MARMARA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

DERİN SİNİR AĞLARI İLE FEDERE ÖĞRENME TABANLI BİR KÖTÜ AMAÇLI YAZILIM TESPİT UYGULAMASI

ZEKİ ÇIPLAK

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Yüksek Lisans Programı

DANIŞMAN

Doç. Dr. Kazım YILDIZ

İSTANBUL, 2024

MARMARA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

DERİN SİNİR AĞLARI İLE FEDERE ÖĞRENME TABANLI BİR KÖTÜ AMAÇLI YAZILIM TESPİT UYGULAMASI

ZEKİ ÇIPLAK

523622981

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Yüksek Lisans Programı

DANIŞMAN

Doç. Dr. Kazım YILDIZ

İSTANBUL, 2024

ÖNSÖZ

Tez çalışmamın her aşamasında hiçbir zaman desteğini esirgemeyen, her fırsatta yardımıma koşan değerli danışmanım Sn. Doç. Dr. Kazım YILDIZ'a,

Eğitim hayatım boyunca destekleri ve sabırlarıyla bugünlere gelmemde büyük pay sahibi olan değerli anneme ve babama

Teşekkürlerimi sunarım.

TEMMUZ, 2024 **Zeki ÇIPLAK**

İÇİNDEKİLER

SAYFA

ÖNSÖZ . . . . . **i**
İÇİNDEKİLER . . . . . **ii**
ÖZET . . . . . **v**
ABSTRACT . . . . . **vi**
KISALTMALAR . . . . . **vii**
ŞEKİL LİSTESİ . . . . . **x**
TABLO LİSTESİ . . . . . **xi**
1. GİRİŞ **1**
1.1. Genel Bakış **2**
1.1.1. Problemin Tanımı 2
1.1.2. Amaç ve Kapsam 3
1.1.3. Tezin Özgün Değeri 4
1.2. Kötü Amaçlı Yazılım (Malware) **4**
1.2.1. Malware Türleri **4**
1.2.1.1. Virüsler 5
1.2.1.2. Solucanlar 5
1.2.1.3. Truva Atları 6
1.2.1.4. Fidye Yazılımları 6
1.2.1.5. Casus Yazılımlar 7
1.2.1.6. Zararlı Reklam Yazılımları 7
1.2.1.7. Rootkitler 8
1.2.1.8. Botnetler 8
1.2.2. Malware Yayılma Yöntemleri **9**
1.2.2.1. E-posta Ekleri ve Bağlantıları 9
1.2.2.2. Sahte Yazılım Güncellemeleri 10
1.2.2.3. Güvenlik Açıkları 10
1.2.2.4. Taşınabilir Bellek Aygıtları 10
1.2.2.5. Sosyal Mühendislik Saldırıları 11
1.3. Federe Öğrenme **11**
1.3.1. Federe Öğrenmenin Özellikleri **13**

1.3.2. Federe Öğrenme Sisteminin Çalışma Prensipleri. 13
1.3.3. Federe Öğrenme Sisteminde Durdurma Kriterleri. 15
1.3.3.1. Hedef Doğruluk Oranı. 15
1.3.3.2. Eğitim Döngüsü (Epoch) Sayısı. 15
1.3.3.3. Model Parametrelerinin Stabilizasyonu. 16
1.3.3.4. Eğitim Kaybı (Loss) Azalması. 16
1.3.3.5. Kaynak Kullanımı ve Zaman Sınırları. 16
1.3.3.6. Kullanıcı Geri Bildirimi ve Sistem Sağlığı. 16
1.3.4. Temel Federe Öğrenme Birleştirme Algoritmaları. 17
1.3.4.1. Federe Ortalama Algoritması. 17
1.3.4.2. Federe Stokastik Gradyan İnişi Algoritması. 17
1.3.4.3. Federe Proksimal Algoritması. 17
1.3.4.4. Federe Gradyan Birleştirme Algoritması. 18
1.3.4.5. Federe Eşleştirilmiş Ortalama Algoritması. 18
1.3.5. Federe Öğrenmenin Avantajları. 18
1.3.6. Federe Öğrenmenin Uygulama Alanları. 19
1.3.7. Federe Öğrenmenin Uygulanışındaki Bazı Zorluklar. 20
1.3.7.1. Veri Heterojenliği. 21
1.3.7.2. İletişim Verimliliği. 21
1.3.7.3. Hesaplama Gücü. 21
1.3.7.4. Gizlilik ve Güvenlik. 22
1.3.7.5. Altyapı Gereksinimleri. 22
1.4. Literatür Çalışmaları. 22
2. MATERYAL VE YÖNTEM. 25
2.1. Veri Seti. 26
2.1.1. Veri Setinin Özellikleri. 27
2.1.2. Veri Setinin Önişleme Adımları. 28
2.2. Yapay Sinir Ağı. 31
2.2.1. Derin Öğrenme. 32
2.2.2. Aktivasyon Katmanları. 33
2.2.3. İleri Beslemeli Sinir Ağı. 33
2.2.4. Tekrarlamalı Sinir ağı. 34

2.2.5. Uzun Kısa-Süreli Bellek. 34
2.2.6. Vanishing Gradient Problemi. 35
2.2.7. RNN ve LSTM'in Karşılaştırılması. 36
2.3. Önerilen Derin Öğrenme Mimarileri. 37
2.3.1. İkili Sınıflandırma için FNN (FNN-BC). 38
2.3.2. Çoklu Sınıflandırma için FNN (FNN-MC). 40
2.3.3. İkili Sınıflandırma için LSTM (LSTM-BC). 41
2.3.4. Çoklu Sınıflandırma için LSTM (LSTM-MC). 42
3. ÖNERİLEN YAZILIMLARIN AYRINTILARI. 44
3.1. Yazılımlar Hakkında Genel Bilgiler. 44
3.2. Kimlik Doğrulaması (Authorization) 45
3.2.1. Sunucu (Server) Yazılımı. 46
3.2.1.1. JSON Dosyası: *server_main.json*. 46
3.2.1.2. Python Dosyaları. 49
3.2.1.3. Klasörler. 51
3.2.2. İstemci (Client) Yazılımı. 52
3.2.2.1. JSON Dosyası: *client_main.json*. 52
3.2.2.2. Python Dosyaları. 55
3.2.2.3. Klasörler. 56
3.3. Sunucu-İstemci Mesajlaşmasında Kullanılan API Sistemi. 57
3.4. Şifreleme Tekniğinin Ayrıntıları. 59
3.5. Simülasyon Ortamının Ayrıntıları. 61
3.6. Federe Sistem Başlatılmadan Önce Yapılan Hazırlıklar. 62
3.6.1. Sunucu Yazılımında Gerçekleştirilmesi Gereken Önemli Hazırlıklar. 63
3.6.2. İstemci Yazılımında Gerçekleştirilmesi Gereken Önemli Hazırlıklar. 63
3.7. Yazılımların Çalıştırılması. 64
4. BULGULAR VE TARTIŞMA. 69
5. SONUÇLAR. 78
KAYNAKLAR. 80
ÖZGEÇMİŞ. 90

ÖZET

DERİN SİNİR AĞLARI İLE FEDERE ÖĞRENME TABANLI BİR KÖTÜ AMAÇLI YAZILIM TESPİT UYGULAMASI

Kötü amaçlı yazılımların olası tehditleri, veri güvenliği alanının yalnızca bir yönünü teşkil eder. Bunun yanında, veri mahremiyeti de verinin güvenliği için kaçınılmaz bir bileşen olarak öne çıkmaktadır. Uzun zamandır klasik makine öğreniminde benimsenen, verinin merkezi bir cihazda toplanıp-işlenmesi yöntemi, veri mahremiyeti açısından bir dizi problemi de beraberinde getirmiştir. Günümüzde verilerin bu şekilde toplandığı birçok merkezi bilgi sistemi, saldırgan olarak nitelenen kötü niyetli kullanıcılar tarafından hedef haline getirilmiştir. Öte yandan federe öğrenme yöntemi ise hem veri mahremiyetini sağlaması açısından hem de verinin tek merkezde toplanma zorunluluğunu kaldırması açısından, makine öğrenimindeki verinin işlenmesinde, sağlam bir çözüm sunmaktadır.

Bu tez çalışmasında da aynı derin öğrenme mimarisini kullanan, farklı kullanıcı sayılarına sahip federe öğrenme tabanlı modeller ile federe olmayan klasik yaklaşımlı modellerin verdiği sınıflandırma sonuçları, birçok açıdan karşılaştırılmıştır. Kanada Siber Güvenlik Enstitüsü tarafından hazırlanan CIC-MalMem-2022 veri seti kullanılarak hem malware tespiti hem de malware sınıflandırması gerçekleştirilmiştir.

Feedforward Neural Network (FNN) ve Long Short-Term Memory (LSTM) adlı derin öğrenme algoritmalarının kullanıldığı çalışmada, ikili ve çoklu sınıflandırmalar için farklı niteliklere sahip, dördü federe olmayan olan, toplamda 28 adet model test edilmiş ve sonuçları tartışılmıştır.

TEMMUZ, 2024 **Zeki ÇIPLAK**

ABSTRACT

A FEDERATED LEARNING BASED MALWARE DETECTION APPLICATION WITH DEEP NEURAL NETWORKS

The potential threat of malware is only one aspect of data security. Data privacy is also an inevitable component of data security. The method of collecting and processing data on a centralized device, which has long been adopted in classical machine learning, has brought with it a number of problems in terms of data privacy. Today, many centralized information systems where data is collected in this way have been targeted by malicious users who are considered as attackers. On the other hand, federated learning provides a robust solution for processing data in machine learning, both in terms of ensuring data privacy and eliminating the need to centralize the data.

In this thesis, the classification results of federated learning-based models with different numbers of users using the same deep learning architecture are compared with the classification results of non-federated classical approach models in many aspects. Both malware detection and malware classification were performed using the CIC-MalMem-2022 dataset prepared by the Canadian Cyber Security Institute.

In the study, Feedforward Neural Network (FNN) and Long Short-Term Memory (LSTM) deep learning algorithms were used, and a total of 28 models with different attributes for binary and multiple classifications, four of which were non-federated, were tested and the results were discussed.

JULY, 2024 **Zeki CIPLAK**

KISALTMALAR

AB	**:**	Ada Boost
AES	**:**	Advanced Encryption Standard
API	**:**	Application Programming Interface
BC	**:**	Binary Classification
CB	**:**	CatBoost
ÇVS	**:**	Çoklu Veri Seti
DBSCAN	**:**	Density-Based Spatial Clustering of Applications with Noise
DDoS	**:**	Distributed Denial of Service
DHR	**:**	Dağıtılmış Hizmet Reddi
DFFNN	**:**	Deep Feed Forward Neural Network
DÖ	**:**	Derin Öğrenme
DT	**:**	Decision Tree
ET	**:**	Extra Trees
FedAvg	**:**	Federated Averaging
FedGATE	**:**	Federated Gradient Aggregation
FedMA	**:**	Federated Matched Averaging
FedProx	**:**	Federated Proximal
FedSGD	**:**	Federated Stochastic Gradient Descent
FNN	**:**	Feedforward Neural Network

FNN-BC : İkili sınıflandırma için FNN mimarisi

FNN-MC : Çoklu sınıflandırma için FNN mimarisi

FÖ : Federe Öğrenme

FÖS : Federe Öğrenme Sunucusu

GBM : Gradient Boosting Model

GBT : Gradient Boosted Tree

GMM : Gaussian Mixture Model

GNB : Gaussian Naïve Bayes

GNN : Graphical Neural Network

GRU : Gated Recurrent Unit

IoT : Internet of Things

IP : Internet Protocol

İS : İşletim Sistemi

İVS : İkili Veri Seti

JSON : JavaScript Object Notation

kNN : KNeighbors

LGBM : LightGBM

LiM : Less is More

LR : Logistic Regression

LSTM : Long Short-Time Memory

LSTM-BC : İkili sınıflandırma için LSTM mimarisi

LSTM-MC : Çoklu sınıflandırma için LSTM mimarisi

LVSM : Linear Vector Support Machine

MC : Multiclass Classification

MLP : Multilayer Perceptron

MÖ : Makine Öğrenimi

NB : Naive Bayes

ReLU : Rectified Linear Unit

RF : Random Forest

RNN : Recurrent Neural Network

SGD : Stochastic Gradient Descent

SMOTE : Synthetic Minority Oversampling TEchnique

SSL : Secure Sockets Layer

TL : Training Loss

VG : Vanishing Gradient

VL : Validation Loss

XGB : XGBoost

YSA : Yapay Sinir Ağları

YZ : Yapay Zeka

ŞEKİL LİSTESİ

SAYFA

Şekil 1.1. Federe öğrenme sisteminin çalışması. 12

Şekil 2.1. CIC-MalMem-2022 veri setindeki malware türleri. 26

Şekil 2.2. Önişleme, standardizasyon ve veri setlerine göre sınıflandırma işlemleri. . . .31

Şekil 2.3. Temel bir YSA ve katmanları. 32

Şekil 2.4. Tekrarlamalı Sinir Ağı (RNN). 34

Şekil 2.5. FNN-BC mimarisine ait katmanlar ve nöronlar. 39

Şekil 2.6. FNN-MC mimarisine ait katmanlar ve nöronlar. 40

Şekil 2.7. LSTM-BC mimarisine ait katmanlar ve nöronlar. 42

Şekil 2.8. LSTM-MC mimarisine ait katmanlar ve nöronlar. 43

Şekil 3.1. Sunucu yazılımındaki tüm dosya ve klasörler. 46

Şekil 3.2. Sunucu yazılımındaki server_main.json dosyasının içeriği. 48

Şekil 3.3. İstemci yazılımındaki tüm dosya ve klasörler. 52

Şekil 3.4. İstemci yazılımındaki client_main.json dosyasının içeriği. 54

Şekil 3.5. Simülasyon ortamının temsili resmi. 61

Şekil 3.6. Sunucu yazılımı başlatıldığında komut isteminde oluşan görüntü. 66

Şekil 3.7. İstemci yazılımı yüklü olan bir yerel cihazdan model gönderimleri. 67

Şekil 3.8. Yerel cihazların modellerinin birleştirildiği sırada, sunucu yazılımının konsolunda görüntülenen raporlamalar. 68

Şekil 4.1. Model 5'e ait Kayıp (Loss) – Dönemler (Epochs) grafiği. 72

Şekil 4.2. Model 13'e ait Kayıp (Loss) – Dönemler (Epochs) grafiği. 73

TABLO LİSTESİ

SAYFA

Tablo 2.1. CIC-MalMem-2022 veri setinde bulunan tüm özniteliklerin listesi. 27

Tablo 2.2. İVS içindeki iyi huylu ve kötü huylu olarak işaretlenmiş veri miktarları. . . .30

Tablo 2.3. ÇVS içindeki kategoriler ve o kategorilere ait veri miktarları. 30

Tablo 3.1. server_main.json dosyasındaki bazı önemli ayarlar ve açıklamaları. 47

Tablo 3.2. Sunucu yazılımındaki python dosyalarının görevleri. 49

Tablo 3.3. server_functions.py dosyasındaki fonksiyonların isimleri ve görevleri. 49

Tablo 3.4. other.py dosyasında bulunan fonksiyonların isimleri ve görevleri. 50

Tablo 3.5. Sunucu yazılımındaki klasörler ve kullanılma amaçları. 51

Tablo 3.6. client_main.json dosyasındaki bazı önemli ayarlar ve açıklamaları. 53

Tablo 3.7. İstemci yazılımındaki python dosyalarının görevleri. 55

Tablo 3.8. client_functions.py dosyasındaki fonksiyonların isimleri ve görevleri. 56

Tablo 3.9. Sunucu ve istemci yazılımlarında kullanılan API fonksiyonları. 58

Tablo 3.10. Sunucu yazılımının istemciye gönderdiği cevaplar. 58

Tablo 3.11. Dosya şifreleme ve çözme için kullanılan fonksiyonların sözde kodları. . .60

Tablo 3.12. Deneylerde kullanılan bilgisayarların ortak teknik özellikleri. 62

Tablo 3.13. Sunucu yazılımı için kullanılan install_server.bat dosyasının kodları. 64

Tablo 3.14. İstemci yazılımı için kullanılan install_client.bat dosyasının kodları. 65

Tablo 4.1. Çalışmada önerilen tüm modellerin metrikleri. 69

Tablo 4.2. FÖ yaklaşımı olmadan CIC-MalMem-2022 veri setini kullanarak malware tespiti ve malware tür tespiti yapan çalışmalarla kıyaslama tablosu. 75

Tablo 4.3. FÖ yaklaşımı ve ikili sınıflandırma ile malware tespiti yapan ve farklı veri setleri kullanan diğer çalışmalar ile kıyaslama tablosu. 77

1. GİRİŞ

Kötü amaçlı yazılımlar (Malware), geçmişte olduğu gibi günümüzde de bilgiyi işleyen cihazlar (Bilgisayarlar, akıllı cep telefonu ve saatler vb.) üzerinde bir tehdit unsuru olmuştur [1]. Bu tehdit, günden güne gelişerek, daha karmaşık ve tespit edilmesi zor bir hale gelmiştir. *Saldırgan* olarak nitelendirilebilen siber suçlular, sürekli olarak yeni teknikler ve yöntemler geliştirmekte, savunma mekanizmalarını aşmakta ve veri güvenliğini tehlikeye atmaktadırlar. Siber güvenlik uzmanları, bu tehditlere karşı sürekli olarak yeni savunma stratejileri geliştirmektedirler.

Yapay Zeka (YZ) ve Makine Öğrenimi (MÖ) tekniklerinin, kötü amaçlı yazılımların tespit edilmesi ve engellenmesinde önemli bir rolü olabileceği düşünülebilir. Özellikle Federe Öğrenme (FÖ) [2], veri mahremiyetini koruyarak, kötü amaçlı yazılımların tespit edilmesini daha etkili hale getirebilir. Verilerin merkezi bir sistem yerine, yerel olarak işlenmesi sayesinde saldırganların tek bir noktadan tüm verileri ele geçirme olasılığı azaltılabilir. Uzunca bir dönem, klasik MÖ yöntemlerinde, model oluşturma amacıyla kullanılacak olan tüm verilerin, merkezi bir cihazda toplanıp işlenmesi, yaygın bir uygulama olmuştur. Geleneksel malware tespitinde kullanılan yöntemler, merkezi bir sunucuda saklanan bir veri kümesi üzerinde gerçekleştirilmektedir. Kullanıcıların ürettiği tüm veriler, merkezi bir sunucuya gönderildiği için, bu durum veri mahremiyeti açısından büyük bir risk oluşturmaktadır.

Günümüzde, verilerin bu şekilde toplandığı birçok merkezi bilgi sistemi, saldırganların hedefi haline gelmiştir [3, 4]. Dahası, kullanıcı tarafında da bir tehdit söz konusudur. Son yıllarda, dünya üzerindeki birçok ülkede verinin mahremiyetine dair neler yapılabileceği ve bunun nasıl bir standart haline getirilebileceği ile ilgili birçok çalışma yapılmıştır. Bu riskin global bir öneme sahip olduğu ve ciddi bir şekilde ele alınması gerektiği vurgulanmıştır [5-8].

Öte yandan, verinin merkezi bir sunucuda toplanması ve analiz edilmesi süreçleri, oldukça zaman alan, sunucu hizmetleri için de yüksek bant genişliği gerektiren işlemlerdir. Çünkü kullanıcılar, geliştirilecek her model için gerekli olan tüm verilerini o merkezi sunucuya göndermek durumundadırlar. Tam aksine FÖ yöntemi, veri mahremiyetini koruması ve verilerin tek bir merkezde toplanması zorunluluğunu ortadan kaldırmasından ötürü, makine öğreniminde verilerin işlenmesi için güvenilir bir ortam

meydana getirmektedir. FÖ yaklaşımında kullanıcıların tüm verilerini merkezi bir sunucuya göndermesine gerek yoktur. Kullanıcıların kendi verileriyle oluşturdukları modellere ait belirgin bazı katsayıları göndermeleri yeterlidir. Bu yöntem hem kullanıcı verilerinin daha güvenli işlenmesine hem de korunmasına yönelik sağlam bir çözüm sunmaktadır.

Sonuç olarak kötü amaçlı yazılımların sürekli değişen ve gelişen doğası, veri güvenliği ve mahremiyeti için büyük bir meydan okumadır. Ancak gelişen teknoloji ve yenilikçi yöntemler sayesinde, bu tehditlere karşı daha güçlü savunma mekanizmaları geliştirmek de mümkündür. FÖ gibi yaklaşımların, bu mücadelenin önemli bir parçası olarak, veri güvenliğini ve mahremiyetini korumada kritik bir rol oynayacağı öngörülmektedir.

1.1. Genel Bakış

Malware, geçmişte olduğu gibi günümüz dijital dünyasında da önemli bir tehdit oluşturmaya devam etmektedir [9-11]. Bireylerin, kurumların ve hatta devletlerin bilgisayar sistemlerine sızarak veri çalma, kullanıcıları dolandırma ve sistemleri bozma gibi zararlı faaliyetleri vardır [12].

Bu tez çalışması, kötü amaçlı yazılımların oluşturduğu tehditleri, yayılma yöntemlerini ve bu tehditlerden korunma yollarını detaylı bir şekilde incelemektedir. Ayrıca modern teknolojiler ve yenilikçi yaklaşımlar ile bu tehditlere karşı nasıl etkili savunma mekanizmaları geliştirilebileceğine dair kritik çözümler sunmaktadır. Özellikle federe öğrenme gibi yeni yöntemlerin, veri güvenliği ve mahremiyetini korumada oynadığı önemli rolü vurgulamaktadır.

Bu genel bakış, okuyuculara malware konusundaki temel bilgileri sunarak, daha güvenli bir dijital ortam için farkındalık yaratmayı amaçlamaktadır.

1.1.1. Problemin Tanımı

Kötü amaçlı yazılımlar; her tür bilgisayar sistemi, akıllı veya mobil cihazlar ve ağlar üzerinde, kullanıcıların bilgilerini çalabilen, sistem performansını düşürebilen ve cihazları kullanılamaz hale getirebilen yapıdadır. Özellikle yoğun internet kullanımının olduğu dönemlerde saldırganlar bu durumu fırsat bilerek malware yazılımlarını daha geniş kitlelere yayma amacını taşırlar. Malware'in hızlı bir şekilde tespit edilip etkisiz

hale getirilmesi için, sistemlerin gerçek zamanlı olarak izlenebilmesi ve analiz edilmesi gereklidir. Bu amaçla, anlık tespit ve uyarı üreten güvenlik sistemlerinin tasarlanması çok önemlidir. FÖ tekniği ile donatılmış güvenlik sistemleri, kötü amaçlı yazılımların anlık olarak tespit edilmesinde daha etkili olabilir. Her cihazda bağımsız olarak çalışan modeller, yerel olarak kötü amaçlı aktiviteleri öğrenir. Ardından, bu bilgileri anonim ve güvenli bir şekilde paylaşarak genel güvenlik modelini (yani global modeli) sürekli olarak güncellerler. Bu yaklaşım, kötü amaçlı yazılımların anlık olarak tespit edilmesini ve engellenmesini sağlayabilir. Çünkü her cihaz kendi veri kümesi üzerinde çalışarak daha spesifik ve hızlı tespitler yapabilmektedir. Kullanıcıların güvenliğini sağlamak ve veri kayıplarını önlemek için bu tür gelişmiş ve yenilikçi güvenlik önlemleri büyük önem arz etmektedir.

