znaków drogowych, które zostały ręcznie oznakowane i przypisane do jednej z wielu klas.

4.2.1 Opis danych

Zestaw danych GTSRB składa się z obrazów znaków drogowych, które zostały podzielone na różne klasy w zależności od rodzaju znaku. Dane te zawierają:

- **Liczba klas:** 43 klasy różnych znaków drogowych, w 5 nadrzędnych klasach.
- **Liczba obrazów:** Ponad 50 000 obrazów w zestawie treningowym oraz 12 569 obrazów w zestawie testowym.
- **Rozdzielczość obrazów:** Obrazy mają różne rozdzielczości, jednak większość z nich to kwadratowe obrazy o wymiarach 32x32 piksele.
- **Format plików:** Pliki graficzne są w formacie PPM (Portable Pixmap).

4.2.2 Przygotowanie danych

Przed rozpoczęciem trenowania modeli, dane zostały wstępnie przetworzone, aby zapewnić ich jednolitą jakość i format:

- **Skalowanie obrazów:** Wszystkie obrazy zostały przeskalowane do jednakowej rozdzielczości 32x32 piksele.
- **Normalizacja:** Piksele obrazów zostały znormalizowane do wartości z zakresu [0, 1].
- **Etykietowanie:** Obrazy zostały odpowiednio oznakowane, zgodnie z przypisaną klasą znaku drogowego.

4.2.3 Podział danych

Dane zostały podzielone na dwa zestawy:

- **Zestaw treningowy:** Używany do trenowania modeli, zawierał większość obrazów z zestawu danych.
- **Zestaw testowy:** Używany do ostatecznej oceny skuteczności modeli po zakończeniu procesu trenowania.

4.2.4 Zastosowanie danych

Dane z zestawu GTSRB zostały wykorzystane do trenowania i testowania modeli w dwóch scenariuszach uczenia federacyjnego: scenariusza z komunikacją synchroniczną oraz scenariusza z komunikacją aynchroniczną. Wyniki uzyskane z tych modeli pozwalają na porównanie skuteczności i wydajności różnych podejść do uczenia federacyjnego w kontekście klasyfikacji obrazów znaków drogowych.

4.3 Kod i implementacja

4.3.1 Wprowadzenie

W niniejszym rozdziale omówione zostaną szczegóły implementacji kodu do trenowania modeli MobileNet v3, które zostały wykorzystane do rozpoznawania znaków drogowych. Implementacja opiera się na rozwiązaniu przedstawionym w artykule [21], wykorzystując framework PyTorch [22], a dane do trenowania zostały pobrane z zestawu danych GTSRB (German Traffic Sign Recognition Benchmark).

4.3.2 Przygotowanie środowiska i danych

Pierwszym krokiem było przygotowanie odpowiedniego środowiska programistycznego. Wykorzystano biblioteki takie jak PyTorch, torchvision oraz numpy. Zestaw danych GTSRB został pobrany i przetworzony do odpowiedniego formatu. Dane te zostały podzielone na zestawy treningowe i walidacyjne.

```
import torch
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader

# Transformacje danych
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

# Ładowanie danych
train_dataset = ImageFolder(root='path_to_train_data', transform=transform)
train_loader = DataLoader(dataset=train_dataset, batch_size=32, shuffle=True)

val_dataset = ImageFolder(root='path_to_val_data', transform=transform)
val_loader = DataLoader(dataset=val_dataset, batch_size=32, shuffle=False)
```

Transformacje danych obejmowały zmianę rozmiaru obrazów do 224x224 pikseli, normalizację oraz konwersję do tensora. Dane treningowe i walidacyjne zostały załadowane przy użyciu DataLoadera z odpowiednimi ustawieniami batch_size i shuffle.

4.3.3 Architektura modelu MobileNet v3

MobileNet v3 [23] jest najnowszą wersją w rodzinie modeli MobileNet, zaprojektowaną z myślą o urządzeniach mobilnych i wbudowanych systemach. Model ten łączy w sobie zaawansowane techniki optymalizacji z wcześniejszych wersji, takich jak separable convolutions, oraz nowe innowacje w architekturze sieci neuronowych, które poprawiają zarówno wydajność, jak i efektywność obliczeniową.

Funkcje aktywacji h-swish i h-sigmoid

MobileNet v3 wprowadza nowe funkcje aktywacji h-swish (hard-swish) i h-sigmoid (hard-sigmoid), które są bardziej wydajne obliczeniowo w porównaniu do tradycyjnych funkcji aktywacji, takich jak ReLU. Funkcje te są definiowane jako:

$$\text{h-swish}(x) = x \cdot \frac{\text{ReLU6}(x+3)}{6}$$

$$\text{h-sigmoid}(x) = \frac{\text{ReLU6}(x+3)}{6}$$

Funkcje te pozwalają na zachowanie właściwości nieliniowych, które są istotne dla głębokich sieci neuronowych, jednocześnie redukując złożoność obliczeniową.

Dwa warianty: MobileNet v3 Small i Large

Model MobileNet v3 występuje w dwóch wariantach: Small i Large, które są dostosowane do różnych potrzeb obliczeniowych i ograniczeń zasobów:

- **MobileNet v3 Small**: Zaprojektowany do urządzeń z bardzo ograniczonymi zasobami obliczeniowymi, takich jak urządzenia IoT. Model ten jest zoptymalizowany pod kątem minimalnej liczby operacji obliczeniowych i małej liczby parametrów.
- **MobileNet v3 Large**: Przeznaczony do bardziej wymagających zastosowań, gdzie dostępne są większe zasoby obliczeniowe. Model ten oferuje lepszą dokładność kosztem większej złożoności obliczeniowej.

Implementacja w PyTorch

Implementacja modelu MobileNet v3 w PyTorch zaczyna się od wczytania pretrenowanego modelu, który został wstępnie wytrenowany na zbiorze danych ImageNet. Następnie modyfikowana jest ostatnia warstwa w celu dopasowania jej do liczby klas w zestawie danych GTSRB.

```
from torchvision import models

# Wczytanie pretrenowanego modelu MobileNet v3
model = models.mobilenet_v3_large(pretrained=True)

# Modyfikacja ostatniej warstwy w celu dopasowania do liczby klas
model.classifier[3] = torch.nn.Linear(
in_features=model.classifier[3].in_features, out_features=num_classes
)

# Przeniesienie modelu na GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)
```

W powyższym kodzie model MobileNet v3 został wczytany z pretrenowanymi wagami. Ostatnia warstwa klasyfikatora została zmodyfikowana, aby dopasować ją do liczby klas w zadaniu rozpoznawania znaków drogowych.

Zalety i zastosowania MobileNet v3

MobileNet v3 jest szczególnie przydatny w aplikacjach, gdzie kluczowe są ograniczone zasoby obliczeniowe, takich jak:

- **Rozpoznawanie obrazów i obiektów** w aplikacjach mobilnych i systemach wbudowanych.
- **Systemy nawigacji i autonomiczne pojazdy**, gdzie wymagane są szybkie i efektywne modele rozpoznawania wzorców.
- **Internet Rzeczy (IoT)**, gdzie urządzenia mają ograniczoną moc obliczeniową i zasilanie bateryjne.

Dzięki swojej efektywności i lekkości, MobileNet v3 jest jednym z najbardziej zaawansowanych modeli głębokiego uczenia zaprojektowanych z myślą o zastosowaniach mobilnych.

Implementacja w PyTorch

Implementacja modelu MobileNet v3 w PyTorch zaczyna się od wczytania pretrenowanego modelu, który został wstępnie wytrenowany na zbiorze danych ImageNet. Następnie modyfikowana jest ostatnia warstwa w celu dopasowania jej do liczby klas w zestawie danych GTSRB.

```
from torchvision import models

# Wczytanie pretrenowanego modelu MobileNet v3
model = models.mobilenet_v3_large(pretrained=True)

# Modyfikacja ostatniej warstwy w celu dopasowania do liczby klas
model.classifier[3] = torch.nn.Linear(
    in_features=model.classifier[3].in_features,
    out_features=num_classes
)

# Przeniesienie modelu na GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)
```

W powyższym kodzie model MobileNet v3 został wczytany z pretrenowanymi wagami. Ostatnia warstwa klasyfikatora została zmodyfikowana, aby dopasować ją do liczby klas w zadaniu rozpoznawania znaków drogowych.

4.3.4 Proces trenowania

Podczas trenowania, użyto funkcji strat typu Cross-Entropy oraz optymalizatora Adam. Proces trenowania obejmował monitorowanie wskaźników takich jak dokładność oraz strata na zbiorze testowym.

4.3.5 Znaczenie współczynnika uczenia w optymalizatorze Adam

Współczynnik uczenia jest kluczowym hiperparametrem wpływającym na proces optymalizacji sieci neuronowej. Określa on, jak duże zmiany w wagach modelu będą wprowadzane podczas każdej iteracji. Zbyt wysoki współczynnik uczenia może prowadzić do oscylacji wokół minimum funkcji kosztu, podczas gdy zbyt niski może skutkować wolną konwergencją.

Optymalizator Adam (Adaptive Moment Estimation) [24] łączy zalety metod RMSProp i Momentum, dynamicznie dostosowując współczynnik uczenia dla każdego parametru modelu. Aktualizacja wag odbywa się zgodnie z równaniami:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{4.1}$$

gdzie m_t i v_t to momenty pierwszego i drugiego rzędu, a g_t to gradient w momencie t. Następnie skorygowane momenty oblicza się jako:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}. \tag{4.2}$$

Aktualizacja wag jest wykonywana za pomocą:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t, \tag{4.3}$$

gdzie η to współczynnik uczenia, a ϵ to mała wartość zapobiegająca dzieleniu przez zero.

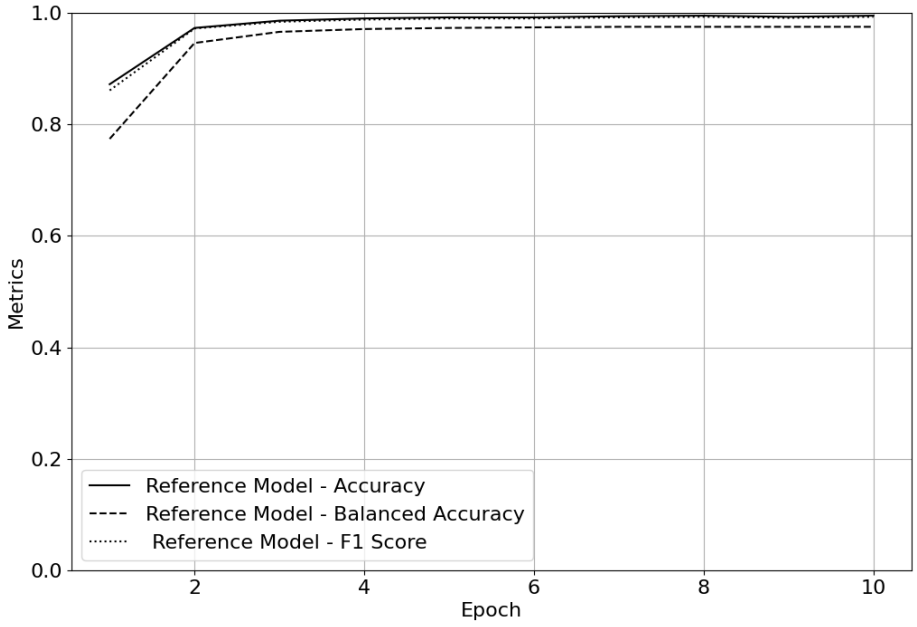
W uczeniu federacyjnym, gdzie w każdej epoce optymalizator Adam jest definiowany od początku, jego adaptacyjność pomaga złagodzić wahania wynikające z wielu wsadów (*ang. batches*) danych. Mimo adaptacyjnego charakteru optymalizatora Adam, wartość η nadal odgrywa istotną rolę w szybkości konwergencji modelu. Dlatego właściwy dobór η jest kluczowy dla stabilności i efektywności trenowania.

Odpowiedni dobór współczynnika uczenia ma kluczowe znaczenie dla skutecznego trenowania modeli. W ramach eksperymentów badane będą różne wartości początkowe oraz podejścia ze zmniejszaniem współczynnika uczenia, aby znaleźć optymalne ustawienia w scenariuszach uczenia federacyjnego.

4.4 Model referencyjny

W celu uzyskania odniesienia, dla porównania skuteczności modeli w różnych scenariuszach uczenia federacyjnego, przeprowadzono trening modelu referencyjnego na pełnym zbiorze danych GTSRB, bez podziału na modele lokalne. Użyto jednego modelu, który trenowano w sposób tradycyjny, z dostępem do wszystkich danych.

Model referencyjny umożliwia ocenę, jak scenariusze federacyjne wypadają w porównaniu do standardowego podejścia. Wyniki, takie jak dokładność, zbalansowana dokładność i F1-Score, uzyskane podczas treningu modelu referencyjnego, stanowią podstawę do porównań z wynikami modeli trenowanych w architekturze federacyjnej. Dane wskaźniki jakości są istotne dla oceny jakości modelu, jednak w tym przypadku ich wartości mają podobny przebieg, dlatego jako główny wskaźnik wybrano dokładność.



Rysunek 4.1: Przebieg metryk dla modelu referencyjnego.

Po 10 epokach uczenia uzyskano następujące wyniki:

- **Dokładność** — 0.995,
- **Zbalansowana dokładność** — 0.975,
- **F1 Score** — 0.993.

Na podstawie wyników przedstawionych na wykresie 4.1 oraz końcowych wartości metryk można stwierdzić, że model referencyjny osiągnął bardzo wysoką skuteczność w klasyfikacji znaków drogowych. Model bardzo szybko, już po trzech do czterech epokach, osiągnął wysokie wartości metryk. Ostateczna wartość dokładności wynosiła 0.995, co oznacza, że model prawidłowo sklasyfikował 99,5% wszystkich próbek w zbiorze testowym. Zbalansowana dokładność osiągnęła 0.975, co wskazuje na niewielkie różnice w skuteczności klasyfikacji między różnymi klasami, jednak nadal na bardzo wysokim poziomie. Dodatkowo, wynik F1 Score, równy 0.993, potwierdza, że model doskonale radził sobie z równoważeniem precyzji i czułości, co jest kluczowe w zadaniach klasyfikacji wieloklasowej. Te wyniki sugerują, że model referencyjny jest niezwykle skuteczny i stabilny.

Rozdział 5

Badanie wybranych scenariuszy uczenia federacyjnego z komunikacją synchroniczną

B. Bok

W niniejszym rozdziale przedstawiono badania przeprowadzone w celu oceny skuteczności i wydajności wybranych scenariuszy uczenia federacyjnego z modelem agregującym oraz komunikacją synchroniczną. Badania te mają na celu zrozumienie, jakie czynniki i w jakim stopniu wpływają na dokładność modeli.

5.1 Parametry maszyny, na której przeprowadzono badania

Badania przedstawione w niniejszym rozdziale zostały przeprowadzone na maszynie z systemem operacyjnym Ubuntu, wyposażonej w procesor Intel Core i7-10750H, pracujący z bazową częstotliwością 2.60 GHz i maksymalną częstotliwością 5.0 GHz. Procesor posiada 6 rdzeni, obsługujących 12 wątków, oraz jest zgodny z architekturą x86_64.

Maszyna dysponuje 32 GB pamięci operacyjnej RAM.

Za akcelerację graficzną odpowiada karta graficzna NVIDIA GeForce GTX 1660 Ti z 6 GB dedykowanej pamięci VRAM, działająca pod kontrolą sterownika w wersji 550.107.02 oraz obsługująca CUDA w wersji 12.4.

5.2 Metodologia

Badania przedstawione w tym rozdziale obejmują eksperymenty, w których analizowane są różne konfiguracje: modele z różnymi wartościami współczynnika uczenia, liczbą modeli lokalnych, a także równomiernym i nierównomiernym rozkładem etykiet. Celem jest określenie, które czynniki, takie jak nierównomierność klas w danych, grupowanie etykiet, czy wartości współczynnika uczenia, najbardziej wpływają na stabilność i jakość modeli. Dzięki temu możliwe jest lepsze zrozumienie wyzwań związanych z komunikacją synchroniczną oraz potencjalne opracowanie metod usprawniających ten proces w kontekście uczenia federacyjnego.

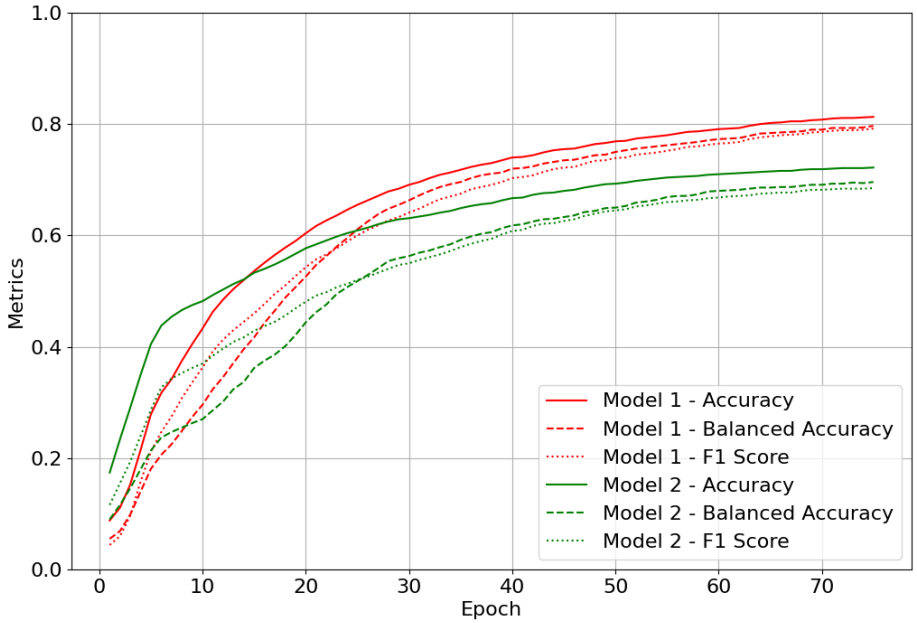
5.3 Scenariusz ze stałym współczynnikiem uczenia, stałymi ważnościami modeli oraz równo rozłożonymi etykietami

W tej sekcji opisano wyniki uzyskane dla scenariusza z modelem agregującym, w którym zastosowano stały współczynnik uczenia i równą ważność modeli lokalnych. Przeprowadzono eksperymenty dla różnych wartości współczynnika uczenia (0.00001, 0.0001, 0.001) oraz różnej liczby modeli lokalnych (2, 3, 4).

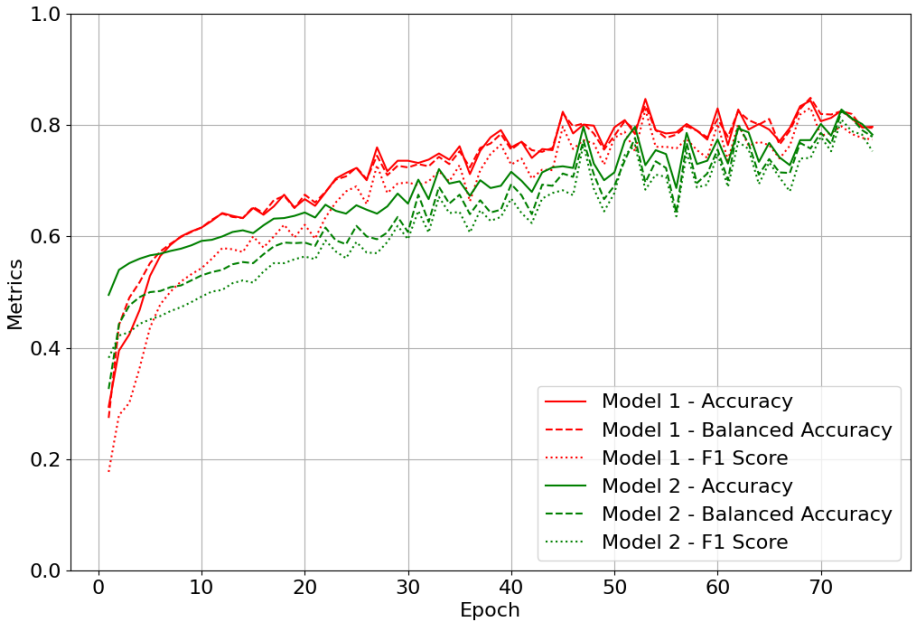
W tym scenariuszu założono, że każdy model lokalny posiadał podobną liczbę etykiet do trenowania, co oznacza, że dane zostały rozdzielone równomiernie pomiędzy wszystkie modele uczestniczące w procesie uczenia federacyjnego. Równo rozłożone etykiety oznaczają, że każdy z modeli lokalnych ma dostęp do podobnej liczby przykładów podczas treningu, co minimalizuje ryzyko niejednorodności danych i umożliwia bardziej spójne trenowanie w środowisku federacyjnym. Dzięki temu można ocenić wpływ innych czynników, takich jak współczynnik uczenia czy liczba modeli lokalnych, na metryki jakości modelu agregującego.

5.3.1 Analiza wyników dla 2 modeli lokalnych

W tej sekcji przedstawiono wyniki eksperymentów przeprowadzonych dla konfiguracji z 2 modelami lokalnymi. Analiza skupia się na zbadaniu, jak zmiana współczynnika uczenia wpływa na metryki takie jak accuracy, balanced accuracy oraz F1 score w miarę postępu treningu.



Rysunek 5.1: Przebieg metryk dla 2 modeli lokalnych oraz współczynnika uczenia równego 0.00001.



Rysunek 5.2: Przebieg metryk dla 2 modeli lokalnych oraz współczynnika uczenia równego 0.0001.

Rysunek 5.3: Przebieg metryk dla 2 modeli lokalnych oraz współczynnika uczenia równego 0.001.

Analizując rysunki 5.1, 5.2 oraz 5.3, można zauważyć, że w przypadku 2 modeli lokalnych, przy niskim współczynniku uczenia (0.00001), model charakteryzuje się wolnym oraz stabilnym przyrostem wartości metryk. Zwiększenie współczynnika do 0.0001 skutkuje szybszym wzrostem wartości metryk, jednak po ok. 20 epoce można

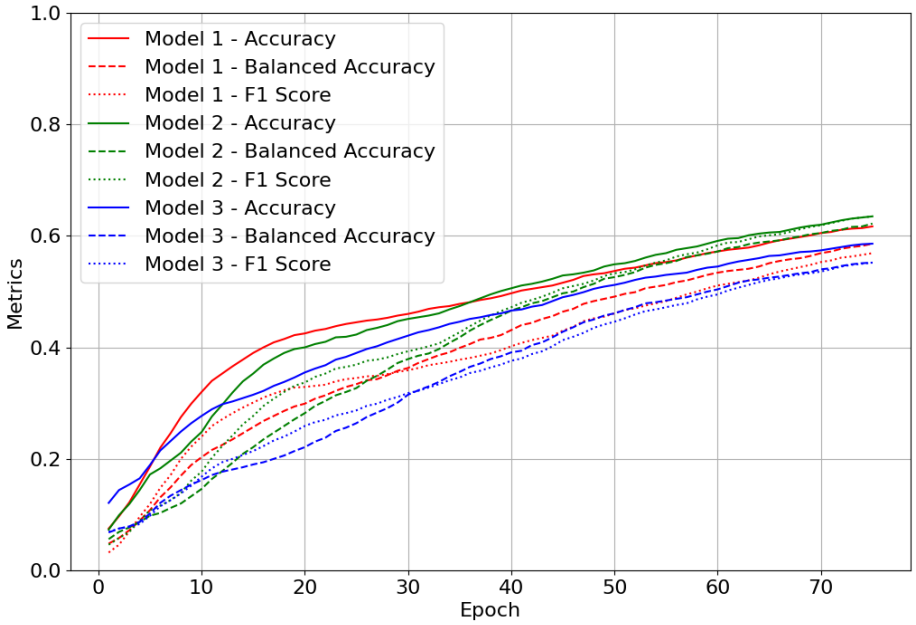
zauważyć zachwianie in stabilności. Jeszcze większą niestabilność występuje przy największej wartości współczynnika uczenia (0.001).

Zauważona niestabilność przy coraz większym współczynniku uczenia może być wynikiem silnego przeuczenia się modeli lokalnych na własnych danych. W przypadku ładowania zagregowanego modelu z serwera, model lokalny przy dużym współczynniku uczenia może znacząco zmienić swoje parametry, co może prowadzić do nadmiernych różnic w stosunku do zagregowanego modelu.