1.1.2. Amaç ve Kapsam

Bu tez çalışmasındaki malware tespiti, *CIC-MalMem-2022* [13] adlı hazır bir veri seti kullanılarak gerçekleştirilmiştir. Bu veri seti, malware yazılım saldırısı sırasında, bir kullanıcının bilgisayar ortamında neleri çalıştıracağının, doğru bir örneğini temsil etmektedir.

Oluşturulan sistemde merkezi bir sunucu (Federe Öğrenme Sunucusu – FÖS) yazılımı ve yerel cihazları temsil eden bir başka yazılım bulunmaktadır. Yerel cihazların ilgili hazır veri setini kullanarak, veri ürettiği varsayılmaktadır. Gerçek dünya uygulamasında ise bu veriler anlık olarak yerel cihazların kendisinde üretilmektedir.

Bu çalışmanın kapsamı, gerçek dünya koşullarına çok yakın bir simülasyon ortamı oluşturmakla sınırlıdır. Her yerel cihaz, kendi verileriyle kendi modelini oluşturmakta ve sadece modelle ilgili bilgileri (model ağırlıklarını) FÖS'e göndermektedir. Merkezi sunucu, verileri görmeden sadece model katsayılarını işlemektedir. Böylece FÖS'ün, kendisine gönderilen modelleri birleştirerek, tüm yerel cihaz modellerini temsil eden global bir model oluşturması hedeflenmiştir. Çalışmanın temel amacı, tüm bu işlemleri gerçekleştirebilecek bir sunucu ve bir de yerel cihazı temsil eden yazılımı tasarlamaktır. Aynı zamanda bu yazılımlar ile oluşturulan modellerin, verdiği sonuçlar üzerinde yorumlar yaparak, FÖ yaklaşımının modeller üzerindeki etkisini incelemektir.

1.1.3. Tezin Özgün Değeri

Malware tespitine yönelik literatür incelendiğinde, bu alanda çok sayıda çalışma yapıldığı gözlemlenmektedir. Literatür çalışmaları bölümünde bu alandaki bazı çalışmaların ayrıntılarına değinilmiştir. Fakat tezin hazırlandığı tarihlerde, CIC-MalMem-2022 veri seti ve FÖ yaklaşımını birlikte kullanarak yapılmış hiçbir çalışma bulunmamaktadır. Ayrıca ulusal tez merkezinde yer alan tezler incelendiğinde, FÖ yaklaşımının kullanıldığı birçok çalışma tespit edilmiştir. Son üç yıla dayanan bu çalışmaların hiçbirisinde, FÖ yaklaşımına dayalı bir malware tespit çalışması da bulunmamaktadır.

Bu tez çalışması, malware tespiti yapmak isteyen araştırmacılara kaynak olabileceği gibi, genel anlamda federe öğrenmenin nasıl uygulanacağı hakkında teknik detaylar edinmek isteyen araştırmacılara da ışık tutacaktır. Ayrıca malware tespiti dışında, farklı bir alanda gerçekleştirilmesi planlanan FÖ çalışmaları için de faydalanılabilecek güncel bir bilgi kaynağı konumundadır.

1.2. Kötü Amaçlı Yazılım (Malware)

Malware, İngilizce "***mal****icious* *soft****ware***" (kötü amaçlı/niyetli yazılım) kelimelerinin kısaltmasıdır ve geniş bir yelpazede farklı türlerde yazılımlar içerir [14].

Malware; bilgisayar sistemlerine, ağlara ve veritabanlarına zarar vermek, veri hırsızlığı yapmak veya yetkisiz erişim sağlamak maksadıyla geliştirilen kötü amaçlı yazılımlardır. Farklı yayılma yöntemleriyle ortaya çıkabilmektedir. Bu yazılımlar, kullanıcıların bilgisi ve izni olmadan bilgisayarlara veya diğer dijital cihazlara sızarak çeşitli zararlı faaliyetler gerçekleştirirler.

Malware; bireysel kullanıcılar, şirketler ve devlet kurumları için ciddi sorunlara yol açabilmektedir. Veri kaybı, finansal zararlar, itibar kaybı ve operasyonel aksaklıklar gibi birçok olumsuz sonuç doğurabilmektedir. Bu nedenle, karşı korunma stratejileri geliştirmek ve bunları sürekli olarak güncellemek son derece önemlidir.

1.2.1. Malware Türleri

Teknolojinin hızla ilerlemesiyle birlikte, malware türleri de gelişerek daha karmaşık hale gelmiştir. Bu bölümde farklı malware türleri ve bunların işleyiş mekanizmaları ayrıntılı

olarak ele alınacak, ayrıca bu tehditlerle başa çıkmak için uygulanabilecek güvenlik önlemleri ve korunma stratejileri üzerinde durulacaktır.

1.2.1.1. Virüsler

Virüsler, kendilerini diğer program veya dosyalara ekleyerek çoğalabilen zararlı yazılımlardır. Bir virüs, genellikle kullanıcı tarafından çalıştırılan veya açılan bir dosya veya program aracılığıyla aktif hale gelir. Aktif hale geldiğinde, virüs kendini çoğaltır ve yayılır, sistemin performansını düşürür ve veri kaybına neden olabilir. Bazı virüsler, sistem dosyalarını bozarak işletim sisteminin çalışmasını engelleyebilirler [15, 16].

1.2.1.2. Solucanlar

Solucanlar (Worms), herhangi bir yardımcı programa ihtiyaç duymadan kendi başlarına çalışabilen ve yayılabilen bağımsız kötü niyetli yazılımlar olarak tanınırlar [17, 18]. Bu tür yazılımlar, özellikle ağlar üzerinden hızlı yayılma kapasitesine sahiptirler ve bir sistemden diğerine geçiş yaparak geniş çaplı ağ trafiği problemlerine neden olurlar.

Solucanlar, genellikle güvenlik açıklarını istismar ederek yayılma sürecini gerçekleştirirler. Ağ üzerinde hızla yayılmaları, onların etkisini ve zarar potansiyelini önemli ölçüde arttırmaktadır. Solucanların yayılma süreci, bir hedef sistemi etkilediklerinde, kendi kopyalarını o sistemde çoğaltmaları ve daha sonra bu kopyaları ağ üzerinde başka sistemlere yaymaları ile başlar. Bu yayılma, çoğu zaman kullanıcı müdahalesine gerek kalmadan otomatik bir şekilde gerçekleşir. Bu nedenle, solucanlar kısa zamanda geniş bir ağ üzerinde çok ciddi hasarlar verebilirler. Ayrıca solucanların çoğu, ağ trafiğini aşırı derecede artırarak hizmet reddi (DoS) saldırılarına neden olabilirler.

Solucanların bu tür saldırılarda kullanılması, ağ performansını düşürmekte ve hizmetlerin aksamasına yol açmaktadır. Bazı solucanlar, kullanıcı verilerini çalma, sistem dosyalarını silme veya değiştirme gibi zararlı eylemler de gerçekleştirebilirler. Geçmişte bunun birçok örneğine rastlamak mümkündür. Örneğin, Mydoom ve SQL Slammer gibi solucanlar, dünya çapında milyarlarca dolarlık zarara yol açmışlardır [19].

1.2.1.3. Truva Atları

Truva atları (Trojans), görünüşte yararlı bir program gibi davranarak, kullanıcıları yanıltan ve bilgisayar sistemlerine zarar veren kötü amaçlı yazılımlardır [20]. Bu tür yazılımlar, kullanıcılar tarafından indirildiğinde veya çalıştırıldığında, arka planda gizlice zararlı faaliyetler yürütür. Truva atları, sistemde arka kapılar (backdoor) [21] oluşturarak saldırganlara uzaktan erişim imkânı tanır. Kullanıcıların kritik verilerini çalar veya daha farklı çeşitte kötü amaçlı yazılımların yüklenmesine neden olur [22]. Bu özellikleri nedeniyle, siber güvenlik alanında çok ciddi bir tehdit olarak kabul edilmektedirler.

Truva atlarının tarihçesi oldukça eskiye dayanır. Truva ismi, Antik Yunan mitolojisindeki Truva Savaşı'nda kullanılan ünlü *Truva Atı* hilesinden gelir. Tıpkı savaşta olduğu gibi, bu kötü amaçlı yazılımlar da masum görünen bir dış görünüşün arkasına gizlenirler ve içlerinde tehlikeli unsurlar barındırırlar. Truva atlarının en büyük tehlikelerinden biri, kullanıcılara hissettirmeden sisteme zarar verebilmeleridir [23]. Bu yazılımlar, genellikle kullanıcıların pek dikkatini çekmeyen ve arka planda sessizce çalışan bazı süreçlerdir.

1.2.1.4. Fidye Yazılımları

Fidye yazılımları (Ransomwares), kullanıcıların dijital verilerini ele geçirerek, çeşitli şifreleme metotlarıyla erişilemez hale getirirler ve bu şifrelemeyi ancak belirli bir fidye miktarı karşılığında çözmeyi teklif ederler [24]. Bu zararlı yazılımların yayılma yöntemleri arasında, sosyal mühendislik saldırıları ve var olan sistem güvenlik açıklarının istismar edilmesi bulunur.

Kullanıcılar, genellikle kişisel verilerinin kalıcı olarak silineceği veya kamuya ifşa edileceği tehditleri ile fidye ödemeye zorlanırlar. Özellikle büyük kurumsal şirketlerde meydana gelen bu tür saldırılar, ciddi miktarda mali kayıplara yol açabilmekte ve şirketlerin işleyişini önemli ölçüde bozabilmektedir. Ayrıca fidye yazılımı saldırıları, şirketlerin müşteriler nezdindeki güvenilirliğini sarsarak uzun vadeli itibar kaybına da yol açabilmektedirler [25]. Bu tür olayların ortaya çıkardığı güvenlik zafiyetleri, müşteri verilerinin korunmasındaki başarısızlıkları gözler önüne serer ve potansiyel yasal sorumlulukları beraberinde getirir. Bu nedenle, şirketlerin fidye yazılımı saldırılarına karşı etkili savunma stratejileri geliştirmesi, düzenli güvenlik denetimleri yapması ve çalışanlarını bu tür tehditlere karşı bilinçlendirmesi büyük önem taşır [26]. Ancak bu

şekilde, zararlı yazılımların oluşturabileceği riskler minimize edilebilir ve organizasyonun sürdürülebilir güvenliği sağlanabilir [27].

1.2.1.5. Casus Yazılımlar

Casus yazılımlar (Spywares), bireylerin bilgisayarlarında farkında olmadan faaliyetleri takip eden ve bu verileri kötü niyetli kişilere aktaran programlardır. Bu programlar, kullanıcıların internet gezinme alışkanlıklarını, kişisel verilerini, şifrelerini ve banka hesap detaylarını dahi ele geçirebilir [28, 29]. Çoğu zaman bu tehlikeli yazılımlar, masum gözüken uygulamaların içine gizlenmiş şekilde bilgisayarlara sızar ve kullanıcıların haberi olmadan arka planda çalışmaya başlar.

Casus yazılımlar, bilgisayar kullanıcılarının en büyük korkularından biridir. Çünkü bireylerin gizliliğini tehdit ederek finansal zararlara yol açabilmektedir. Bu yazılımlar genellikle e-posta ekleri, sahte güncelleme talepleri veya zararlı web siteleri aracılığıyla bulaşabilmektedir [30, 31]. Bir kere sisteme sızdığında, sadece kişisel verileri toplamakla kalmaz, aynı zamanda bu bilgileri uzaktaki sunuculara da aktararak saldırganların bu verilere erişimini sağlar. Özellikle kişisel bilgiler ve finansal veriler gibi hassas verilerin korunması gerektiğinde, casus yazılımların tehlikeleri daha da önem kazanır. Bu yazılımlar, gizlilik ihlallerinin yanı sıra, kimlik hırsızlığı gibi suçların da failleri olabilir. Ayrıca şüpheli linklere tıklamamak ve bilinmeyen kaynaklardan gelen dosyaları açmamak, casus yazılımlara karşı korunmada etkili yöntemler arasındadır.

1.2.1.6. Zararlı Reklam Yazılımları

Zararlı reklam yazılımları (Adwares), kullanıcının rızası dışında bilgisayarlarına veya mobil cihazlarına yerleştirilerek izinsiz reklam yayınlayan yazılımlardır [32, 33]. Bu tür yazılımlar, kullanıcının internet üzerindeki gezinme alışkanlıklarını, özellikle tarayıcı geçmişini takip ederek kişiye özel reklamlar sunar. Bu reklamlar, kullanıcının daha önce ziyaret ettiği siteler veya arattığı ürünlerle ilgili olabilir, böylece daha etkili bir pazarlama stratejisi hedeflenir. Adware, bu süreçte elde ettiği verileri reklam verenlere satarak gelir elde eder.

Adware programları ilk bakışta zararlı olmayabilir ve sadece reklam gösterimi yapar gibi görünebilir. Ancak bu yazılımlar sadece reklam göstermekle kalmaz, aynı zamanda

kullanıcıların gizliliğine müdahale edebilir [34]. Kullanıcının çevrimiçi davranışlarını sürekli olarak izlemek, kişisel verilerin toplanması ve hatta bu bilgilerin üçüncü taraflarla paylaşılması gizlilik ihlallerine sebep olabilir. Bu durum, kullanıcı deneyimini olumsuz yönde etkiler ve çeşitli güvenlik risklerini beraberinde getirebilir. Öte yandan, adware'in neden olduğu diğer bir sorun da bilgisayarın performansında düşüş olmasıdır. Sürekli olarak arka planda çalışan bu tür programlar, işlemci gücünü ve belleği kullanarak sistemi yavaşlatabilirler. Ayrıca bazı adware türleri daha zararlı yazılımların bilgisayara sızmasına da aracı olabilmektedir [35].

1.2.1.7. Rootkitler

Rootkitler, kötü niyetli saldırganların bilgisayar sistemlerine derinlemesine nüfuz etmesini sağlayan ileri seviye yazılımlardır. Saldırganlara sistem üzerindeki en yüksek seviyede yetkileri, yani yönetici haklarını verir ve bu erişimi sistemin geri kalanından gizler [36].

Rootkitlerin en tehlikeli özelliklerinden bir tanesi de işletim sisteminin çekirdek seviyesinde aktif olabilmeleridir. Bu seviyede çalıştıkları için, saldırganlar sistemde fark edilmeden kalabilmekte ve istedikleri diğer zararlı yazılımları sisteme sızdırabilmektedirler [37]. Rootkitlerin bir başka tehlike arz eden özelliği ise tespit edilmelerinin ve sistemden uzaklaştırılmalarının oldukça güç olmasıdır. Çünkü bu yazılımlar, kendilerini gizlemek için işletim sisteminin temel işlevlerine derinden entegre olurlar ve genellikle antivirüs programları tarafından da tespit edilemeyebilirler.

Rootkitler, antivirüs yazılımlarını ve sistem güvenlik araçlarını etkisiz hale getirebilir, böylece kendilerini daha da gizli tutabilirler [38]. Bu zararlı yazılımların saldırganlara sağladığı gizlilik, onların uzun süreler boyunca bir sistemde kalabilmelerine ve sürekli olarak bilgi çalmalarına, izinsiz izleme yapmalarına veya diğer zararlı faaliyetlerde bulunmalarına imkan tanımaktadır. Ayrıca sistemdeki diğer güvenlik açıklarını da kullanarak daha fazla zararlı yazılımın yüklenmesine olanak sağlarlar.

1.2.1.8. Botnetler

Botnetler, siber saldırganların yönetimi altında olan ve zararlı eylemler gerçekleştirmek üzere kullanılan enfekte olmuş bilgisayar ağlarıdır [39-41]. Bu ağlar, çeşitli siber suçlar

için araç olarak kullanılabilir; bunlara örnek olarak Dağıtılmış Hizmet Reddi (DHR) saldırıları [42], istenmeyen reklam içerikli e-postaların yayılması ve kriptopara madenciliği sayılabilir. Botnet yapıları, saldırganlara, tek bir merkezden çok sayıda bilgisayarı kontrol etme ve böylece koordineli ve geniş çaplı siber saldırılar yapma imkanı verir.

Botnetler, genellikle malware kullanılarak oluşturulur. Bu yazılımlar, internet üzerinden yayılır ve bilgisayar sistemlerine sızar. Bir kez enfekte olduktan sonra, bu bilgisayarlar bot olarak adlandırılır ve bir komuta kontrol merkezi aracılığıyla uzaktan yönetilir. Bu merkez, botnet operatörünün botları tek bir noktadan kontrol etmesine ve onlara çeşitli emirler vermesine olanak tanır. Örneğin bir botnet sahibi, enfekte bilgisayarları kullanarak bir web sitesini aşırı yükleme yoluyla çevrimdışı bırakabilir veya büyük miktarda spam e-posta gönderebilir.

Distributed Denial of Service (DDoS) saldırıları [43], botnetlerin en yaygın kullanımlarından biridir. Bu tür saldırılarda, çok sayıda enfekte bilgisayar hedeflenen bir sunucuya veya ağa sürekli istekler gönderir. Bu durum, sunucunun kaynaklarını tüketir ve normal kullanıcıların hizmetlere erişimini engeller. Böyle bir saldırı, etkilenen işletmeler için ciddi kesintilere ve itibar kaybına neden olabilir.

1.2.2. Malware'in Yayılma Yöntemleri

Kötü amaçlı yazılımların yayılma yöntemleri oldukça çeşitlidir ve her geçen gün daha da sofistike hale gelmektedir. Yaygın yöntemler arasında; zararlı e-posta ekleri ve bağlantıları, zararlı yazılım güncellemeleri, güvenlik açıkları, taşınabilir bellek aygıtları ve sosyal mühendislik saldırıları bulunmaktadır. Bu yöntemler, bilgisayar korsanlarının hedefledikleri sistemlere sızmasını ve kontrol altına almasını sağlayarak, kullanıcıların hassas bilgilerini çalmalarına veya sistemleri zarara uğratmalarına olanak tanır.

1.2.2.1. E-posta Ekleri ve Bağlantıları

Kötü amaçlı yazılımlar, çoğunlukla zararlı e-posta ekleri ve bağlantıları üzerinden bulaşır [44-46]. Bilgisayar korsanları, kullanıcıların güvenini kazanmak için çeşitli sosyal mühendislik tekniklerini kullanarak bu tür zararlı içerikleri masum gibi gösterirler. Bu nedenle, kullanıcıların tanımadıkları veya güvenmedikleri kişilerden gelen e-postalardaki

ekleri açmaması ve bağlantılara tıklamaması büyük önem taşır. Ayrıca e-posta adresine gelen şüpheli içerikleri dikkatlice incelemek ve gerektiğinde bu tür mesajları silmek veya spam olarak işaretlemek de iyi bir güvenlik pratiğidir. Ek olarak, düzenli aralıklarla yazılım güncellemelerini yapmak ve güvenlik yamalarını uygulamak da bilgisayar güvenliğini artıracaktır. Bilgi güvenliği bilincinin artırılması, bu tür siber tehditlerle başa çıkmada önemli bir adımdır.

1.2.2.2. Sahte Yazılım Güncellemeleri

Sahte yazılım güncellemeleri de kötü amaçlı yazılımların yayılmasında kullanılan bir yöntemdir. Kullanıcılara sahte güncelleme bildirimleri gönderilir ve bu bildirimlere tıklayan kullanıcılar, farkında olmadan malware yazılımlarını bilgisayarlarına indirirler [47]. Bu tür siber saldırılardan korunmak için, yazılım güncellemelerini yalnızca resmi web sitelerinden veya güvenilir kaynaklardan indirmenin önemini vurgulamak gerekir. Ayrıca yazılım güncellemeleri konusunda dikkatli olmak ve her zaman doğrulama yapmadan gelen bildirimlere güvenmemek, kullanıcıların güvenliğini arttıracaktır.

1.2.2.3. Güvenlik Açıkları

Güvenlik açıkları, yazılım ve donanımlarda bulunan zayıflıkların, saldırganlar tarafından kullanılmasıyla malware yazılımların yayılmasını sağlar. Bilgisayar korsanları, bu tür açıkları bulup değerlendirerek sistemlere izinsiz giriş yapar ve zararlı yazılımları yayarlar. Bu gibi durumlara düşmemek adına, yazılım ve donanımların düzenli olarak güncellenmesi ve güvenlik yamalarının uygulanması büyük önem taşır [48].

Güvenli parola kullanımının teşvik edilmesi, kimlik doğrulama yöntemlerinin güçlendirilmesi ve şüpheli e-postalar veya bağlantılar konusunda dikkatli olunması, siber güvenlik önlemlerinin etkinliğini artırır.

1.2.2.4. Taşınabilir Bellek Aygıtları

Taşınabilir bellek aygıtları da kötü amaçlı yazılımların yayılmasında sıkça başvurulan yöntemler arasındadır. Örneğin; USB bellekler, harici sabit diskler ve diğer taşınabilir depolama aygıtları, zararlı yazılımları barındırabilir ve bu cihazlar bilgisayara bağlandığında otomatik olarak bulaşabilir [49, 50]. Bilgisayar korsanları, bu cihazlara kötü amaçlı yazılımlar yükleyerek, kullanıcıların farkında olmadan bu zararlıları

yaymalarını sağlar. Bu tür tehditlerden korunmak için, bilinmeyen veya şüpheli cihazları bilgisayarlara bağlamaktan kaçınmak önemlidir. Ayrıca güvenilir olmayan kaynaklardan gelen taşınabilir bellek aygıtlarının kullanımı da risklidir. Ek olarak, taşınabilir bellek aygıtlarının otomatik çalıştırma (autorun) özelliğini devre dışı bırakmak da, zararlı yazılımların yayılmasını önlemede etkili bir yöntemdir. Bu özellik kapatıldığında, cihazlar bağlandığında otomatik olarak çalışmaz ve kullanıcıların manuel olarak dosya açması gerekir, bu da zararlı yazılımların otomatik olarak çalışmasını engellemektedir.

1.2.2.5. Sosyal Mühendislik Saldırıları

Sosyal mühendislik saldırıları, insanların psikolojik manipülasyon yoluyla kandırılmasıyla zararlı yazılımların yayılmasını mümkün kılar. Bu tür saldırılarda, bilgisayar korsanları, kullanıcıların güvenini kazanmak ve onları aldatmak için çeşitli stratejiler kullanırlar [51, 52]. Örneğin; sahte e-postalar, telefon aramaları veya sosyal medya mesajları aracılığıyla, kullanıcıları zararlı yazılımları indirmeye veya hassas bilgilerini paylaşmaya ikna ederler.