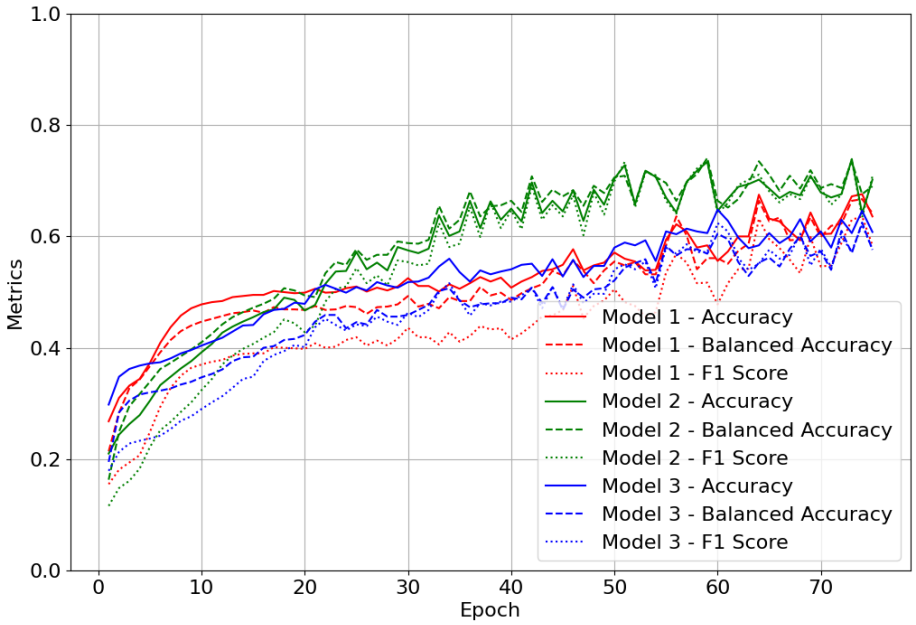
Warto również zwrócić uwagę, że wskaźniki jakości są znacząco gorsze od referencyjnych, co może być wynikiem kilku czynników. Przede wszystkim, rozproszenie danych pomiędzy dwie instytucje powoduje, że każdy model lokalny uczy się na swoim unikalnym podzbiorze danych, co prowadzi do trudności w uogólnieniu wyników. Dodatkowo, niski współczynnik uczenia ogranicza tempo dostosowywania parametrów modeli lokalnych, co może powodować powolny wzrost dokładności i wydajności w początkowych fazach treningu. Ostatecznie, kombinacja tych czynników może prowadzić do znacznie niższych wskaźników jakości w porównaniu do modelu referencyjnego, który trenował na pełnym zbiorze danych bez konieczności agregacji wyników z różnych źródeł.

5.3.2 Analiza wyników dla 3 modeli lokalnych

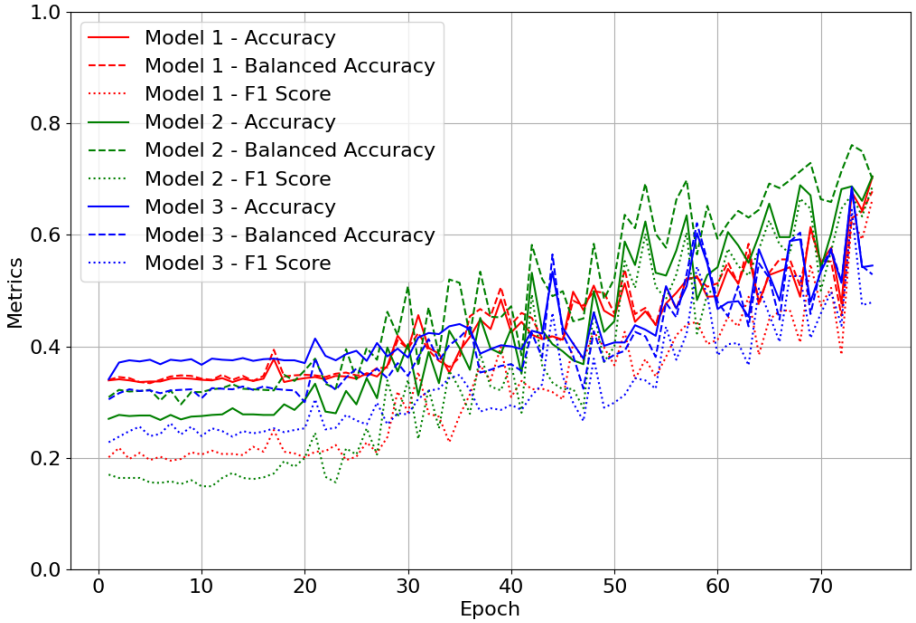
Podrozdział ten zawiera analizę wyników uzyskanych w przypadku użycia 3 modeli lokalnych.



Rysunek 5.4: Przebieg metryk dla 3 modeli lokalnych oraz współczynnika uczenia równego 0.00001.



Rysunek 5.5: Przebieg metryk dla 3 modeli lokalnych oraz współczynnika uczenia równego 0.0001.

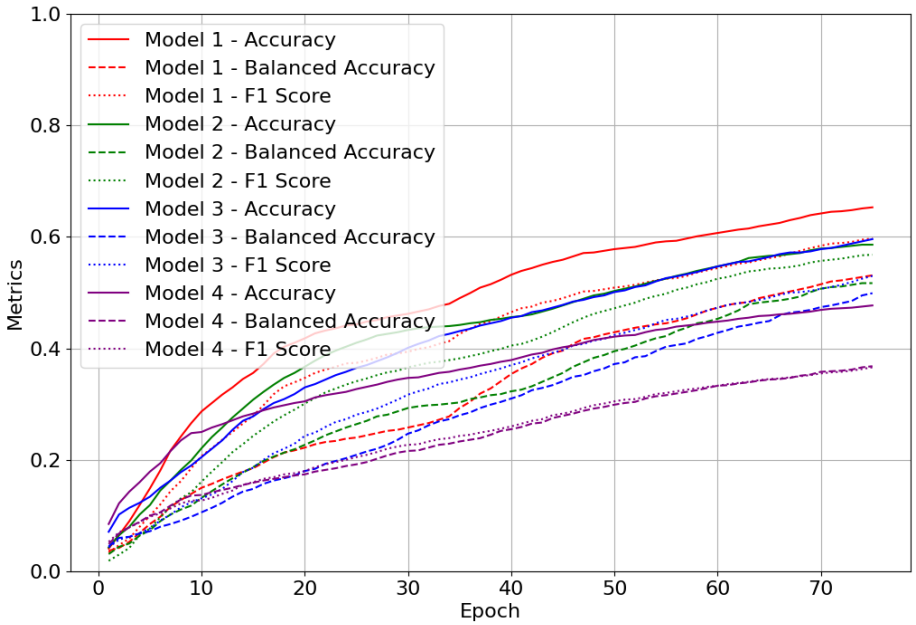


Rysunek 5.6: Przebieg metryk dla 3 modeli lokalnych oraz współczynnika uczenia równego 0.001.

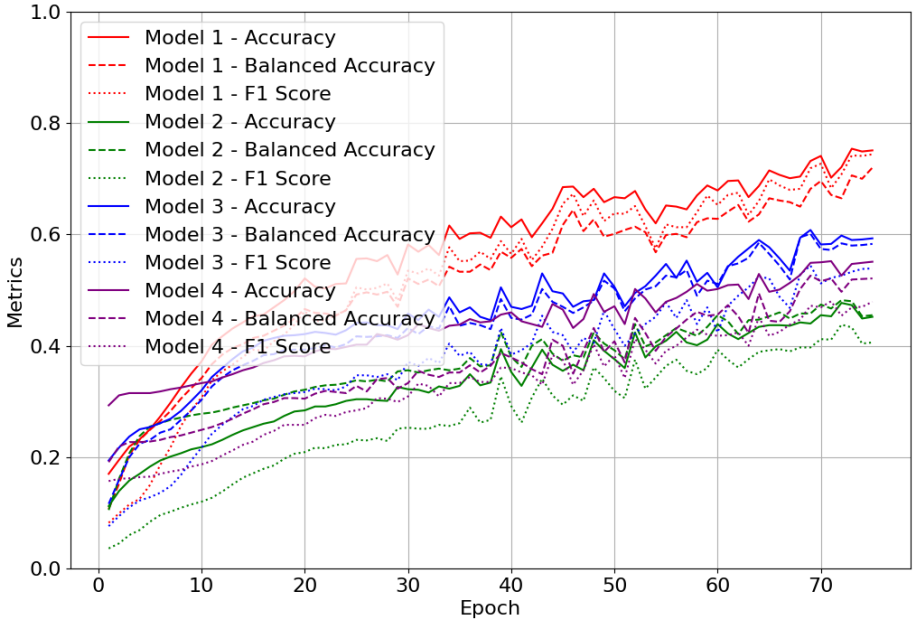
Analizując rysunki 5.4, 5.5 oraz 5.6, przedstawiające wyniki eksperymentów dla 3 modeli lokalnych, można wyciągnąć podobne wnioski, co do zmiany przebiegu wartości metryk, w zależności od wielkości współczynnika uczenia. Porównując jednak bezpośrednio wyniki dla 3 oraz 2 modeli lokalnych, widoczny jest znaczący spadek wskaźników jakości modeli, dla większej ich liczby.

5.3.3 Analiza wyników dla 4 modeli lokalnych

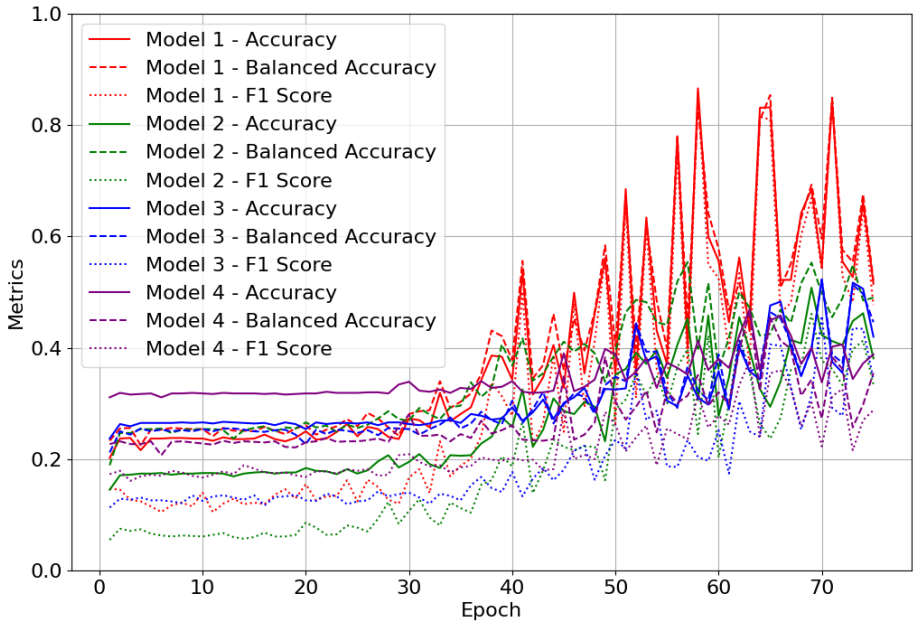
W tym podrozdziale omówiono wyniki eksperymentów przeprowadzonych z 4 modelami lokalnymi.



Rysunek 5.7: Przebieg metryk dla 4 modeli lokalnych oraz współczynnika uczenia równego 0.00001.



Rysunek 5.8: Przebieg metryk dla 4 modeli lokalnych oraz współczynnika uczenia równego 0.0001.



Rysunek 5.9: Przebieg metryk dla 4 modeli lokalnych oraz współczynnika uczenia równego 0.001.

W przypadku metryk 4 modeli lokalnych, przedstawionych na rysunkach 5.7, 5.8 oraz 5.9, można zauważyć analogiczne zależności jak w przypadku 2 oraz 3 modeli. Przy najniższym współczynniku uczenia (0.00001), model cechuje się powolnym, lecz stabilnym wzrostem wartości metryk. Wraz ze zwiększeniem współczynnika uczenia do 0.0001, przyrost wartości metryk staje się szybszy, jednak pojawia się niestabilność. Największa niestabilność również występuje dla najwyższego współczynnika uczenia (0.001), co może wskazywać na przeuczenie modeli lokalnych.

Porównując wyniki dla 4 modeli z wynikami dla 2 i 3 modeli, można zauważyć wyraźniejszy spadek wartości metryk przy większej liczbie modeli lokalnych. Może to wynikać z większych różnic między parametrami modeli lokalnych a zagregowanym modelem, co prowadzi do trudniejszej synchronizacji i spójności modeli lokalnych przy większym współczynniku uczenia.

5.3.4 Podsumowanie

Na podstawie analizy wyników dla różnych konfiguracji liczby modeli lokalnych (2, 3, i 4) można zauważyć, że niezależnie od liczby modeli, wpływ wzrostu współczynnika uczenia jest podobny: coraz większa jego wartość powoduje szybsze uczenie się, ale jednak większą niestabilność, która najprawdopodobniej wynika z przeuczenia się na danych lokalnych.

Wraz ze wzrostem liczby modeli można również zauważyć spadek wartości metryk, co może wynikać z trudniejszej agregacji większej liczby modeli. danych.

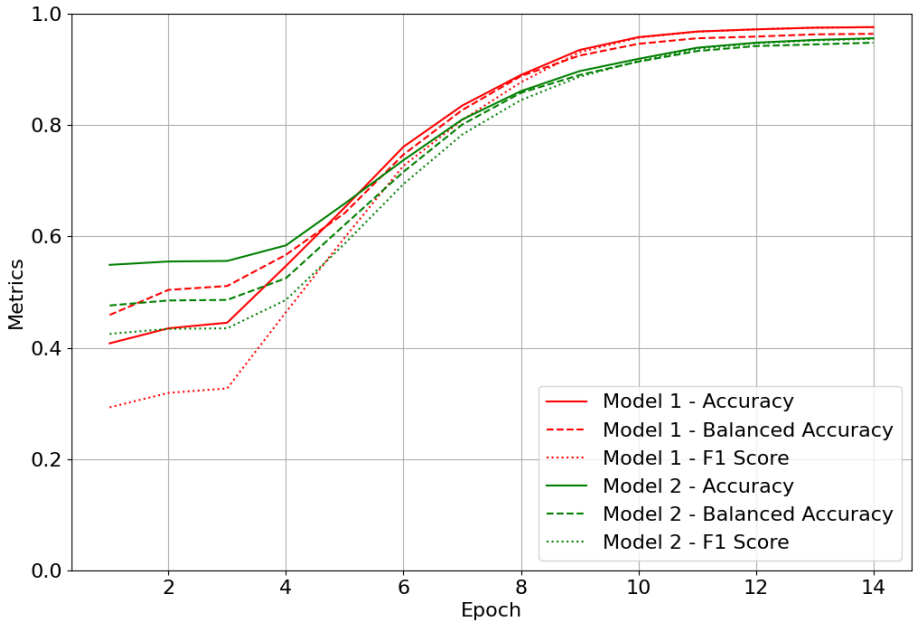
W porównaniu do modelu referencyjnego, można zauważyć o wiele wolniejszy przyrost metryk. Dla niższej wartości współczynnika uczenia, wspomagającej stabilność wyników, trening na nawet 75 epokach nie przyniósł przybliżonych wyników do modelu referencyjnego, który już po kilku epokach uzyskał wysokie wartości metryk.

Wysoka wartość współczynnika uczenia pozwala na szybszy przyrost metryk, jednak z czasem wprowadza dużą niestabilność. Mała wartość tego współczynnika charakteryzuje się większą stabilnością, jednak powoduje wolną naukę modeli lokalnych. Dlatego też w następnym podrozdziale postanowiono przyjąć początkowo większą wartość współczynnika uczenia oraz zmniejszać jego wartość w kolejnych epokach treningu.

5.4 Scenariusz ze zmiennym współczynnikiem uczenia, stałymi ważnościami modeli oraz równo rozłożonymi etykietami

W tym rozdziale przedstawiono wyniki uzyskane dla scenariusza z modelem agregującym, w którym zastosowano zmienny współczynnik uczenia, jednakową ważność modeli oraz równo rozłożone etykiety. Modele trenowano z początkowym współczynnikiem uczenia wynoszącym 0.001, który zmniejszano dwukrotnie w każdej kolejnej epoce. Zmienny współczynnik uczenia został wprowadzony, aby zapobiec problemowi przeuczania, który zaobserwowano przy jego stałej wartości. Początkowa jego wartość nie powinna być jednak za mała, aby proces treningu nie trwał zbyt długo, co można było zauważyć w poprzednim rozdziale.

5.4.1 Analiza wyników dla 2 modeli lokalnych

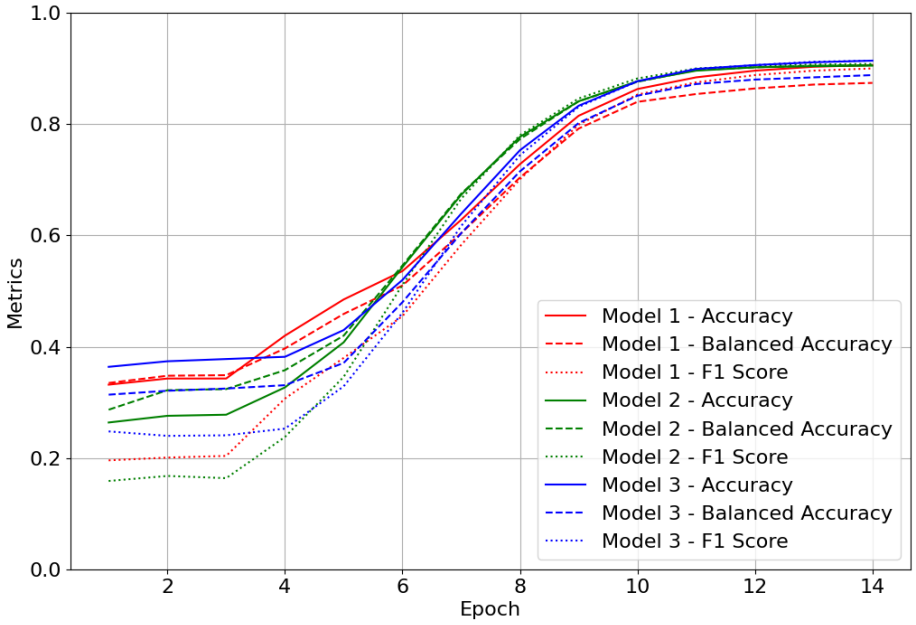


Rysunek 5.10: Przebieg metryk dla 2 modeli i malejącego learning rate.

Jak widać na rysunku 5.10, zmniejszanie współczynnika uczenia skutecznie zapobiega problemowi przeuczenia w późniejszych epokach. Modele stopniowo osiągają wysokie wartości metryk, a zmniejszenie wartości learning rate prowadzi do

stabilizacji i dalszej poprawy, nawet po wielu epokach, co sugeruje zdolność do precyzyjnego dostrajania wag modelu. Zrównoważona dokładność poprawia się wraz ze zmniejszaniem learning rate, co może wskazywać na lepszą generalizację.

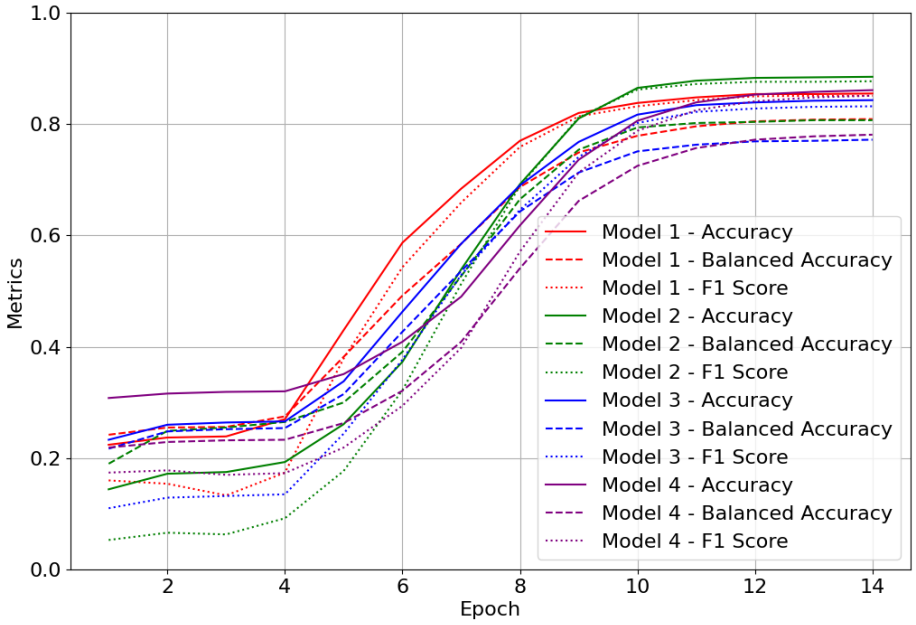
5.4.2 Analiza wyników dla 3 modeli lokalnych



Rysunek 5.11: Przebieg metryk dla 3 modeli i malejącego learning rate.

Na podstawie analizy rysunku 5.11 można zauważyć, że w przypadku 3 modeli, podobnie jak dla 2, wprowadzenie malejącego współczynnika uczenia skutkuje zauważalnym wzrostem metryk w początkowych epokach, a następnie stabilizacją po dalszym zmniejszaniu wartości współczynnika uczenia. Wyniki pokazują, że spadek dokładności wynosi około 10% w porównaniu do modelu referencyjnego, co najprawdopodobniej wynika z rozproszenia danych pomiędzy trzy instytucje oraz uśredniania parametrów modeli lokalnych podczas procesu agregacji. W rezultacie modele lokalne nie są w stanie osiągnąć tych samych wyników co model referencyjny, który trenuje na pełnym zbiorze danych bez konieczności agregacji i synchronizacji wyników z różnych źródeł.

5.4.3 Analiza wyników dla 4 modeli lokalnych



Rysunek 5.12: Przebieg metryk dla 4 modeli i malejącego learning rate.

Na podstawie analizy rysunku 5.12 można stwierdzić, że w przypadku 4 modeli lokalnych wprowadzenie malejącego współczynnika uczenia również przyczynia się do wzrostu metryk na wczesnych etapach treningu, po czym następuje stabilizacja. Jednakże, w porównaniu z mniejszą liczbą modeli, poprawa wydajności jest mniej dynamiczna, co może sugerować większą trudność w efektywnym dostrajaniu większej liczby modeli jednocześnie. Spadek wartości wskaźników jakości wynosi ok. 20%.

5.4.4 Podsumowanie

Na podstawie wyników uzyskanych w tm podrozdziale, można stwierdzić, że dwukrotnie zmniejszanie współczynnika uczenia co epokę było trafionym pomysłem.

Warto zwrócić uwagę, że na prezentowanych wykresach przedstawiono wyniki dla 14 epok. Liczba ta została dobrana w taki sposób, aby można było zauważyć charakterystykę przyrostu metryk modeli lokalnych, przed ich względną stabilizacją.

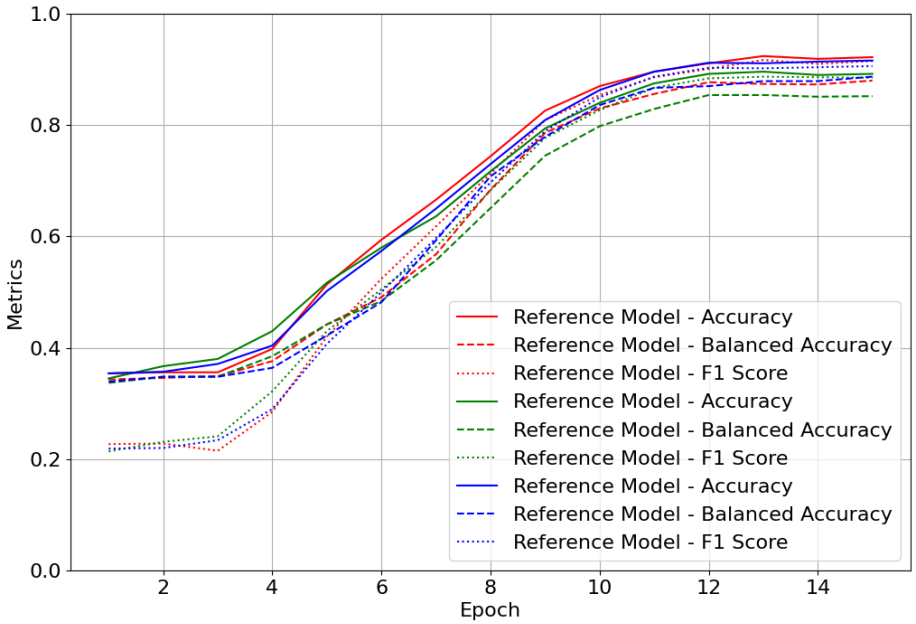
5.4.5 Wpływ losowości wag ostatniej warstwy na wyniki

W tym podrozdziale omówiony zostanie wpływ losowości początkowych wag ostatniej warstwy na wyniki osiągane przez modele w scenariuszach federacyjnego uczenia się. Losowe inicjalizowanie wag ostatniej warstwy może prowadzić do znacznych różnic w wynikach, nawet przy identycznych pozostałych parametrach modelu. Celem analizy było zbadanie, w jaki sposób losowość ta wpływa na stabilność i skuteczność modeli lokalnych.

Zakłada się, że w uczeniu federacyjnym bierze udział wiele instytucji, więc podczas badań poddano analizie wyłącznie scenariusz z 3 i 4 modelami lokalnymi, bez uwzględnienia przypadku z 2 modelami.

Analiza wyników dla 3 modeli lokalnych

Na początku poddano analizie scenariusz z 3 modelami lokalnymi.

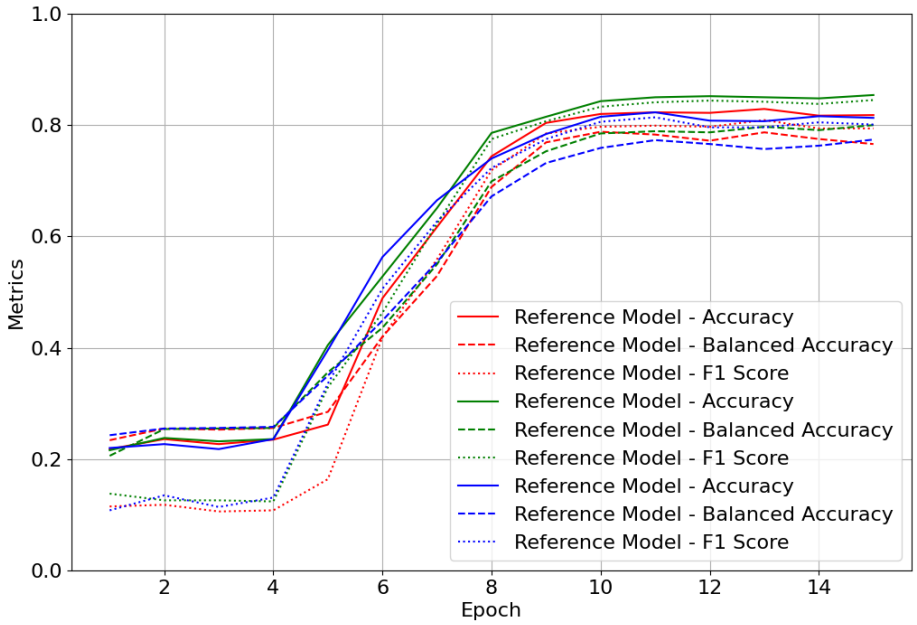


Rysunek 5.13: Przebieg metryk dla 3 modeli.

Na podstawie rysunku 5.13 można zauważyć, że w eksperymencie z 3 modelami lokalnymi, losowość wag ostatniej warstwy doprowadziła do nieznaczących różnic, wynoszących ok. 2% w wartościach metryk.

Analiza wyników dla 4 modeli lokalnych

Następnie poddano analizie scenariusz z 4 modelami lokalnymi.



Rysunek 5.14: Przebieg metryk dla 4 modeli.

Na podstawie rysunku 5.14 można zauważyć, że w przypadku 4 modeli lokalnych, wpływ losowości wag również był nieznaczny, jednak trochę większy niż w przypadku 3 modeli. Nieznacznie większa różnica jest widoczna wyłącznie w końcowej fazie treningu, gdzie różnica może sięgać ok. 4%.

5.4.6 Podsumowanie

Losowość wag ostatniej warstwy nie wpływa znacząco na uzyskiwane wyniki. Modele z czasem konwergują do podobnych wartości metryk, co wskazuje, że wpływ losowej inicjalizacji wag jest minimalny w długoterminowej perspektywie treningu. Nie ma konieczności większej liczby epok treningowych lub bardziej zaawansowanych metod inicjalizacji wag w celu stabilizacji wyników.

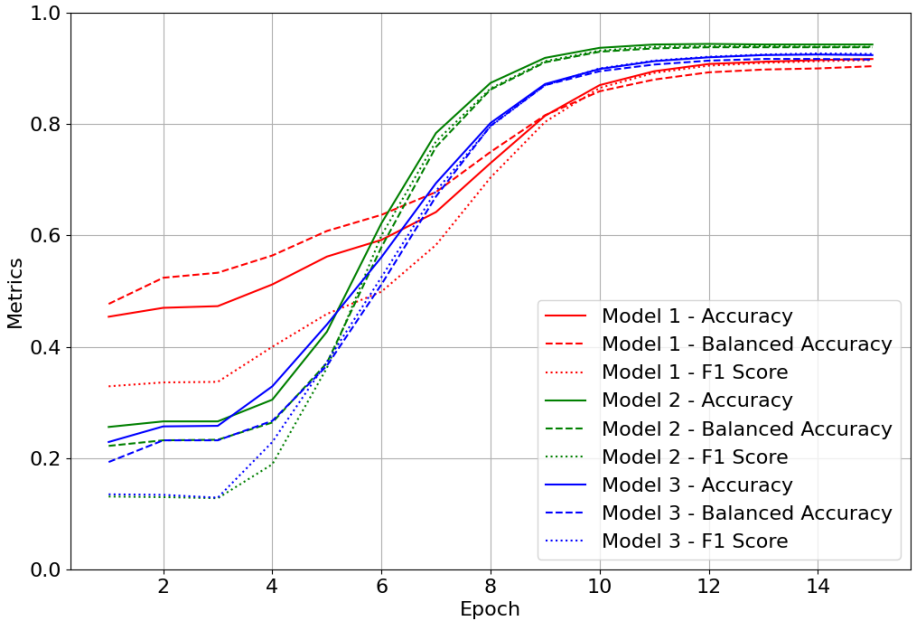
5.5 Scenariusz z malejącym współczynnikiem uczenia oraz nierównomiernie rozłożonymi etykietami danych

Do tej pory zakładano w badaniach równomiernie rozłożone etykiety danych, wraz z równą ważnością modeli lokalnych. W niniejszym podrozdziale zostanie przeprowadzona analiza wpływu nierównomiernego podziału etykiet, dla dwóch podejść: z jednakową ważnością modeli lokalnych oraz ważnością dostosowaną do liczby próbek, która jest skorelowana liniowo z liczbą obsługiwanych etykiet. W tym przypadku badania również przeprowadzono dla przypadków z 3 i 4 modelami lokalnymi.

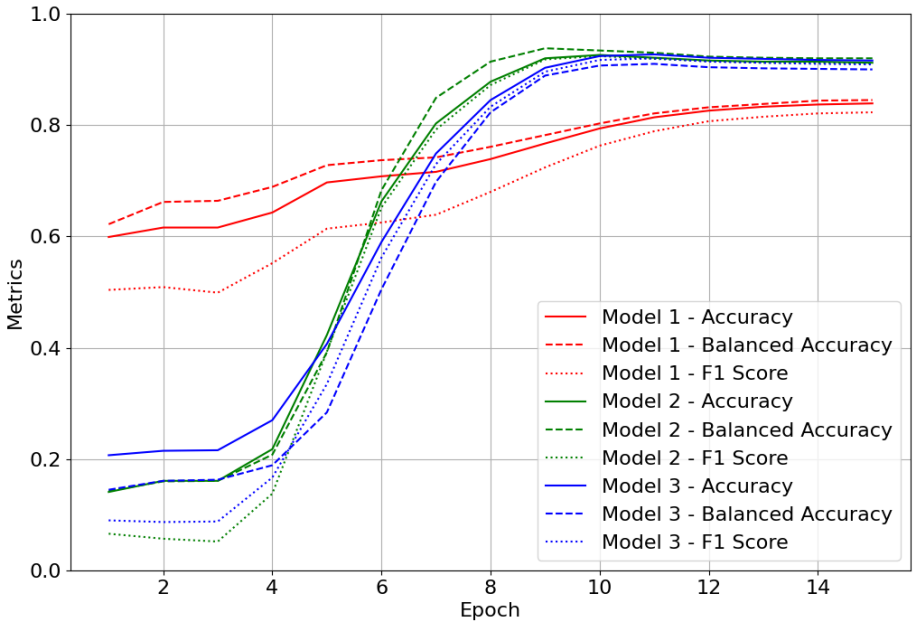
5.5.1 Analiza wyników dla 3 modeli lokalnych dla ważności modeli dostosowanej do liczby danych

W pierwszej kolejności zbadano scenariusz dla ważności modeli dostosowanej do liczby próbek, dla przypadku z 3 modelami lokalnymi oraz klasami rozłożonymi na 2 przypadki:

1. Podział etykiet w stosunku 23 : 10 : 10
2. Podział etykiet w stosunku 29 : 7 : 7



Rysunek 5.15: Przebieg metryk dla 3 modeli, malejącego współczynnika uczenia i wagami zależnymi od liczby etykiet, w stosunku 23:10:10.

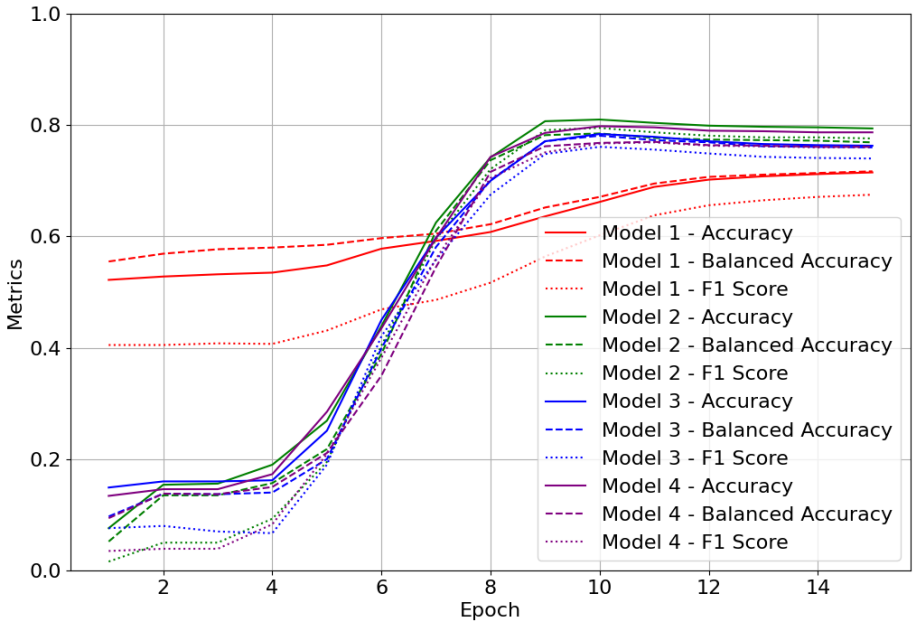


Rysunek 5.16: Przebieg metryk dla 3 modeli, malejącego współczynnika uczenia i wagami zależnymi od liczby etykiet, w stosunku 29:7:7.

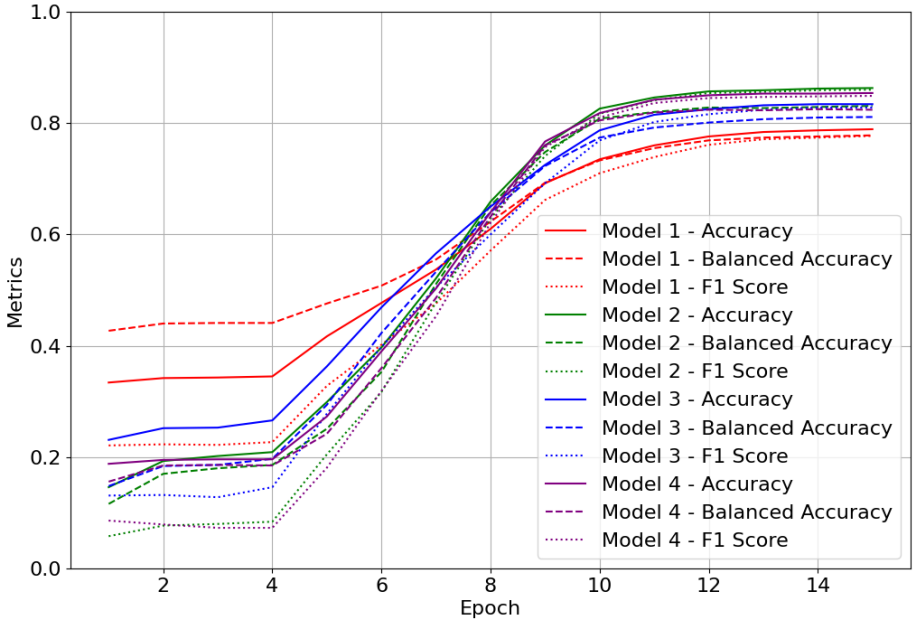
Na podstawie rysunków 5.15 oraz 5.16 można zauważyć, że dostosowanie wag modeli lokalnych do liczby dostępnych danych zauważalnie poprawiło metryki oraz przyspieszyło trening modeli zawierających mniejszą liczbę etykiet, jednak w przypadku ich większej liczby, proces treningu został spowolniony. Model mający większą liczbę danych zdominowały proces agregacji, co skutkowało dużą wiedzą przekazywaną do pozostałych modeli oraz mniejszą wiedzą pobieraną przez niego. Aby temu zaradzić, można przypisać większe wagi modelom lokalnym z mniejszą liczbą danych podczas procesu agregacji. Dzięki temu ich wyniki będą miały większy wpływ na końcowy model agregujący, co może zrównoważyć wpływ modeli dysponujących większą ilością danych.

5.5.2 Analiza wyników dla 4 modeli lokalnych dla ważności modeli dostosowanej do liczby danych

Następnie wzięto pod uwagę przypadek z 4 modelami lokalnymi.



Rysunek 5.17: Przebieg metryk dla 4 modeli, malejącego współczynnika uczenia i wagami zależnymi od liczby etykiet, w stosunku 25:6:6:6.



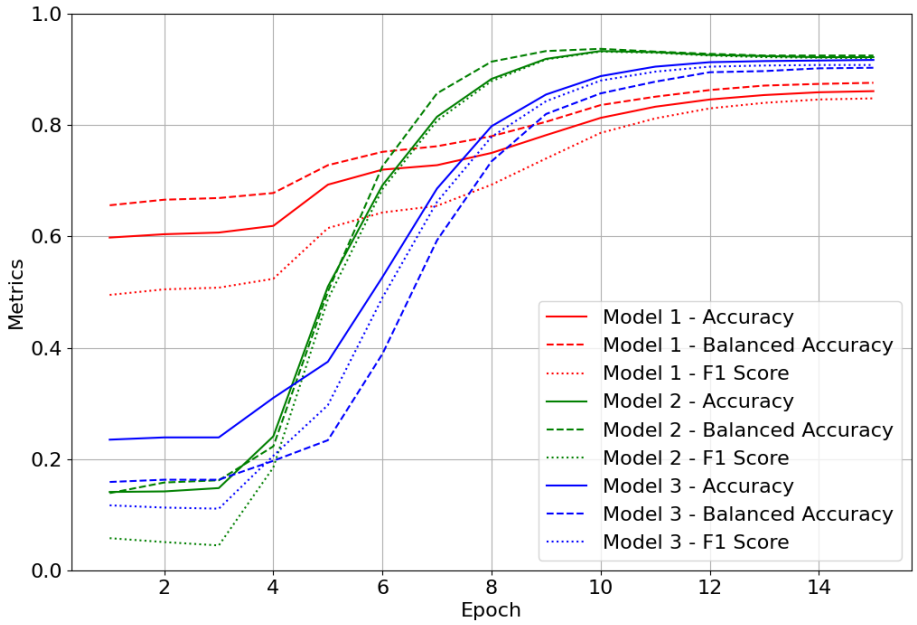
Rysunek 5.18: Przebieg metryk dla 4 modeli, malejącego współczynnika uczenia i wagami zależnymi od liczby etykiet, w stosunku 19:8:8:8.

Na podstawie rysunków 5.17 oraz 5.18 można zauważyć, że wprowadzenie 4 modeli lokalnych skutkowało podobnymi skutkami, jak przy 3 modelach. Modele z mniejszą liczbą etykiet szybciej się trenowały, zdobywając większą wiedzę od modelu z dużą liczbą danych, a w przypadku modelu z dużą liczbą danych efekt był do nich odwrotny.

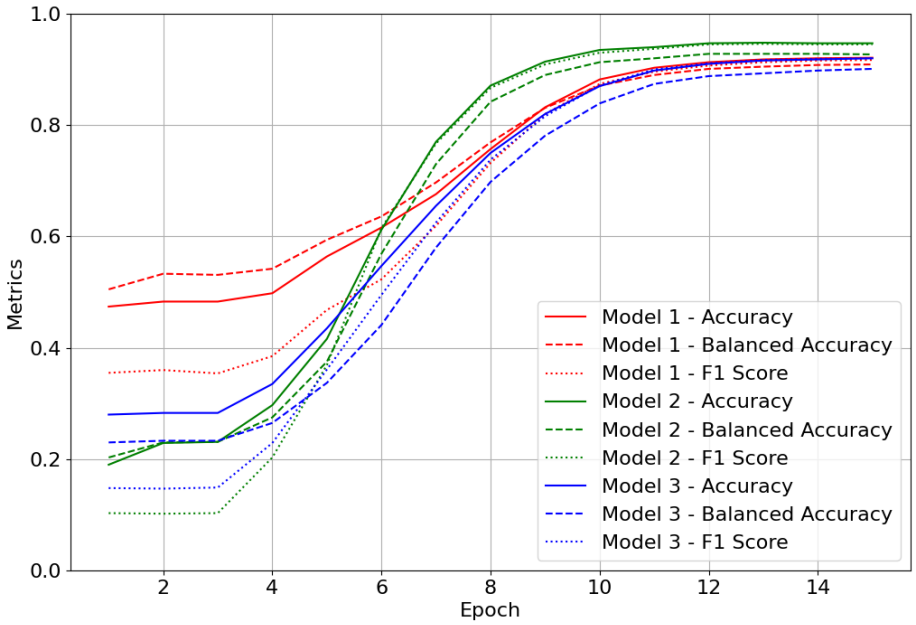
Jak widać, większa ważność modeli posiadających większą liczbę etykiet może skutkować lepszymi metrykami i szybszym uczeniem się modeli posiadających małą liczbę etykiet, jednak wolniejsze trenowanie modelu z dużą liczbą etykiet. Najprawdopodobniej jest to skutek dużego wpływu modelu zawierającego dużą liczbą etykiet. Dlatego też w następnych podrozdziałach postanowiono zrównać ważność modeli, niezależnie od liczby posiadanych etykiet.

5.5.3 Analiza wyników dla 3 modeli lokalnych dla stałej ważności modeli

W pierwszej kolejności postanowiono przeprowadzić badania dla przypadku z 3 modelami lokalnymi.



Rysunek 5.19: Przebieg metryk dla 3 modeli, malejącego współczynnika uczenia i wagami zależnymi od liczby etykiet, w stosunku 29:7:7.

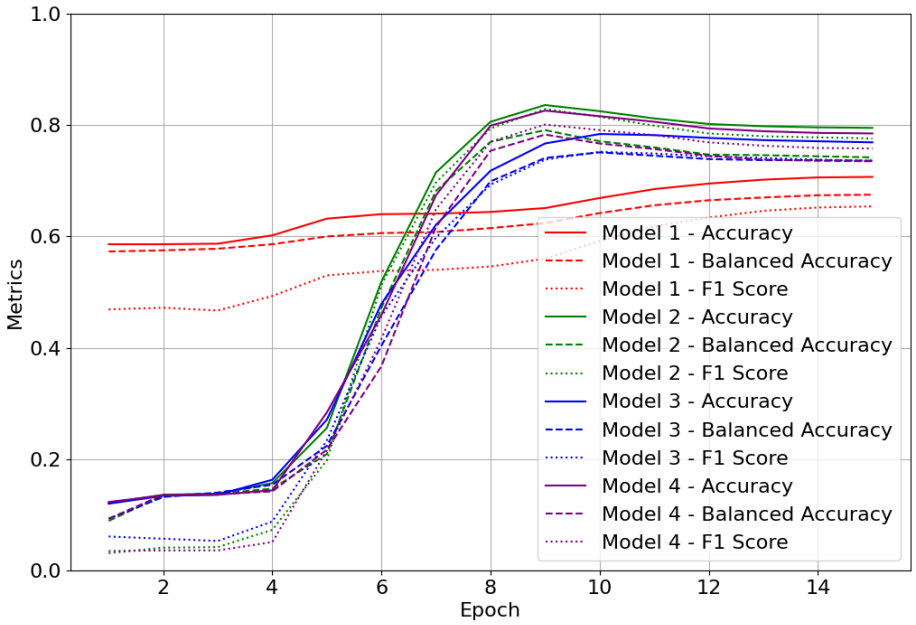


Rysunek 5.20: Przebieg metryk dla 3 modeli, malejącego współczynnika uczenia i wagami zależnymi od liczby etykiet, w stosunku 23:10:10.

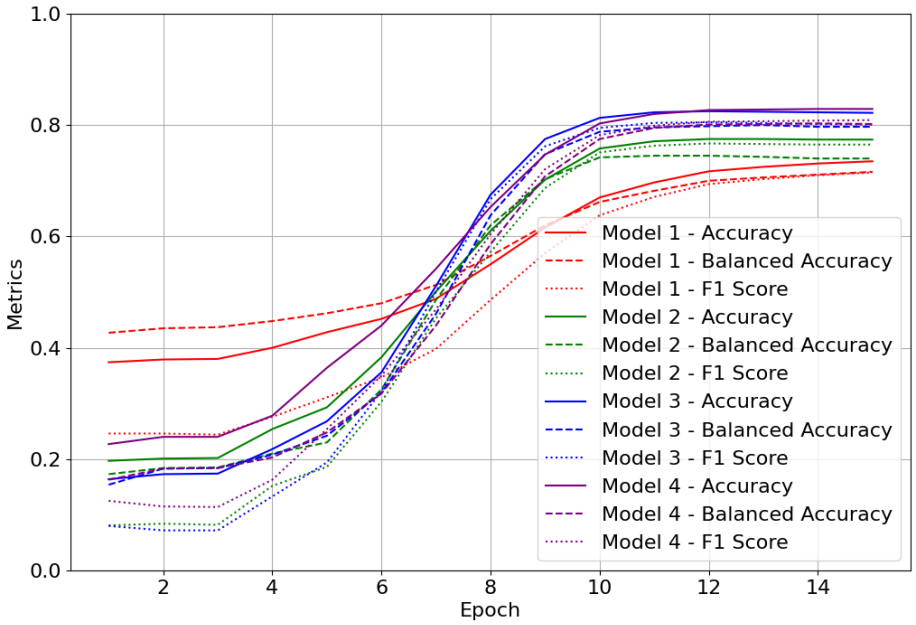
Przy stałej ważności modeli lokalnych niezależnie od liczby danych, zauważalny jest wolniejszy wzrost metryk w porównaniu do sytuacji, gdy wagi były dostosowane. Modele o mniejszej ilości danych nie były w stanie w pełni wykorzystać swojego potencjału, co wpłynęło na niższe wartości accuracy i F1 score w początkowych epokach. Mimo to, zmniejszenie współczynnika uczenia pomogło w stabilizacji wyników w późniejszych etapach treningu.

5.5.4 Analiza wyników dla 4 modeli lokalnych dla stałej ważności modeli

Podobnie jak w przypadku trzech modeli, analiza wyników dla czterech modeli lokalnych z równą wagą pokazała wyraźne różnice w jakości wyników w porównaniu do scenariuszy z dostosowaną wagą modeli.



Rysunek 5.21: Przebieg metryk dla 4 modeli, malejącego współczynnika uczenia i wagami zależnymi od liczby etykiet, w stosunku 25:6:6:6.



Rysunek 5.22: Przebieg metryk dla 4 modeli, malejącego współczynnika uczenia i wagami zależnymi od liczby etykiet, w stosunku 19:8:8:8.

W przypadku czterech modeli lokalnych o stałej ważności, metryki rosły wolniej niż w scenariuszach z wagami dostosowanymi do liczby danych. Modele z mniejszą ilością danych miały trudności z dostosowaniem się do całościowego modelu agregującego, co przełożyło się na niższe wartości metryk w początkowych epokach. Jednakże, zmniejszenie współczynnika uczenia stopniowo poprawiało stabilność i

dokładność modeli, co sugeruje, że mimo początkowych trudności, modele mogły osiągnąć akceptowalny poziom generalizacji.

5.5.5 Podsumowanie

Analiza wskazuje, że dobór strategii ważenia modeli w federacyjnym uczeniu się ma istotny wpływ na wydajność i jakość modelu agregującego. Dostosowanie wag do liczby danych może poprawić wyniki, ale może również spowodować dominację modeli z większą ilością danych. Zrównoważenie tego efektu wymaga zastosowania odpowiednich technik, takich jak dynamiczne ważenie lub regulacja współczynnika uczenia, aby uzyskać optymalną wydajność w różnych scenariuszach.