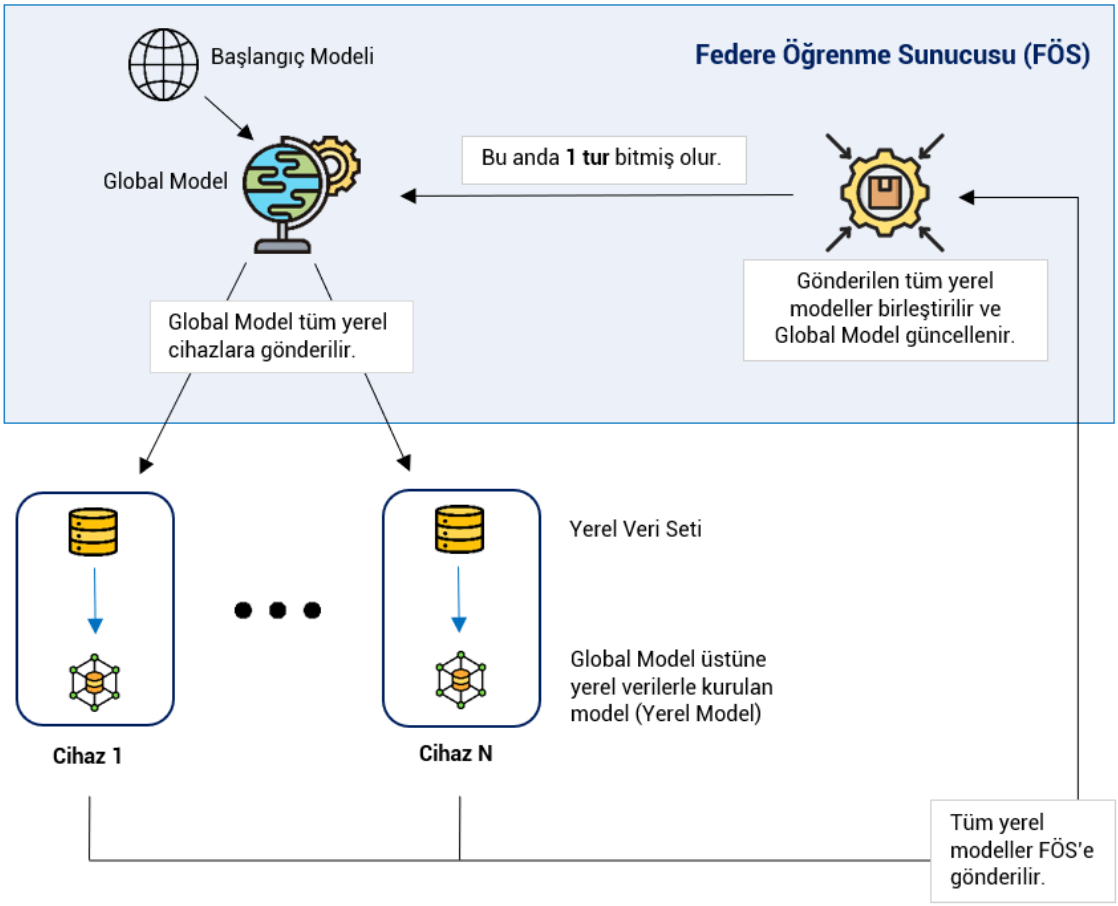
Sosyal mühendislik saldırılarının yaygın bir taktiği, kullanıcıların güvenliğini tehdit eden acil durum senaryoları oluşturmaktır. Saldırganlar, bir kullanıcının banka hesabının tehlikede olduğunu belirten bir e-posta gönderebilir ve bu durumu düzeltmek için acil olarak bir bağlantıya tıklamasını isteyebilir. Kullanıcı da paniğe kapılarak bu bağlantıya tıkladığında, zararlı yazılım bilgisayarına yüklenebilir veya kişisel bilgileri çalınabilir.

1.3. Federe Öğrenme

Federe Öğrenme olarak da bilinen "işbirliğine dayalı öğrenme", ilk kez 2016 yılında Google tarafından önerilmiştir [2, 53, 54]. Son dönemde yapılan çeşitli çalışmalar, FÖ tekniğinin makine öğrenimi alanında giderek daha yaygın ve popüler bir yöntem haline geldiğini ortaya koymaktadır [55-63].

FÖ; verilerin, üretildiği yerel cihazlarda saklanmaya devam etmesi ve bu cihazlarda eğitilen model parametrelerinin, merkezi bir sunucuya (FÖS) gönderilmesi esasına dayanmaktadır. Bu yöntem, veri mahremiyetini korumanın yanı sıra, makine öğrenmesi modellerinin daha hızlı ve doğru bir şekilde oluşturulmasına da olanak tanımaktadır.

FÖ tabanlı sistemlerde yer alan yerel cihazlar, klasik yaklaşımlara kıyasla büyük bir iş yükünü aralarında paylaşmaktadır. Başka bir ifadeyle, FÖ sistemindeki her yerel cihaz, daha büyük bir modelin belirli bir bileşenini geliştirmekte ve bu yerel modeller daha sonra FÖS üzerinde birleştirilmektedir. Bu sayede FÖ yaklaşımı, tüm işlemlerin tek bir merkezde yürütülmesi sonucu, oluşabilecek zaman ve kaynak israfını önlemektedir. Bu bağlamda, kötü amaçlı yazılımların tespitinde federe öğrenme yönteminin kullanılması, özellikle veri güvenliği açısından önemli avantajlar sunmaktadır. Temel bir FÖ sisteminin çalışması Şekil 1.1'de gösterilmiştir.



Şekil 1.1. Federe öğrenme sisteminin çalışması.

FÖ yaklaşımının kullanıldığı bir malware tespit uygulaması, kullanıcıların kendi cihazlarındaki tehditleri tespit etmelerine ve daha büyük bir modelin geliştirilmesine katkıda bulunmalarına imkan tanır. Kullanıcılar, kendi modellerini oluşturup bu modellerin parametrelerini merkezi bir sunucu olan FÖS'e iletebilirler.

Böylece FÖS, farklı kullanıcı cihazlarından gelen çeşitli tehdit bilgilerini içeren yerel modelleri işleyerek, bu alandaki potansiyel tehditlerin büyük çoğunluğunu öngörebilen ve bunları engelleyebilen bir global model oluşturabilmektedir. Global modelin parametreleri, yeniden tüm yerel cihazlara gönderilir ve böylelikle her bir yerel cihaz kullanıcısı, diğer tüm kullanıcıların deneyimlerinden, onların verilerini görmeden faydalanmış olur.

1.3.1. Federe Öğrenmenin Özellikleri

FÖ, diğer makine öğrenimi yöntemlerinden ayrılan birçok temel özellikle dikkat çekmektedir. Öncelikle, federe öğrenmenin en belirgin farkı, verilerin merkezi bir sunucuya taşınmadan, verilerin üretildiği yerel cihazlarda kalması ve burada işlenmesidir. Bu sayede, kurumlar veya bireyler, hassas ve paylaşmak istemedikleri veri kümelerini merkezileştirmeden kullanarak model eğitimi gerçekleştirebilirler [2].

Federe öğrenmede model eğitimi dağıtık bir şekilde gerçekleştirilir. Yerel cihazlarda oluşturulan modellerin parametreleri, merkezi bir sunucuya (FÖS) gönderilir. Fakat bu süreçte verilerin kendisi değil, sadece model parametreleri paylaşılır. Bu dağıtık yapı, model eğitiminin hızını artırır ve merkezi sunucunun hesaplama yükünü azaltır. Böylece merkezi sunucunun enerji tüketimi de önemli ölçüde azaltılmış olur. Bu sonuç, federe öğrenmeyi daha sürdürülebilir bir yöntem haline getirir [64].

Federe öğrenmenin bir diğer önemli özelliği, veri güvenliği ve mahremiyetini sağlamadaki üstünlüğüdür. Yerel cihazlardan gönderilen model parametreleri, FÖS'e şifrelenerek iletilir. Bu şifreleme işlemi, verilerin güvenli bir şekilde taşınmasını ve yetkisiz erişimlere karşı korunmasını sağlar. Böylece verilerin güvenliği en üst düzeyde sağlanmış olur. Ayrıca FÖS, farklı cihazlardan gelen bu şifrelenmiş parametreleri birleştirerek global bir model oluşturur. Global model, tekrar yerel cihazlara gönderilir ve böylece her bir cihaz, diğer cihazlardan öğrenilen bilgilerden faydalanabilir, ancak hiçbir zaman diğer cihazların verilerine doğrudan erişim sağlayamaz [53].

1.3.2. Federe Öğrenme Sisteminin Çalışma Prensipleri

FÖ mimarileri arasında bazı farklılıklar bulunsa da çoğu FÖ yaklaşımı genellikle aşağıdaki temel adımları izlemektedir. Bu adımlar, verilerin yerel cihazlarda işlenmesini

ve merkezi sunucuyla şifreli bir şekilde paylaşılmasını sağlayarak, dağıtık bir makine öğrenimi sürecini mümkün kılar.

Tipik bir FÖ sistemi aşağıdaki gibi çalışmaktadır [65].

I) ***Başlangıç Modelinin Oluşturulması:*** FÖS, bir başlangıç modeli oluşturur. Bu başlangıç modeli, rastgele parametrelerle de oluşturulabilir, önceden eğitilen bir modelin kullanılması şeklinde de olabilmektedir. Başlangıç modeli, federe sistemdeki tüm yerel cihazların, ortak olarak kullanacağı ve üzerinde eğitim gerçekleştireceği ilk modeldir.

II) ***Başlangıç Modelinin Dağıtımı:*** Oluşturulan ilk model, ilk olarak federe sistemdeki tüm yerel cihazlara gönderilir. Yerel cihazlar da bu gönderilen modeli bir anlamda başlangıç noktası olarak kullanırlar.

III) ***Yerel Cihazlarda Eğitim:*** Başlangıç modelini elde eden yerel cihazlar, daha sonra kendi ürettikleri veriler ile başlangıç modeli üzerinde eğitime başlarlar. Modelin parametrelerinde yerel cihazlarda üretilen verilerle yeni güncellemeler yapılır. Böylece her bir yerel cihazda yerel bir model oluşturulmuş olur.

IV) ***Yerel Modellerin FÖS'e Gönderilmesi:*** Federe sistemdeki tüm yerel cihazlarda model eğitimlerinin tamamlanmasının ardından, her yerel cihaz, sahip olduğu yerel modelin bütün parametrelerini (yani ağırlıkları ifade eden katsayıları) FÖS'e gönderir.

V) ***Global Modelin Güncellenmesi:*** FÖS, federe sistemdeki cihazlardan elde ettiği modellere ait tüm katsayıları, çeşitli birleştirme algoritmalarını kullanarak (1.3.4. başlığında ayrıntılarına yer verilmiştir) tek global modelde birleştirir.

VI) ***Güncel Global Modelin Dağıtımı:*** Tüm yerel cihazlardan elde edilen model katsayılarının birleştirilmesiyle oluşturulan güncel global modelin parametreleri, yeniden tüm yerel cihazlara gönderilir. Aslında bu madde, yukarıda belirtilmiş olan II. maddenin bir benzeridir. Bundan sonra yeniden III. maddeden devam edilerek aynı işlemler tekrarlanır. Buraya kadar gerçekleştirilen işlemler ile tam bir *Round* (Tur) tamamlanmış olmaktadır. Global modelin başarısının arttırılması için daha fazla Round'a ihtiyaç duyulabilmektedir.

Bahsi geçen ve yukarıda ayrıntılarına değinilen adımlar, sürekli tekrarlanarak, global modelin belli bir başarı seviyesine gelmesi hedeflenir. Fakat sistemin, bir zaman sonra da

durdurulması gerekmektedir. Bu sistem, bu döngüyle sonsuza dek devam ettirilemeyeceği için önceden belirlenmiş bir kriter uygulanmalıdır.

1.3.3. Federe Öğrenme Sisteminde Durdurma Kriterleri

Sürekli olarak yerel cihazlardan gelen yeni yerel modellerin gönderiminin ne zaman sonlanacağına, belli bir kritere göre karar verilebilmektedir. Böylece FÖS'ün ve yerel cihazların model eğitimleri durdurulabilir. Aşağıda FÖ sistemlerinde yaygın olarak kullanılan bazı durdurma kriterleri ve bunların ayrıntıları açıklanmaktadır.

1.3.3.1. Hedef Doğruluk Oranı

FÖ sistemlerinde en yaygın kullanılan durdurma kriterlerinden biri, belirli bir hedef doğruluk oranına ulaşmaktır. Bu kriter, global modelin doğruluk oranının önceden belirlenen bir eşiği (örneğin %99.9 gibi) yakalaması durumunda eğitimin durdurulmasını öngörür [66]. Hedef doğruluk oranı, modelin istenilen performansa ulaşmasını sağlar ve gereksiz eğitim döngülerinin önüne geçer.

Bu tez çalışmasında, hedef bir doğruluk oranına göre sistemin çalışmasının durdurulması yoluna gidilmiştir. FÖS'teki global modelin belli bir doğruluk oranını yakalayıncaya kadar FÖ sisteminin devam etmesi öngörülmüştür. Farklı model mimarileri için, farklı oranlarda doğruluk değeri ayarlaması yapılmıştır.

Hedef doğruluk oranı yaklaşımı, global modelin belirli bir performans seviyesine ulaşmasını sağlayarak, hem hesaplama kaynaklarının verimli kullanılmasını hem de eğitim sürecinin optimize edilmesini mümkün kılar. Modelin doğruluk oranına dayalı olarak durdurulması, sadece yüksek kaliteli modellerin oluşturulmasını garanti etmekle kalmaz, aynı zamanda gereksiz eğitim döngülerinin de önüne geçerek enerji ve zaman tasarrufu sağlar.

1.3.3.2. Eğitim Döngüsü (Epoch) Sayısı

Belirli bir sayıda eğitim döngüsünün (epoch) tamamlanması, başka bir durdurma kriteri olarak kullanılabilir. Bu kriter, sistemin belirli bir süre boyunca çalışmasını garanti eder ve bu süre sonunda eğitimin sonlandırılmasını sağlar. Eğitim döngüsü sayısına dayalı

durdurma, belirli bir süre zarfında elde edilen performansı değerlendirmek için kullanışlıdır [67].

1.3.3.3. Model Parametrelerinin Stabilizasyonu

Model parametrelerinin belirli bir stabiliteye ulaşması, durdurma kriteri olarak kullanılabilir [68]. Bu durumda, global model parametrelerinin değişim oranı, belirli bir eşik değerin altına düştüğünde (örneğin %0.01'den az değişimde), eğitim durdurulabilir. Bu kriter, modelin yeterince eğitildiğini ve daha fazla değişikliğin model performansına kayda değer bir katkı sağlamayacağını gösterir.

1.3.3.4. Eğitim Kaybı (Loss) Azalması

Eğitim kaybının (loss) belirli bir seviyeye inmesi veya belirli bir süre boyunca kayda değer bir azalma göstermemesi de bir durdurma kriteri olarak kullanılabilir [69, 70]. Bu durumda, eğitim kaybı belirli bir eşiğin altına düştüğünde veya belirli bir sayıda ardışık eğitim döngüsü boyunca önemli bir iyileşme göstermediğinde, eğitim durdurulabilir. Bu kriter, modelin minimum hata ile çalışmasını sağlamak için kullanılır.

1.3.3.5. Kaynak Kullanımı ve Zaman Sınırları

FÖ sistemlerinde, hesaplama kaynaklarının ve zamanın verimli kullanımı da önemlidir. Belirli bir zaman sınırının aşılması veya hesaplama kaynaklarının (örneğin CPU/GPU kullanımı, bellek tüketimi) belirli bir limitin üstüne çıkması durumunda, eğitim durdurulabilir [71]. Bu kriter, sistemin aşırı yüklenmesini ve enerji tüketiminin kontrolsüz bir şekilde artmasını önler.

1.3.3.6. Kullanıcı Geri Bildirimi ve Sistem Sağlığı

Kullanıcı geri bildirimleri ve sistem sağlığı göstergeleri (örneğin cihazların pil durumu, ağ bağlantı durumu) da durdurma kriterleri arasında yer alabilir. Kullanıcı cihazlarının pil seviyeleri kritik bir seviyeye düştüğünde veya ağ bağlantısı stabil olmadığında, eğitim süreci durdurulabilir [71]. Bu kriter, kullanıcı deneyimini ve sistem kararlılığını korumak için önemlidir.

1.3.4. Temel Federe Öğrenme Birleştirme Algoritmaları

FÖ tekniğinin başarısının kaynağında, yerel cihazlarda eğitilen modellerin parametrelerinin merkezi bir sunucuda etkili bir şekilde birleştirilmesi yatmaktadır. Bu birleştirme süreci, global modelin doğruluğunu ve genel performansını doğrudan etkileyen kritik bir aşamadır. Yazının devamında, yaygın olarak kullanılan beş federe öğrenme birleştirme algoritması ve bunların işleyişi detaylandırılmıştır.

1.3.4.1. Federe Ortalama Algoritması

Federe Ortalama (Federated Averaging – FedAvg) algoritması, en yaygın kullanılan FÖ birleştirme algoritmalarından biridir. Bu algoritma, yerel cihazlarda eğitilen model parametrelerinin ağırlıklı ortalamasını alarak global modeli güncellemektedir. FedAvg, basit ve etkili bir yöntem olup, çeşitli uygulamalarda yüksek performans göstermektedir. Bu tez çalışmasındaki yerel cihaz modellerinin birleştirilmesinde de FedAvg algoritması kullanılmıştır [2].

1.3.4.2. Federe Stokastik Gradyan İnişi Algoritması

Federe Stokastik Gradyan İnişi (Federated Stochastic Gradient Descent – FedSGD) algoritması, temel olarak Stokastik Gradyan İnişi (Stochastic Gradient Descent – SGD) algoritmasının FÖ tekniğine uyarlanmış halidir. Bu algoritmada, her bir yerel cihaz kendi veri kümesi üzerinde bir adım SGD uygular ve elde ettiği gradient güncellemelerini merkezi sunucuya gönderir. Merkezi sunucu, bu gradient güncellemelerini birleştirerek global modelin parametrelerini günceller. FedSGD, özellikle büyük veri kümeleri ve yüksek boyutlu modeller için etkili bir yöntemdir [64].

1.3.4.3. Federe Proksimal Algoritması

Federe Proksimal (Federated Proximal – FedProx) algoritması, FedAvg algoritmasının bir genellemesidir. Heterojen veri ve cihaz ortamlarında daha stabil bir eğitim süreci sağlamayı amaçlar. Bu algoritma, her bir yerel cihazın kendi model güncellemelerini hesaplarken, global modelden çok uzaklaşmamasını sağlayan bir düzenleme terimi ekler. Bu düzenleme, cihazlar arasındaki veri heterojenliğinin ve hesaplama kaynaklarındaki

farklılıkların eğitim sürecine olan olumsuz etkilerini azaltır. FedProx, cihazlar arasındaki uyumsuzlukları minimize ederek daha sağlam bir birleştirme süreci sunar [72].

1.3.4.4. Federe Gradyan Birleştirme Algoritması

Federe Gradyan Birleştirme (Federated Gradient Aggregation – FedGATE) algoritması, gradyan tabanlı bir birleştirme algoritmasıdır ve gradyanların belirli bir eşiğin altında olup olmadığını kontrol ederek birleştirme işlemini gerçekleştirir. Bu algoritma, özellikle güvenli ve gizlilik odaklı uygulamalarda kullanılır. FedGATE, gradyanların merkezileştirilmesi sırasında oluşabilecek potansiyel bilgi sızmalarını minimize etmek için tasarlanmıştır. Bu algoritma, gradyan güncellemelerinin hassas verileri açığa çıkarmasını önler ve güvenli bir birleştirme süreci sunar [73].

1.3.4.5. Federe Eşleştirilmiş Ortalama Algoritması

Federe Eşleştirilmiş Ortalama (Federated Matched Averaging – FedMA) algoritması, FedAvg algoritmasının daha sofistike bir versiyonudur. Bu algoritma, yerel modellerdeki katmanları ve nöronları eşleştirerek birleştirme işlemini gerçekleştirir. FedMA, özellikle derin öğrenme modellerinde katmanlar arası uyumu sağlayarak daha etkili bir birleştirme süreci sunar. Yerel modellerdeki nöronların benzerliğine dayanarak eşleştirme yapan bu algoritma, heterojen veri ortamlarında daha yüksek performans ve doğruluk sağlar [74].

1.3.5. Federe Öğrenmenin Avantajları

FÖ tekniğinin avantajları, yalnızca veri mahremiyeti ve model doğruluğu ile sınırlı değildir. Bu yöntemin bir diğer önemli avantajı, merkezi sunucu üzerindeki hesaplama yükünü azaltarak sistem performansını artırmasıdır. FÖ sayesinde çeşitli cihazlardan toplanan veriler, merkezi bir veri havuzu oluşturulmadan analiz edilebilir, bu da özellikle veri depolama maliyetlerini düşürmektedir [64].

Federe öğrenmenin enerji verimliliği konusundaki avantajları da göz ardı edilemezdir. Enerji verimliliği ve sürdürülebilirlik, günümüzde giderek daha fazla önem kazanan konular arasındadır [75-78]. Özellikle IoT cihazlarının sayısının hızla arttığı ve bu cihazların enerji tüketiminin büyük bir sorun haline geldiği bir dünyada, FÖ tekniğinin sunduğu enerji verimliliği çözümleri kritik öneme sahiptir. FÖ, merkezi sunucuya yapılan veri transferlerini minimize etmesi bakımından, hem enerji tasarrufu sağlar hem de veri

transferi sırasında oluşan karbon ayak izini azaltır. Bu durum, büyük veri merkezlerinin enerji tüketiminin ve çevresel etkilerinin azaltılması açısından da önemli bir adımdır. FÖ'nün bu sürdürülebilir yaklaşımı, çevresel sorumluluk taşıyan şirketler ve kurumlar için de cazip bir seçenek olarak öne çıkmaktadır. Enerji verimliliği ve sürdürülebilirlik, uzun vadede maliyet tasarrufu sağlamaktadır. Bu sonuç da FÖ'nün benimsenmesini teşvik eder. Federe öğrenmenin, enerji verimliliği konusundaki bir diğer önemli avantajı, cihazların pil ömrünü uzatmasıdır. Geleneksel merkezi veri işleme yöntemlerinde, cihazlar sürekli olarak büyük miktarda veri gönderip almak zorunda kalırken, FÖ sisteminde veri işleme yerel olarak gerçekleştirilir. Böylece veri transferi sırasında oluşan enerji tüketimi azaltılarak cihazların pil ömrü uzatılmış olur. Özellikle cep telefonu ve taşınabilir IoT gibi pil ömrünün kritik olduğu elektronik cihazlarda, federe öğrenmenin bu avantajı kullanıcı deneyimini ve cihazların genel performansını olumlu yönde etkiler.

Federe öğrenmenin bir diğer avantajı ise, çok çeşitli veri kaynaklarından elde edilen bilgilerin entegre edilmesine olanak tanımasıdır. Farklı cihazlardan gelen veriler, merkezi bir modelde birleştirilerek daha geniş kapsamlı ve daha doğru tahminler yapılabilir. Özellikle sağlık hizmetleri, akıllı şehirler ve endüstriyel uygulamalar gibi alanlarda büyük bir potansiyele sahiptir. Örneğin sağlık hizmetlerinde, hastaların verileri yerel cihazlarda tutulabilir ve bu veriler anonimleştirilmiş model parametreleri aracılığıyla merkezi bir sunucuya iletilerek daha doğru teşhis ve tedavi modelleri oluşturulabilir. Bazı hastalıkların önceden tahmin edilmesi ile önlemler arttırılabilir [79].