5.6 Wpływ grupowania etykiet próbek na wyniki

Celem tego rozdziału jest zbadanie, w jaki sposób różne strategie grupowania etykiet próbek mogą wpływać na skuteczność lokalnym modeli w uczeniu federacyjnym, zwłaszcza w kontekście różnorodności danych i nierównomierności w liczbie przykładów należących do poszczególnych klas. Przeanalizowano tutaj różne scenariusze, w których etykiety zostały podzielone na grupy na podstawie wspólnych cech semantycznych, takich jak typy znaków drogowych. Wyniki tych eksperymentów dostarczą wglądu w to, jak najlepiej dostosować proces trenowania modeli federacyjnych do rzeczywistych warunków operacyjnych, gdzie dane są często niespójnie rozproszone pomiędzy różnymi źródłami.

W tabeli 5.1 przedstawiono opis klas znaków drogowych.

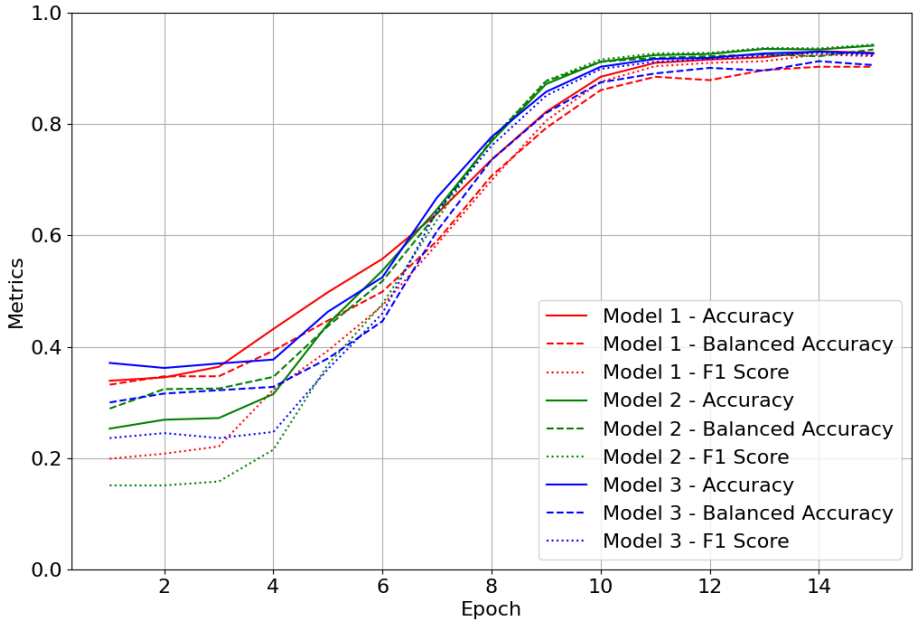
id	nazwa znaku
0	Ograniczenie prędkości (20km/h)
1	Ograniczenie prędkości (30km/h)
2	Ograniczenie prędkości (50km/h)
3	Ograniczenie prędkości (60km/h)
4	Ograniczenie prędkości (70km/h)
5	Ograniczenie prędkości (80km/h)
6	Koniec ograniczenia prędkości (80km/h)
7	Ograniczenie prędkości (100km/h)
8	Ograniczenie prędkości (120km/h)
9	Zakaz wyprzedzania
10	Zakaz wyprzedzania dla pojazdów powyżej 3,5 t
11	Pierwszeństwo na następnym skrzyżowaniu
12	Droga z pierwszeństwem
13	Ustąp pierwszeństwa
14	Stop
15	Zakaz ruchu
16	Zakaz wjazdu pojazdów powyżej 3,5 t
17	Zakaz wjazdu
18	Uwaga! Ogólne zagrożenie
19	Niebezpieczny zakręt w lewo
20	Niebezpieczny zakręt w prawo
21	Niebezpieczne zakręty
22	Wyboista droga
23	Śliska droga
24	Zwężenie drogi z prawej strony
25	Roboty drogowe
26	Sygnalizacja świetlna
27	Przejście dla pieszych
28	Przejście dla dzieci
29	Przejazd dla rowerzystów
30	Uwaga na lód/śnieg
31	Przejście dzikich zwierząt
32	Koniec wszelkich ograniczeń prędkości i zakazów wyprzedzania
33	Nakaz skrętu w prawo
34	Nakaz skrętu w lewo
35	Nakaz jazdy prosto
36	Nakaz jazdy prosto lub w prawo
37	Nakaz jazdy prosto lub w lewo
38	Nakaz trzymania się prawej strony
39	Nakaz trzymania się lewej strony
40	Nakaz jazdy okrężnej (rondo)
41	Koniec zakazu wyprzedzania
42	Koniec zakazu wyprzedzania dla pojazdów powyżej 3,5 t

Tabela 5.1: Tabela nazw klas (znaków drogowych) w wykorzystanym zbiorze danych

5.6.1 Analiza wyników dla 3 modeli lokalnych

W tej sekcji przeanalizowano skuteczność federacyjnego uczenia się z 3 modelami lokalnymi, gdzie etykiety danych podzielono na trzy grupy:

1. **Znaki ograniczenia prędkości i zakazu wyprzedzania:**
   - Klasa: 0-8 (Ograniczenia prędkości)
   - Klasa: 9-10 (Zakaz wyprzedzania)
   - Klasa: 41-42 (Koniec zakazu wyprzedzania)
2. **Znaki zakazu, nakazu i ostrzegawcze:**
   - Klasa: 11-17 (Pierwszeństwo, zakazy wjazdu itp.)
   - Klasa: 18-31 (Ostrzeżenia, np. niebezpieczne zakręty, śliska droga)
3. **Znaki kierunku jazdy i inne:**
   - Klasa: 32 (Koniec ograniczeń prędkości i zakazów wyprzedzania)
   - Klasa: 33-40 (Kierunki jazdy, rondo itp.)



Rysunek 5.23: Przebieg metryk dla 3 modeli, z pogrupowanymi etykietami.

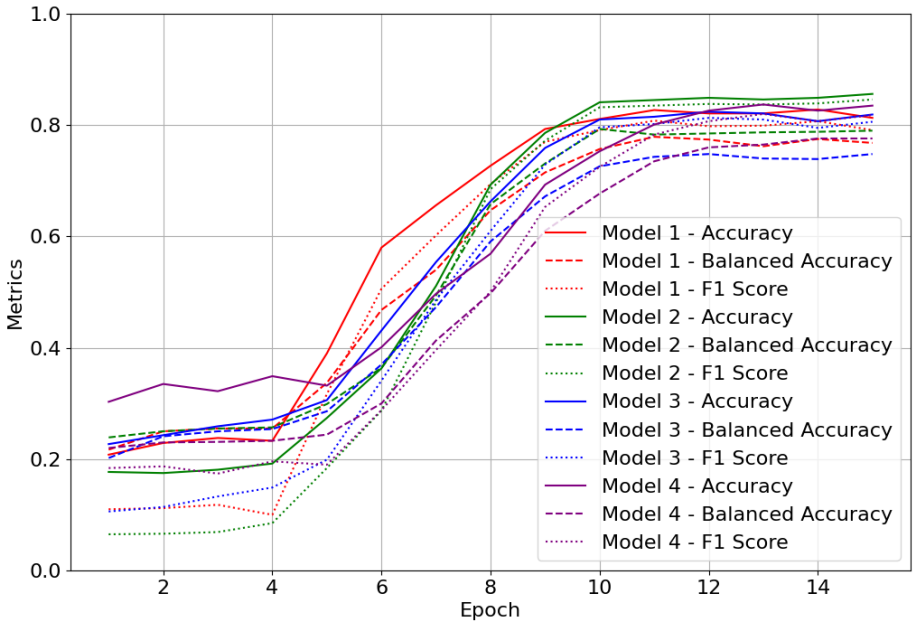
Na podstawie rysunku 5.23 można zauważyć, że podział etykiet na grupy pozwolił na lepsze dopasowanie modeli do specyficznych kategorii znaków, co przełożyło się na ich większą dokładność i stabilność w rozpoznawaniu, podwyższając uzyskane metryki o ok. 0.02. Wyniki wskazują, że każdy z modeli skutecznie nauczył się rozpoznawać swoją grupę znaków oraz uzyskać wiedzę z innych modeli lokalnych, co potwierdza efektywność zastosowanego podejścia.

5.6.2 Analiza wyników dla 4 modeli lokalnych

Następnie przeprowadzono badania dla podziału etykiet na 4 grupy. Podział wyglądał następująco:

1. **Znaki ograniczenia prędkości:**
   - Klasa: 0-8 (Ograniczenia prędkości)
2. **Znaki zakazu:**
   - Klasa: 9-10 (Zakaz wyprzedzania)
   - Klasa: 15-17 (Zakaz ruchu, Zakaz wjazdu)
   - Klasa: 16 (Zakaz wjazdu pojazdów powyżej 3,5 tony)
3. **Znaki nakazu i kierunku jazdy:**
   - Klasa: 33-40 (Kierunki jazdy, rondo itp.)
4. **Znaki ostrzegawcze i informacyjne:**
   - Klasa: 11-14 (Pierwszeństwo, ustąp pierwszeństwa, stop)
   - Klasa: 18-31 (Ostrzeżenia, np. niebezpieczne zakręty, śliska droga)
   - Klasa: 32 (Koniec wszelkich ograniczeń prędkości i zakazów wyprzedzania)
   - Klasa: 41-42 (Koniec zakazu wyprzedzania)

Uzyskane wyniki są zaprezentowane na rysunku poniżej.



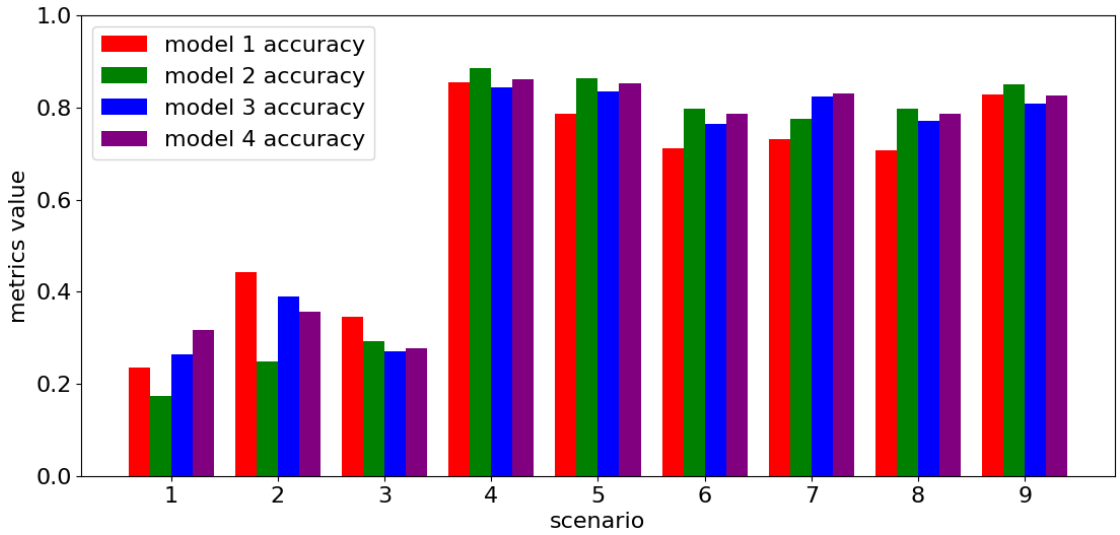
Rysunek 5.24: Przebieg metryk dla 4 modeli, z pogrupowanymi etykietami.

Na podstawie rysunku 5.24 można zauważyć, że podział etykiet na grupy nie zmienił znacząco metryk uzyskanych przez modele lokalne. Może to wynikać z nieoptymalnego pogrupowania etykiet lub z faktu, że dla dużej liczby modeli lokalnych to podejście nie wpływa znacząco na wyniki.

5.7 Wnioski

Przeprowadzone badania podkreśliły kluczową rolę adaptacyjnego (malejącego) współczynnika uczenia w procesie federacyjnego uczenia się. Początkowo wysoka wartość tego parametru umożliwia szybkie przekazywanie wiedzy między modelami lokalnymi, co sprzyja efektywnej synchronizacji i agregacji. W miarę postępu treningu, stopniowe obniżanie współczynnika uczenia pomaga zapobiegać problemowi przeuczenia na danych lokalnych, co zapewnia stabilność modeli i lepszą generalizację do nowych danych. Dzięki takiemu podejściu możliwe jest uzyskanie bardziej spójnych i efektywnych wyników.

Na rysunku 5.25 przedstawiono metryki dokładności modeli lokalnych po 14 epokach, ze wszystkich scenariuszy badań przeprowadzonych w tym rozdziale. Pod rysunkiem znajduje się opis danego numeru scenariusza oraz tabela 5.2 przedstawiająca dokładne wartości dokładności dla modeli po 14 epoce w badanych scenariuszach.



Rysunek 5.25: Wyniki metryk poszczególnych scenariuszy badań z 4 modelami lokalnymi.

1. Scenariusz ze współczynnikiem uczenia równym 0.001 (rozdział 5.3).
2. Scenariusz ze współczynnikiem uczenia równym 0.0001 (rozdział 5.3).
3. Scenariusz ze współczynnikiem uczenia równym 0.00001 (rozdział 5.3).
4. Scenariusz z malejącym współczynnikiem uczenia, nierównomiernie rozłożonymi etykietami (w stosunku 19:8:8:8) oraz z ważnością modeli dostosowanymi do liczby próbek (rozdział 5.5).
5. Scenariusz z malejącym współczynnikiem uczenia, nierównomiernie rozłożonymi etykietami (w stosunku 25:6:6:6) oraz z ważnością modeli dostosowanymi do liczby próbek (rozdział 5.5).
6. Scenariusz z malejącym współczynnikiem uczenia, nierównomiernie rozłożonymi etykietami (w stosunku 19:8:8:8) oraz ze stałą ważnością modeli (rozdział 5.5).
7. Scenariusz z malejącym współczynnikiem uczenia, nierównomiernie rozłożonymi etykietami (w stosunku 25:6:6:6) oraz ze stałą ważnością modeli (rozdział 5.5).
8. Scenariusz z malejącym współczynnikiem uczenia, nierównomiernie rozłożonymi etykietami (w stosunku 19:8:8:8) oraz ze stałą ważnością modeli (rozdział

5.5).

9. Scenariusz z malejącym współczynnikiem uczenia, pogrupowane etykietami oraz ze stałą ważnością modeli (rozdział 5.6).

Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
model 1	0.236	0.442	0.345	0.855	0.787	0.712	0.731	0.706	0.828
model 2	0.174	0.248	0.292	0.885	0.862	0.796	0.774	0.796	0.849
model 3	0.265	0.390	0.270	0.843	0.834	0.764	0.823	0.771	0.807
model 4	0.318	0.356	0.278	0.861	0.853	0.787	0.829	0.786	0.826

Tabela 5.2: Wyniki dokładności dla modeli w badanych scenariuszach po 14 epokach.

Wyniki uzyskane przez modele lokalne w procesie federacyjnego uczenia się nie dorównują rezultatom modelu referencyjnego. Prawdopodobnie jest to spowodowane faktem, że każdy model lokalny uczy się na swoich unikalnych danych, co prowadzi do dostrajania parametrów w kierunku optymalnym jedynie dla jego własnego zestawu danych. Proces agregacji uśrednia te parametry, co uniemożliwia znalezienie bardzo niskiego minimum globalnej funkcji straty. W efekcie model agregujący zostaje dostrojony w regionie przestrzeni parametrów, który jest akceptowalny dla wszystkich modeli lokalnych, ale nie optymalny dla żadnego z nich.

Rozdział 6

Badanie wpływu konfiguracji infrastruktury na wybrany scenariusz uczenia federacyjnego

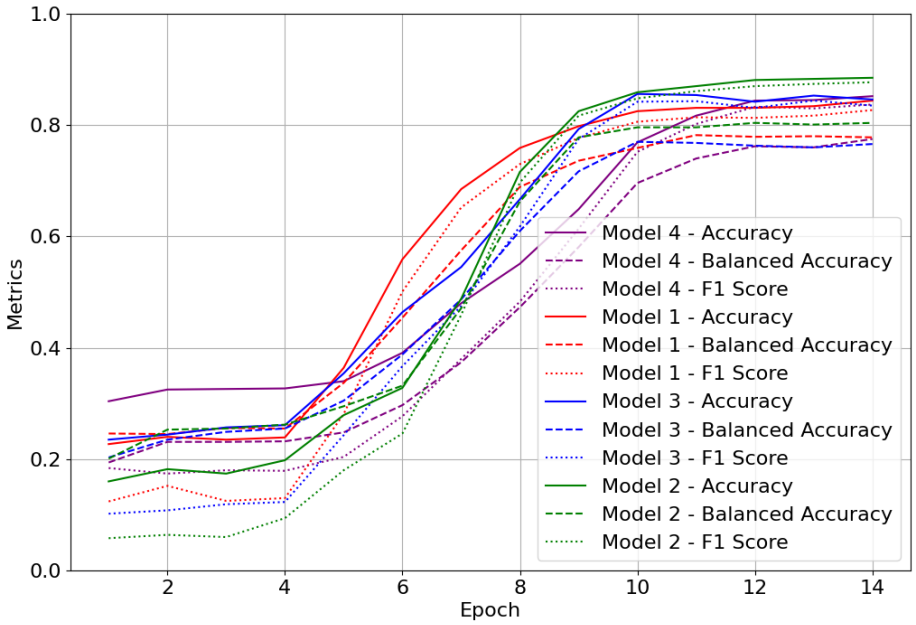
P. Mazur

W niniejszym rozdziale zostanie omówiony wpływ wybranych parametrów konfiguracyjnych infrastruktury chmurowej na przebieg oraz wynik uczenia federacyjnego. Dokonana zostanie optymalizacja kosztowa uczenia federacyjnego. Na podstawie dobranych wcześniej parametrów wykonana zostanie analiza kosztowa zestawiająca koszt uczenia federacyjnego, wraz z uśrednioną dokładnością poszczególnych modeli.

6.1 Wybrany scenariusz uczenia federacyjnego oraz jego parametry

Podczas wszystkich eksperymentów w niniejszym rozdziale wykorzystano stałe parametry związane z uczeniem federacyjnym. Wykorzystano agregację 4 modeli lokalnych oraz malejący learning rate, zaczynając od wartości równej **0.001**. Dla tej wartości odnotowano najlepsze wyniki metryk w poprzednim rozdziale. Uczenie przeprowadzono dla 14 epok. W poprzednim rozdziale najniższe wartości badanych metryk otrzymano dla 4 modeli. Pomimo tego, zdecydowano się na tę ilość ze względu na możliwość przeprowadzenia szerszego zakresu eksperymentów.

Każdy model lokalny trenowany jest na dedykowanym klastrze, z dostępem do danych uczących oraz połączony z klastrem centralnym. W przedstawionych wynikach zapis "model 1"oznaczać będzie eksperyment przeprowadzony na klastrze, na którym przeprowadzany jest trening wersji modelu pierwszego.



Rysunek 6.1: Metryki modeli dla przebiegu referencyjnego.

Na rysunku 6.1 przedstawiono przebieg metryk modeli otrzymanych dla parametrów opisanych w niniejszym rozdziale. Wartości te zostaną użyte w podrozdziale poświęconym analizie kosztowej.

6.2 Wpływ rodzaju nośnika danych na proces uczenia

Dostawcy chmurowi oferują różnego rodzaju dyski, na których przechowywane są podczas procesu uczenia dane uczące, kontrolne oraz generowane modele. Autorzy [25] podkreślają w swojej pracy znaczenie pamięci lokalnej dla skonteneryzowanych aplikacji, dlatego wybór odpowiedniego rodzaju pamięci masowej może mieć znaczny wpływ na działanie aplikacji. W większości przypadków dostępny jest wybór pomiędzy dyskiem HDD oraz SSD. Dyski SSD charakteryzuje się większymi prędkościami zapisu i odczytu, jednak wiąże się to z większymi kosztami.

Podczas procesu uczenia modelu lokalnego wykorzystywane są 3 woluminy przechowujące dane uczące, modele lokalne oraz metryki. Przeprowadzono, wykorzystując tylko dyski HDD, następnie tylko dyski SSD. Najdłużej i najczęściej wykorzystywany jest wolumin z zapisanymi danymi uczącymi. To na nim następuje początkowy zapis danych uczących i późniejsze ładowanie danych podczas treningu. Z tego powodu wykonano również trzecie badanie, w którym wykorzystano dyski SSD tylko dla woluminu zawierające dane uczące, pozostałe pozostawiono z dyskami HDD.

Wyniki eksperymentu przedstawione w 6.1 pokazują, że użycie dysków SSD znacząco skraca czas treningu w porównaniu do dysków HDD. Jest to szczególnie istotne w kontekście odczytu i zapisu danych uczących, które są najczęściej używane podczas treningu. Dyski SSD, dzięki szybszym operacjom I/O, poprawiają ogólną wydajność procesu.

Rozwiązanie hybrydowe, gdzie tylko dane uczące są na dyku SSD, jest niemal tak samo szybkie jak pełne rozwiązanie SSD. Średni czas różni się zaledwie o minutę,

Tabela 6.1: Czas eksperymentu dla różnych wersji nośnika danych.

Nośnik	model 1	model 2	model 3	model 4	Średnia
HDD	02:36:22	02:36:38	02:36:40	02:36:30	02:36:30
SSD	01:55:33	01:55:30	01:55:45	01:55:33	01:55:33
SSD i HDD	01:56:07	01:56:15	01:56:12	01:56:29	01:56:24

co sugeruje, że głównym wąskim gardłem w przypadku dysków HDD są operacje na danych uczących. Użycie SSD tylko dla tego woluminu pozwala zatem na osiągnięcie znacznych oszczędności czasowych przy jednoczesnym ograniczeniu kosztów.

6.3 Wpływ rodzaju maszyny wirtualnej

Istotny wpływ na czas treningu ma dostępność zasobów obliczeniowych. Wraz ze wzrostem liczby rdzeni zwiększa się prędkość obliczeń, dzięki możliwości przetwarzania równoległego.

Wykonano badania dla 3 rodzajów maszyn wirtualnych, charakteryzujących się poniższymi parametrami:

- **small** - 4 vCPU i 16 GB RAM
- **medium** - 8 vCPU i 32 GB RAM
- **large** - 16 vCPU i 64 GB RAM

Proporcje ilości vCPU do pamięci RAM są wielkościami stałymi, wyznaczonymi przez dostawcę chmury. Wybierane są zazwyczaj na podstawie wiedzy eksperckiej. Często spotykane są różne konfiguracje maszyn, w zależności od ich przeznaczenia. Wyniki eksperymentów przedstawiono w tabeli 6.2. Warunkiem kończącym pomiar czasu dla danego modelu jest ukończenie treningu zadanej liczby epok.

Tabela 6.2: Czas eksperymentu dla różnych wielkości maszyn wirtualnych.

Rodzaj maszyny	model 1	model 2	model 3	model 4	Średnia
small	02:28:12	02:28:38	02:28:40	02:28:30	02:28:29
medium	02:15:33	02:15:30	02:15:45	02:15:33	02:15:29
large	01:50:07	01:50:15	01:50:12	01:50:29	01:50:13

Wyniki przedstawione w tabeli 6.2 jednoznacznie pokazują, że zwiększenie wielkości maszyn wirtualnych prowadzi do skrócenia czasu treningu. To wskazuje na efektywne skalowanie wydajności w zależności od dostępnych zasobów obliczeniowych. Większe maszyny są w stanie szybciej przetwarzać zadania uczenia maszynowego, co jest zgodne z oczekiwaniami. Dwukrotne zwiększenie mocy obliczeniowej nie prowadzi jednak do dwukrotnego skrócenia czasu oczekiwania. Wynika to z faktu, że poza samym treningiem w każdej epoce następuje ładowanie danych do pamięci RAM, co jest procesem zależnym głównie od rodzaju nośnika, na którym znajdują się dane do treningu.

Choć większe maszyny **large** oferują najwyższą wydajność, warto również rozważyć koszt takiego rozwiązania w kontekście specyficznych wymagań biznesowych. Dla zadań, gdzie czas nie jest krytyczny, maszyny **small** mogą być bardziej opłacalnym rozwiązaniem, mimo dłuższego czasu treningu.