Akıllı şehirlerde ise çeşitli sensörlerden toplanan veriler analiz edilerek; trafik yönetimi, enerji tüketimi optimizasyonu ve kamu güvenliği gibi konularda daha etkin çözümler sunulabilir [80].

1.3.6. Federe Öğrenmenin Uygulama Alanları

FÖ, sağlık hizmetlerinden finans sektörüne, akıllı şehirlerden otonom araçlara kadar geniş bir yelpazede uygulama alanı sunmaktadır. Aşağıda federe öğrenmenin en temel uygulama alanları ve bu alanlardaki kullanımı ele alınmıştır. Sağlık hizmetleri, federe öğrenmenin en çok potansiyel vaat ettiği alanlardan biridir. FÖ, hastaların hassas sağlık verilerini merkezileştirmeden analiz edebilme yeteneği sayesinde veri gizliliğini korurken, geniş kapsamlı ve doğru teşhis modelleri geliştirilmesine olanak tanır [81-83].

Finans sektöründe, müşteri verilerinin gizliliği ve güvenliği büyük önem taşır. FÖ, bankaların ve finans kurumlarının müşterilere ait hassas verileri paylaşmadan kredi risk değerlendirmesi, dolandırıcılık tespiti ve müşteri segmentasyonu gibi işlemleri gerçekleştirmesine olanak tanır [84-86].

Akıllı şehirlerde ve nesnelerin interneti (Internet of Things – IoT) olarak adlandırılan cihazlarda büyük miktarda veri üretilmekte ve bu verilerin merkezi bir noktaya taşınması hem maliyetli hem de zaman alıcı olabilmektedir. FÖ, bu sorunun üstesinden gelerek, verilerin taşınması yerine model parametrelerinin ilgili sunucuya gönderilmesini sağlar. Böylece hem maliyet hem zaman tasarrufu elde edilmiş olur. Literatürde akıllı şehirler ve IoT cihazlarında federe öğrenmenin uygulanmasıyla ilgili birçok yayınlanmış çalışma bulunmaktadır [87-89].

Otonom araçlar, sürekli olarak veri toplar ve bu verilerin analizi, güvenli sürüş ve yol optimizasyonu için önemlidir. FÖ, bu verilerin yerel olarak işlenmesine imkan tanıyarak, veri gizliliğini korumakta ve merkezi veri işlemeye kıyasla daha hızlı ve verimli çözümler sunabilmektedir. Bu alanda da federe öğrenmenin kullanıldığı birçok çalışma gerçekleştirilmiştir [90-93].

Siber saldırıların önlenmesi noktasında da FÖ, hızlı ve sağlam çözümler sunmaktadır. Farklı kurum ve kuruluşların siber saldırı tespiti ve önleme sistemlerini birbirleriyle veri paylaşmadan geliştirmelerine imkan tanır. Bu yöntem, kurumların birbirleriyle hassas veri paylaşımı yapmadan, siber güvenliklerini artırmalarını sağlar. FÖ ile geliştirilen siber saldırı önleme sistemleri, sürekli olarak güncellenir ve genişleyen bir "tehdit veri tabanı" oluşturarak, yeni ortaya çıkan saldırı türlerine karşı daha hızlı ve etkili bir savunma mekanizması sunar [94-96].

1.3.7. Federe Öğrenmenin Uygulanışındaki Bazı Zorluklar

Federe öğrenmenin birçok avantajı olmasına rağmen, bu tekniğin geniş ölçekte doğru bir şekilde uygulanabilmesi, çeşitli zorlukları da beraberinde getirmektedir. Bu zorluklar, sistemin verimliliğini etkileyebilen, göz ardı edilmemesi ve çözümler geliştirilmesi gereken alanlardır.

Aşağıda federe öğrenmenin uygulanışındaki bazı temel zorluklar ve bu zorlukların literatürdeki yansımaları ele alınmaktadır.

1.3.7.1. Veri Heterojenliği

FÖ sisteminde, farklı cihazlar ve kullanıcılar tarafından üretilen veriler genellikle heterojendir. Veri heterojenliği, verilerin dağılımında, örnek sayısında ve veri kalitesinde büyük farklılıklar meydana getirebilmektedir. Bu da oluşturulacak olan global modelin, genel performansını olumsuz etkilemekte ve modelin tüm veri çeşitliliğini kapsamasını zorlaştırabilmektedir.

Bazı yerel cihazlar, diğer yerel cihazlara nazaran, daha fazla veya daha farklı çeşitte verilere sahip olabilirler. Bu durumda, bu yerel cihazlar, global modeli kendi yerel modeline daha fazla yaklaştırabilmektedir. Bir başka ifadeyle, ilgili yerel cihaz, global modeli manipüle edecek güce erişmiş demektir. Eğer kurulacak olan bir FÖ sisteminde, veri heterojenliği varsa, bu etkenin de göz önünde bulundurularak, çözüme kavuşturulması gerekmektedir. Literatürdeki birçok makalede, bu durumu anlatan çalışmalar bulunmaktadır [97-99].

1.3.7.2. İletişim Verimliliği

FÖ sürecinde, yerel cihazlar ve merkezi sunucu arasında sık sık model güncellemeleri ve parametre transferleri yapılmaktadır. Bu durum, özellikle büyük ölçekli sistemlerde ve sınırlı bant genişliği olan ortamlarda iletişim maliyetlerini artırabilmektedir. Bu olumsuzluğun önüne geçilmesi amacıyla, parametre sıkıştırma ve azaltma teknikleri gibi birçok yöntem geliştirilmiştir [100-104].

1.3.7.3. Hesaplama Gücü

FÖ, yerel cihazların hesaplama gücüne dayanır. Fakat bu cihazlar, genellikle sınırlı hesaplama kapasitesine ve enerji kaynaklarına sahiptir. Bu durum, özellikle büyük ve karmaşık modellerin eğitimi sırasında zorluklar çıkartabilmektedir.

Yurochkin ve arkadaşlarının çalışması, sınırlı kaynaklarla federe öğrenme yapmanın zorluklarını ve bu zorlukların üstesinden gelmek için geliştirilen yöntemleri tartışmıştır [74].

Ek olarak literatürde bulunan birçok çalışmada, FÖ sistemlerindeki hesaplama gücünün etkilerinden bahsetmektedir [105-107].

1.3.7.4. Gizlilik ve Güvenlik

FÖ, veri gizliliğini çok etkili bir şekilde korurken, yerel veya global model parametrelerinin paylaşımı sırasında hala gizlilik ve güvenlik riskleri barındırmaktadır. Bu riskler, tersine mühendislik ile tehlikeli hale gelebilmektedir.

Tersine mühendislik, bir modelin parametrelerinden hareketle, modelin eğitildiği verilerin özelliklerini çıkarmaya yönelik yöntemleri ifade eder. FÖ sistemlerinde, yerel cihazlarda eğitilen modellerin parametreleri merkezi sunucuya gönderildiğinde, bu parametrelerin analiziyle orijinal verilere dair ipuçları elde edilebilmektedir. Örneğin, model parametrelerinin dikkatli bir şekilde incelenmesi, belirli kullanıcıların verilerinin yapısını veya istatistiksel özelliklerini ortaya çıkarabilmektedir.

Bu yüzden, FÖ sistemlerinin güvenliği azami düzeyde artırılmalıdır. Literatürde, bu tür saldırılara karşı çeşitli önlemler geliştirilmiştir [108-111].

1.3.7.5. Altyapı Gereksinimleri

FÖ sistemlerinin başarılı bir şekilde uygulanabilmesi için sağlam bir ağ ve bilgi işlem altyapısına ihtiyaç vardır. Sistem içerisindeki yerel cihaz sayısı arttıkça, merkezi sunucu üzerindeki ağ trafiğinin yükü de artmaktadır. Ayrıca kurulacak modellerin nöron ve katman sayılarındaki artış, merkezi sunucuya gönderilecek olan bilginin miktarını da arttırmaktadır.

Model eğitiminin binlerce cihaz arasında koordine edildiği düşünülecek olursa, bu yük hissedilir derecede kendini gösterecektir. Bunun bir sonucu olarak; yüksek bant genişliği, düşük gecikme süreleri ve güvenilir ağ bağlantıları gereksinimleri daha da artacaktır. Bu gereksinimler, özellikle büyük ölçekli FÖ sistemlerinde altyapı maliyetlerini arttırabilmekte ve sistemin uygulanabilirliğini zorlaştırabilmektedir [112, 113].

1.4. Literatür Çalışmaları

Malware tespitine yönelik literatür incelendiğinde, bu alanda çok sayıda çalışma yapıldığı gözlemlenmektedir. Bu çalışmalar temelde iki tür olarak sayılabilir. İlk türdeki çalışmalar, CIC-MalMem-2022 veri setini FÖ yaklaşımını kullanmaksızın ele alarak, malware tespiti yapan çalışmalardır. Diğer türdeki çalışmalar ise FÖ yaklaşımını benimseyerek malware tespiti gerçekleştiren, fakat CIC-MalMem-2022 veri setini

kullanmayan çalışmalardan oluşmaktadır. Dolayısıyla hem CIC-MalMem-2022 veri seti hem de FÖ yaklaşımını kullanarak yapılmış hiçbir çalışma bulunmamaktadır.

Maya Hilda Lestari Louk ve arkadaşları [114]; BODMAS, Kaggle ve CIC-MalMem-2022 gibi birçok veri setini kullanarak gerçekleştirdikleri malware tespiti çalışmasında, çeşitli ağaç tabanlı topluluk öğrenme yöntemlerini karşılaştırmışlardır. Her veri seti için birçok ağaç tabanlı makine öğrenmesi algoritması test edilmiştir. CIC-MalMem-2022 veri seti için ise CatBoost (CB), XGBoost (XGB), LightGBM (LGBM), Random Forest (RF) ve Gradient Boosting Model (GBM) gibi birçok algoritmanın test edildiği çalışmada en iyi sonucu RF algoritmasının verdiği tespit edilmiştir. Accuracy, Presicion ve F1 metriklerinde %100, Recall'da ise %99.99 başarı oranının elde edildiği görülmüştür.

Talukder ve arkadaşları [115], ağ saldırı tespit sistemleri üzerine gerçekleştirdikleri çalışmada makine öğrenimi ve derin öğrenme tekniklerini birleştirerek hibrit bir model önermişlerdir. Synthetic Minority Oversampling TEchnique (SMOTE) ile veri dengeleme ve XGB ile baskın özelliklerin çıkarılması sağlanmıştır. KDDCUP'99 [116] ve CIC-MalMem-2022 veri setlerinin kullanıldığı çalışmada; overfitting, type-1 ve type-2 gibi sorunlar yaşanmadan sırasıyla %99.99 ve %100 başarı oranları elde etmişlerdir.

Daryle Smith ve arkadaşları [117], gözetimli ve gözetimsiz öğrenme algoritmalarını kullandıkları çalışmada malware tespiti yapmışlardır. Bu amaçla çalışmada, Malware-Exploratory ve CIC-MalMem-2022 şeklinde iki ayrı veri kümesi kullanılmıştır. K-Means, Density-Based Spatial Clustering of Applications with Noise (DBSCAN) ve Gaussian Mixture Model (GMM) gibi kümeleme algoritmaları ve Decision Tree (DT), KNeighbors (kNN), RF, Stokastik Gradient Descent (SGD), Ada Boost (AB), Extra Trees (ET), Gaussian Naïve Bayes (GNB) gibi toplamda yedi adet sınıflandırma algoritmasını kullanmışlardır. Her iki veri seti için de Ada Boost algoritması en iyi sonuçları vermiştir. Malware-Exploratory veri seti için %100 ve CIC-MalMem-2022 için %99.99 doğruluk oranları elde edilmiştir.

Mezina ve arkadaşları [118], bellekte gizlenen kötü amaçlı yazılımı tespit etme amaçlı bir çalışma gerçekleştirmişlerdir. Çalışmada CIC-MalMem-2022 veri seti kullanılarak, öncelikle geleneksel makine öğrenimi yöntemleri test edilmiş ve ardından Genişletilmiş Evrişimli Sinir Ağı modeli önerilmiştir. Malware tespitinde tüm yöntemlerin doğruluk oranı yaklaşık 0.99 bulunmuştur. Bu yöntemlerden en iyisinin RF olduğu görülmüştür.

Malware türü tespiti için ise en iyi yöntemin, önerilen sinir ağı mimarisi olduğu belirlenmiştir. Malware sınıflandırması için kullanılan sinir ağının doğruluk oranı 0.83 olarak tespit edilmiştir.

M. Dener ve arkadaşları [119], bellek verilerinin kullanımı üzerinden hareketle malware tespiti çalışması gerçekleştirmişlerdir. Çok çeşitli makine öğrenimi ve derin öğrenme yaklaşımlarının denendiği çalışmada, büyük veri platformlarından da faydalanılmıştır. Google Colab platformunda, PySpark büyük veri yönetim aracı ile CIC-MalMem-2022 veri seti kullanılmıştır. RF, DT, Gradient Boosted Tree (GBT), Logistic Regression (LR), Naive Bayes (NB), Linear Vector Support Machine (LVSM), Multilayer Perceptron (MLP), Deep Feed Forward Neural Network (DFFNN) ve Long Short-Term Memory (LSTM) algoritmaları kullanılarak ikili sınıflandırma yapılmıştır. Kötü amaçlı yazılım tespitinde LR algoritması için 99.97, GBT için 99.94 ve NB için 98.41 doğruluk oranları elde edilmiştir.

Kuang-Yao Lin ve arkadaşları [120], VirusTotal tarafından sağlanan ve 10907 kayıt içeren bir veri seti ile bir malware sınıflandırma çalışması gerçekleştirmişlerdir. SVM ve LSTM makine öğrenmesi modellerinin federe öğrenme ve federe öğrenme olmadan kurgulandığı çalışmada toplam 4 adet model test edilmiştir. En yüksek doğruluk oranı 0.9167 olarak sağlanmıştır.

V. Rey ve arkadaşları [121], gözetimli ve gözetimsiz öğrenme modelleri için gerçekleştirdikleri IoT malware tespiti çalışmasında, federe öğrenme yaklaşımının veri gizliliğini sağlarken, modelin performansından da ödün vermediğini göstermişlerdir. Çalışma içerisinde ağ trafiğini modelleyen N-BaIoT adlı veri seti kullanılmıştır. Birçok farklı modelin test edildiği çalışmada hesaplanan tüm metriklerde (Accuracy, TPN, TPR) %97 üzerinde bir başarı sağlanmıştır.

Ruei-Hau Hsu ve arkadaşları [122], Opera Mobile Store'dan toplanmış olan 87182 adet APK dosyası ile SVM algoritması ve FÖ yaklaşımını kullanarak, malware tespiti yapmışlardır. Eğlence, Sağlık, Oyun, Seyahat, E-book vb. gibi toplam 12 kategoride uygulamalar incelenmiştir. Birçok farklı senaryonun test edildiği çalışmada, 5 kullanıcı ile yapılan federe öğrenme yaklaşımlı SVM ve verilerin merkezileştirildiği klasik yaklaşım karşılaştırılmıştır. Federe yaklaşımın doğruluk oranı %93.45 olarak, klasik yaklaşımdan düşük çıkmıştır. Klasik yaklaşımdaki doğruluk oranı ise %94.05 çıkmıştır.

Doğruluk oranları açısından incelendiğinde daha düşük çıkmasına rağmen; precision, recall, fpr ve f-measure metriklerinde FÖ yaklaşımı, klasik yaklaşıma göre daha yüksek değerlerde seyretmiştir.

Rafa Gálvez ve arkadaşları [123], Less is More (LiM) adını verdikleri kötü amaçlı yazılım sınıflandırma çerçevesini kullanarak, federe öğrenme yaklaşımı ile klasik sınıflandırmaya göre daha performanslı bir sonuç elde ettiklerini göstermişlerdir. Çalışmada 200 kullanıcı ile 25 bin temiz uygulama ve 25 bin kötü amaçlı uygulama teste katılmıştır. MaMaDroid adlı veri setinin kullanıldığı çalışmada, başarı metriği olarak F1 skorun %95 olarak hesaplandığı görülmüştür.

Changnan Jiang ve arkadaşları [124], gerçekleştirdikleri çalışmada, Android cihazlardaki kötü amaçlı yazılımları, federe öğrenmeye dayalı bir şekilde sınıflandıran, FedHGCDroid adlı yeni bir sınıflandırma şeması önermişlerdir. Öncelikle kötü amaçlı yazılım sınıflandırması için, malware yazılımların davranış özelliklerini çıkartan HGCDroid isimli, CNN ve Grafik Sinir Ağı (Graphical Neural Network – GNN) kullanmışlardır. Androzoo veri kümesi ile yapılan testlerde, FedHGCDroid'in diğer yöntemlere göre daha fazla uyarlanabilirlik ve daha yüksek doğruluk elde ettiği tespit edilmiştir. Başarı metrikleri incelendiğinde, %91.3 doğruluk oranı ve %91.25 F1 skor ile malware tespitini gerçekleştirdikleri görülmüştür.

Taheri ve arkadaşları [125], malware tespiti için Fed-IIoT adını verdikleri, katılımcı ve sunucu taraflı iki bileşenden oluşan, federe öğrenme yaklaşımlı Android mimarisi önermişlerdir. Üç ayrı veri setinin kullanıldığı çalışmada, en iyi sonucun alındığı veri seti Genome olmuştur ve kullanılan FedGAN modellerinin (FedGAN-BM ve FedGAN-BK), doğruluk oranları %89.51 ve %93.24 şeklinde ölçülmüştür.

Ulusal tez merkezinde yer alan tezler incelendiğinde, FÖ tekniğinin kullanıldığı altı adet yüksek lisans tez çalışması [126-131] tespit edilmiştir. Son üç yıla dayanan bu çalışmaların hiçbirisinde FÖ yaklaşımına dayalı bir malware tespit çalışması bulunmamaktadır.

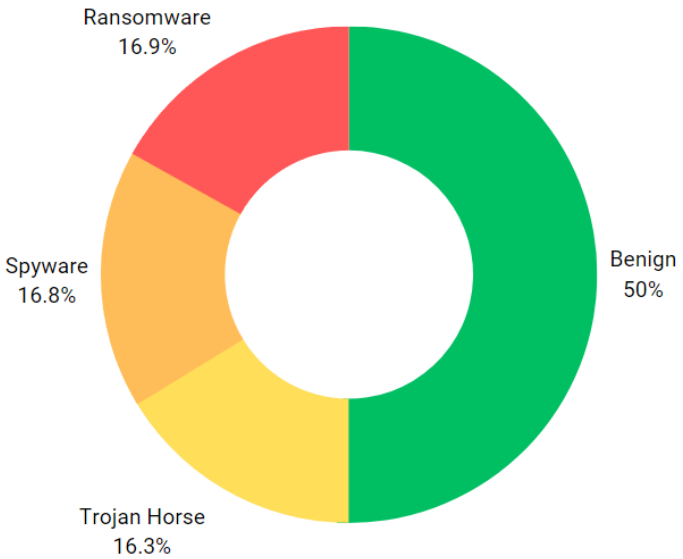
2. MATERYAL VE YÖNTEM

Bu bölümde, tez çalışmasında kullanılan veri seti ve oluşturulan modellerde kullanılan teknik yöntemler hakkında bilgiler verilmiştir.

2.1. Veri Seti

Bu çalışmada kullanılan temel veri kaynağı, Kanada Siber Güvenlik Enstitüsü tarafından hazırlanan *CIC-MalMem-2022* [13] adlı veri setidir. 2022 yılında yayınlanan bu veri seti, günümüzde malware tespiti yapmak için kullanılabilecek en güncel kaynaklardan biridir. Güncel bir kaynağın kullanılması, gerçekleştirilen çalışmanın değerini önemli ölçüde arttırmaktadır. Söz konusu veri seti, gerçek dünyada yaygın olarak kullanılan belirli kötü amaçlı yazılımların "bellekteki etkilerini" mümkün olduğunca gerçekçi bir biçimde yansıtan değerli bir kaynaktır. Bu veri seti, kötü amaçlı yazılım saldırısı sırasında bir kullanıcının bilgisayar ortamında (işletim sisteminde) çalıştırabileceği süreçlerin doğru bir örneğini sunmaktadır. Toplamda 58296 kayıttan oluşan bu veri setinde, dengeli bir dağılımla, her biri 29298 kayıt içeren iyi huylu (malware sayılmayan, benign) ve kötü huylu (malware sayılan) veri bulunmaktadır.

Ayrıca kötü huylu olarak işaretlenmiş olan veriler de kendi aralarında üç çeşit malware kategorisine (Trojan Horse [132], Spyware [133], Ransomware [134, 135]) ayrılmış durumdadır. İyi huylu ve diğer kötü huylu üç kategorinin, tüm veri setindeki miktarlarına ait oranlar Şekil 2.1'de verilmiştir.



Şekil 2.1. CIC-MalMem-2022 veri setindeki malware türleri.

2.1.1. Veri Setinin Özellikleri

CIC-MalMem-2022 veri seti toplamda 57 öznitelik içermektedir. Bu öznitelikler genel olarak; İşletim Sistemi (İS) içerisinde çalıştırılan süreçler (process), cihazdaki anlık bellek durumu, oturumlar ve İS tarafından oluşturulup çağrılan, belirli işlevleri (fonksiyonları) yerine getiren yapılar (handler, event vb. [136, 137]) hakkında bilgiler sunmaktadır.

Veri setinde bulunan öznitelikler, sırasına uygun olarak ve ayrıntılı açıklamaları ile birlikte Tablo 2.1'de sunulmuştur.

Tablo 2.1. CIC-MalMem-2022 veri setinde bulunan tüm özniteliklerin listesi.