6.4 Analiza kosztowa

W niniejszym rozdziale przedstawiona została analiza kosztów związanych z wykorzystaniem różnych konfiguracji maszyn wirtualnych oraz dysków do przechowywania danych w ramach przeprowadzanych eksperymentów. Celem analizy jest ocena efektywności kosztowej zastosowanych rozwiązań oraz ich wpływ na czas przeprowadzanych eksperymentów. Zestawiony zostanie również koszt otrzymania określonych dokładności modeli lokalnych.

6.4.1 Koszty wykorzystania maszyn wirtualnych

Aby obliczyć koszt samych maszyn wirtualnych (VM) dla poszczególnych konfiguracji, pomnożymy średni czas trwania eksperymentów w godzinach przez odpowiednie stawki godzinowe.

Na podstawie cennika dostawcy chmury publicznej opartej o Openstack [26] zebrano poniższe **stawki godzinowe za użycie VM:**

- small: 0.186 EUR/godz.
- medium: 0.372 EUR/godz.
- large: 0.744 EUR/godz.

Przeliczenie czasów na godziny i obliczenie kosztów:

1. **Konfiguracja small:**
   - Średni czas: 02:58:29
   - Czas w godzinach:
   $$2 + \frac{58}{60} + \frac{29}{3600} = 2{,}9747 \text{ godziny}$$
   - Koszt:
   $$2{,}9747 \text{ godziny} \times 0{,}186 \text{ EUR/godz.} = 0{,}5533 \text{ EUR}$$
2. **Konfiguracja medium:**
   - Średni czas: 02:15:29
   - Czas w godzinach: 2,2581 godziny
   - Koszt: 0,8400 EUR
3. **Konfiguracja large:**
   - Średni czas: 01:38:13
   - Czas w godzinach: 1,6369 godziny
   - Koszt: 1,2179 EUR

Podsumowanie kosztów samych VM:

- small: 0,5533 EUR
- medium: 0,8400 EUR
- large: 1,2179 EUR

Koszty te pokazują, że bardziej zaawansowane i większe maszyny wirtualne, takie jak konfiguracje medium i large, są droższe, ale mogą zakończyć trening szybciej, co może być korzystne w zależności od wymagań biznesowych.

6.4.2 Koszty związane z przechowywaniem danych

Na podstawie cennika tego samego dostawcy, co w przypadku maszyn wirtualnych - [27] zebrano poniższe **stawki godzinowe za użycie dysków:**

- SSD: 0,0001389 EUR/godz./GB
- HDD: 0,0000556 EUR/godz./GB

Przeliczenie czasów na godziny i obliczenie kosztów dysków:

1. **Konfiguracja HDD:**
   - Średni czas: 02:36:30
   - Czas w godzinach: 2,6083 godziny
   - Koszt dysków: 0,000435 EUR
2. **Konfiguracja SSD:**
   - Średni czas: 01:55:33
   - Czas w godzinach: 1,9258 godziny
   - Koszt dysków: 0,000803 EUR
3. **Konfiguracja hybrydowa (SSD dla danych uczących):**
   - Średni czas: 01:56:24
   - Czas w godzinach: 1,9400 godziny
   - Koszt dysków:
     - SSD (1 GB dla danych uczących): 0,000269 EUR
     - HDD (2 GB dla pozostałych woluminów): 0,000216 EUR
   - Łączny koszt dysków: 0,000485 EUR

Podsumowanie kosztów dysków:

- HDD: 0,000435 EUR
- SSD: 0,000803 EUR
- Hybrydowe (SSD dla danych uczących): 0,000485 EUR

Koszty dysków pokazują, że wykorzystanie dysków SSD jest droższe w porównaniu do HDD. W rozwiązaniu hybrydowym, gdzie tylko dane uczące są na SSD, a pozostałe woluminy są na HDD, koszt dysków jest niższy niż pełne rozwiązanie SSD, ale nieznacznie wyższy niż pełne rozwiązanie HDD. Biorąc pod uwagę niewielką różnicę w cenie, a znaczący wpływ na prędkość obliczeń, rekomendowane jest używanie wyższej jakości dysków pod przechowywanie danych.

6.4.3 Zestawienie kosztów uczenia federacyjnego z dokładnością modeli

Podczas uczenia maszynowego dążymy do osiągnięcia jak najlepszych metryk dla modeli. W zależności od kontekstu biznesowego, koszty poniesione na przeprowadzenie obliczeń mogą stanowić ważny kontekst. Postanowiono przedstawić koszty osiągania poszczególnych dokładności modeli. W tym celu szczegółowo zebrano czasy, po których zakończono trening konkretnych epok modeli.

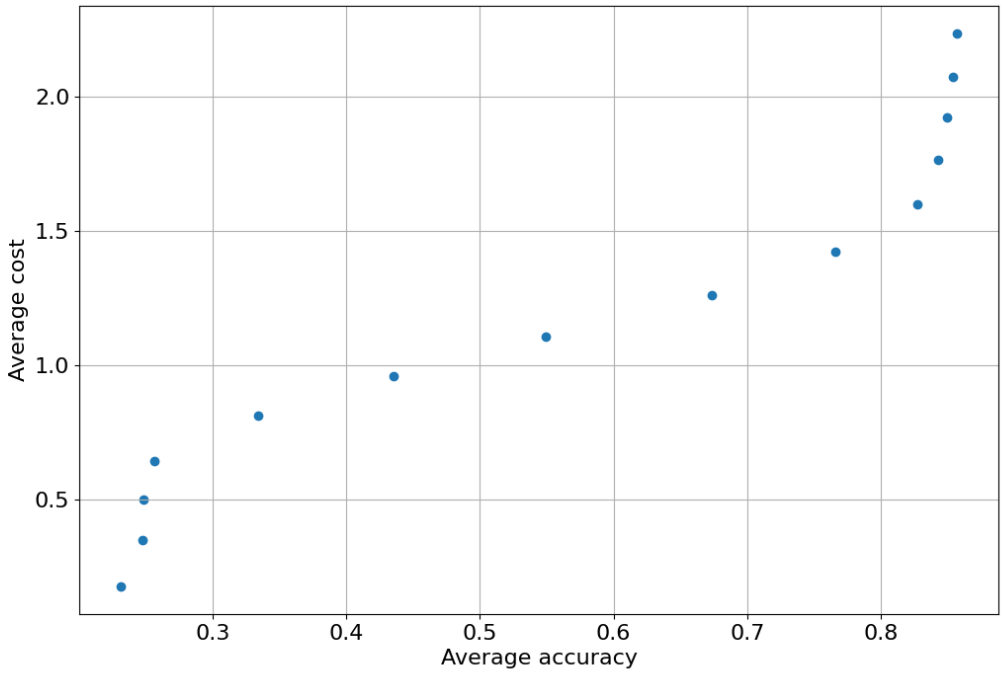
Epoka	model 1	model 2	model 3	model 4	Średni czas
1	00:07:25	00:08:43	00:09:06	00:09:57	00:08:48
2	00:17:04	00:16:35	00:18:12	00:19:28	00:17:20
3	00:24:02	00:23:44	00:25:00	00:26:19	00:24:46
4	00:30:50	00:31:25	00:31:58	00:33:07	00:31:50
5	00:38:58	00:39:44	00:40:17	00:41:28	00:40:12
6	00:46:37	00:46:11	00:47:45	00:48:58	00:47:23
7	00:54:23	00:53:10	00:54:46	00:55:59	00:54:34
8	01:01:16	01:01:00	01:02:26	01:03:38	01:02:05
9	01:08:46	01:09:19	01:10:55	01:12:03	01:10:16
10	01:19:11	01:17:10	01:18:47	01:20:00	01:18:47
11	01:25:58	01:25:39	01:28:07	01:28:19	01:27:01
12	01:34:57	01:33:39	01:35:07	01:36:19	01:35:00
13	01:41:45	01:41:30	01:43:42	01:44:07	01:42:46
14	01:49:35	01:49:10	01:50:49	01:51:57	01:50:23

Tabela 6.3: Czasy zakończenia treningów konkretnych epok wraz z uśrednieniem.

Na podstawie średnich czasów treningów poszczególnych epok z tabeli 6.3 obliczono średni koszt obliczeń. Posłużono się kosztem dla konfiguracji z wirtualnymi maszynami typu large oraz konfiguracją hybrydową dysków. Łączny koszt za godzinę pracy w tej konfiguracji wynosi:

$$0{,}000485 + 1{,}2179 = 1,218385 \text{ EUR/godz}$$

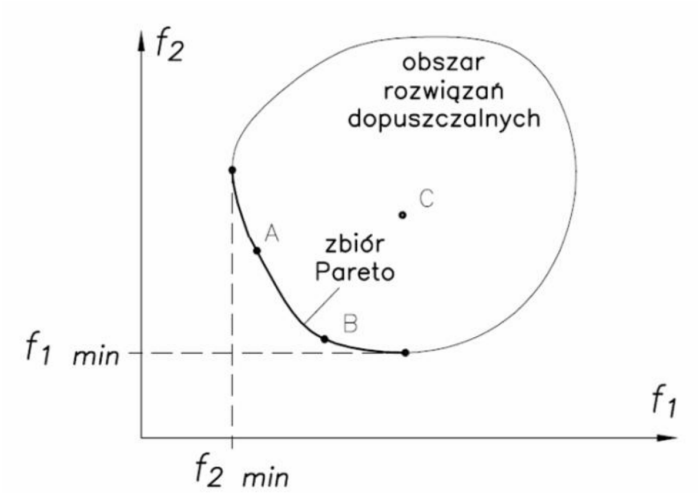
Dla tej stawki godzinowej obliczono średni koszt treningu konkretnych epok modeli. Posiadając uśrednione koszty otrzymania konkretnych epok podczas uczenia dokonano zestawienia tych wartości ze średnią dokładnością. Metryka została obliczona na podstawie przebiegu 6.1 z rozdziału 6.1.



Rysunek 6.2: Zestawienie średniego kosztu do średniej dokładności.

Zgodnie z rysunkiem 6.2 można zauważyć, że wzrost kosztów nie jest liniowy. Początkowo, przy niskich dokładnościach, koszty rosną powoli, ale wraz z rosnącą dokładnością koszty zaczynają rosnąć coraz szybciej. Początek obliczeń to moment, w którym niezauważalny jest wysoki wzrost dokładności. Dopiero w środkowej fazie dokładność zaczyna znacząco rosnąć, sprawiając, że wzrost jakości modeli zaczyna być proporcjonalny do ponoszonych kosztów. W końcowej fazie eksperymentu koszty zaczynają rosnąć wykładniczo w porównaniu do otrzymywanej dokładności. Uzyskanie bardzo wysokiej dokładności jest również bardzo kosztowne. W zależności od konkretnej potrzeby biznesowej może okazać się, że wcześniejsze zakończenie procesu uczenia i wykorzystanie modeli wystarczająco dobrych przyniesie oszczędności rzędu kilkudziesięciu procent, przy kilkuprocentowej stracie na dokładności.

W realnych zastosowaniach organizacje muszą znaleźć optymalny punkt, w którym dokładność modelu jest wystarczająco wysoka, ale koszty są nadal akceptowalne. Jedną z metod wspomagania takiej decyzji jest znalezienie rozwiązania optymalnego w sensie Pareto, występujące gdy nie jest możliwe znalezienie rozwiązania lepszego z uwagi na co najmniej jedno kryterium bez pogorszenia z uwagi na pozostałe. W przypadku punktów z rysunku 6.2 możemy zauważyć, że każdy punkt należy do takich rozwiązań. Po przeprowadzenia większej liczby eksperymentów, np. dla różnych dostawców chmurowych, możliwe byłoby wybranie zbioru Pareto, spośród otrzymanych rozwiązań. Tworzenie tego zbioru dla zadania minimalizacji przedstawiono na rysunku 6.3.



Rysunek 6.3: Wybór zbioru Pareto [28].

Z uwagi na otrzymywanie wielu rozwiązań optymalnych w sensie Pareto wykorzystywane są dodatkowe metody umożliwiające podjęcie odpowiedniej decyzji. Możliwe jest wprowadzenie dodatkowego kryterium, będącego sumą ważoną wszystkich kryteriów, umożliwiając wybór rozwiązania. Innym podejściem, również prostym w implementacji jest ograniczenie kryteriów. W rzeczywistych zastosowaniach naturalną granicą jest wyznaczenie maksymalnego kosztu, dla którego trenowanie modeli będzie mieściło się w zaplanowanym budżecie. W zależności od kontekstu biznesowego możliwy jest również wybór minimalnej dokładności jaką model powinien osiągać. Przykładowo dla granicy akceptowalnej dokładności równej przynajmniej 0.8, na podstawie rysunku 6.2 wynika, że optymalnym rozwiązaniem jest trening 10 epok, gdzie dokładność wynosi 0.83, przy koszcie równym 1.6.

6.5 Podsumowanie

W niniejszym rozdziale zbadano wpływ podstawowych parametrów konfiguracji sprzętowej na czas procesu uczenia federacyjnego. Zastosowanie pamięci masowej lepszej jakości ma znaczący wpływ na przebieg eksperymentów. Wynika to z konieczności ładowania danych uczących przy każdej epoce. Zwiększenie mocy obliczeniowej również ma wpływ na czas trwania procesu, jest on skalowalny, jednak ze względu na wspomnianą wcześniej specyfikę działania treningu korzyść płynąca z większej ilości vCPU jest mniej zauważalna. Przeprowadzono również analizę kosztową, pozwalającą na zrozumienie kompromisów między kosztami a dokładnością oraz pomagającą w optymalizacji kosztowej procesów związanych z trenowaniem modeli.

Rozdział 7

Badanie wpływu konfiguracji infrastruktury na asynchroniczne uczenie federacyjne

W niniejszym rozdziale podjęto próbę analizy wpływu różnorodności zasobów w klastrach na wyniki federacyjnego uczenia maszynowego. W przeciwieństwie do poprzedniego rozdziału, po skończeniu treningu modelu lokalnego, agregacja następuje natychmiast, na podstawie najnowszych modeli lokalnych. Eliminuje to konieczność oczekiwania na wykonanie treningu wszystkich modeli dla danej epoki. Przeprowadzono szereg eksperymentów, które miały na celu ocenę wydajności oraz jakości uczenia dla różnych scenariuszy i konfiguracji maszyn wirtualnych. Wyniki te pozwalają zrozumieć, jak różnice w dostępnych zasobach mogą wpłynąć na końcowy rezultat procesu uczenia oraz wskazują na potencjalne optymalizacje, które można zastosować w celu minimalizacji negatywnego wpływu omawianych przypadków. Należy zwrócić uwagę, że tak jak w przypadku uczenia synchronicznego, każdy model lokalny wykonywał 14 epok. *P. Mazur*

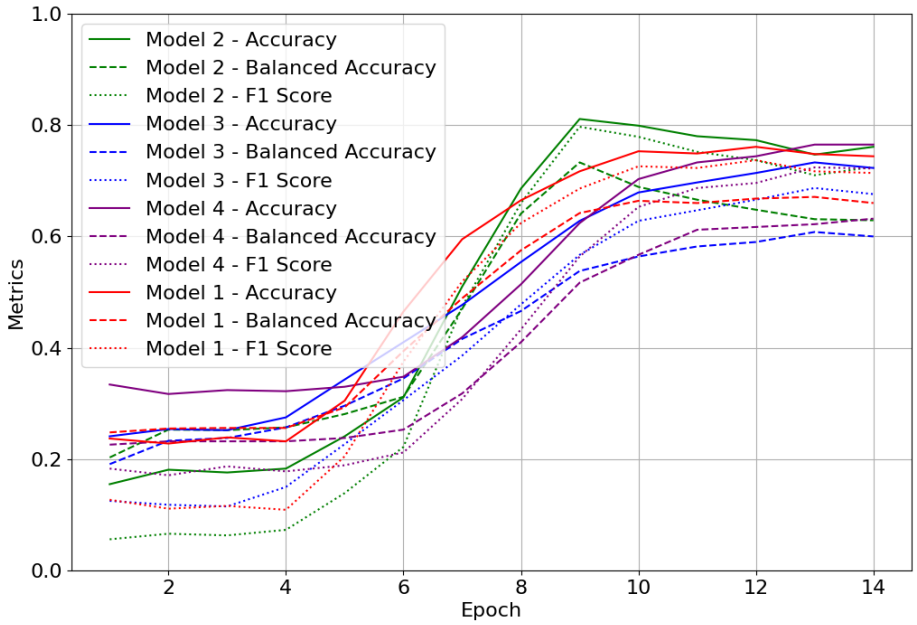
7.1 Heterogeniczne zasoby modeli lokalnych

W rzeczywistych scenariuszach podczas uczenia federacyjnego rzadko zdarza się, aby modele lokalne miały identyczne zasoby obliczeniowe. Często spotykamy się z sytuacją, gdzie zasoby są heterogeniczne, czyli różnią się pod względem mocy obliczeniowej, dostępnej pamięci RAM, czy prędkości i rodzaju pamięci masowej. Tego rodzaju niejednorodność może wpływać na wydajność procesu uczenia, szczególnie w kontekście synchronicznego uczenia federacyjnego, gdzie opóźnienia w przetwarzaniu danych na jednym z węzłów mogą spowolnić cały proces. *P. Mazur*

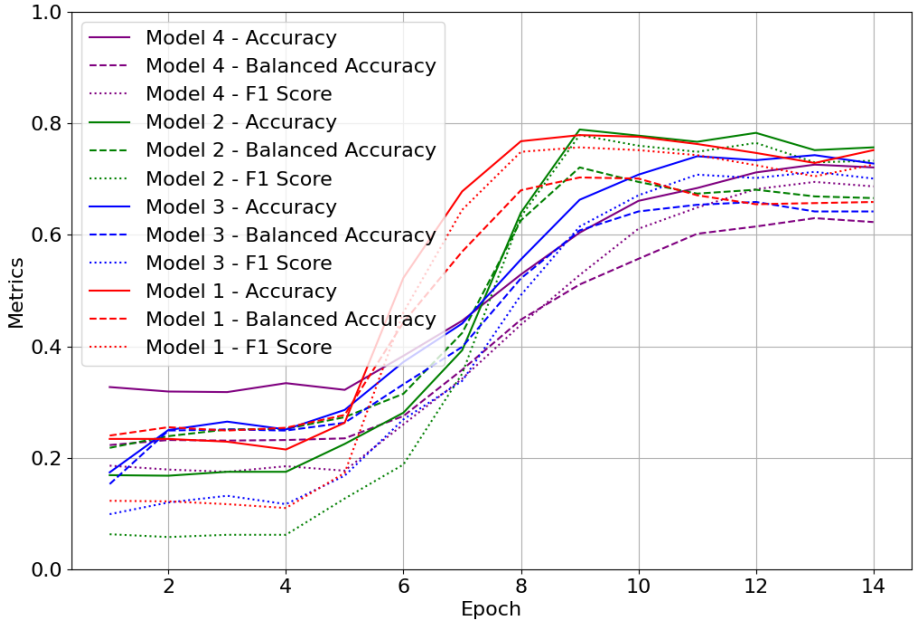
W niniejszym rozdziale skupiono się na analizie klastrów o konfiguracji wielkości maszyn wirtualnych. W eksperymentach wykorzystano 3 konfiguracje:

- **small** - 4 vCPU i 16 GB RAM,
- **medium** - 8 vCPU i 32 GB RAM,
- **large** - 16 vCPU i 64 GB RAM.

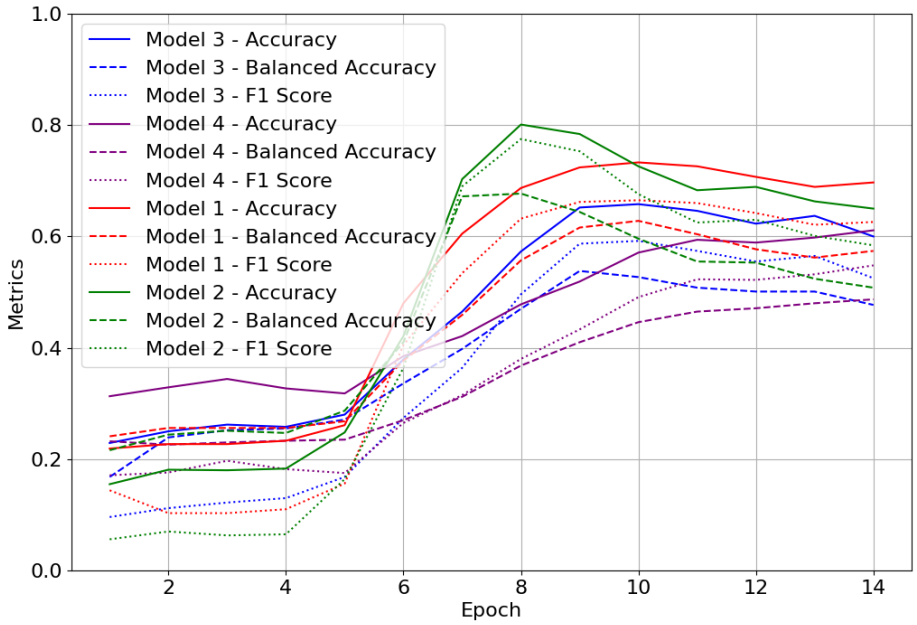
Wykonano eksperymenty, w których jeden z klastrów posiadał większą moc obliczeniową od pozostałych, w kolejnym eksperymencie ten sam klaster posiadał mniejszą moc obliczeniową w stosunku do pozostałych klastrów. W ostatniej iteracji jeden z klastrów charakteryzował się zwiększoną mocą obliczeniową, a jeden mniejszą w porównaniu do pozostałych modeli lokalnych. *P. Mazur*



Rysunek 7.1: Przebieg metryk dla modelu 1 z dwukrotnie większą liczbą vCPU.



Rysunek 7.2: Przebieg metryk dla modelu 1 z dwukrotnie zmniejszoną liczbą vCPU.



Rysunek 7.3: Przebieg metryk dla modelu 1 z dwukrotnie większą liczbą vCPU i modelu 4 z dwukrotnie zmniejszoną ilością vCPU.

Na podstawie rysunków 7.1, 7.2 oraz 7.3 można zauważyć, że w przypadku klastra z dwukrotnie większą liczbą vCPU wskaźniki jakości osiągnięte na końcu treningu były lepsze od dwóch pozostałych przypadków o ok. 0.05. Wskaźniki jakości dla przypadku z jednym klastrem z mniejszą liczbą zasobów były porównywalne do przypadku z różnymi wielkościami maszyn wirtualnych. *B. Bok*

Wyniki czasów poszczególnych eksperymentów przedstawiono w tabeli poniżej, zgodnie z poniższą listą:

1. **Eksperyment 1** - model 1 posiada konfigurację large, co odpowiada **dwukrotnie większej** wartości w porównaniu z pozostałymi klastrami.
2. **Eksperyment 2** - model 1 posiada konfigurację small, co odpowiada **dwukrotnie mniejszej** wartości w porównaniu z pozostałymi klastrami.
3. **Eksperyment 3**: model 1 posiada konfigurację large, model 4 posiada konfigurację small, natomiast pozostałe klastry posiadają konfigurację medium. *P. Mazur*

Tabela 7.1: Czasy eksperymentów dla różnych wielkości maszyn wirtualnych.