No	Öznitelik	Özniteliğin Açıklaması
1	*Category*	Malware Türleri veya İyi Huylu (Benign) kategorisi ile birlikte toplam dört farklı değer içerir.
2	*pslist.nproc*	Çalıştırılan toplam süreç sayısı.
3	*pslist.nppid*	Çalıştırılan toplam ebeveyn süreç sayısı.
4	*pslist.avg_threads*	Her işlem için oluşturulan ortalama iş parçacığı (thread) sayısı.
5	*pslist.nprocs64bit*	Çalıştırılan toplam 64-bit işlem sayısı.
6	*pslist.avg_handlers*	İS tarafından çağrılan ortalama handler sayısı.
7	*dllist.ndlls*	Tüm işlemler için yüklenen toplam kütüphane sayısı.
8	*dllist.avg_dlls_per_proc*	Her süreç için yüklenen ortalama kütüphane sayısı.
9	*handles.nhandles*	Toplam handler sayısı.
10	*handles.avg_handles_per_proc*	Her süreç için açılan ortalama İS işlevi sayısı.
11	*handles.nport*	İS tarafından oluşturulan toplam port işlevi sayısı.
12	*handles.nfile*	İS tarafından oluşturulan toplam dosya işlevi sayısı.
13	*handles.nevent*	İS tarafından oluşturulan toplam event işlevi sayısı.
14	*handles.ndesktop*	İS tarafından oluşturulan toplam masaüstü işlevi sayısı.
15	*handles.nkey*	İS tarafından üretilen toplam anahtar işlevi sayısı.
16	*handles.nthread*	İS tarafından oluşturulan toplam thread işlevi sayısı.
17	*handles.ndirectory*	İS tarafından oluşturulan toplam dizin işlevi sayısı.
18	*handles.nsemaphore*	İS tarafından oluşturulan toplam semafor işlevi sayısı.
19	*handles.ntimer*	İS tarafından oluşturulan toplam zamanlayıcı sayısı.
20	*handles.nsection*	İS tarafından oluşturulan toplam bölüm işlevi sayısı.
21	*handles.nmutant*	İS tarafından oluşturulan toplam mutant işlevi sayısı.
22	*ldrmodules.not_in_load*	Yükleme listesinde eksik olan toplam modül sayısı.
23	*ldrmodules.not_in_init*	Başlatma listesinde eksik olan toplam modül sayısı.
24	*ldrmodules.not_in_mem*	Bellek listesinde eksik olan toplam modül sayısı.
25	*ldrmodules.not_in_load_avg*	Yükleme listesinde eksik olan ortalama modül sayısı.
26	*ldrmodules.not_in_init_avg*	Başlatma listesinde eksik olan ortalama modül sayısı.
27	*ldrmodules.not_in_mem_avg*	Bellek listesinden eksik olan ortalama modül sayısı.
28	*malfind.ninjections*	Tespit edilen gizli kod enjeksiyonlarının toplam sayısı.

29	*malfind.commitCharge*	Toplam commit charge sayısı.
30	*malfind.protection*	Toplam koruma sayısı.
31	*malfind.uniqueInjections*	Toplam benzersiz enjeksiyon sayısı.
32	*psxview.not_in_pslist*	PsList içinde olmayan İS süreçlerinin toplam sayısı.
33	*psxview.not_in_eprocess_pool*	PsScan içinde olmayan İS süreçlerinin toplam sayısı.
34	*psxview.not_in_ethread_pool*	ThrdProc içinde olmayan İS süreçlerinin toplam sayısı.
35	*psxview.not_in_pspcid_list*	PspCid içinde olmayan İS süreçlerinin toplam sayısı.
36	*psxview.not_in_csrss_handles*	Csrss içinde olmayan İS süreçlerinin toplam sayısı.
37	*psxview.not_in_session*	Oturum içinde olmayan İS süreçlerinin toplam sayısı.
38	*psxview.not_in_deskthrd*	Deskthrd içinde olmayan İS süreçlerinin toplam sayısı.
39	*psxview.not_in_pslist_false_avg*	PsList'in ortalama yanlış oranı.
40	*psxview.not_in_eprocess_pool_false_avg*	PsScan'in ortalama yanlış oranı.
41	*psxview.not_in_ethread_pool_false_avg*	ThrdProc'in ortalama yanlış oranı.
42	*psxview.not_in_pspcid_list_false_avg*	Süreç kimliğinin ortalama yanlış oranı.
43	*psxview.not_in_csrss_handles_false_avg*	Csrss işlevlerinin ortalama yanlış oranı.
44	*psxview.not_in_session_false_avg*	Oturumun ortalama yanlış oranı.
45	*psxview.not_in_deskthrd_false_avg*	Deskthrd'in ortalama yanlış oranı.
46	*modules.nmodules*	Toplam modül sayısı.
47	*svcscan.nservices*	Toplam hizmet sayısı.
48	*svcscan.kernel_drivers*	Toplam çekirdek sürücüleri sayısı.
49	*svcscan.fs_drivers*	Toplam dosya sistemi sürücüleri sayısı.
50	*svcscan.process_services*	Toplam Windows 32 sahipli İS süreçleri.
51	*svcscan.shared_process_services*	Toplam Windows 32 paylaşımlı İS süreçleri.
52	*svcscan.interactive_process_services*	Toplam etkileşimli hizmet İS süreçleri.
53	*svcscan.nactive*	Aktif çalışan hizmet İS süreçlerinin toplam sayısı.
54	*callbacks.ncallbacks*	Toplam geri çağırım sayısı.
55	*callbacks.nanonymous*	Toplam bilinmeyen süreç sayısı.
56	*callbacks.ngeneric*	Toplam genel süreç sayısı.
57	*Class*	İyi huylu veya kötü kuylu şeklinde iki değer içerir.

Bu tez çalışmasında geliştirilen bazı modeller, belirtilen öznitelikleri kullanarak kötü amaçlı yazılım tespitine (İkili Sınıflandırma) yönelik çalışırken, bazı modeller ise kötü amaçlı yazılımın türünü (Çoklu Sınıflandırma) tahmin etmeye yöneliktir.

2.1.2. Veri Setinin Önişleme Adımları

CIC-MalMem-2022 veri seti, çalışmada kullanılmadan önce bazı önişlemlerden geçirilmiştir. Bu önişlemlerin amacı, ikili ve çoklu sınıflandırma için ayrı ayrı kullanılabilecek iki adet veri seti elde etmek olmuştur. Öncelikle tüm değerleri aynı olan değişkenler, orijinal veri setinden çıkarılmıştır. Çünkü bir özniteliğin tüm değerlerinin aynı olması, oluşturulacak olan makine öğrenimi modeline herhangi bir fayda sağlamamaktadır. Bu sebeple veri setinden çıkarılan öznitelikler; *pslist.nprocs64bit*,

handles.nport ve *svcscan.interactive_process_services*'dir. Bu işlemle birlikte veri setindeki toplam öznitelik sayısı 54'e düşmüştür. Ardından veri setinde sonsuz değere sahip bir verinin olup olmadığı, boş bırakılmış (missing value) bir satır olup olmadığı gibi, bazı temel veri analizi kontrolleri gerçekleştirilmiştir.

Ardından *Category* kolonunda bulunan değerlerdeki sayılar ve malware ile ilgili diğer kullanılmayacak olan detaylar kaldırılarak, ilgili kolonda malware ailelerinin sadece isimlerinin gözükmesi sağlanmıştır. Örneğin *Category* kolonunda "***Ransomware-Shade-fa03be3078d1b9840f06745f160eb...***" biçiminde yer alan bir değer, sadece "*Ransomware*" olacak şekilde yeniden düzenlenmiştir. *Class* özniteliğinde Label Encoding gerçekleştirilerek; "Benign" değerleri 0, Malware değerleri ise 1'e dönüştürülmüştür. Bu noktada orijinal veri seti birçok açıdan değişikliğe uğratılmıştır. Bundan sonra gerçekleştirilen önişlemlerde orijinal veri setinin bu en son, biraz değiştirilmiş olan hali kullanılmıştır. Bu amaçla bu dosyanın iki kopyası oluşturulup, birine İkili Veri Seti (İVS), diğerine de Çoklu Veri Seti (ÇVS) adı verilmiştir.

İkili sınıflandırma için kullanılacak veri setini meydana getirmek amacıyla *Class* özniteliği hedef değişken olarak kullanılmıştır ve *Category* özniteliği İVS'den kaldırılmıştır. Bu işlemin sebebi şöyle açıklanabilir. *Category* özniteliği barındırdığı değerler bakımından, hedef değişkeni en çok etkileyen özniteliktir. *Category* değişkeni; "Benign", "Ransomware", "Trojan Horse" ve "Spyware" değerlerini barındırmaktadır ve oluşturulacak olan bir malware detection modeli, sadece *Category* değişkenini değerlendirerek, ilgili satırdaki verinin "Benign" veya "Malware" olup olmadığına karar verebilecektir. Böyle bir durum, geriye kalan tüm özniteliklerin öneminin yitirilmesine sebep olacaktır. Bu amaçla, diğer özniteliklerin de modeli etkileyebilmesi ve daha sağlıklı bir malware tespit modeli oluşturulabilmesi adına *Category* değişkeni İVS'den kaldırılmıştır. Sonuç olarak İVS içerisindeki toplam değişken sayısı 53'e düşmüştür. Ardından hedef değişken hariç, geriye kalan 52 bağımsız değişkene ait tüm satırlar, Z-Score Normalization (Standardization) işlemine tabi tutulmuştur. Veri setleri üzerine yapılan standardizasyon işlemlerinin, genelde model performanslarını arttırıcı bir etkisi olduğu bilinmektedir [138]. Bu şekilde İVS hazır hale gelmiştir. İVS içerisindeki *Class* kolonunda, "Benign" ve "Malware" olarak etiketlenmiş veri miktarları Tablo 2.2'de gösterilmiştir.

Tablo 2.2. İVS içindeki iyi huylu ve kötü huylu olarak işaretlenmiş veri miktarları.

Sınıf	Veri Miktarı
İyi Huylu (Benign: 0)	29298
Kötü Huylu (Malware: 1)	29298

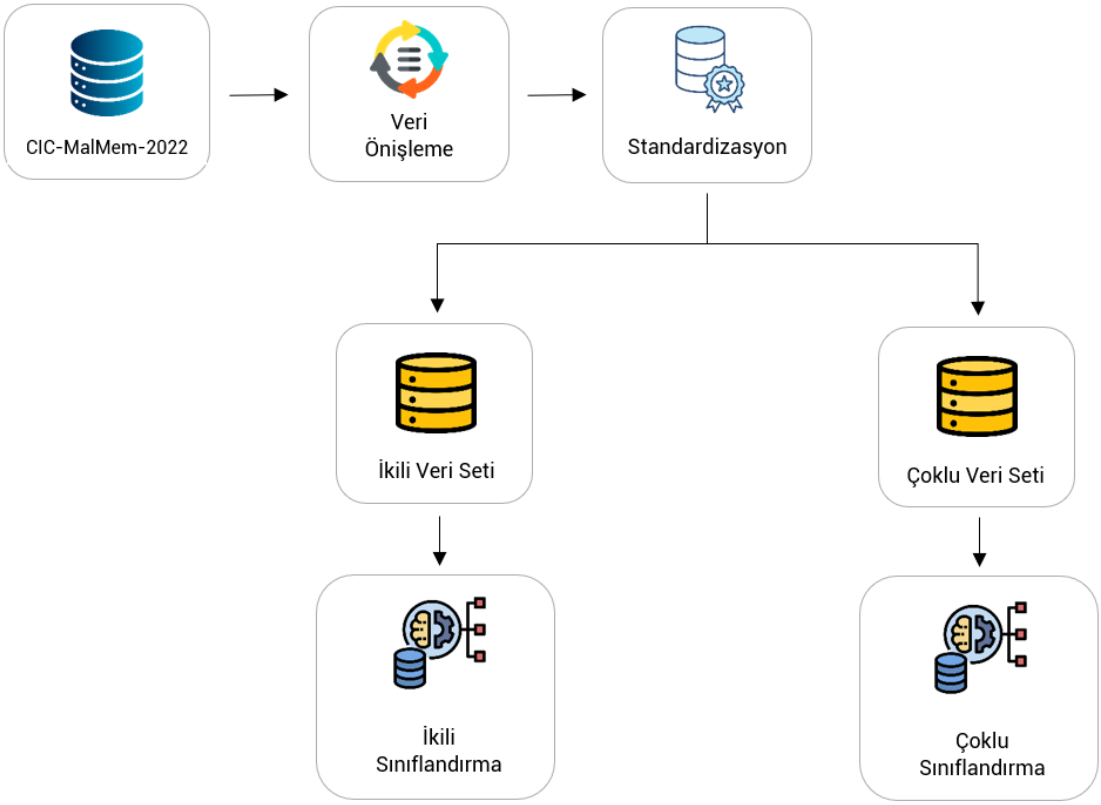
Çoklu (Multiclass) sınıflandırma için kullanılacak olan ÇVS'de ilk anda 54 öznitelik bulunmaktadır. İVS oluşturulurken, *Category* kolonunun veri setinden çıkarılması gibi, ÇVS oluşturulurken de *Class* kolonu veri setinden çıkartılmıştır. *Class* kolonu, ÇVS'de bağımsız bir değişken olarak ve *Category* üzerinde belirli miktarda bir etkiye sahipmiş gibi algılanabilir. Fakat bu bir yanılsamadır. Çünkü gerçek dünya koşulları düşünülecek olunursa, malware olarak kabul edilen hiçbir veride bir etiketleme olmayacağı aşikardır. *Class* değişkeni, ilgili veri satırının iyi huylu veya kötü huylu (malware) olup olmadığına dair direkt bir bilgi içerdiği için, ÇVS'den çıkartılmıştır. Son olarak İVS'deki gibi, hedef değişken harici tüm değişkenlere Standardizasyon işlemi uygulanmıştır. Bu şekilde ÇVS kullanılmaya hazır hale getirilmiştir.

Özetlemek gerekirse, son durumda İVS ve ÇVS'nin her ikisinde de 52 adet bağımsız değişken yer almıştır. İVS, ikili sınıflandırma yapan modellerde ve *Class* değişkeninin değerinin tahmin edilmesinde kullanılmıştır. ÇVS ise çoklu sınıflandırma yapan modellerde ve *Category* değişkeninin değerinin tahmin edilmesinde kullanılmıştır. ÇVS içerisindeki *Category* kolonunda, yer alan etiketlerin veri miktarları Tablo 2.3'te verilmiştir.

Table 2.3. ÇVS içindeki kategoriler ve o kategorilere ait veri miktarları.

Kategori	Veri Miktarı
Benign (0)	29298
Ransomware (1)	9791
Spyware (2)	10020
Trojan Horse (3)	9487

Önişleme ve sonrasındaki tüm aşamalar özetlenmiş olarak Şekil 2.2'de gösterilmiştir.

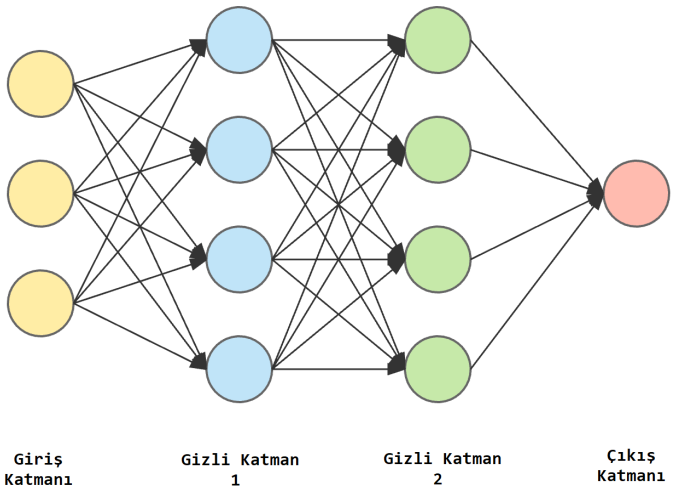


Şekil 2.2. Önişleme, standardizasyon ve veri setlerine göre sınıflandırma işlemleri.

2.2. Yapay Sinir Ağı

Yapay Sinir Ağı (YSA), biyolojik sinir ağlarından ilham alarak geliştirilmiş, öğrenme ve karar verme süreçlerini simüle eden algoritmalardır [139]. YSA'nın temel amacı, veriler arasındaki karmaşık ilişkileri öğrenmek ve genel bir fonksiyonun yaklaşık bir modelini oluşturmaktır [140]. YSA, genellikle düğümler (neuron - nöron) ve bu düğümler arasındaki bağlantılardan (synapse) oluşan katmanlar halinde yapılandırılır. Temel olarak; Giriş Katmanı (Input Layer), Gizli Katmanlar (Hidden Layers) ve Çıkış Katmanı (Output Layer) şeklinde üç bölümden oluşur.

Giriş katmanı, ham verilerin ağa sunulduğu ilk katmandır. Bu katman, verinin boyutuna göre düğümler içerir. Gizli katmanlar, girdi katmanı ile çıktı katmanı arasında yer alan katmanlardır. Verilerin işlenmesi ve özelliklerinin çıkarılması işlemi bu katmanlarda gerçekleşir. Gizli katmanların sayısı ve her katmandaki düğüm sayısı, ağın kapasitesini ve öğrenme yeteneğini belirler. Şekil 2.3'te yapay bir sinir ağına ait görsel bulunmaktadır.



Şekil 2.3. Temel bir YSA ve katmanları.

Çıktı katmanı ise işlenen verilerin nihai sonuçlarının elde edildiği katmandır. Bu katman, çözülmesi istenen probleme bağlı olarak farklı sayıda düğüm içerebilmektedir. YSA'ların temel öğrenme yöntemi, geri yayılım (Back Propagation) algoritmasıyla gerçekleştirilir. Geri yayılım, hatayı minimize etmek için ağırlıkların güncellenmesini sağlayan bir optimizasyon tekniğidir.

2.2.1. Derin Öğrenme (Deep Learning)

Derin Öğrenme (DÖ), YSA'nın bir alt dalı olup, çok katmanlı yapıları ve büyük veri setleri üzerinde öğrenme yeteneği ile öne çıkan bir makine öğrenimi yöntemidir. Bu teknik, geleneksel YSA'lara kıyasla daha karmaşık ve derin yapıların kullanılmasıyla, daha yüksek performans ve doğruluk oranlarına ulaşmayı hedeflemiştir. DÖ modelleri, birden fazla gizli katmana sahip sinir ağları olarak tanımlanır. Bu gizli katmanlar, verilerin daha soyut ve anlamlı temsillerini öğrenir [140].

DÖ modelleri, genellikle büyük veri setleri ve güçlü hesaplama kaynakları gerektirir. Öğrenme süreci, ileri yayılım (Forward Propagation) ve geri yayılım adımlarından oluşur. İleri yayılımda girdi verileri, ağın katmanları boyunca ilerletilir ve her katmanda belirli özellikler çıkarılır. Üretilen çıktılar, beklenen sonuçlarla karşılaştırılarak hata hesaplanır. Ardından ise hata, ağın ağırlıklarını güncellemek için geri yayılır ve bu sayede model optimize edilmiş olur. DÖ yakın zaman literatüründeki birçok çalışmada kullanılan bir tekniktir [43, 141-144].

2.2.2. Aktivasyon Katmanları

DÖ modelleri, verilerin işlenmesi ve özelliklerin çıkarılması için birden fazla katman kullanır. Her katman, girdiler üzerinde belirli işlemler gerçekleştirir ve bu işlemler, modelin öğrenme yeteneğini arttırır. Çeşitli amaçlarla kullanılan birçok DÖ katmanı bulunmaktadır.

DÖ modellerinde kullanılan aktivasyon fonksiyonları, doğrusal olmayan ilişkilerin modellenmesine olanak tanır. Bu katmanlar, sinir ağının her bir nöronunun aktivasyon seviyesini belirler. Yaygın olarak kullanılan aktivasyon fonksiyonları arasında ReLU (Rectified Linear Unit), Sigmoid ve Tanh bulunur [145]. ReLU'da girdi negatif ise 0, pozitif ise aynı değer döndürülür. Hesaplama açısından verimli ve yaygın olarak kullanılır. Sigmoid aktivasyon fonksiyonu, girdiyi 0 ile 1 arasında bir değere sıkıştırır. Genellikle çıktı katmanında kullanılır. Tanh ise girdiyi -1 ile 1 arasında bir değere sıkıştırır. Sigmoid fonksiyonuna benzer, fakat çıktı aralığı farklıdır.

2.2.3. İleri Beslemeli Sinir Ağı

İleri Beslemeli Sinir Ağı (Feedforward Neural Network – FNN), YSA'nın en temel ve en yaygın kullanılan türlerinden biridir [146]. Bu tür ağlar, bilgiyi yalnızca ileri yönde, yani giriş katmanından çıkış katmanına doğru iletir ve geri besleme içermez. FNN'ler; sınıflandırma, regresyon ve örüntü tanıma gibi çeşitli görevlerde kullanılır ve geniş bir uygulama yelpazesi sunar.

İleri beslemeli sinir ağları, verileri girdi katmanından alır ve gizli katmanlar üzerinden geçirerek çıktı katmanına iletir. Her düğüm, aldığı girdileri belirli ağırlıklarla çarpar ve bu değerlerin toplamına bir aktivasyon fonksiyonu uygular [147-149].

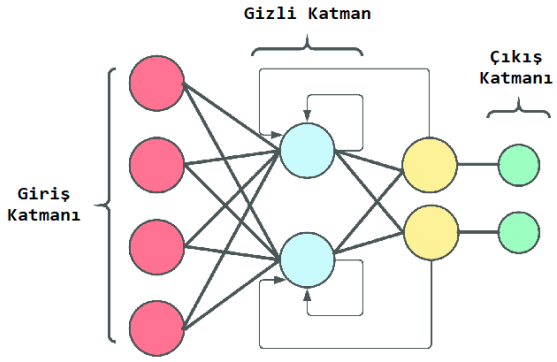
Avantajları arasında, basit bir yapıda olmaları vardır. FNN'ler, ileri beslemeli yapısı sayesinde daha kolay anlaşılır ve uygulanabilirdirler. Genel amaçlı olarak birçok problemde kullanılabilirler.

Dezavantajlı olduğu yönler bakımından incelendiğinde, derin ve karmaşık veri yapılarında performansları sınırlı düzeydedir. Ayrıca çok katmanlı FNN yapılarında, özellikle derin ağlarda, gradyanların sıfıra yaklaşması problemi (Vanishing Gradient – VG) nedeniyle öğrenme süreci zorlaşabilmektedir.

2.2.4. Tekrarlamalı Sinir Ağı

Tekrarlamalı Sinir Ağı (Recurrent Neural Network – RNN), ardışık verilerle çalışmak üzere tasarlanmış yapay sinir ağı türleridir. RNN'ler, ardışık verilerdeki zaman bağımlılıklarını öğrenebilir ve bu bilgiyi kullanarak tahminlerde bulunabilir. Bu anlamda, geçmiş bilgilere dayanarak gelecekteki verileri tahmin edebilme yeteneği ile öne çıkmaktadır. Bu tür ağlar; özellikle zaman serileri, dil modelleme ve ardışık veri analizlerinde kullanılmaktadır [150].

RNN'lerin temel özelliği, döngüsel bağlantılarının olmasıdır. Bu sayede bu tür ağlar, önceki zaman adımlarının bilgilerini hatırlayabilmekte ve bu bilgileri mevcut adımın işlenmesinde kullanabilmektedir. Her zaman adımında, girdi verisi ve önceki gizli durum birlikte işlenir. Gizli katmandaki her nöron, hem mevcut zaman adımının girdisini hem de önceki adımın gizli durumunu dikkate alarak bir çıktı üretir. Bu işlem sürekli olarak tekrarlanır. Çıktı katmanında üretilen değerler, gerçek değerlerle karşılaştırılarak hata hesaplanır. Şekil 2.4'te tekrarlamalı sinir ağına ait temsili bir görsel verilmiştir.



Şekil 2.4. Tekrarlamalı Sinir Ağı (RNN).

Genellikle çapraz entropi veya ortalama kare hatası gibi kayıp fonksiyonları kullanılır. Hata zaman boyunca geri yayılır ve ağın ağırlıkları güncellenmiş olur. Bu işlem, döngüsel yapısı nedeniyle "zaman içinde geri yayılım" olarak adlandırılır [140].

2.2.5. Uzun Kısa-Süreli Bellek

Uzun Kısa-Süreli Bellek (Long Short-Term Memory – LSTM), RNN türlerinden biridir ve özellikle uzun vadeli bağımlılıkları öğrenmede etkilidir [151]. LSTM'ler, RNN'lerin karşılaştığı VG problemini aşmak için geliştirilmiştir ve bu sayede daha karmaşık ve uzun

süreli bağımlılıkları modelleyebilmektedir. LSTM hücreleri, klasik RNN hücrelerinden daha karmaşık bir yapıya sahiptir.

Her LSTM hücresi, bir dizi kapı mekanizması içerir. Giriş Kapısı (Input Gate), hangi yeni bilginin hücre durumuna ekleneceğine karar verir. Unutma Kapısı (Forget Gate), hangi bilginin hücre durumundan çıkarılacağına karar verir. Çıkış Kapısı (Output Gate), hücre durumunun bir kısmını çıktı olarak verir.

Bu kapılar, LSTM'nin uzun vadeli bağımlılıkları öğrenmesini sağlar ve vanishing gradient problemini minimize eder. LSTM'ler, daha karmaşık bir yapıya sahip olduklarından hesaplama maliyetleri daha yüksektir. LSTM hücreleri, klasik RNN hücrelerine göre daha fazla parametre içermektedir. Bu durum, modelin eğitimi ve çalıştırılması sırasında daha fazla kaynak gerektirmektedir.

2.2.6. Vanishing Gradient Problemi

Bu problem, derin sinir ağlarının (özellikle RNN ve LSTM gibi) eğitimi sırasında karşılaşılan temel zorluklardan biridir. Bu problem, ağın öğrenme sürecini önemli ölçüde etkileyerek modelin performansını sınırlar [152].

VG Problemi, gradyan iniş algoritmasının geri yayılım (backpropagation) aşamasında ortaya çıkar. Geri yayılım algoritması, hata fonksiyonunun gradyanını hesaplar ve bu gradyanı kullanarak ağın ağırlıklarını günceller. Fakat derin ağlarda veya uzun zaman serilerinde gradyanlar, katmanlar boyunca geriye doğru yayıldıkça eksponansiyel olarak küçülebilir. Bu durum, özellikle sigmoid veya tanh gibi aktivasyon fonksiyonları kullanıldığında daha belirgin bir şekilde ortaya çıkar. Bu fonksiyonlar, çıktı değerlerini sınırlı bir aralıkta tutar. Bu nedenle, türevleri de küçük değerlere sahip olur. Gradyanlar bu fonksiyonlardan geçerken küçülme eğilimindedir. Derin sinir ağlarında veya uzun süreli zaman serilerinde, gradyanlar her katmanda küçülerek yayıldıkça çok küçük değerlere ulaşır. Bu durum, ağın alt katmanlarının veya önceki zaman adımlarının etkili bir şekilde öğrenememesine neden olur.

VG Problemi, ağın belirli bölgelerinde öğrenmenin durmasına neden olabilir. Bu durum, modelin genelleme yeteneğini azaltır ve performansını sınırlar. Çok küçük gradyanlar, bazı ağırlıkların eğitim süreci boyunca neredeyse sabit kalmasına neden olabilir, bu da modelin yeterince esnek olamamasına yol açar [153, 154].

Bu probleme çözüm olarak; aktivasyon fonksiyonlarının veya ağ mimarilerinin özel RNN türleri olan LSTM veya Gated Recurrent Unit (GRU) ile değiştirilmesi düşünebilir. Bundan başka ağırlık başlatma teknikleri üzerinde yapılacak değişiklikler bir çözüm olabilir. Xavier veya He başlatma gibi yöntemler, ağın başlangıç ağırlıklarını uygun şekilde ayarlayarak gradyanların başlangıçta uygun büyüklükte olmasını sağlar [155]. Yine ek olarak, Batch normalization gibi teknikler, her mini-batch'in ortalamasını ve varyansını normalize ederek gradyanların istikrarlı kalmasını sağlayabilmektedir [156].

2.2.7. RNN ve LSTM'in Karşılaştırılması

RNN ve LSTM ağları, ardışık verilerin ve zaman serilerinin modellenmesi için kullanılan iki temel DÖ yapısıdır. Her iki model de geçmiş bilgileri hatırlayarak gelecekteki tahminleri yapmayı hedefler, ancak bu iki ağ türü arasında bazı önemli farklar ve benzerlikler bulunmaktadır.

Temel olarak hücre yapılarındaki farklılıklar ön plana çıkar. RNN hücreleri, yalnızca bir gizli durumdan (hidden state) oluşur. Bu hücreler, her zaman adımında girdi verisini işler ve bir önceki gizli durum ile birlikte güncellenmiş bir gizli durum üretir. Ancak RNN'ler uzun vadeli bağımlılıkları öğrenmede zorluk yaşamakta ve VG problemiyle karşı karşıya kalmaktadırlar. LSTM hücreleri ise daha karmaşık bir yapıya sahiptir ve unutma kapısı, giriş kapısı ve çıkış kapısı şeklinde kapı mekanizmalarını içerir. Bu kapılar, hücre durumunu (cell state) ve gizli durumu (hidden state) kontrol eder. Bu yapı, LSTM'lerin uzun vadeli bağımlılıkları daha etkili bir şekilde öğrenmesini sağlar ve VG problemini büyük ölçüde azaltır.

RNN'ler daha basit bir yapıya sahip olduklarından hesaplama açısından daha verimlidir. Ancak model doğrulukları, uzun vadeli bağımlılıkları öğrenmede yetersiz kalabilir. LSTM'ler ise daha fazla hesaplama kaynağı gerektirir. Olumsuz bir faktör gibi gözüken bu maliyet, elde edilen modelin performans artışıyla dengelenmiş olur.

Hem RNN'ler hem de LSTM'ler, ardışık verileri işlemek için tasarlanmıştır ve zaman serileri, dil modelleme, konuşma tanıma ve benzeri birçok uygulamada kullanılır. Bu tez çalışmasında da kötü amaçlı yazılımların tespiti için kullanılmıştır. LSTM, bu çalışmada olduğu gibi, ardışık olmayan veriler üzerinde de kullanılabilen bir derin öğrenme

algoritmasıdır. LSTM'in farklı problem türleri için güçlü ve esnek bir araç olması, onun popülerleşmesinin bir sebebi olarak görülebilir [157, 158].

2.3. Önerilen Derin Öğrenme Mimarileri

Bu tez çalışmasında iki farklı sınıflandırma tekniği ile toplamda dört farklı DÖ mimarisi önerilmiştir. Bu dört mimari ile temel anlamda 16 adet model geliştirilmiştir. Temel modellerin dışında, bazı modellerin varyant modelleri de oluşturulup, o modellerin verdiği sonuçlar ve etkileri de çalışmada tartışılmıştır.

Model mimarilerine özel olarak değinmeden önce, bazı temel özelliklere dikkat çekmek gerekir. Örneğin bir sinir ağında hangi aktivasyon fonksiyonunun kullanılacağı kritik bir karar noktasıdır. Aktivasyon fonksiyonları, sinir ağlarının doğrusal olmayan ilişkileri modelleyebilme yeteneğini kazandırır ve bu da modelin öğrenme kapasitesini doğrudan etkiler. Herhangi bir aktivasyon fonksiyonunun performansı, genellikle kullanılan uygulamaya ve veri kümesine bağlı olarak değişir. Örneğin ReLU, çoğu derin öğrenme modeli için yaygın olarak kullanılır ve genellikle hızlı ve etkili öğrenme sağlar. Fakat her zaman en etkili olacak tek bir aktivasyon fonksiyonu yoktur.

Bu nedenle, belirli bir problem veya veri kümesi için en iyi sonucu veren aktivasyon fonksiyonunu bulmak amacıyla çeşitli fonksiyonların denendiği ve değerlendirildiği deneme-yanılma süreçlerini işletmek gerekir. Bu bağlamda yapılan çalışmalar, farklı aktivasyon fonksiyonlarının performansını karşılaştırarak belirli senaryolar için en uygun seçenekleri belirlemeyi amaçlar.

Ramachandran ve arkadaşları tarafından yapılan bir araştırma [159], çeşitli aktivasyon fonksiyonlarının performansını kapsamlı bir şekilde değerlendirerek, DÖ modellerinde kullanılan yaygın aktivasyon fonksiyonlarının avantajlarını ve dezavantajlarını ortaya koymuştur. Bu nedenle, aktivasyon fonksiyonlarının seçimi, modelin genel başarısını belirleyen önemli bir faktördür ve bu seçim dikkatli bir şekilde yapılmalıdır. Bu tez çalışmada da hem ikili sınıflandırma için hem de çoklu sınıflandırmada bir dizi testler sonrası, en iyi sonucu verdiği görülen aktivasyon fonksiyonları tercih edilmiştir. İkili sınıflandırma için kullanılan tüm mimarilerde ReLU aktivasyon fonksiyonu, çoklu sınıflandırma için kullanılan tüm mimarilerde ise Tanh kullanılmıştır.

Aktivasyon fonksiyonlarının yanı sıra, nöron ve katman sayısının belirlenmesi de bir sinir ağı mimarisini oluşturma aşamasında, dikkate alınması gereken önemli parametrelerdendir. DÖ literatüründe, optimum nöron ve katman sayısının tespitine dair, geçmişte yapılmış birçok çalışma bulunmaktadır [160-162].

Bu çalışmada, sinir ağlarının performansını optimize etmek amacıyla nöron ve katman sayılarının belirlenmesi için bir dizi testler gerçekleştirilmiştir. Yapılan bu testler sonucunda, doğruluk oranlarını olumsuz etkilemeyecek şekilde, mümkün olan en az sayıda katman ve nöron kullanılmasına özen gösterilmiştir.

Bu yaklaşım, modelin verimliliğini artırmanın yanı sıra, hesaplama gereksinimlerini de minimumda tutmayı hedeflemektedir. Nöron ve katman sayılarının minimize edilmesinin bir diğer önemli nedeni, FÖ sisteminde kullanılan cihazların işlemci kapasitelerinin genellikle kişisel bilgisayarlara kıyasla daha düşük olabilme ihtimalidir.

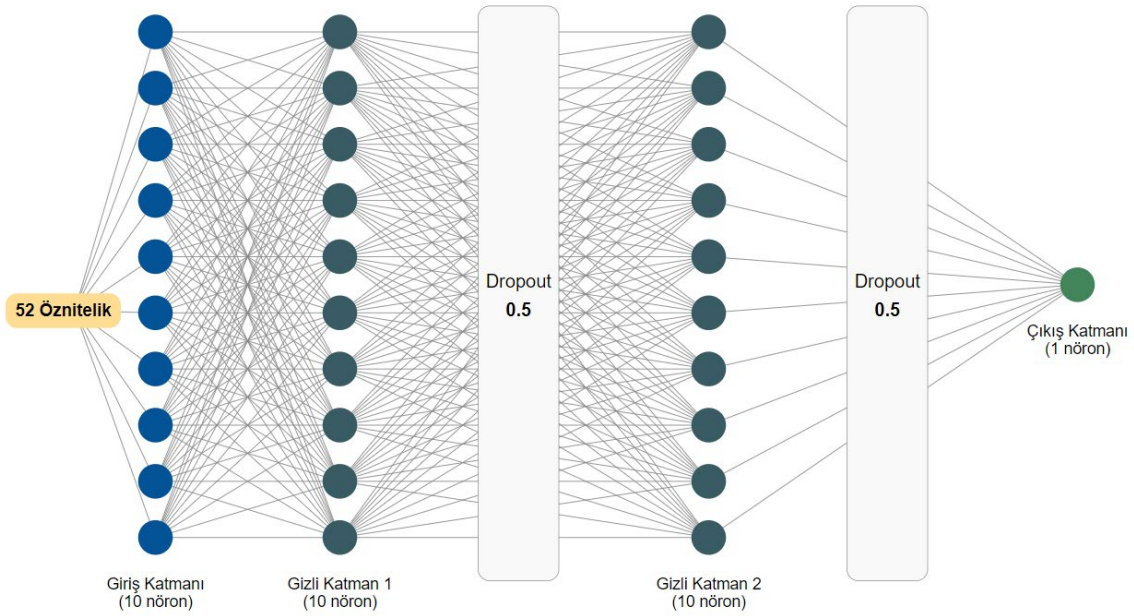
Modelin karmaşıklığı azaltılarak, eğitim süresinin kısaltılması ve hesaplama gereksinimlerinin düşürülmesi mümkündür. Böylece FÖ sistemindeki düşük işlemci kapasiteli cihazların da etkin bir şekilde çalışabilmesi sağlanabilir. McMahan ve arkadaşları tarafından yapılan bir çalışmada [2], FÖ sistemlerinde model eğitim süreçlerinin optimize edilmesi ve yerel cihazların sınırlı kaynaklarının verimli olarak kullanabilmesi adına stratejiler geliştirilmiştir.

Kısaca özetlenirse, az katmanlı ve düşük nöron sayısına sahip bir sinir ağı, federe sistemlerde daha düşük hesaplama gücüyle etkili bir performans sergileyebilmektedir. Bu önemli özellik, yerel cihazların sistem içerisindeki görevlerini sağlıklı bir şekilde yerine getirebilmelerini sağlamaktadır.

Aşağıda sunulan alt bölümlerde, çalışmadaki mimariler ve bu mimarilerden oluşturulan modeller hakkında ön bilgiler verilmiştir. Bu modeller eğitildikten sonra elde edilen sonuçlar ise *Bulgular ve Tartışma* bölümünde değerlendirilmiştir.

2.3.1. İkili Sınıflandırma için FNN Mimarisi (FNN-BC)

İkili sınıflandırma (Binary Classification – BC) problemini çözmek için kullanılan ileri beslemeli bir derin sinir ağı mimarisi olan FNN-BC, belirli katman yapılarına ve nöron sayısına sahiptir. FNN-BC mimarisinin görseli, Şekil 2.5'te gösterilmiştir.



Şekil 2.5. FNN-BC mimarisine ait katmanlar ve nöronlar.

Bu tez çalışmasında, DÖ mimarilerine ait görseller, *D3* adlı JavaScript kütüphanesi kullanılarak oluşturulmuştur. *D3.js*, veri odaklı belgelerin manipülasyonu için kullanılan güçlü bir kütüphanedir ve bu tür grafiklerin dinamik ve etkileşimli bir şekilde oluşturulmasına imkan tanır [163].

Bu mimari, giriş katmanından sonra iki adet gizli katmana ve bir adet çıkış katmanına sahiptir. FNN-BC'nin giriş katmanı, 52 giriş özelliği alır ve bu verilerden 10 adet çıkış nöronu üretir. Bu katmanda kullanılan aktivasyon fonksiyonu ReLU olup, her bir nöronun çıkışını hesaplamak için kullanılır. İki gizli katman da sırasıyla 10 giriş ve 10 çıkış nöronuna sahiptir ve bu katmanlarda da ReLU aktivasyon fonksiyonu kullanılmaktadır. Son olarak çıkış katmanı, son gizli katmandan gelen 10 nöronun çıktısını alır ve bu verileri tek bir çıkış nöronuna dönüştürür. Bu çıkış nöronu, ikili sınıflandırma problemi için gerekli olan son kararı verir.

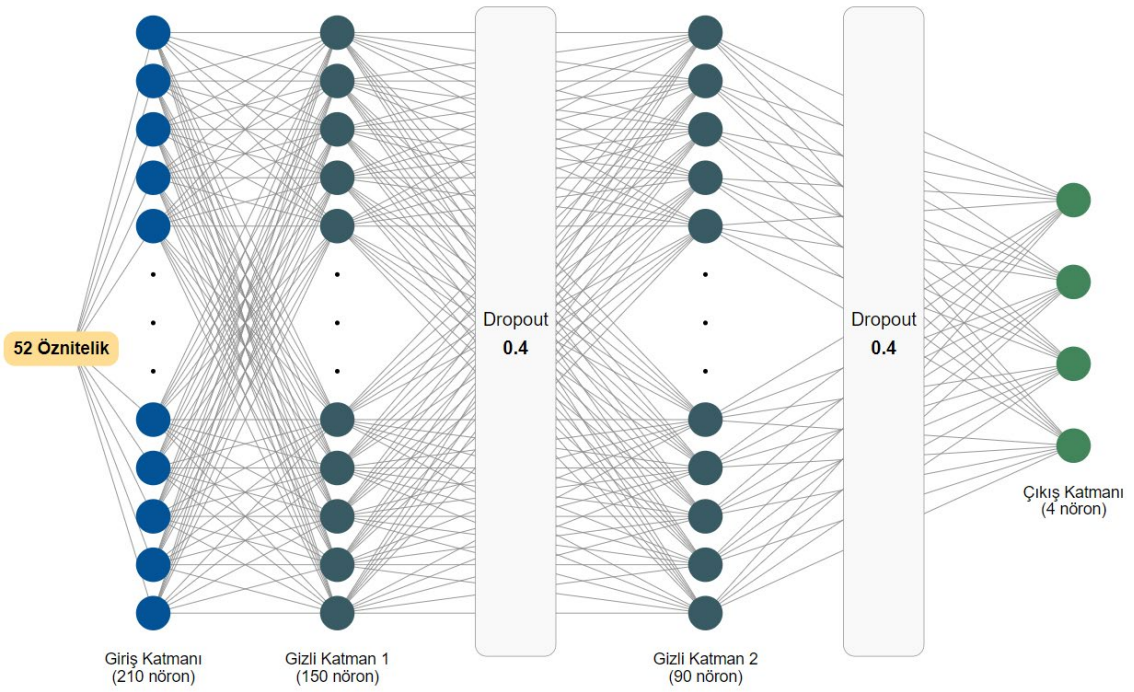
FNN-BC mimarisindeki her gizli katmanın ardında bir Dropout katmanı [164] bulunur. Dropout katmanları da dahil edilirse, bu mimarinin toplam 6 katmandan oluşan bir mimari olduğunu söylemek mümkündür. Dropout katmanlarının görevi, overfitting (aşırı öğrenme) oluşmasını engellemek ve belirtilmiş bir yüzdelik ile (bu mimaride 0.5) rastgele seçilen nöronların aktivasyonlarını sıfıra ayarlamaktır [165].

FNN-BC, ikili sınıflandırma amacıyla kullanılacağı için çıkış katmanında 1 nöron bulunur. Çıkış katmanında aktivasyon fonksiyonu olarak da Sigmoid fonksiyonu

kullanılır. Bu fonksiyon herhangi bir reel sayıyı giriş olarak alır ve sonuçta 0 veya 1 elde edilebilmesini sağlar. Çıktının olasılıksal olarak yorumlanması gereken durumlarda kullanılır. Bu sayede ikili sınıflandırma tahmini yapılabilmektedir. Sigmoid aşırı değerlerin etkisini azaltmakta ve modellerin daha kararlı olmasını sağlamaktadır. Sigmoid doğrusal olmadığı için de bir sinir ağının daha ayrıntılı öğrenebilmesine imkân vermektedir [145].

2.3.2. Çoklu Sınıflandırma için FNN Mimarisi (FNN-MC)

Çoklu sınıflandırma (Multiclass Classification – MC) ile malware türlerini tespit etmek için tasarlanmış bir ileri beslemeli derin sinir ağı mimarisidir. Bu mimari, bir giriş katmanı, iki gizli katman ve bir çıkış katmanından oluşur. Gizli katmanlar arasında Dropout katmanları yer alır ve bu katmanlardaki Dropout oranı %40 olarak belirlenmiştir. Şekil 2.6'da FNN-MC mimarisi görselleştirilmiştir.



Şekil 2.6. FNN-MC mimarisine ait katmanlar ve nöronlar.

Bu mimarinin ilk katmanı, 52 giriş özelliğini alır ve 210 adet çıkış nöronu üretir. Bu nöronlar, birinci gizli katmana giriş olarak kullanılır. Birinci gizli katman 150 nörona, ikinci gizli katman ise 90 nörona sahiptir. Hem giriş katmanında hem de iki gizli katmanda aktivasyon fonksiyonu olarak Tanh kullanılmıştır.

Tanh aktivasyon fonksiyonu, her nöronun çıkışını hesaplamak için kullanılır ve çıktı değerlerini -1 ile 1 arasında sınırlar. Bu, sinir ağının öğrenme sürecinde daha dengeli bir davranış sergilemesine yardımcı olur. Son katman, ikinci gizli katmandan gelen 90 nöronu giriş olarak alır ve 4 çıkış nöronu üretir. Bu çıkış nöronları, malware türlerini belirlemek amacıyla kullanılır ve her biri farklı bir sınıfa karşılık gelir. Çıkış katmanında kullanılan Softmax aktivasyon fonksiyonu [166], bu 4 çıkış nöronunun ürettiği değerleri normalleştirir ve bunları bir olasılık dağılımına dönüştürür.

Softmax, her sınıf için bir olasılık değeri hesaplar ve bu değerlerin toplamı 1'e eşit olur. Bu özellik, multiclass sınıflandırma problemleri için Softmax'ı özellikle uygun kılar. Bu tür bir mimarinin etkinliği, çeşitli araştırmalarla desteklenmektedir. 2012 yılında Krizhevsky ve arkadaşları tarafından yapılan çalışmada [167], Softmax fonksiyonunun çoklu sınıflandırma problemlerinde yüksek performans gösterdiği ve Tanh aktivasyon fonksiyonunun derin sinir ağlarındaki verimliliği artırdığı belirtilmiştir.

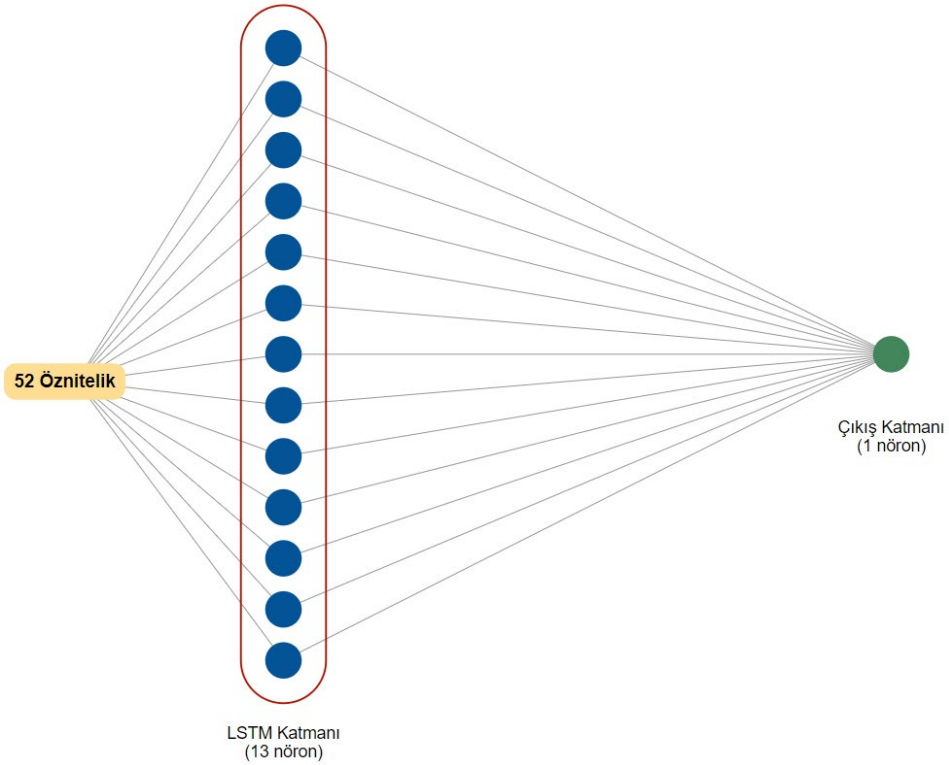
2.3.3. İkili Sınıflandırma için LSTM Mimarisi (LSTM-BC)

İkili sınıflandırma problemleri için FNN-BC mimarisine alternatif olarak geliştirilen bir başka derin öğrenme mimarisidir. Bu mimari, temelde LSTM katmanı ve Çıkış (Output) katmanı şeklinde iki ana katmandan oluşur. LSTM katmanı, modelin ardışık veri yapısını öğrenebilmesi için kullanılan rekürsif bir katmandır. Bu katman, zaman serisi veya dizisel veri yapılarında uzun süreli bağımlılıkları öğrenmede oldukça etkilidir.