Rodzaj eksperymentu	**model 1**	**model 2**	**model 3**	**model 4**	**Średnia**
eksperyment 1	01:54:31	02:05:55	02:05:42	02:08:16	02:03:56
eksperyment 2	02:26:39	02:17:44	02:16:10	02:19:00	02:19:58
eksperyment 3	01:50:34	01:56:14	02:04:02	02:24:24	01:58:43

Na podstawie tabeli 7.1 można zauważyć, że dzięki podejściu asynchronicznemu udaje się redukować średni czas trwania eksperymentu, z racji braku konieczności

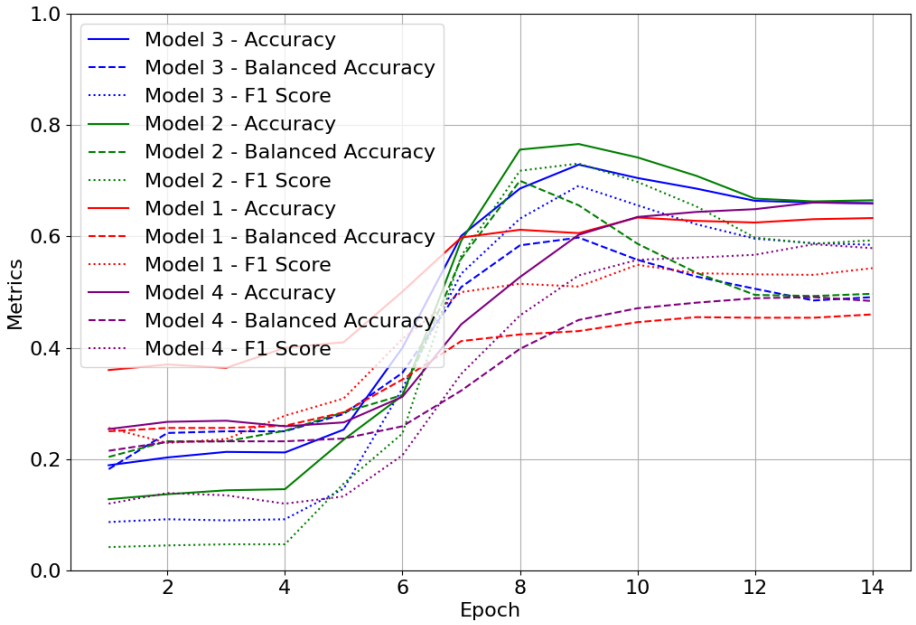
oczekiwania na wyniki treningu pozostałych modeli lokalnych. W przypadku dostępności większej mocy obliczeniowej uczestnik procesu jest w stanie zakończyć uczenie wcześniej i zwolnić zbędne zasoby. *P. Mazur*

Uczenie asynchroniczne pozwala na efektywne wykorzystanie różnic w mocach obliczeniowych. Dzięki temu, cały proces uczenia staje się tańszy i szybszy. W tradycyjnym podejściu, czas uczenia byłby równy czasowi potrzebnemu dla klastra z najmniejszą ilością zasobów. Warto jednak zauważyć, ze występowanie jednego klastra wykonującego obliczenia szybciej od reszty nie wpływa negatywnie na dokładność modeli, natomiast obecność klastra wolniejszego dokładność obniża. *P. Mazur*

7.2 Nierównomierna ilość danych między klastrami

W procesie federacyjnego uczenia maszynowego jednym z kluczowych wyzwań jest nierównomierne rozłożenie danych treningowych pomiędzy poszczególne węzły lub klastry. W rzeczywistych aplikacjach dane często są rozproszone w sposób niezrównoważony, co może prowadzić do wydłużenia czasu trwania treningu w przypadku bardziej obciążonych węzłów. Nierównomierne ilości danych mogą skutkować różnicami w tempie uczenia oraz w jakości modelu uzyskiwanego przez poszczególne węzły, co z kolei wpływa na globalną wydajność całego systemu. *P. Mazur*

W tym rozdziale analizowane są skutki nierównomiernego rozłożenia danych na efektywność federacyjnego uczenia maszynowego. Przeprowadzono szereg eksperymentów, w których model pierwszy miał zduplikowane dane dla swojej wersji, co symuluje sytuację, w której ilość danych na tym węźle jest znacznie większa niż na pozostałych. Konieczność wprowadzenia mechanizmów kompensacji wynika z potrzeby zapewnienia optymalnej wydajności systemu i uniknięcia sytuacji, w której bardziej obciążone węzły stają się wąskim gardłem procesu treningu. Jest to istotne zarówno dla dostawców usług chmurowych, którzy dążą do minimalizacji kosztów operacyjnych, jak i użytkowników końcowych korzystających z federacyjnego uczenia maszynowego. W kolejnych eksperymentach badano różne podejścia do kompensacji tego obciążenia, takie jak zmiany wielkości maszyn wirtualnych oraz zmiana rodzaju dysków na szybsze SSD. Wyniki tych eksperymentów pozwalają ocenić, które techniki są najbardziej efektywne w radzeniu sobie z nierównomiernym rozkładem danych pomiędzy modelami lokalnymi. Takie działania mogą zostać podejmowane w celu redukcji kosztów przez organizacje posiadające mniejszą ilość danych. *P. Mazur*

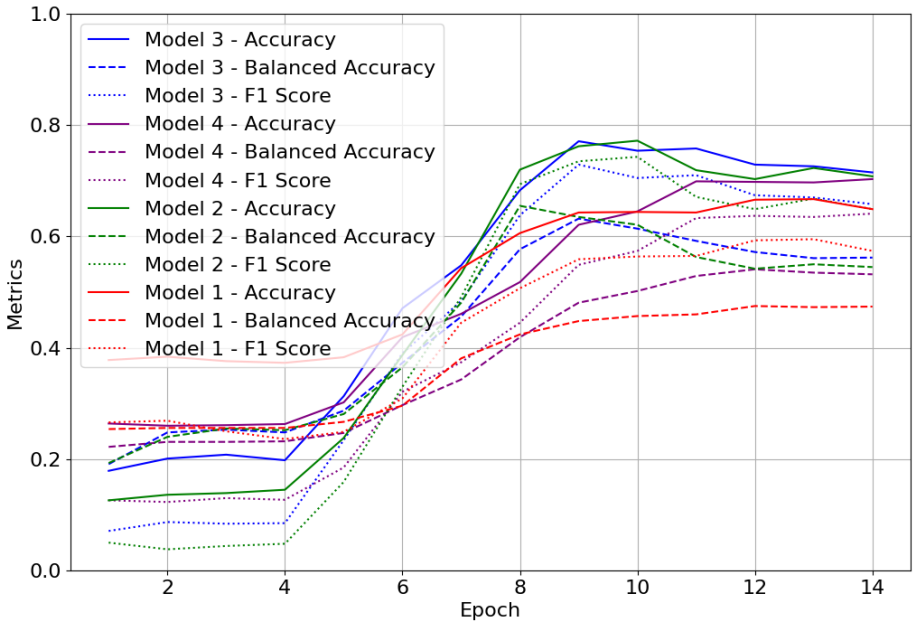


Rysunek 7.4: Przebieg metryk dla zduplikowanych danych dla klastra 1.

Na podstawie rysunku 7.4 można zauważyć, że metryki pogorszyły się w stosunku do równomiernie rozłożonych danych. Najprawdopodobniej wynika to z faktu rzadszej aktualizacji modelu 1, co było spowodowane dużą liczbą danych do przetrenowania. *B. Bok*

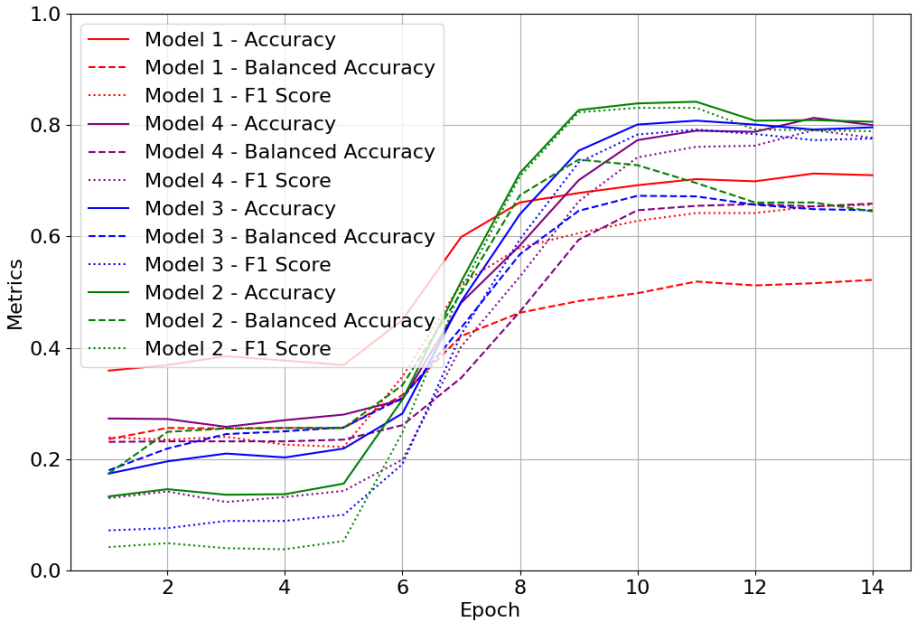
7.2.1 Propozycje kompensacji

W celu zredukowania negatywnego wpływu nierównomiernej ilości danych między klastrami w federacyjnym uczeniu maszynowym, przeprowadzono szereg eksperymentów mających na celu kompensację zwiększonego obciążenia jednego z klastrów. Model 1, który posiadał zduplikowaną ilość danych, był przedmiotem tych eksperymentów, podczas których testowano różne strategie kompensacji, takie jak zastosowanie konfiguracji maszyn wirtualnych o zwiększonej ilości mocy obliczeniowej oraz zastosowanie szybszego nośnika danych w postaci dysku SSD. *P. Mazur*



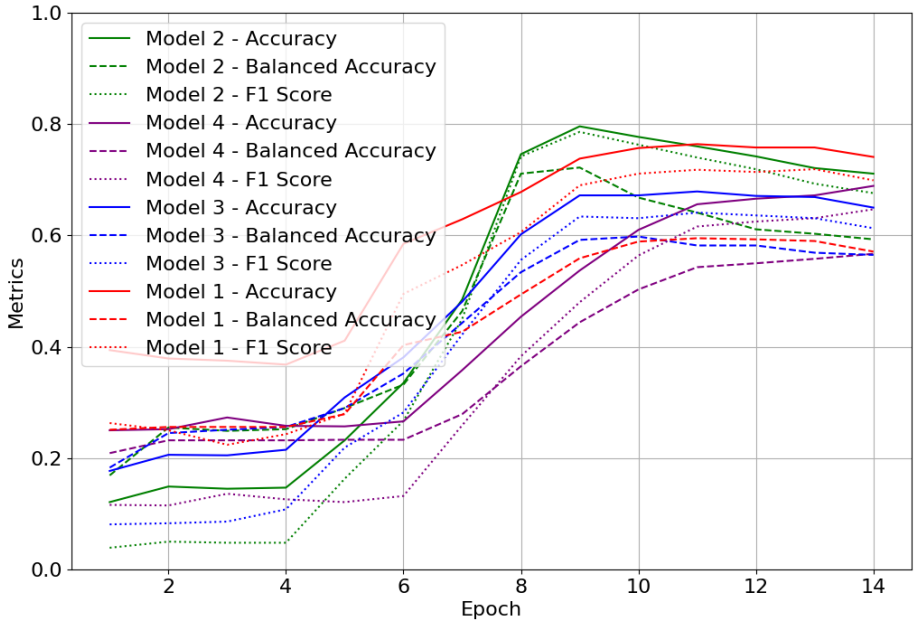
Rysunek 7.5: Przebieg metryk dla zduplikowanych danych dla modelu pierwszego, z maszyną wirtualną w konfiguracji medium.

Na rysunku 7.5 dla scenariusza z maszyną wirtualną w konfiguracji medium dla modelu lokalnego posiadającego zduplikowane dane, nie widać zauważalnej różnicy w metrykach osiąganych przez modele. Jest to najprawdopodobniej skutkiem zbyt małego zwiększenia możliwości trenowania modelu. *B. Bok*



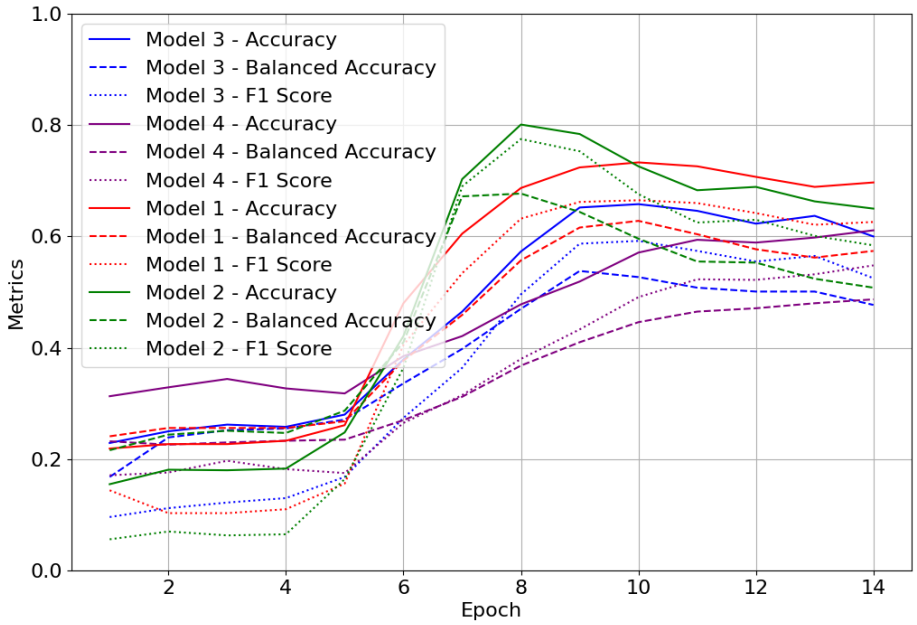
Rysunek 7.6: Przebieg metryk dla zduplikowanych danych dla modelu pierwszego, z maszyną wirtualną w konfiguracji large.

Dla przypadku z zastosowaniem konfiguracji large (przedstawionego na rysunku 7.6) różnica jest już zauważalna. Modele osiągają wyższe metryki accuracy o ok. 0.05. Widać więc, że czterokrotne zwiększenie liczby vCPU dla klastra posiadającego więcej danych, umożliwia kompensację strat wynikających z większej liczby danych. *B. Bok*



Rysunek 7.7: Przebieg metryk dla zduplikowanych danych dla modelu pierwszego, z dyskami SSD.

Zamiana dysku dla modelu pierwszego na SSD poprawiła znacząco przebieg wskaźników jakości. Potwierdza to, że pogorszenie wskaźników jakości przez zduplikowanie danych na klastrze jest spowodowane długim zaczytywaniem danych do pamięci RAM.



Rysunek 7.8: Przebieg metryk dla modelu pierwszego z maszyną wirtualną w konfiguracji large.

Na podstawie rysunku 7.8 można zauważyć, że zwiększenie mocy obliczeniowej dla modelu lokalnego, który posiada podobną liczbę próbek, co pozostałe klastry, nie zmienia znacząco przebiegu wskaźników jakości. *B. Bok*

Zastosowanie maszyny wirtualnej w konfiguracji z większą ilością zasobów obliczeniowych może wpłynąć na przyspieszenie procesu treningu. Należy jednak zauważyć, że w omawianych scenariuszach, w każdej epoce dane na klastrze były zaczytywane do pamięci RAM, co było czasochłonnym procesem. W przypadku zduplikowanych danych proces zaczytywanie był znacząco dłuższy co spowodowało opóźnienie epok danego modelu względem innych, co przeniosło się na pogorszenie metryk wszystkich modeli lokalnych. *P. Mazur*

Przeprowadzono poniższe eksperymenty z próbą kompensacji dwukrotnie większej liczby danych:

1. **Eksperyment 1** - model 1 posiada dysk SSD, pozostałe modele korzystają z dysków HDD.
2. **Eksperyment 2** - model 1 posiada konfigurację medium, co odpowiada **dwukrotnie większej** wartości w porównaniu z pozostałymi klastrami.
3. **Eksperyment 3** - model 1 posiada konfigurację large, co odpowiada **czterokrotnie większej** wartości w porównaniu z pozostałymi klastrami.

Wyniki czasów poszczególnych eksperymentów przedstawiono w tabeli poniżej. *P. Mazur*

Tabela 7.2: Czas eksperymentu dla różnych rodzajów obciążeń

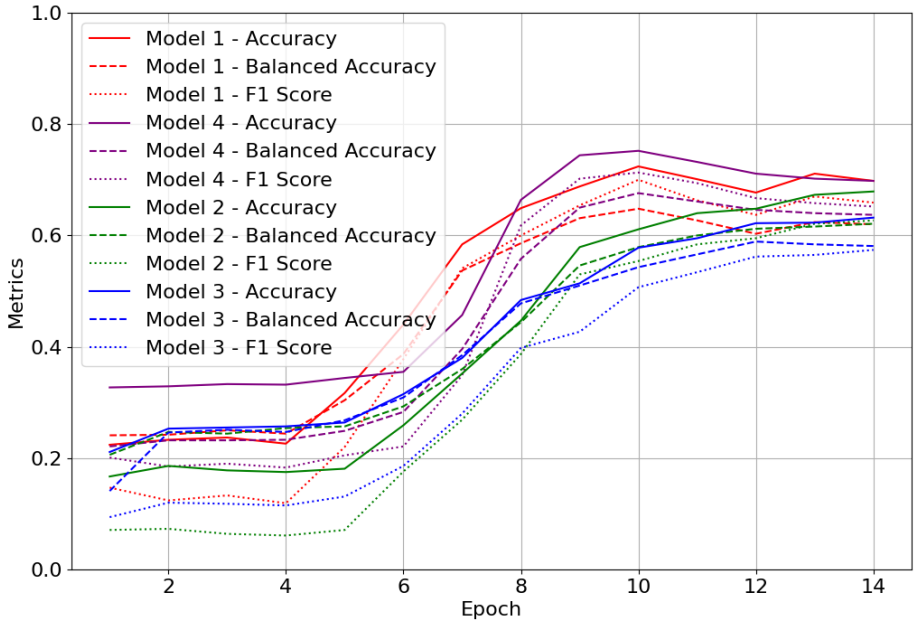
Rodzaj eksperymentu	**model 1**	**model 2**	**model 3**	**model 4**	**Średnia**
eksperyment 1	03:00:37	02:57:51	03:31:45	03:28:45	03:09:54
eksperyment 2	02:33:09	02:11:56	02:19:20	02:17:49	02:20:34
eksperyment 3	02:22:57	02:11:35	02:19:34	02:16:38	02:17:41

Na podstawie 7.7 można zauważyć, że zastosowanie SSD dla modelu pierwszego przyniosło pewne korzyści, jednak znacznie wydłuża czas trwania eksperymentu, co odnotowano w tabeli 7.2. Wymusza to zastosowanie wolniejszych dysków w pozostałych klastrach. To zwiększenie liczby vCPU okazało się bardziej efektywną strategią kompensacji dodatkowego obciążenia wynikającego z nierównomiernego rozkładu danych. Co ciekawe, dwukrotne zwiększenie mocy obliczeniowej okazało się niewystarczające, dokładność modeli okazała się mniejsza niż w porównaniu z zastosowaniem różnego rodzaju dysków. Najlepsze wyniki osiągnięto przy zastosowaniu 16 vCPU, co sugeruje, że przy dużych różnicach w ilości danych pomiędzy klastrami warto zainwestować w większą moc obliczeniową. *P. Mazur*

7.3 Losowe opóźnienia czasowe w komunikacji

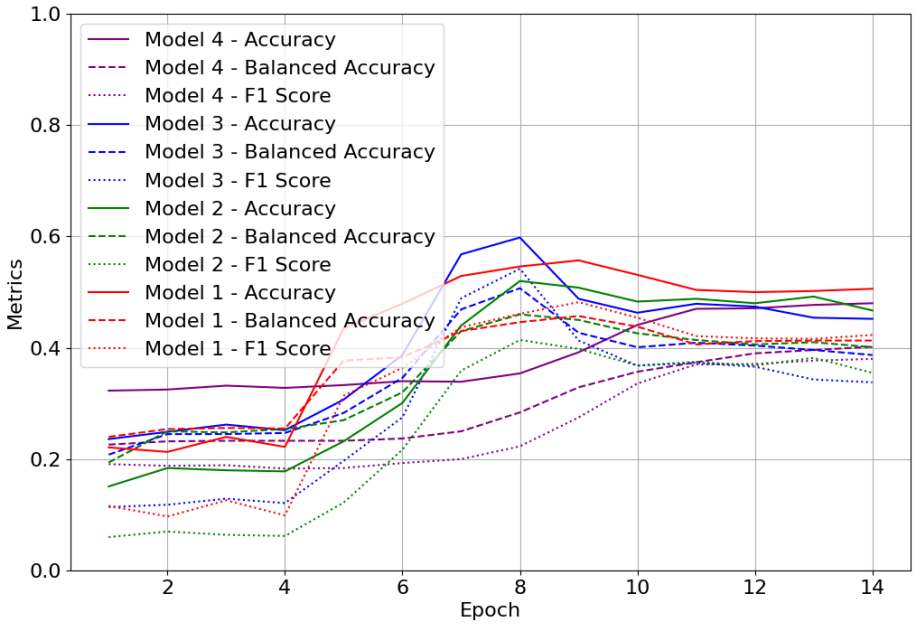
W rzeczywistych warunkach pracy sieci, zwłaszcza w rozproszonych systemach uczenia maszynowego, takich jak federacyjne uczenie maszynowe, nieuniknione są losowe opóźnienia w komunikacji między węzłami. Opóźnienia te mogą wynikać z różnorodnych czynników, takich jak obciążenie sieci, przerwy w dostępie do zasobów, czy zmienne przepustowości łączy. Opóźnienia te wpływają na czas synchronizacji modelu między węzłami, co z kolei może prowadzić do degradacji wydajności i efektywności procesu uczenia. *P. Mazur*

W tym rozdziale analizowane są skutki losowych opóźnień w komunikacji na przebieg procesu uczenia. Skupiono się na dwóch scenariuszach: pierwszy zakłada 25% szans na 5-minutowe opóźnienie w przesyłaniu modelu lokalnego, a drugi 10% szans na 30-minutowe opóźnienie. Porównanie wyników dla obu przypadków pozwala lepiej zrozumieć, jak różne poziomy i częstości opóźnień wpływają na czas trwania eksperymentów oraz końcowe metryki jakości modelu. *P. Mazur*



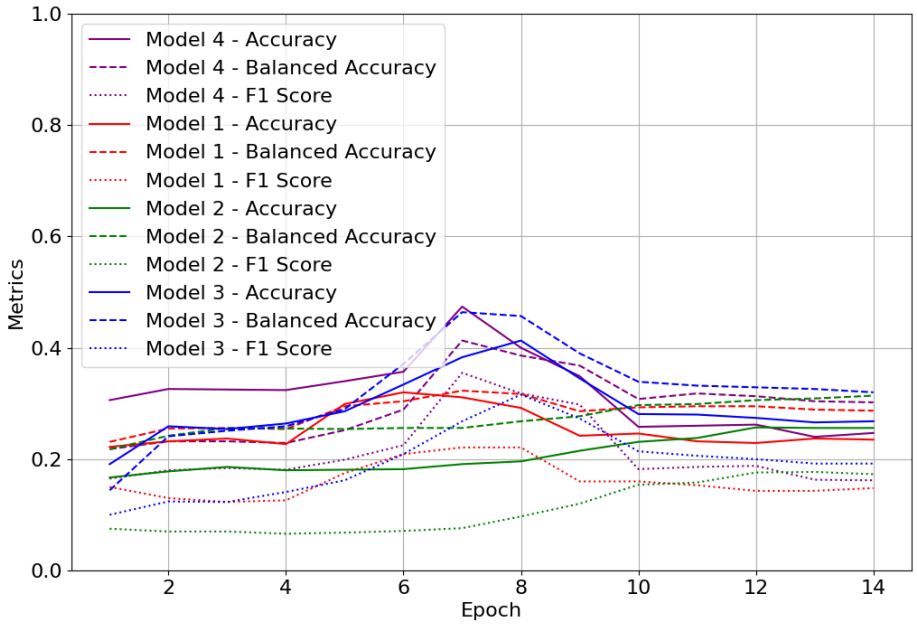
Rysunek 7.9: Przebieg metryk z 25% szans na 5 minut opóźnienia z przesłaniem modelu lokalnego.

Na podstawie rysunku 7.9 można zauważyć, że 25% szansa na 5-minutowe opóźnienia nie wpłynęła znacząco na wyniki. Wynika to najprawdopodobniej z małej wartości opóźnienia, które nie jest większa niż trwanie całej epoki. *B. Bok*



Rysunek 7.10: Przebieg metryk z 10% szans na 15 minut opóźnienia z przesłaniem modelu lokalnego.

W przypadku 10% szans na 15-minutowe opóźnienia następuje znaczące pogorszenie metryk, co jest przedstawione na rysunku 7.10. Pomimo mniejszej szansy na opóźnienia, w porównaniu do wcześniejszego przypadku, spadek jakości modeli jest już znaczący. Wynika to z faktu, że modele bez opóźnień otrzymają zagregowaną wersję modelu na potencjalnie starszych epokach pozostałych modeli, które niwelują wpływ aktualizacji parametrów, po przetrenowaniu najbardziej aktualnych modeli. *B. Bok*



Rysunek 7.11: Przebieg metryk z 10% szans na 30 minut opóźnienia z przesłaniem modelu lokalnego.