LSTM katmanı, 52 giriş özelliği alır ve toplamda 13 nöron içerir. Bu katmanın çıktısı 13 boyutlu bir vektördür ve bu vektör çıkış katmanına giriş olarak sunulur. LSTM katmanında, her bir nöronun çıktısını hesaplamak için ReLU aktivasyon fonksiyonu kullanılmıştır. Çıkış katmanı ise tam bağlantılı (Dense) bir katmandır ve modelin nihai tahminini üretir. İkili sınıflandırma problemi söz konusu olduğunda, çıkış katmanında sadece bir adet nöron bulunur. Bu nöron, modelin ikili sınıflandırma sonucunu sağlar. Çıkış katmanında aktivasyon fonksiyonu olarak Sigmoid fonksiyonu kullanılmıştır.

LSTM-BC, basit ve etkili bir LSTM mimarisidir. Bu yüzden, çalışmadaki diğer mimarilerde bulunan Dropout katmanına bu mimaride ihtiyaç duyulmamıştır. Dropout katmanları, genellikle aşırı öğrenmeyi önlemek için kullanılır, ancak basit LSTM-BC yapısında yeterli sayıda nöron ve katman bulunmadığı için bu gereksinim doğmamıştır.

Şekil 2.7'de, LSTM-BC mimarisine ait tüm nöronlar ve katmanlar detaylı bir şekilde gösterilmiştir.



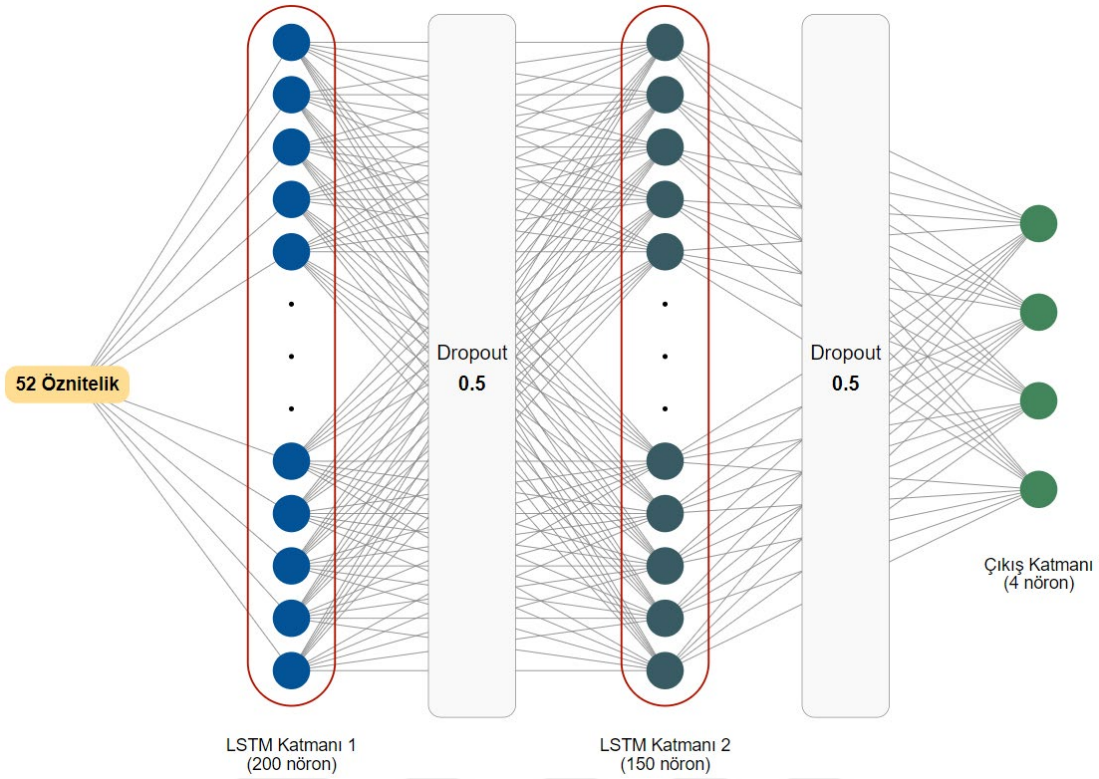
Şekil 2.7. LSTM-BC mimarisine ait katmanlar ve nöronlar.

LSTM katmanları, çalışmadaki diğer tam bağlantılı katmanlardan yapısal olarak farklıdır. Bu farklılığı vurgulamak amacıyla, LSTM-BC ve LSTM-MC mimarilerini gösteren figürlerde LSTM katmanları kırmızı bir çizgi ile belirtilmiştir.

2.3.4. Çoklu Sınıflandırma için LSTM Mimarisi (LSTM-MC)

Çoklu sınıflandırma uygulamalarını gerçekleştirmek için tasarlanan ve FNN-MC mimarisine alternatif olarak kullanılan LSTM-MC mimarisi, bir önceki başlıkta anlatılan LSTM mimarisine göre daha karmaşık ve daha güçlüdür.

Bu mimari, ardışık verilerin işlenmesi ve farklı malware türlerinin tespiti için özel olarak tasarlanmıştır. Şekil 2.8'de LSTM-MC mimarisine ait nöronlar ve katmanlar görülmektedir.



Şekil 2.8. LSTM-MC mimarisine ait katmanlar ve nöronlar.

LSTM-MC mimarisinin ilk katmanı, 200 adet nörondan oluşan bir LSTM katmanıdır. Bu katman, 52 giriş özelliğini alır ve her nöron için aktivasyon fonksiyonu olarak Tanh kullanılır. İlk LSTM katmanının ardından, aşırı öğrenmeyi engellemek amacıyla Dropout katmanı yer alır. Burada dropout katmanının oranı %50 olarak belirlenmiştir, yani nöronların yarısı rastgele kapatılarak modelin genelleme kabiliyeti arttırılmıştır.

Dropout katmanından sonra, ikinci bir LSTM katmanı bulunmaktadır. Bu katmanda 150 nöron yer alır ve yine Tanh aktivasyon fonksiyonu kullanılır. Bu katmanı takiben, tekrar %50 oranında dropout uygulanmıştır. Böylece modelin aşırı öğrenme riski daha da azaltılmıştır. Mimarinin son katmanı ise 4 nörona sahip bir çıkış katmanıdır. Bu katman, modelin nihai tahminlerini üreten son katmanıdır.

LSTM-MC, bir çoklu sınıflandırma problemi çözmek amacıyla kullanıldığından, çıkış katmanında veri setinde tahmin edilmek istenen sınıf sayısı kadar nöron bulunmaktadır. Çıkış katmanında softmax aktivasyon fonksiyonu kullanılmıştır.

3. ÖNERİLEN YAZILIMLARIN AYRINTILARI

Bu bölümde tez çalışmasında oluşturulan yazılımların kodları ve diğer teknik detayları hakkında bilgiler verilmiştir.

3.1. Yazılımlar Hakkında Genel Bilgiler

Bu tez çalışmasında FÖ yaklaşımı ve hazır bir veri seti [13] kullanılarak, malware tespiti ve malware tür tespiti yapan, DÖ modelleri geliştiren uygulamalar tasarlanmıştır. Bu amaçla iki farklı yazılım oluşturulmuştur. Bunlardan biri, yerel kullanıcıların kendi verileriyle model eğitimlerini gerçekleştirip, model bilgilerini merkezi sunucuya gönderebilmelerini sağlayan istemci (client) uygulaması, diğeri de yerel kullanıcıların ürettiği modellerin birleştirilmesi, global modelin başarısını ölçme ve güncel global modelin bilgilerinin yeniden yerel kullanıcılara gönderilebilmesi için kullanılacak olan sunucu (server) uygulamasıdır. İki yazılımın da açık kaynak kodlarına GitHub sitesindeki ***zkcplk*** hesabından ulaşılabilmektedir[1]. Bu iki uygulamanın benzer yönleri olduğu gibi, birbirinden bağımsız bölümleri de bulunmaktadır. Örneğin yerel kullanıcılara ait olan istemci uygulamasında, verilerle model eğitimi gerçekleştirilirken; sunucu uygulamasında model eğitimi gerçekleştirilmemekte, sadece model birleştirme ve model başarı değerlendirme (model testing) işlemi yapılmaktadır.

Önerilen sistemde, sunucu ve istemci arasındaki tüm bilgi alışverişi, Secure Sockets Layer (SSL) [168] üzerinden gerçekleştirilmektedir. Bu amaçla kendi kendine imzalı (Self-Signed) sertifikalar kullanılmaktadır. İstemciler, ürettikleri yerel modelin tüm ağırlıklarını (weights) bir dosya içerisine kaydederek, bu dosyayı Gelişmiş Şifreleme Standardı (Advanced Encryption Standard – AES) şifreleme algoritması [169] ile şifrelemektedir. Böylece istemciden sunucuya gönderilen model bilgileri, şifrelenmiş bir dosya şeklinde ve SSL tabanlı bir bağlantı üzerinden gönderilmektedir. Bu durumda model bilgileri iki katmanlı bir güvenlik koruması altına alınmıştır.

Bu çift katmanlı güvenlik yaklaşımı, model bilgilerinin üçüncü şahıslar tarafından ele geçirilme riskini minimize etmektedir. Sunucu tarafında, alınan şifrelenmiş dosya öncelikle AES özel anahtarı ile deşifre edilmektedir. Bu işlem sonucunda elde edilen

[1] https://github.com/zkcplk/Federe_Ogrenme_ile_Malware_Tespiti

model ağırlıkları, merkezi modelin güncellenmesi için kullanılmaktadır. Sunucu, tüm istemcilerden gelen bu verileri birleştirir ve yeni bir global model oluşturur. Bu global model, istemcilere geri gönderilmeden önce yine AES ile şifrelenir ve SSL üzerinden güvenli bir şekilde dağıtılır. Bu süreç, istemcilerin kendi yerel modellerini güncellemeleri ve merkezi modelin genel performansının iyileştirilmesi açısından önemlidir. Bu şifreleme ve güvenlik katmanları, bir yandan istemcilerin ve sunucunun güvenliğini sağlarken, diğer yandan model bilgilerini potansiyel saldırılardan korumuş olur. Sistemin bütünlüğü ve gizliliği, bu güvenlik önlemleri sayesinde korunur. Ek olarak, kullanılan SSL ve AES teknikleri, bilgi güvenliği alanında yaygın olarak kabul edilen ve test edilmiş protokollerdir. SSL, ağ üzerindeki veri iletimini güvenli hale getirirken; AES, verilerin sadece doğru alıcılar tarafından okunabilmesini sağlamaktadır. Bu teknolojilerin entegrasyonu, model eğitimi ve güncellenmesi süreçlerinde yüksek güvenlik standartlarının uygulanmasını mümkün hale getirir.

Sunucu ile istemci arasındaki iletişim için bir de Uygulama Programlama Arayüzü (Application Programming Interface – API) oluşturulmuştur. Kimlik kontrolleri, çeşitli sorgulamalar ve dosya gönderimleri API içerisinde tanımlı, *API fonksiyonları* yardımıyla sağlanmıştır.

İki yazılım da *Python* programlama diliyle kodlanmıştır. *Python*'ın 3.11.9 versiyonu ile birlikte geliştirme ortamı olarak *PyCharm Community Edition 2023.3.3* kullanılmıştır. FÖ yaklaşımlı DÖ modellerini oluşturmak için python dilindeki kullanımı destekleyen; *Scikit-learn* [170], *Numpy* [171], *Pandas* [172], *Tensorflow* [173] ve *Keras* [174] kütüphanelerinden faydalanılmıştır. *Scikit-learn*, *Numpy* ve *Pandas* gibi kütüphaneler veri işleme ve analizinde yaygın olarak kullanılırken, *Tensorflow* ve *Keras* DÖ modelleri için güçlü araçlar sunmaktadır.

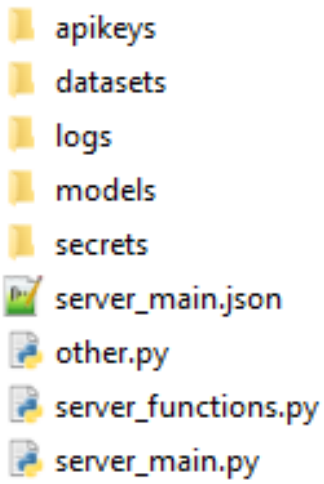
3.2. Kimlik Doğrulaması (Authorization)

Sunucu yazılımı, yerel bir cihazdan model bilgilerini içeren şifreli dosyayı almadan önce, o yerel cihazın mutlaka kimlik kontrolünü yapmak zorundadır. Yerel cihaz, daha önce kendisine verilen *API Anahtarı* (API Key) yardımıyla kendini tanıtarak sunucuya ilgili model dosyasını göndermelidir. Ancak o takdirde, yerel cihazın ürettiği modele ait tüm ağırlıklar, şifrelenmiş bir dosya olarak sunucuya kabul edilebilmektedir. Kimlik doğrulamasının başarıyla tamamlanmasının ardından sunucu yazılımı, şifreli dosyayı

çözüp model bilgilerini işlemeye başlar. Kimlik doğrulaması esnasında sunucu-istemci arasında gerçekleşen tüm mesajlaşmalara ait ayrıntılar, API fonksiyonlarının açıklandığı bölümde detaylandırılmıştır.

3.2.1. Sunucu (Server) Yazılımı

Şekil 3.1'de sunucu yazılımındaki dosya ve klasörler gösterilmiştir.



Şekil 3.1. Sunucu yazılımındaki tüm dosya ve klasörler.

Sunucu yazılımı; *apikeys*, *datasets*, *logs*, *models* ve *secrets* adlı klasörler ile *server_main.json*, *other.py*, *server_functions.py* ve *server_main.py* adlı dosyalardan oluşmaktadır. Bu yazılım, *server_main.py* dosyasının tetiklenmesi ile başlatılmaktadır. Bu dosya, *server_main.json* dosyasındaki ilgili ayarlara göre çalışmaktadır.

3.2.1.1. JSON Dosyası: server_main.json

JavaScript nesne notasyonu (JavaScript Object Notation – JSON) dosyası, sunucu ayarlarının tanımlandığı yapılandırma dosyasıdır. Sunucu başlatılmadan önce bu JSON dosyasındaki ayarların doğruluğu kontrol edilmelidir. Bu dosyada sunucunun IP (Internet Protocol) adresi ve portu, kurulacak olan modelin mimarisi ve sınıflandırma türü, kullanılan veri setine ait bilgiler ve diğer konfigürasyon detayları yer almaktadır.

Sunucu yazılımı, çalıştırılma esnasında gerekli gördüğü durumlarda bu JSON dosyasının içeriğini anlık olarak değiştirebilmektedir. Örneğin *round_num* değişkeni, bu dosyada yer alan bir ayardır ve FÖ sistemindeki round (tur) değerini temsil etmektedir. Bu değer,

round değeri arttıkça JSON dosyasında anlık olarak güncellenmektedir. Buna benzer birçok ayardan en önemlileri, Tablo 3.1'de ayrıntıları ile yer almaktadır.

Tablo 3.1. *server_main.json* dosyasındaki bazı önemli ayarlar ve açıklamaları.

Ayar	Açıklaması
users	Federe sistemin kaç kullanıcılı olacağını belirler. Bu tez çalışmasında 2, 4 ve 8 kullanıcılı olarak üç farklı değer almıştır.
arc_type	Kullanılacak olan DÖ mimarisini belirtir. (0 ise FNN, 1 ise LSTM)
class_type	Kullanılacak olan sınıflandırma türünü belirtir. (0 ise ikili sınıflandırma, 1 ise çoklu sınıflandırma)
round_num	Federe sistemin kaçıncı round'da olduğunu sayan değerdir.
server_host	Sunucu IP adresini belirtir. İstemciler bu IP'den bağlantı sağlar.
server_port	Sunucu port bilgisini belirtir. İstemciler bu porttan bağlantı sağlar.
csv_files	İkili ve çoklu sınıflandırma için kullanılacak iki adet veri setine ait dosya dizininin bulunduğu, liste formatında bir değere sahiptir.
csv_test_size	Veri setinin yüzde kaçının test verisi olarak kullanılacağını belirtir.
hedef_acc	FÖ sisteminin durdurulacağı doğruluk değeridir. 0 ile 1 arası bir değerdir.

Bu ayarların dışında otomatik olarak değer alabilen ayarlar da vardır. Örneğin *csv_file* ayarı, girilen *class_type* bilgisine göre değer almaktadır. Yine aynı mantıkla, *local_models* ayarı da sunucu çalışırken, yerel cihazların gönderdiği yerel model dosyalarının isimlerinin sunucu dizininde yer aldığı bölgeyi belirten bir ayardır. *loss* ayarı da önceden belirlenen sınıflandırma türüne göre (ikili veya çoklu), otomatik olarak "binary_crossentropy" veya "sparse_categorical_crossentropy" değerini almaktadır.

Ayrıca sunucu çalıştırılmadan önce değerleri belirlenen ve sunucu çalışırken hiç değişmeyen ayarlar da bu JSON dosyası içerisinde bulunmaktadır. Bu ayarlardan olan *losses*, *loss* ayarının değeri belirlenirken kullanılan bir listedir. *types* ayarı, hangi sınıflandırma türü seçilmişse ona göre bir değer almaktadır. *metrics* ayarı ise *Keras API*'de kullanılan fonksiyonların istediği bir parametre olarak yer alır. Aynı şekilde *verbose* da bu türden bir ayardır ve sunucuda model birleştirme yapılırken, modelle ilgili

ayrıntıların konsola yazdırılıp yazdırılmamasını belirler. *local_models_prefix ayarı*, yerel cihazlar tarafından gönderilen dosyaların, şifresi çözülmüş hallerinin, sunucudaki hangi dizinde yer aldığını belirten ön ektir. *local_models_enc_prefix* ayarı da bir önceki ayar gibi çalışır ve gönderilen dosyaların şifreli hallerinin yer aldığı dizini belirtir. *global_model_prefix* ayarı, global modelin, model bilgilerinin şifresiz olarak yer aldığı dosya dizinini ifade eder. *global_model_enc_prefix* ise global modelin şifreli olarak yer aldığı dizindir. Şekil 3.2'de *server_main.json* dosyasının içeriği not defterinde açılarak gösterilmiştir.

server_main.json - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

```
{
    "users": 1,
    "arc_type": 0,
    "class_type": 0,
    "csv_files": [
        "datasets/binary_standardize.csv",
        "datasets/multiclass_standardize.csv"
    ],
    "losses": [
        "binary_crossentropy",
        "sparse_categorical_crossentropy"
    ],
    "types": [
        "binary",
        "multiclass"
    ],
    "csv_file": "datasets/binary_standardize.csv",
    "loss": "binary_crossentropy",
    "round_num": 2,
    "server_host": "0.0.0.0",
    "server_port": 65432,
    "csv_test_size": 0.2,
    "hedef_acc": 0.999,
    "metrics": [
        "accuracy"
    ],
    "verbose": 0,
    "local_models": [
        "models/local_models/model_enc_001_2.weights.h5"
    ],
    "local_models_prefix": "models/local_models/model",
    "local_models_enc_prefix": "models/local_models/model_enc",
    "global_model_prefix": "models/global/model.weights.h5",
    "global_model_enc_prefix": "models/global/model_enc.weights.h5",
    "certfile": "secrets/server.crt",
    "keyfile": "secrets/server.key"
}
```

Şekil 3.2. Sunucu yazılımındaki *server_main.json* dosyasının içeriği.

Son olarak *certfile* ve *keyfile* ayarları, SSL bağlantısı için kullanılacak olan sertifikaların dizinlerini gösteren ayarlar olarak bu dosyada yer almaktadır.

3.2.1.2. Python Dosyaları

Sunucu yazılımı üç temel python dosyası ile çalışmaktadır. Bu dosyalar; *server_main.py*, *server_functions.py* ve *other.py*'dır. Bunun dışında bazı klasörlerin içinde de python dosyaları bulunmaktadır. Yeri geldikçe o dosyalardan da bahsedilecektir. Tablo 3.2'de bahsi geçen bu temel üç dosyanın sunucu yazılımındaki görevleri belirtilmiştir.

Tablo 3.2. Sunucu yazılımındaki python dosyalarının görevleri.

Python Dosyası	Sunucu Yazılımındaki Görevi
server_main.py	Sunucu yazılımının çalışması için tetiklenen ilk dosyadır. Bu dosya çalıştıktan sonra, istemci bağlantıları kabul edilmeye başlanır.
server_functions.py	Sunucuyla ilgili işlemleri gerçekleştiren fonksiyonların yer aldığı dosyadır.
other.py	Sunucu ve istemci yazılımında ortak kullanılan (JSON dosya okuma ve güncelleme, işletim sisteminde dosya silme, model dosyalarını şifreleme ve çözme gibi) fonksiyonların yer aldığı dosyadır.

Sunucu yazılımı, istemci isteklerinin kabul etmeye başlanmasının ardından, *server_functions.py* ve *other.py* dosyalarındaki fonksiyonları kullanarak, sunucuya dair tüm görevleri yerine getirir.

Bu fonksiyonların hangi sıralamayla ve nasıl çalıştığıyla ilgili daha ayrıntılı açıklamalar, *3.7. Yazılımların Çalıştırılması* bölümünde yer almaktadır. Tablo 3.3'te *server_functions.py* dosyasındaki fonksiyonların isimleri ve görevleri hakkında bilgiler verilmiştir.

Tablo 3.3. *server_functions.py* dosyasındaki fonksiyonların isimleri ve görevleri.