10% szans na 30-minutowe opóźnienie jeszcze bardziej potęguje efekt niwelowania aktualizacji wag najbardziej aktualnych modeli, co jest przedstawione na rysunku 7.11. W pierwszej fazie eksperymentu modele jeszcze miały szansę na zwiększenie swoich metryk, jednak z czasem nastąpił spadek. Najprawdopodobniej wynika to z faktu zmniejszającej się wartości współczynnika uczenia, która po agregacji ze starszymi wersjami modeli, nie jest już w stanie w późniejszych epokach przeuczyć modelu na swoich danych. *B. Bok*

Tabela 7.3: Liczba opóźnień w poszczególnych eksperymentach

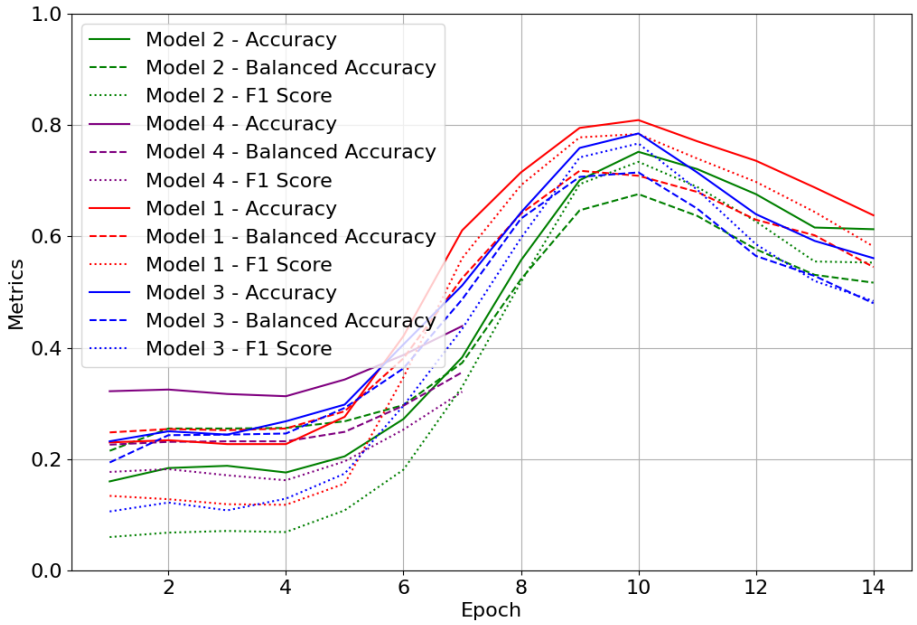
Eksperyment	**model 1**	**model 2**	**model 3**	**model 4**
25% szans na 5 min. opóźnienia	4	3	2	4
10% szans na 15 min. opóźnienia	1	0	1	2
10% szans na 30 min. opóźnienia	1	0	1	1

Na podstawie tabeli 7.3 można zauważyć, że losowe opóźnienia rozkładały się równomiernie po klastrach, pomimo tego efekty opóźnień są wyraźnie widoczne w postaci obniżenia wskaźników jakości modeli. *P. Mazur*

7.4 Awarie modeli lokalnych

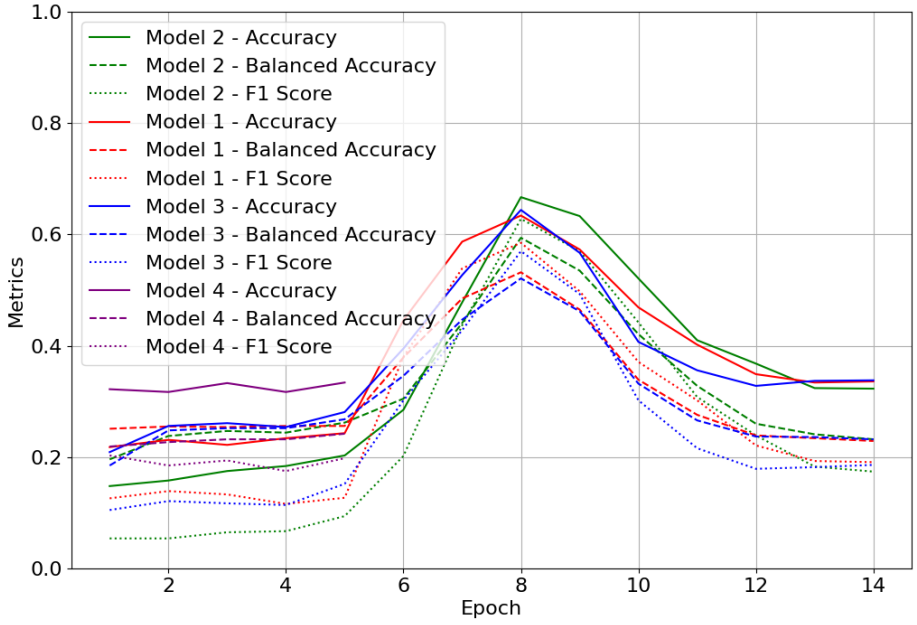
Awarie są nieodłącznym elementem systemów rozproszonych, w tym także federacyjnego uczenia maszynowego. Mogą one wynikać z różnych przyczyn, takich jak problemy sprzętowe, awarie sieci, czy błędy oprogramowania. *P. Mazur*

W tym rozdziale badano wpływ awarii jednego z klastrów na proces uczenia. Eksperyment polegał na symulacji awarii jednego z węzłów po 5 i 7 epoce uczenia i monitorowaniu, jak ta awaria wpływa na przebieg metryk. *P. Mazur*



Rysunek 7.12: Przebieg metryk po awarii jednego z klastrów po 7 epoce.

Jak widać na rysunku 7.12, awaria po 7 epoce znacząco wpłynęła na pogorszenie metryk wszystkich modeli. W początkowej fazie działające modele były w stanie się jeszcze douczyć przy większym współczynniku uczenia. Jednak w kolejnych epokach współczynnik ten był na tyle mały, że podczas agregacji modele bez awarii miały parametry uśredniane z modelem po awarii, nie będąc w stanie mocno douczyć się na swoich danych. *B. Bok*



Rysunek 7.13: Przebieg metryk po awarii jednego z klastrów po 5 epoce.

Podobna sytuacja ma miejsce z awarią po 5 epoce, co jest przedstawione na rysunku 7.13. Efekt ten jest jeszcze mocniejszy ze względu na szybszą awarię. Powoduje to szybszy spadek metryk oraz większe pogorszenie, ze względu na starszą wersję modelu podczas agregacji. Warto zwrócić uwagę, że bezpośrednio po awarii następował przyrost metryk przez kilka epok, które dopiero później zaczęły opadać. Najprawdopodobniej wynika to z faktu zmniejszającego się współczynnika uczenia, który bezpośrednio po awarii był jeszcze na tyle duży, że pozwalał modelom lokalnym mocno przeuczyć się na swoich danych. *B. Bok*

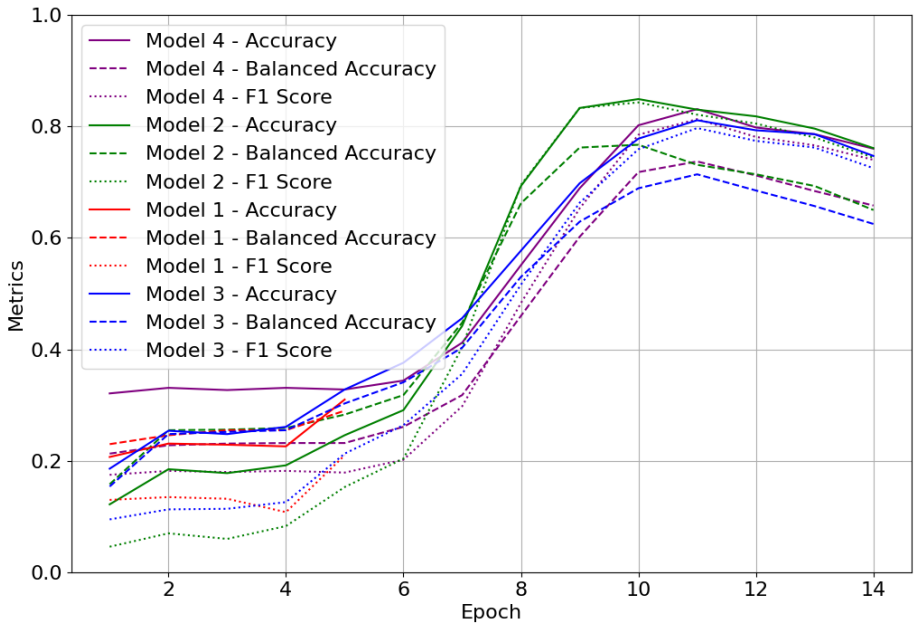
7.4.1 Przeciwdziałanie negatywnym skutkom awarii

Zaproponowaną metodą przeciwdziałaniu negatywnym skutkom awarii jest uzależnienie wag danego modelu lokalnego od wartości jego ostatniej epoki, podzielonej przez sumę ostatnich epok wszystkich modeli lokalnych. Tę metodę przedstawia poniższy wzór:

$$w_i = \frac{e_i}{\sum_{j=1}^{n} e_j} \tag{7.1}$$

gdzie w_i to ważność i-tego modelu, a e_i to najnowsza epoka i-tego modelu. *B. Bok*

Po zaimplementowaniu danego rozwiązania otrzymano wyniki przedstawione na rysunku 7.14.



Rysunek 7.14: Przebieg metryk po awarii jednego z klastrów po 5 epoce z zastosowaniem kompensacji.

Na podstawie rysunku 7.14 można zauważyć, że próba naprawy się powiodła, pozwalając modelom bez awarii na swobodne douczanie się na swoich danych, bez tak drastycznego spadku wartości metryk przy małym współczynniku uczenia na końcu. *B. Bok*

Inne sposobem na przeciwdziałanie negatywnym skutkom awarii może być zmiana koncepcji agregacji modeli. W dotychczas zakładanych scenariuszach, klaster cen-

tralny po otrzymaniu nowego modelu lokalnego, agregował go ze wszystkimi najnowszymi wersjami pozostałych modeli. Powoduje to jednak równy wpływ modelu z awarią, z pozostałymi. Możliwym rozwiązaniem może być agregacja otrzymanego modelu lokalnego z ostatnią wersją modelu zagregowanego. Pozwoli to agregować model częściej z modelami, które wysyłają więcej swoich aktualizacji oraz zniwelować wpływ modelu z awarią. *B. Bok*

7.5 Podsumowanie

B.Bok, P.Mazur

Federacyjne uczenie maszynowe w trybie asynchronicznym lepiej odzwierciedla rzeczywiste warunki pracy rozproszonych systemów. W praktyce, różnorodność zasobów obliczeniowych, nierównomierne rozłożenie danych oraz występowanie opóźnień i awarii w komunikacji między węzłami to częste problemy, które wpływają na wydajność całego procesu uczenia.

Eksperymenty wykazały, że lepsze zasoby obliczeniowe, takie jak zdecydowanie większa moc obliczeniowa lub szybsze dyski SSD, prowadzą do wyższej dokładności modeli lokalnych i skracają czas trwania treningu. Niemniej jednak, gdy występują znaczące opóźnienia lub awarie, wszystkie modele w klastrze tracą na jakości. Dzieje się tak, ponieważ modele najczęściej operują już na niskim współczynniku uczenia w późniejszych etapach treningu, a agregacja modeli z opóźnieniem prowadzi do zniekształcenia ich parametrów.

Wyniki prowadzą do wniosku, że konieczne jest opracowanie nowych metod radzenia sobie z problemami asynchronicznej komunikacji. Jednym z możliwych rozwiązań może być wprowadzenie globalnej wartości współczynnika uczenia, która byłaby dostosowywana dynamicznie w zależności od stanu pozostałych klastrów. Inną opcją jest wyznaczenie współczynnika uczenia w funkcji wartości ostatnich epok wszystkich modeli lokalnych, co mogłoby pomóc w minimalizacji negatywnego wpływu starszych modeli na proces agregacji.

Na przyszłość warto rozważyć także inne podejścia optymalizacyjne, takie jak adaptacyjna synchronizacja między węzłami, gdzie węzły z większymi zasobami obliczeniowymi lub szybszym dostępem do danych mogą aktualizować model częściej niż wolniejsze węzły. Ponadto, wprowadzenie mechanizmów predykcji awarii i opóźnień mogłoby umożliwić wcześniejsze przygotowanie systemu do wystąpienia problemów, np. poprzez dynamiczne alokowanie zasobów czy priorytetyzowanie komunikacji między klastrami.

Podsumowując, chociaż asynchroniczna komunikacja odzwierciedla realne warunki i jest bardziej elastyczna, wymaga ona dalszych badań i optymalizacji w celu minimalizacji wpływu opóźnień, awarii oraz nierównomierności zasobów na jakość uzyskanych modeli.

7.6 Porównanie z uczeniem synchronicznym

7.6.1 Porównanie jakości modeli

B. Bok

Uczenie asynchroniczne okazuje się być bardziej elastycznym podejściem, pozwalającym na minimalizację kosztów operacyjnych, jednak często wiąże się z mniejszą kontrolą nad jakością metryk. W sytuacji, gdy występują opóźnienia lub awarie w

komunikacji, końcowa dokładność modeli może znacząco spaść. W związku z tym, w przypadku federacyjnego uczenia maszynowego, w którym kluczowe jest osiągnięcie jak najwyższej jakości modeli, warto rozważyć zastosowanie podejścia synchronicznego lub implementację dodatkowych mechanizmów, które pozwolą na bardziej kontrolowane i spójne przesyłanie oraz agregację modeli.

W sytuacjach, gdzie jeden z klastrów posiada większą ilość danych, należy rozważyć zwiększenie liczby vCPU lub zamianę dysku na SSD, aby minimalizować negatywny wpływ nierównomiernego obciążenia.

W przypadku asynchronicznego uczenia maszynowego, warto rozważyć wprowadzenie mechanizmów monitorowania jakości modeli lokalnych przed ich agregacją. Na przykład, można uwzględniać epokę, dla której model został wygenerowany, lub inne wskaźniki jakości, aby zapobiec pogorszeniu metryk globalnych.

Optymalizacja infrastruktury sieciowej i zapewnienie jak najbardziej spójnych warunków komunikacji między węzłami może również pomóc w redukcji wpływu losowych opóźnień i zapewnić większą stabilność procesu uczenia.

7.6.2 Wydajność i optymalizacja infrastruktury

P. Mazur

Federacyjne uczenie maszynowe może być realizowane w różnych trybach, w tym synchronicznym i asynchronicznym. W trybie synchronicznym wszystkie węzły muszą zsynchronizować się na koniec każdej epoki, co może prowadzić do opóźnień, jeśli któryś z węzłów jest znacznie wolniejszy. Z kolei tryb asynchroniczny pozwala na bardziej elastyczne podejście, gdzie węzły mogą aktualizować model globalny bez konieczności oczekiwania na pozostałe.

Podejście asynchroniczne dobrze sprawdza się w kwestii minimalizacji kosztów, jednak na podstawie przeprowadzonych eksperymentów lepsza dokładność modeli otrzymywana jest przy podejściu synchronicznym. W przypadkach awarii lub dłuższych opóźnień z dostarczeniem modelu lokalnego końcowa dokładność znacząco spada. W przypadku uczenia asynchronicznego warto rozważyć analizę jakości modelu lokalnego przed agregacją, np. biorąc pod uwagę epokę, dla której został wygenerowany.

Rozdział 8

Możliwości predykcji zasobów podczas uczenia federacyjnego

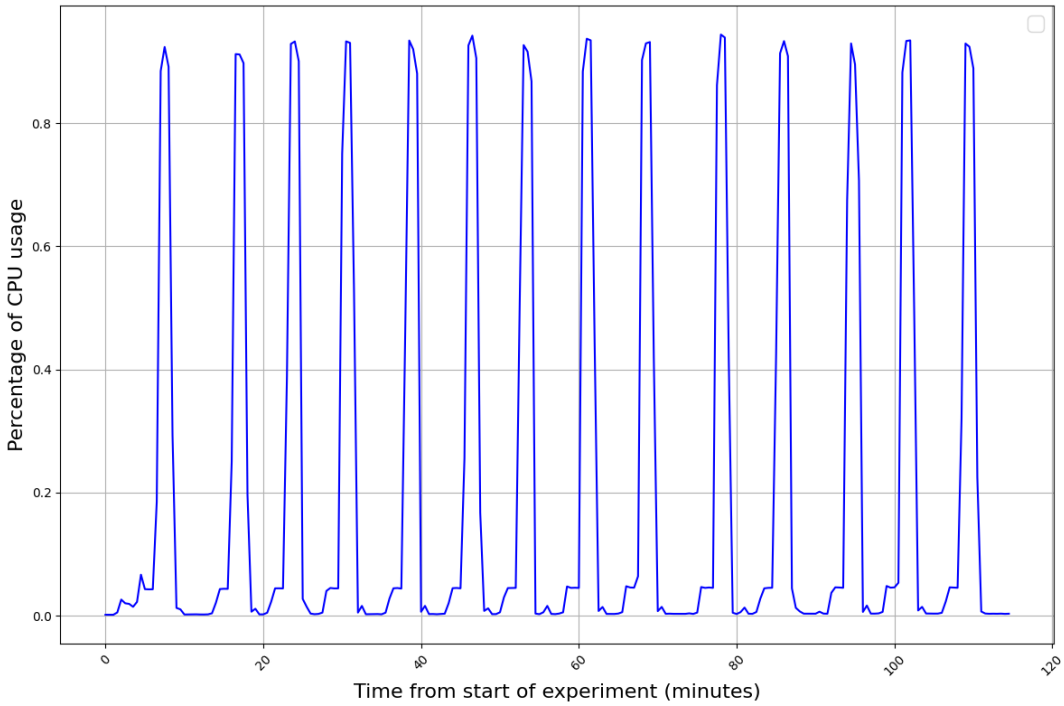
P. Mazur

8.1 Wstęp

W niniejszym rozdziale poruszone zostaną możliwości predykcji zasobów oraz automatycznego skalowania klastrów w oparciu o przewidywane zapotrzebowanie. Z uwagi na charakterystykę pracy uczenia federacyjnego, zwłaszcza w przypadku **podejścia synchronicznego**, w procesie mogą występować długie przerwy pomiędzy treningami kolejnych epok. Korzystne może być wyłączanie maszyn z dużą ilością zasobów na czas oczekiwania na pozostałem modele. Zbierając dane historyczne z poprzednich procesów uczenia możliwe jest również podjęcie próby przewidywania jakiej wielkości maszyna wirtualna jest potrzebna do przeprowadzenia obliczeń w zadanym czasie.

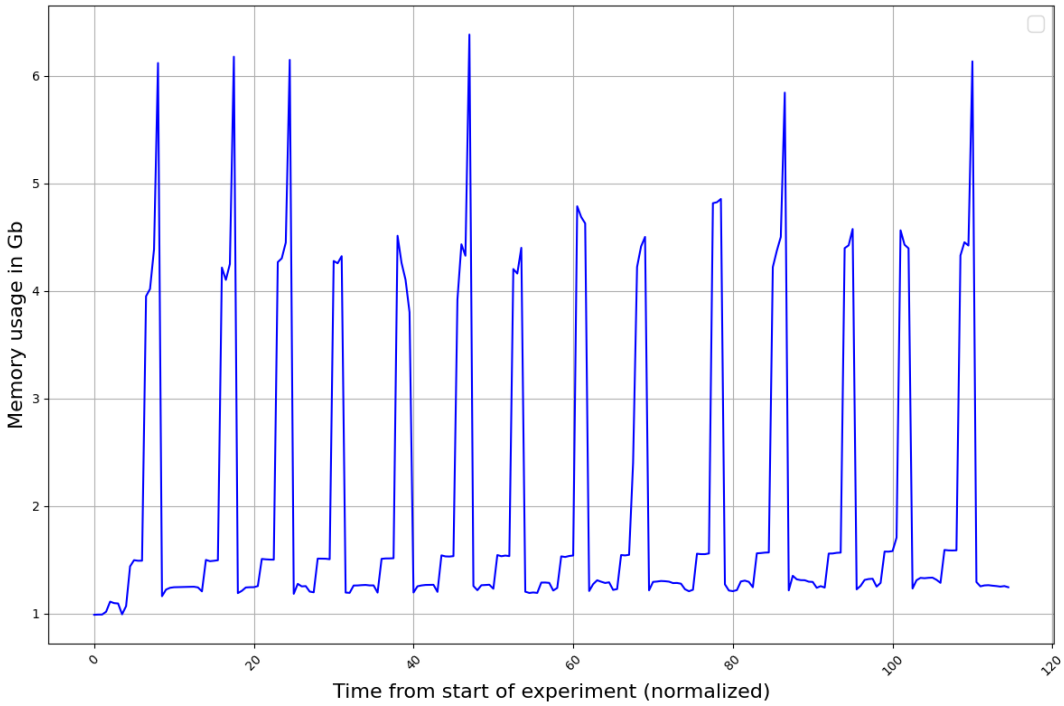
8.2 Zapotrzebowanie na ograniczenie kosztów uczenia

W tej sekcji przedstawione zostaną wykresy zużycia CPU i RAM podczas eksperymentów z uczeniem federacyjnym, uzasadniające potrzebę optymalizacji wykorzystywanych zasobów podczas uczenia federacyjnego. Przedstawiono przypadek uczenia synchronicznego dla modelu 2 , następnie zwiększono ilość danych dla modelu 1.



Rysunek 8.1: Wykres zużycia CPU w czasie eksperymentu.

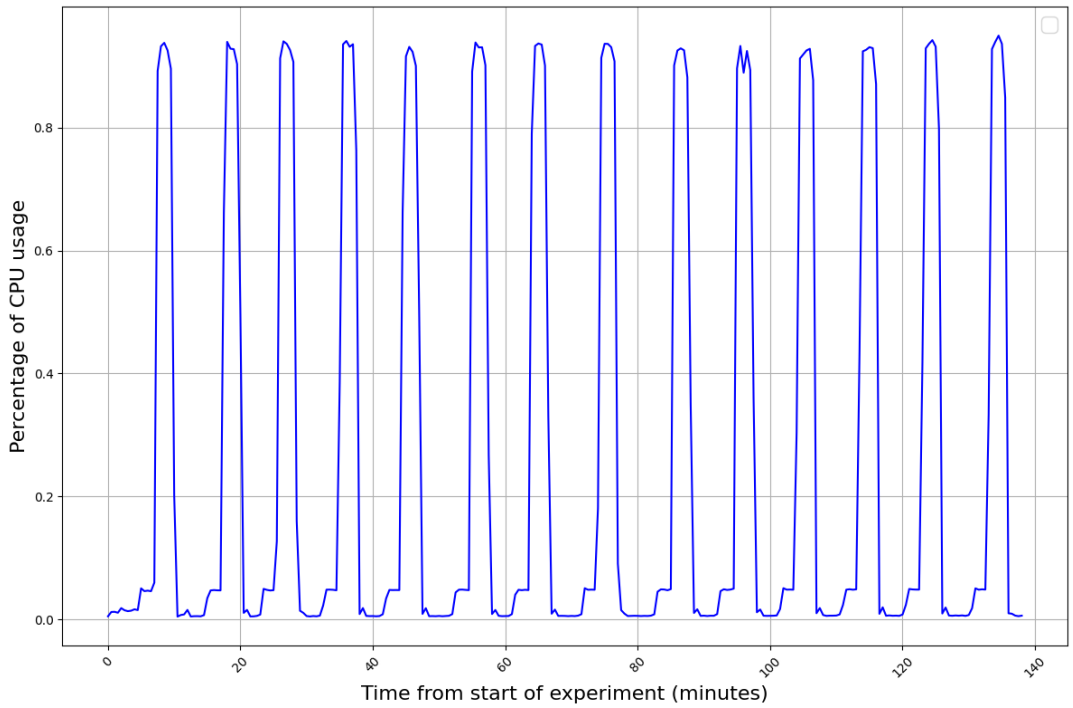
Na rysunku 8.1 przedstawiono zużycie CPU w trakcie trwania eksperymentu. Wysokie zapotrzebowanie na zasoby skorelowane jest z treningiem kolejnych epok modelu. Pomiędzy nimi jest czas oczekiwania na model po agregacji oraz czynności niewymagające wysokiej mocy obliczeniowej, takie jak przesłanie modelu lokalnego czy komunikacja z klastrem centralnym.



Rysunek 8.2: Wykres zużycia RAM w czasie eksperymentu.

Na rysunku 8.2 widzimy również wzrosty zapotrzebowania podczas treningu mo-

deli, podobnie jak w przypadku CPU. Jeżeli wszystkie modele lokalne dysponują zbliżonymi zasobami oraz danymi uczącymi proces przebiega sprawnie, jeżeli jednak te warunki nie zostaną spełnione lub pojawią się opóźnienia koszty związane z korzystaniem z maszyn wirtualnych rosną. Poniżej zamieszczono wykres zużycia CPU w przypadku gdy jeden z modeli lokalnych posiada dwukrotnie zwiększoną liczbę danych.



Rysunek 8.3: Wykres zużycia CPU w czasie eksperymentu dla zduplikowanych danych.

Na rysunku 8.3 oprócz wydłużenia samego procesu uczenia, zwiększa się również okres przestoju między kolejnymi treningami epok. Im więcej dysproporcji lub opóźnień w całym procesie, tym efekt ten będzie coraz bardziej widoczny. Awaria jednego z modeli lokalnych może nieść za sobą bardzo długie przestoje, często wymagające interwencji ze strony użytkowników.

Biorąc pod uwagę powyższe argumenty, potrzeba optymalizacji wykorzystywanych zasobów wydaje się realna, a przynajmniej warta dłuższego zastanowienia.

8.3 Istniejące rozwiązania automatycznego skalowania klastrów

Automatyczne skalowanie klastrów odnosi się do dynamicznego dostosowywania liczby dostępnych maszyn wirtualnych w zależności od bieżących potrzeb.

Systemy te skalują zasoby na podstawie wcześniej zdefiniowanych reguł, takich jak osiągnięcie określonego poziomu zużycia CPU czy RAM. Większość dostawców chmurowych oferuje wbudowane narzędzia skalowania. Przykłady to:

- AWS Auto Scaling [29],
- Google Cloud AutoScaler [30],
- Cluster Autoscaler [31]

Cluster Autoscaler to narzędzie pozwalające wdrożyć autoskalowanie bezpośrednio na klastrach zarządzanych bezpośrednio przez użytkownika. Wspiera szereg dostawców chmurowych, dzięki czemu nie jest konieczne korzystanie tylko z największych chmur publicznych. Jego działanie jest bardzo zbliżone do narzędzi oferowanych przez AWS czy Google.

8.4 Koncepcja implementacji

Z uwagi na charakterystykę pracy cyklicznej uczenia federacyjnego oraz możliwości dynamicznej zmiany wielkości danych warto przeprowadzać częste predykcje zasobów. Propozycja automatycznego skalowanania w oparciu o uczenie maszynowe została zaproponowana w artykule [32], gdzie autorzy skupiają się na rozszerzeniu Horizontal Pod Autoscaler (HPA). Horizontal Pod Autoscaler automatycznie dostosowuje liczbę podów w aplikacji na podstawie obserwowanych wskaźników, takich jak zużycie CPU lub pamięci. W przypadku uczenia w postaci sekwencji zadań takie podejście nie zapewni oczekiwanego rezultatu, ponieważ uczenie przeprowadzane jest tylko w jednym podzie, HPA natomiast zwiększa liczbę replik. Odpowiednim rozwiązaniem wydaje się zastosowanie Admission Webhook, omówionego poniżej. Przed powstaniem poda odpytywany jest serwer, który dokonuje predykcji potrzebnych zasobów, a następnie modyfikuje manifest o przewidziane zasoby. Tworzony pod zawiera niezbędne żądanie zasobów, które następnie mogą wyzwolić stworzenie odpowiedniej wielkości nową maszynę wirtualną za pomocą Cluster Autoscalera, przedstawionego w rozdziale 8.3.

8.4.1 Admission webhook do modyfikacji zasobów

Kluczowym komponentem rozwiązania jest admission webhook. Zgodnie z dokumentacją Kubernetes [33] są to obiekty, które przechwytują żądania HTTP i wykonują na nich określone czynności. Można zdefiniować dwa typy admission webhooków validating i **mutating**. W proponowanym rozwiązaniu wykorzystywany jest drugi rodzaj, który modyfikuje wybrane żądania do API serwera. Poniżej zamieszczono przykład zasobu mutating admission webhook:

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  name: resource-mutator
webhooks:
  - name: resource-mutator.default.svc
    clientConfig:
      service:
        name: resource-mutator
        namespace: mutation-webhook-prediciton
        path: "/mutate"
      caBundle: <caBundle>
    rules:
      - operations: ["CREATE"]
        apiGroups: [""]
        apiVersions: ["v1"]
        resources: ["pods"]
        scope: Namespaced
    admissionReviewVersions: ["v1"]
    sideEffects: None
    timeoutSeconds: 10
```

Po standardowym wysłaniu żądania HTTP z manifestem opisującym dany obiekt, webhook przekazuje manifest do serwera dostępnego pod serwisem resource-mutator w przestrzeni nazw mutation-webhook-prediciton, pod ścieżką /mutate. Po otrzymaniu zwrotu przesyła zmodyfikowany manifest do API serwera. Webhook jest wy-

woływany przy tworzeniu pod'ów w wybranej przestrzeni nazw. Dalsza logika zależy od tego w jaki sposób zostanie zaimplementowany serwer modyfikujący. To w nim można wykorzystać predykcję zasobów, aby zmodyfikować żądania zasobów pod'a oraz ograniczyć się tylko do obiektów związanych z procesem trenowania modelu.

8.4.2 Serwer obsługujący mutowanie obiektów

Serwer, do którego żądania przesyła mutating admission webhook powinien pod zdefiniowaną ścieżką przyjąć, a następnie zwrócić modyfikację obiektu. Manifesty przesyłane są w formacie json, w którym należy przyjąć dane żądanie, a następnie zwrócić obiekt admission review, na podstawie którego mutating admission webhook przeprowadzi modyfikacje poda. Przykładowy admission review zamieszczono poniżej:

```
admission_response = {
    "apiVersion": "admission.k8s.io/v1",
    "kind": "AdmissionReview",
    "response": {
        "uid": request_json['request']['uid'],
        "allowed": True,
        "patchType": "JSONPatch",
        "patch": <patch for reesource>
    }
}
```

Poza standardowym wskazaniem rodzaju obiektu oraz wersji API należy w odpowiedzi zdefiniować uid odczytane z przyjętego żądania oraz rodzaj modyfikacji, wygodnym i najczęściej stosowany jest format json. W polu patchType należy umieścić wygenerowaną zmianę. Należy pamiętać, aby używać certyfikatów zgodnych z CA użytym w definicji mutating admission webhook.

Predykcja zasobów może być realizowana w ramach aplikacji serwera lub w osobnej aplikacji. Z punktu widzenia operacyjnego lepszym podejściem jest wykorzystanie drugiego podejścia, w celu rozdzielenia logiki obu aplikacji oraz łatwiejszego wprowadzania zmian. Aplikacja powinna na podstawie przesłanych danych zwrócić przewidywane wielkości zasobów, z wykorzystaniem wybranego algorytmu. Przykładowy algorytm do tego zadania przedstawiono w późniejszym podrozdziale.

8.4.3 Konfiguracja Cluster Autoscaler

Narzędzie w postaci Cluster Autoscaler umożliwia zdefiniowanie wybranego rodzaju i ilości maszyn wirtualnych, jakie są w stanie utworzyć się w przypadku braku wystarczającej ilości zasobów na klastrze. Zmodyfikowany przez admission webhook manifest posiada powinien posiadać przewidywane zapotrzebowanie na zasoby. Zazwyczaj są to wysokie wartości, których brakuje w klastrze. W takiej sytuacji pod nie zostanie przydzielony do żadnej maszyny wirtualnej w klastrze. Aktywuje to autoskalowanie, które utworzy najmniejszą maszynę wirtualną, która spełni żądania zasobów poda.

8.5 Propozycja wykorzystania algorytmu kNN

Algorytm K-Nearest Neighbors (kNN) jest jednym z najprostszych i najbardziej intuicyjnych algorytmów uczenia maszynowego. Jego podstawowym założeniem jest to, że podobne punkty danych znajdują się blisko siebie w przestrzeni cech. Nie wymaga również zebrania dużej ilości danych historycznych, dzięki czemu możliwe jest wykorzystanie go we wczesnej fazie procesu.

8.5.1 Opis matematyczny algorytmu kNN

W artykule Zhongheng Zhang [34] opisuje działanie oraz najważniejsze założenia dla algorytmu. Domyślnie kNN używa odległości euklidesowej, którą można obliczyć za pomocą następującego równania:

$$D(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$$

gdzie p i q są obiektami porównywanymi pod względem n cech. Kolejnym pojęciem jest parametr k, który decyduje, ilu sąsiadów zostanie wybranych dla algorytmu kNN. Odpowiedni wybór k ma znaczący wpływ na wydajność diagnostyczną algorytmu kNN. Duże k zmniejsza wpływ wariancji spowodowanej błędem losowym, ale niesie ryzyko pominięcia małych, ale istotnych wzorców.

Algorytm kNN działa w następujących krokach:

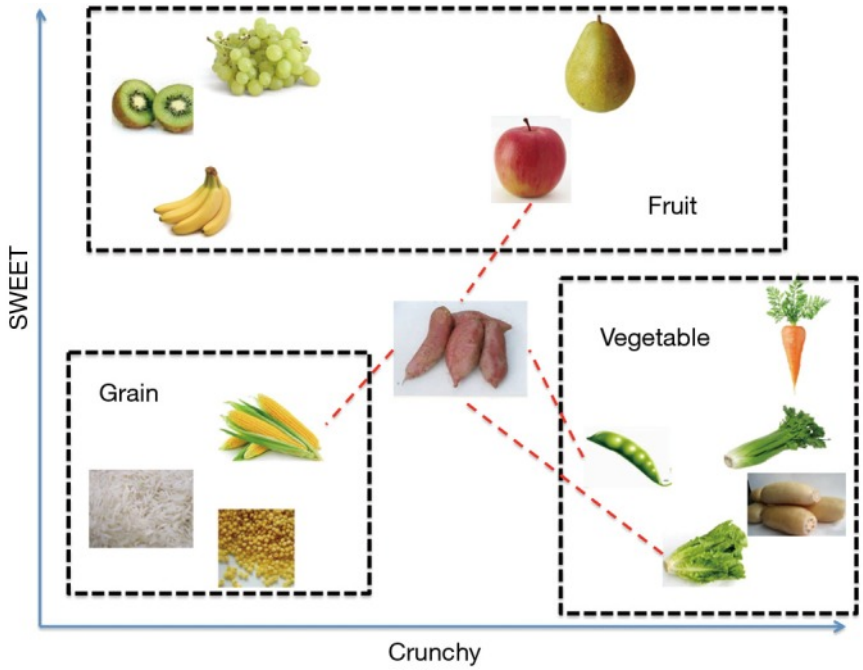
1. Oblicza odległość D między p a wszystkimi punktami q w zbiorze treningowym.
2. Wybiera k najbliższych sąsiadów na podstawie obliczonych odległości.
3. Przewiduje wartość dla p na podstawie wartości jego k najbliższych sąsiadów. W przypadku regresji, wartość ta jest zazwyczaj średnią wartości sąsiadów.

8.5.2 Zastosowanie kNN do predykcji zasobów

Aby zastosować algorytm kNN do predykcji zasobów potrzebnych do uczenia federacyjnego, możemy użyć następujących cech:

- wielkość danych uczących,
- liczba próbek danych uczących,
- oczekiwany czas trenowania epoki.

Im większa ilość zasobów obliczeniowych, tym szybszy proces uczenia, dlatego podczas predykcji powinno się wybrać oczekiwany czas trwania eksperymentu. Dzięki temu skrócony zostanie okres oczekiwania na pozostałe modele lokalne. Dane te mogą być reprezentowane jako wektory cech w przestrzeni wielowymiarowej. Na podstawie historycznych danych dotyczących zużycia zasobów, algorytm kNN może przewidzieć przyszłe zużycie CPU i RAM dla nowych zadań uczenia federacyjnego.



Rysunek 8.4: Wizualizacja działania algorytmu kNN [34].

Rysunek 8.4 przedstawia wizualizację działania algorytmu kNN. W zależności od danych cech obiektu i k najbliższych sąsiadów zostanie on przydzielony do określonej grupy. W analogiczny sposób na podstawie wskazanych danych możemy określić jakiego rodzaju maszyna wirtualna jest potrzebna, żeby spełnić zadane wymagania. Przykład zastosowania:

1. Dostępne są dane historyczne dotyczące zużycia zasobów dla różnych zadań uczenia federacyjnego.
2. Dla nowego poda, które ma określoną wielkość danych uczących i liczbę danych, używamy algorytmu kNN, aby znaleźć k najbardziej podobnych zadań w historycznych danych.
3. Klasyfikujemy zadanie do odpowiedniej grupy, związanej z wielkością maszyny wirtualnej.

Algorytm kNN jest szczególnie przydatny w kontekście uczenia federacyjnego, gdzie dane mogą być heterogeniczne i nieprzewidywalne. Dzięki swojej prostocie i elastyczności, kNN może być skutecznym narzędziem do predykcji zasobów w dynamicznych środowiskach, jednak ze względu na swoje działanie, wraz ze wzrostem ilości danych historycznych jego skuteczność może spadać.

8.6 Podsumowanie

Efektywne zarządzanie zasobami w uczeniu federacyjnym jest kluczowe dla zapewnienia płynności i wydajności procesu trenowania modeli. W niniejszym rozdziale przedstawiono, w jaki sposób automatyczne skalowanie klastrów oraz predykcja zapotrzebowania na zasoby mogą wpłynąć na optymalizację kosztów i efektywności pracy w środowiskach chmurowych.

Wykazano, że charakterystyka pracy uczenia federacyjnego, szczególnie w podejściu synchronicznym, może prowadzić do długich okresów przestoju maszyn wirtualnych pomiędzy kolejnymi epokami trenowania. Przeprowadzone eksperymenty

wskazują, że również w przypadku uczenia asynchronicznego ważne jest utrzymanie całego procesu na zbliżonym etapie. W takich sytuacjach wyłączanie niepotrzebnych zasobów może znacząco obniżyć koszty operacyjne. Wykorzystanie danych historycznych pozwala na przewidywanie wielkości zasobów potrzebnych do realizacji konkretnych zadań, co z kolei umożliwia dynamiczne dostosowywanie infrastruktury w odpowiedzi na bieżące potrzeby.

Istniejące rozwiązania, takie jak AWS Auto Scaling, Google Cloud AutoScaler oraz Cluster Autoscaler, oferują mechanizmy automatycznego skalowania zasobów na podstawie zdefiniowanych reguł. Jednakże w kontekście uczenia federacyjnego, gdzie dynamika procesów jest wysoce zmienna, konieczne jest połączenie tych rozwiązań z predykcją zasobów.

Przedstawiono koncepcję implementacji rozwiązania opartego na mutating admission webhook, który umożliwia dynamiczne dostosowywanie żądań zasobów dla tworzonych podów w Kubernetes. Serwer odpowiedzialny za mutację obiektów, wyposażony w mechanizm predykcji zasobów, może znacząco zwiększyć efektywność wykorzystania infrastruktury.

W ramach proponowanego rozwiązania omówiono także zastosowanie algorytmu K-Nearest Neighbors (kNN) do predykcji zasobów. Algorytm ten, dzięki swojej prostocie i intuicyjności, pozwala na szybkie przewidywanie potrzebnych zasobów na podstawie danych historycznych. Wybór odpowiednich cech, takich jak wielkość danych uczących, liczba danych oraz oczekiwany czas trenowania epoki, umożliwia dokładne dopasowanie zasobów do konkretnych zadań.

Rozdział 9

Podsumowanie

9.1 Wnioski

Praca magisterska koncentruje się na analizie i ocenie różnych scenariuszy uczenia federacyjnego, ze szczególnym uwzględnieniem uczenia synchronicznego i asynchronicznego oraz wpływu konfiguracji infrastruktury na skuteczność i wydajność modeli. Podejście asynchroniczne jest bliższe rzeczywistości w porównaniu do uczenia synchronicznego, jednak jest bardziej podatne na pogorszenie wartości metryk podczas występowania opóźnień czy awarii. Możliwe jest częściowe zniwelowanie wpływu awarii poprzez zmniejszenie ważności modeli, które doświadczają awarii, co może pomóc w minimalizacji strat jakościowych.

Podejście synchroniczne w uczeniu federacyjnym zazwyczaj prowadzi do lepszych wyników, szczególnie w przypadku równomiernie rozłożonych danych i modeli o stałej ważności, ale jest trudniejsza do zrealizowania, ponieważ wymaga od uczestniczących instytucji ścisłego przestrzegania harmonogramu cyklicznego przesyłania modeli. Taki rygor może być trudny do utrzymania, zwłaszcza w środowiskach o zmiennej dostępności zasobów. Z kolei podejście asynchroniczne jest bardziej elastyczne dla instytucji, jednak może skutkować potencjalnie gorszymi wynikami ze względu na większe ryzyko rozbieżności i opóźnień w aktualizacjach modeli. Grupowanie etykiet tematycznych nie ma znaczącego wpływu na wyniki modeli w przypadku komunikacji synchronicznej.

Warto zauważyć, że zastosowanie adaptacyjnego (malejącego) współczynnika uczenia może przynieść korzyści w kontekście zarówno podejścia synchronicznego, jak i asynchronicznego, poprawiając konwergencję modeli i efektywność uczenia. Jednocześnie im większa wartość metryk, tym większe koszty operacyjne, co sugeruje potrzebę optymalizacji parametrów procesu uczenia w celu zrównoważenia jakości modeli i kosztów ich treningu.

Wydajność uczenia federacyjnego zależy znacząco od rodzaju nośnika danych i używanych maszyn wirtualnych. Modele trenowane na szybszych nośnikach danych i wydajniejszych maszynach wykazują lepsze wyniki metryk. Koszty związane z przechowywaniem danych oraz wykorzystaniem maszyn wirtualnych różnią się w zależności od scenariusza, przy czym optymalizacja infrastruktury może znacząco obniżyć koszty operacyjne, szczególnie w kontekście dużych zbiorów danych.

9.2 Dalsze badania

Dalsze badania mogą dotyczyć przeprowadzenia innych scenariuszy uczenia federacyjnego. Można również rozwinąć opisane w pracy scenariusze o bardziej szczegó-

łowe badania, przykładowo eksplorację różnych technik adaptacyjnego współczynnika uczenia, szczególnie w kontekście komunikacji asynchronicznej. Jednym z potencjalnych podejść może być zastosowanie globalnego współczynnika uczenia, który byłby dynamicznie dostosowywany w oparciu o liczbę epok treningowych danych modeli lokalnych, co mogłoby zoptymalizować proces uczenia. Dodatkowo, warto analizować inne metody zapobiegania awarii, takie jak agregacja modelu na podstawie nowego modelu lokalnego i starszej wersji modelu zagregowanego, co mogłoby zwiększyć odporność systemu na nieprzewidziane zdarzenia. Można również zbadać metryki modeli, rozważając scenariusz z rozdzieleniem próbek danych z tymi samymi etykietami pomiędzy różne modele lokalne.

Dalsze badania mogą skoncentrować się na testach automatycznego skalowania wraz z predykcją zasobów, z wykorzystaniem zaproponowanego rozwiązania z algorytmem k-Nearest Neighbors (kNN). Dodatkowową możliwością jest integracja z bardziej zaawansowanymi algorytmami predykcji zasobów, co może poprawić efektywność wykorzystania zasobów i zmniejszyć koszty operacyjne. Warto również skupić się na badaniu, jak różnorodność danych (pod względem ilości, jakości oraz typu danych) wpływa na skuteczność uczenia federacyjnego. Może to obejmować analizę różnych metod przetwarzania wstępnego danych oraz ich wpływ na stabilność i dokładność modelu. Prace nad nowymi algorytmami, które uwzględniają specyficzne wymagania uczenia federacyjnego, mogą przyczynić się do poprawy wydajności i skalowalności systemów uczenia maszynowego, w tym rozwój hybrydowych metod łączących cechy różnych podejść do predykcji i zarządzania zasobami.

Dalsze badania mogą również dotyczyć aspektów bezpieczeństwa i prywatności w uczeniu federacyjnym, szczególnie w kontekście współdzielenia danych między wieloma podmiotami. Opracowanie bardziej zaawansowanych metod ochrony danych [35], takich jak mechanizmy kryptograficzne lub techniki anonimizacji danych, może zwiększyć zaufanie i akceptację tej technologii w różnych sektorach. Te kierunki badań mogą przyczynić się do dalszego rozwoju uczenia federacyjnego, czyniąc je bardziej efektywnym, skalowalnym i bezpiecznym rozwiązaniem w kontekście nowoczesnych aplikacji uczenia maszynowego.

Bibliografia

[1] CL Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information sciences*, 275:314–347, 2014.

[2] Pranav Rajpurkar, Emma Chen, Oishi Banerjee, and Eric J Topol. Ai in health and medicine. *Nature medicine*, 28(1):31–38, 2022.

[3] Itay Goldstein, Chester S Spatt, and Mao Ye. Big data in finance. *The Review of Financial Studies*, 34(7):3213–3225, 2021.

[4] Jan Lies. Marketing intelligence and big data: Digital marketing techniques on their way to becoming social engineering techniques in marketing. 2019.

[5] Gang Wang, Angappa Gunasekaran, Eric WT Ngai, and Thanos Papadopoulos. Big data analytics in logistics and supply chain management: Certain investigations for research and applications. *International journal of production economics*, 176:98–110, 2016.

[6] Mohammed Binjubeir, Abdulghani Ali Ahmed, Mohd Arfian Bin Ismail, Ali Safaa Sadiq, and Muhammad Khurram Khan. Comprehensive survey on big data privacy protection. *IEEE Access*, 8:20067–20079, 2019.

[7] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.

[8] Puneet Himthani, Ghanshyam Prasad Dubey, Brij Mohan Sharma, and Ankur Taneja. Big data privacy and challenges for machine learning. In *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, pages 707–713. IEEE, 2020.

[9] Tianyi Chen, Yuejiao Sun, and Wotao Yin. Communication-adaptive stochastic gradient methods for distributed learning. *IEEE Transactions on Signal Processing*, 69:4637–4651, 2021.

[10] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[11] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[12] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[13] Chenhao Xu, Youyang Qu, Yong Xiang, and Longxiang Gao. Asynchronous federated learning on heterogeneous devices: A survey. *Computer Science Review*, 50:100595, 2023.

[14] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, David Huba, Alexandra Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Steven Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. In *Proceedings of the 2nd SysML Conference*. MLSys, 2019.

[15] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[16] Yue Zhou, Yue Yu, and Bo Ding. Towards mlops: A case study of ml pipeline platform. In *2020 International conference on artificial intelligence and computer engineering (ICAICE)*, pages 494–500. IEEE, 2020.

[17] Xiaoda Wang, Yuan Tang, Tengda Guo, Bo Sang, Jiewei Wu, Jian Sha, Ke Zhang, Jiang Qian, and Mingjie Tang. Couler: Unified machine learning workflow optimization in cloud. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 5224–5237. IEEE, 2024.

[18] Gigi Sayfan. *Mastering Kubernetes: Master the art of container management by using the power of Kubernetes*. Packt Publishing Ltd, 2018.

[19] Linkerd Authors. Architecture. https://linkerd.io/2.16/reference/architecture, Wrzesień 2024.

[20] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *The 2011 international joint conference on neural networks*, pages 1453–1460. IEEE, 2011.

[21] Sovit Ranjan RathSovit Ranjan Rath. Traffic sign recognition using pytorch and deep learning. https://debuggercafe.com/traffic-sign-recognition-using-pytorch-and-deep-learning/, Wrzesień 2022.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[23] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[24] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] Janki Bhimani, Jingpei Yang, Zhengyu Yang, Ningfang Mi, Qiumin Xu, Manu Awasthi, Rajinikanth Pandurangan, and Vijay Balakrishnan. Understanding performance of i/o intensive containerized applications for nvme ssds. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2016.

[26] Cloudferro SA. Cennik maszyn wirtualnych. https://cloudferro.com/pl/cennik/tabele-cenowe/waw3-2-cloud/maszyny-wirtualne-vm/, Wrzesień 2024.

[27] Cloudferro SA. Cennik storage. https://cloudferro.com/pl/cennik/tabele-cenowe/waw3-2-cloud/storage/, Wrzesień 2024.

[28] Ewa Szlachcic. Optymalizacja wielokryterialna. http://staff.iiar.pwr.wroc.pl/ewa.szlachcic/materialy%20dydaktyczne/air_studia_2_stopnia/metody_optymalizacji_wielokryterialnej_Mlynek.pdf, Wrzesień 2024.

[29] Inc. Amazon Web Services. Aws auto scaling. https://aws.amazon.com/autoscaling/, Wrzesień 2024.

[30] Google LLC. Gcp autoscaler. https://https://cloud.google.com/compute/docs/autoscaler, Wrzesień 2024.

[31] Kubernetes Autoscaler authors. Kubernetes autoscaler documentation. https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler, Wrzesień 2024.

[32] László Toka, Gergely Dobreff, Balázs Fodor, and Balázs Sonkoly. Machine learning-based scaling management for kubernetes edge clusters. *IEEE Transactions on Network and Service Management*, 18(1):958–972, 2021.

[33] The Kubernetes Authors. Architecture. https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/, Wrzesień 2024.

[34] Zhongheng Zhang. Introduction to machine learning: k-nearest neighbors. *Annals of translational medicine*, 4(11), 2016.

[35] Mohamed Abdel-Basset, Nour Moustafa, Hossam Hawash, Weiping Ding, Mohamed Abdel-Basset, Nour Moustafa, Hossam Hawash, and Weiping Ding. Federated learning for privacy-preserving internet of things. *Deep Learning Techniques for IoT Security and Privacy*, pages 215–228, 2022.