Fonksiyon	Görevi
handle_client	Yerel cihazların isteklerini organize eden fonksiyondur. Cihazların gönderdikleri API komutlarına göre, gerekli işlemlerin gerçekleştirilmesini sağlamaktadır.

login_user	Yerel cihazların kimlik doğrulamasında kullanılır. İstek gönderen yerel cihazın, gönderdiği *user_no* ve API anahtarının doğruluğunu kontrol eder.
model_update	Tüm yerel cihazlar, yerel model bilgilerini FÖS'e gönderdikten sonra, global modelin güncellenmesini sağlar. *federated_averaging*, *model_test* ve *logla* fonksiyonları bu fonksiyon içerisinde çalıştırılır.
federated_averaging	FedAvg algoritmasını kullanarak, yerel cihazların gönderdiği tüm modelleri birleştirir.
model_test	Güncellenen global modelin başarısını ölçer. *Loss* kayıp değeri ile birlikte *Accuracy*, *Precision*, *Recall* ve *F1* skorlarını hesaplar.
logla	*model_test* fonksiyonunun çalışması sonrası elde edilen metrikleri *logs* klasörüne belirli bir isimlendirme ile kaydeder.
print_global_round	Çalıştırıldığı andaki *round_num* değerini *server_main.json* dosyasından okuyarak konsola yazdırır.

Son olarak *other.py* dosyası, sunucu işlemleri için kullanılan bir başka python dosyasıdır. Tablo 3.4'te *other.py* dosyasındaki fonksiyonların isimleri ve görevleri hakkında bilgiler verilmiştir.

Tablo 3.4. *other.py* dosyasında bulunan fonksiyonların isimleri ve görevleri.

Fonksiyon	**Görevi**
info_load	*server_main.json* dosyasındaki verileri okuyup, python nesnesine dönüştürür. Ardından yazılım içinde, *info* adlı bir değişkenle, JSON dosyasındaki tüm veriler erişilebilir olur.
info_update	*server_main.json* dosyasındaki verilerin güncellenmesi için kullanılır.
load_json_file	Herhangi bir JSON dosyasını okuyup, python nesnesine dönüştürür.
update_json_file	Herhangi bir JSON dosyasındaki verileri güncellemek için kullanılır.
encrypt_file	Model ağırlıklarının yer aldığı dosyaları şifrelemek için kullanılır.

decrypt_file	Model ağırlıklarının yer aldığı şifreli dosyaları çözmek için kullanılır.
remove_files	Sunucu veya istemci yazılımında, belirtilen dizindeki tüm dosyaları silme işlemini gerçekleştirir.

3.2.1.3. Klasörler

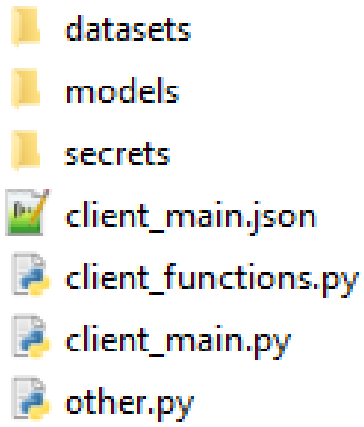
Her biri ayrı görevler için kullanılan, sunucu yazılımındaki klasörler; *apikeys*, *datasets*, *logs*, *models* ve *secrets* olmak üzere beş adettir. Bu klasörlerdeki dosyalar ve bu dosyaların ne amaçla kullanıldığıyla ilgili açıklamalar Tablo 3.5'te verilmiştir.

Tablo 3.5. Sunucu yazılımındaki klasörler ve kullanılma amaçları.

Klasör	**Açıklaması**
apikeys	Sunucuya bağlanabilme ve model gönderebilme yetkisi olan istemcilerin *user_no* ve API anahtarlarının yer aldığı *users.json* dosyasını barındırır.
datasets	İkili ve çoklu sınıflandırma modellerini test etmek amacıyla kullanılacak olan veri setlerini barındırır.
logs	Hedef doğruluğa ulaşıp, sonlandırılmış olan federe modellerin, her round'una dair başarı metriklerinin tutulduğu JSON dosyalarını barındırır.
models	Model mimarilerinin bulunduğu *model.py* adlı python dosyasını barındırır. Bu klasörün içinde *global* ve *local_models* adlı iki klasör daha vardır. Güncel global modelin ağırlıklarına ait bir dosya ve o dosyanın şifreli hali *global* klasöründe yer alır. Yerel kullanıcılar tarafından gönderilen modellerin dosyaları ve onların şifrelenmiş halleri de *local_models* klasöründe yer almaktadır.
secrets	Sunucu ve istemci arasında SSL bağlantısının kurulması için gerekli olan *server.crt* ve *server.key* dosyaları bu klasörde yer alır. Ayrıca model ağırlıklarının yer aldığı dosyaları şifrelemek ve çözmek için kullanılan gizli anahtarın bulunduğu *cipher.key* dosyası da bu klasörde bulunur.

3.2.2. İstemci (Client) Yazılımı

Yerel cihazlarda çalıştırılan istemci yazılımı; *datasets*, *models* ve *secrets* adlı klasörler ile *client_main.json*, *other.py*, *client_functions.py* ve *client_main.py* adlı dosyalardan oluşmaktadır. Şekil 3.3'te istemci yazılımındaki dosya ve klasörler gösterilmiştir.



Şekil 3.3. İstemci yazılımındaki tüm dosya ve klasörler.

İstemci yazılımı da sunucu yazılımına benzer bir şekilde, *client_main.py* dosyasının tetiklenmesi ile başlatılmaktadır. Bu dosya *client_main.json* dosyasındaki ilgili ayarlara göre çalışmaktadır.

3.2.2.1. JSON Dosyası: *client_main.json*

Bu JSON dosyası istemci cihaz ayarlarının tanımlandığı yapılandırma dosyasıdır. Sunucu yazılımında bulunan *server_main.json* dosyasıyla benzerlikleri vardır. İstemci cihazı çalıştırılmadan önce bu JSON dosyasındaki ayarların doğruluğu kontrol edilmelidir. Bu dosyada bağlanılacak olan sunucunun IP adresi ve portu, istemciye daha önceden verilmiş olan numara (*user_no)* ve API anahtarı (*api_key)* bilgisi, kurulacak olan modelin mimarisi ve sınıflandırma türü ve diğer birçok konfigürasyon detayları yer almaktadır.

İstemci yazılımı, çalıştırılma esnasında gerekli gördüğü durumlarda bu JSON dosyasının içeriğini anlık olarak değiştirebilmektedir. Sunucu yazılımındaki JSON dosyasının anlatımında verilen *round_num* örneği, istemci yazılımı için de aynen geçerlidir. İstemci yazılımındaki bu JSON dosyasını, sunucu yazılımındaki benzer JSON dosyasından ayıran en önemli fark, modellere ilişkin hiperparametre ayarlarıdır.

Sunucu yazılımında bir model eğitimi olmadığı için bu ayarlara gerek duyulmamıştır. Fakat istemci yazılımı başlı başına bir model eğitici rölü üstlendiği için hiperparametre ayarları da bu JSON dosyasında yer bulmuştur. Tablo 3.6'da *client_main.json* dosyasındaki en önemli ayarlar ve o ayarlara ilişkin açıklamalar verilmiştir.

Tablo 3.6. *client_main.json* dosyasındaki bazı önemli ayarlar ve açıklamaları.

Ayar	Açıklaması
user_no	Sunucu tarafında kimlik doğrulama için kullanılacak iki bilgiden ilkidir.
api_key	Sunucu tarafında kimlik doğrulama için kullanılacak iki bilgiden ikincisidir. *user_no* ve *api_key* bilgileri federe sistem çalışmaya başlamadan önce yerel kullanıcılara dağıtılmış olmalıdır.
arc_type	Kullanılacak olan DÖ mimarisini belirtir. 0 ise FNN, 1 ise LSTM.
class_type	Kullanılacak olan sınıflandırma türünü belirtir. 0 ise ikili sınıflandırma, 1 ise çoklu sınıflandırma.
round_num	Federe sistemin kaçıncı round'ta olduğunu sayan değerdir.
server_host	Sunucu IP adresini belirtir. İstemciler bu IP'den bağlantı sağlar.
server_port	Sunucu port bilgisini belirtir. İstemciler bu porttan bağlantı sağlar.
csv_files	İkili ve çoklu sınıflandırma için kullanılacak iki adet veri setine ait dosya dizininin bulunduğu, liste formatında bir değere sahiptir.
csv_test_size	Veri setinin yüzde kaçının test verisi olarak kullanılacağını belirtir.
send_time	İstemci bir yerel modeli merkezi sunucuya gönderdikten sonra, yeni bir model göndermek için *send_time* saniye kadar bekler.
epochs	Model eğitilirken kullanılan dönem değeri hiperparametresidir.
batch_size	Her bir epoch dönemi boyunca verilerin ne kadarının kullanılacağını belirleyen bir hiperparametredir.
learning_rate	Öğrenme oranı, model eğitiminde kullanılan bir başka önemli hiperparametredir.

beta_1, beta_2, epsilon	Adam optimizasyon algoritmasında kullanılan parametreleri belirten ayarlardır.

Sunucu yazılımındaki benzer JSON dosyasında olduğu gibi, yukarıda belirtilen ayarların dışında bu JSON dosyasında da otomatik olarak değer alabilen ayarlar vardır. Şekil 3.4'te *client_main.json* dosyasının içeriği not defterinde açılarak gösterilmiştir.

client_main.json - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

```
{
    "user_no": "001",
    "api_key": "zekiCIPLAKMarmaraBilgMUHyuksekLisansTEZi==",
    "arc_type": 0,
    "class_type": 0,
    "csv_files": [
        "datasets/binary_standardize.csv",
        "datasets/multiclass_standardize.csv"
    ],
    "losses": [
        "binary_crossentropy",
        "sparse_categorical_crossentropy"
    ],
    "csv_file": "datasets/binary_standardize.csv",
    "loss": "binary_crossentropy",
    "round_num": 3,
    "send_time": 20,
    "server_host": "localhost",
    "server_port": 65432,
    "csv_test_size": 0.2,
    "epochs": 1,
    "batch_size": 128,
    "learning_rate": 0.001,
    "metrics": [
        "accuracy"
    ],
    "beta_1": 0.9,
    "beta_2": 0.999,
    "epsilon": 1e-08,
    "verbose": 0,
    "local_model_prefix": "models/local/model",
    "local_model_enc_prefix": "models/local/model_enc",
    "global_model_prefix": "models/global/model",
    "global_model_enc_prefix": "models/global/model_enc"
}
```

Şekil 3.4. İstemci yazılımındaki *client_main.json* dosyasının içeriği.

Örneğin *csv_file* ayarı, bu dosyaya girilen *class_type* bilgisine göre değer almaktadır. *loss* ayarı da önceden belirlenen sınıflandırma türüne göre, otomatik bir şekilde "binary_crossentropy" veya "sparse_categorical_crossentropy" değerini almaktadır. Ayrıca istemci yazılımı çalıştırılmadan önce değerleri belirlenen ve istemci çalışırken hiç değişmeyen ayarlar da bu JSON dosyası içerisinde bulunmaktadır. Bunların bazıları hiperparametre ayarları olarak Tablo 3.6 içinde bahsedilmiştir. Bunların dışında *losses*, *loss* ayarının değeri belirlenirken kullanılan bir listedir. Aynı şekilde *verbose* da bu türden bir ayardır ve yerel cihazda bir model eğitilirken, model eğitiminin ayrıntılarının konsola yazdırılıp yazdırılmamasını belirler. *metrics* ayarı ise *Keras API*'de kullanılan fonksiyonların istediği bir parametre olarak yer alır. *local_model_prefix* ayarı, yerel cihazda üretilen modelin katsayılarının olduğu (ve merkezi sunucuta gönderilecek olan) dosyanın dizinine ait ön ektir. *local_model_enc_prefix* ise aynı dosyanın şifrelenmiş halinin bulunduğu dizine ait ön ektir. *global_model_prefix* ise FÖS'ten alınan model bilgilerinin bulunduğu dosyanın dizinine ait ön ektir. Aynı dosyanın şifrelenmiş halini belirten ön ek ayarı ise *global_model_enc_prefix*'dir.

3.2.2.2. Python Dosyaları

İstemci yazılımı da sunucu yazılımındaki gibi üç temel python dosyası ile çalışmaktadır. Bu dosyalar; *client_main.py*, *client_functions.py* ve *other.py*'dır. Bunun dışında *models* klasörünün içinde *model.py* dosyası bulunmaktadır. Tablo 3.7'de bahsi geçen bu temel üç dosyanın sunucu yazılımındaki görevleri belirtilmiştir.

Tablo 3.7. İstemci yazılımındaki python dosyalarının görevleri.

Python Dosyası	**İstemci Yazılımındaki Görevi**
client_main.py	İstemci yazılımının çalışması için tetiklenen ilk dosyadır. Bu dosya çalıştıktan sonra, istemci model oluşturur ve model bilgilerini FÖS'e gönderir.
client_functions.py	İstemciyle ilgili işlemleri gerçekleştiren fonksiyonların yer aldığı dosyadır.
other.py	Sunucu ve istemci yazılımında ortak kullanılan (JSON dosya okuma ve güncelleme, işletim sisteminde dosya silme, model dosyalarını şifreleme ve çözme gibi) fonksiyonların yer aldığı dosyadır.

İstemci yazılımı, çalışmaya başladıktan sonra *client_functions.py* ve *other.py* dosyalarındaki fonksiyonları kullanarak, ilgil istemcinin yapması gereken tüm görevleri yerine getirir. Bu fonksiyonların hangi sıralamayla ve nasıl çalıştığıyla ilgili daha ayrıntılı açıklamalar, *3.7. Yazılımların Çalıştırılması* bölümünde yer almaktadır. Tablo 3.8'de *client_functions.py* dosyasındaki fonksiyonların isimleri ve görevleri hakkında bilgiler verilmiştir.

Tablo 3.8. *client_functions.py* dosyasındaki fonksiyonların isimleri ve görevleri.

Fonksiyon	**Görevi**
tekrarla	Veri setinden alınan belirli miktarda veriyle; model oluşturma ve eğitimi işlemlerini gerçekleştirir. Oluşturulan modelin katsayılarını bir dosyaya yazar ve o dosyayı şifreler. Ardından o dosyanın şifreli halinin FÖS'e gönderilmesi için *client* fonksiyonunu çağırır. Bu işlemler bir döngü içerisinde *send_time* saniye süreyle sürekli olarak tekrarlanır.
client	FÖS ile SSL bağlantısı kurarak, round_num sorgulama ve model gönderimi işlemlerini gerçekleştirir.
local_model_gonderilemedi	FÖS'e model gönderilemediğinde, konsola bilgi verir.

Son olarak *other.py* dosyası, istemci işlemleri için kullanılan bir başka python dosyasıdır. İstemci yazılımındaki *other.py* dosyası ile sunucu yazılımındaki *other.py* dosyası birebir aynıdır. Sadece dosyanın ilk satırlarında tanımlanan *json_file* değişkeninin değerinde bir farklılık vardır. Sunucu yazılımındaki *other.py* dosyasında *json_file* değişkeninin değeri, "server_main.json" olarak, istemci yazılımındaki dosyada ise aynı değişken, "client_main.json" olarak belirlenmiştir.

3.2.2.3. Klasörler

İstemci yazılımındaki klasörler; *datasets*, *models* ve *secrets* olmak üzere üç adettir. Bu klasörlerden datasets ve secrets klasörünün içeriği ve çalışma şeklide sunucu yazılımdakilerle birebir aynıdır. Sadece models klasöründe, istemci yazılımı açısından bazı farklılıklar bulunmaktadır. İstemci yazılımındaki *models* klasöründe, model mimarilerinin bulunduğu *model.py* adlı bir python dosyası bulunur. Bu klasörün içinde *global* ve *local* adlı iki klasör daha vardır. Güncel global modelin ağırlıklarına ait bir

dosya ve o dosyanın şifreli hali *global* klasöründe yer alır. İstemci tarafından gönderilen yerel modele dosya ve onun şifrelenmiş hali ise *local* klasöründe yer almaktadır. Daha açık olarak ifade edilecek olunursa; sunucu yazılımında *local* klasörü *local_models* adını alır ve tüm yerel modellere ait dosyaları içerir. İstemci yazılımında ise aynı klasör *local* adını alır ve sadece ilgili yerel cihazın yerel modeline ait bilgileri içeren dosyayı ve onun şifreli halini barındırır.

3.3. Sunucu-İstemci Mesajlaşmasında Kullanılan API Sistemi

API sistemi, istemciler ve sunucu arasındaki veri alışverişini yönetir ve güvenli bir şekilde gerçekleştirilmesine olanak tanır. Bu tez çalışmasında, sunucu ve istemci arasındaki iletişimi sağlamak ve belirli görevleri yerine getirebilme amacıyla belirli *API fonksiyonları* tanımlanmıştır. Bu fonksiyonlar, istemcilerin sunucuya veri göndermesine ve sunucudan veri almasını sağlayan komutlardır. Ayrıca sunucu yazılımının, her istemci ile iletişim kurabilmesi, *API fonksiyonları* ile mümkün hale gelmiştir.

API fonksiyonlarının kullanımı "KOMUT" anahtar kelimesiyle sağlanmaktadır. Sunucu ile istemci arasındaki bağlantı kurulurken veya dosya transferi gerçekleştirilirken, veri gönderimi bu anahtar kelime ile başlatılır. Ardından kullanıcı numarası (user_no) ve API anahtarı (api_key) gelir ve son kelime olarak, işletilmek istenen API fonksiyonunun ismi yazılır. Örneğin bir istemci federe sistemin hangi round için global modeli birleştirdiğini sorgulamak isterse, aşağıdaki mesajı sunucuya gönderecektir.

KOMUT USER_NO API_KEY CHECK_ROUND_GET

API sistemi, istemci kaynaklı çalışmaktadır. Yani sunucu sadece cevap veren konumda olacak şekilde tasarlanmıştır. Bunun bir sebebi, gereksiz mesajlaşmalardan ve uzun süreli bağlantılardan kaçınma isteğidir. İstemci, sunucuya yerel modelinin dosyasını yükleyeceği zaman bunu mesaj yoluyla ve ilgili API fonksiyonunu kullanarak bildirir. Sunucu olumlu veya olumsuz cevap verir. Örneğin FÖS'te global model güncellendiği zaman, sunucu tüm yerel kullanıcılara bu modeli hemen yollamaz. Cevap veren modda çalıştığı için, sadece kendisinden talep eden istemcilere global modelin son halini şifrelenmiş olarak gönderir. Çalışmada kullanılan tüm API fonksiyonları ve görevleri Tablo 3.9'da gösterilmiştir.

Tablo 3.9. Sunucu ve istemci yazılımlarında kullanılan API fonksiyonları.

Fonksiyon	**Görevi**
GET_MODEL	İstemci yazılımı, merkezi sunucudaki en güncel global modelin dosyasını bu fonksiyon ile yükler. (Download)
POST_MODEL	İstemci yazılım, sunucuya yerel modelini göndermek için bu fonksiyonu kullanır. (Upload)
CHECK_ROUND_GET	İstemci hangi round için model birleştirme yapıldığını bu fonksiyon ile sorgular ve ardından GET_MODEL fonksiyonunu çalıştırır.
CHECK_ROUND_POST	İstemci, kendi verileriyle ürettiği yerel modeli sunucuya göndermeden önce hangi round için birleştirme yapıldığını bu fonksiyon ile sorgular. Eğer kendi round_num değeri ile sunucunun round_num değeri eşitse POST_MODEL fonksiyonu çalıştırılır.

İstemci yazılımının sunucuya gönderdiği mesajlara karşılık olarak sunucu yazılımının istemciye gönderdiği cevaplar vardır. İstemci yazılımı, bu cevaplara göre yeni işlemler gerçekleştirmektedir. Eğer bu mesajlaşmalar düzgün bir şekilde gerçekleşmezse, hata meydana gelmekte ve ilgili fonksiyon görevini yerine getiremeden iptal edilmektedir. Tablo 3.10'da istemcinin sunucuya gönderdiği mesajlara karşılık, sunucunun istemciye gönderdiği cevaplar yer almaktadır.

Tablo 3.10. Sunucu yazılımının istemciye gönderdiği cevaplar.

İstemcinin İsteği	**Sunucunun Gönderdiği Cevap**
CHECK_ROUND_GET	ROUND_NUM_GET**:** *round_num*
CHECK_ROUND_POST	ROUND_NUM_POST**:** *round_num*
GET_MODEL	Öncesinde round_num sorgulaması yapıldığı için istemcinin bu isteğine cevap olarak, sunucudan istemciye, global model bilgilerinin yer aldığı dosyanın şifrelenmiş hali gönderilir.
POST_MODEL	Öncesinde round_num sorgulaması yapıldığı için, sunucu istemcinin bu isteğine cevap vermez ve yerel model bilgilerinin yer aldığı dosyanın şifrelenmiş halini göndermesini bekler.

3.4. Şifreleme Tekniğinin Ayrıntıları

Bu çalışmada önerilen federe sistemde, sunucu ve istemci yazılımları arasında model bilgilerinin paylaşılması, dosya gönderimi şeklinde gerçekleşmektedir. Sunucu veya istemci yazılımı, karşı tarafla paylaşacağı bir modele ait katsayıları, öncelikle bir dosyaya kaydetmekte, ardından o dosyayı karşı tarafla paylaşmadan önce AES algoritması [169] ile şifrelemektedir. Şifrelenmiş olan dosya, ek olarak SSL tabanlı bir bağlantı yoluyla sunucudan istemciye (veya istemciden sunucuya) gönderilmektedir. Böylece iki katmanlı bir güvenlik sistemi uygulanmış olmaktadır. İlgili model dosyasının şifrelenmesi aşamasında kullanılmak üzere, 4096 bitlik RSA [175] anahtarı üretilmiştir. OpenSSL programı [176] yardımıyla üretilen bu anahtar sayesinde, şifreleme ve şifre çözme fonksiyonları sorunsuz bir şekilde işletilebilmiştir. Öncelikle SSL bağlantısının sağlanabilmesi adına, kendi kendine imzalı sertifikalar üretme ihtiyacı ortaya çıkmıştır. OpenSSL'in Windows sürümü, resmi sayfasından[2] indirilerek, kurulum gerçekleştirildikten sonra, komut isteminde aşağıdaki kod parçası çalıştırılmıştır ve **server.key** ile **server.crt** adlı iki adet sertifika dosyası üretilmiştir.

```
openssl req -x509 -newkey rsa:4096 -keyout server.key -out
server.crt -days 365 -nodes
```

Üretilen bu iki dosya daha sonra hem sunucu yazılımında hem de istemci yazılımında SSL bağlantılarında kullanılmak üzere, daha önceki bölümlerde de bahsedilen *secrets* klasörünün içine eklenmiştir. Kendi kendine imzalı sertifikalar, güvenlik zincirinin bir parçası olan sertifika doğrulama sürecinde güvenilir bir otorite tarafından onaylanmaz. Fakat şifreleme sürecini, SSL protokolünün sunduğu veri bütünlüğü ve gizlilik korumasını etkinleştirirler. Bu sertifikalar, sunucu ile istemci arasındaki iletişimin, üçüncü taraflarca dinlenilmesini engeller. Ek olarak verilerin değiştirilmesine karşı da korurlar. Bu çalışmadaki sertifika kullanımı, test ve simülasyon amaçlı olduğu için yeterli görülmüştür. Fakat gerçek dünya uygulamalarında, ilgili sertifikaların güvenilir bir sertifika otoritesi tarafından imzalanmış olması gerekmektedir [177]. SSL dışında, sunucu-istemci arasındaki dosya gönderiminde ayrıca bir dosya şifrelemesi de

[2] https://openssl.org

gerçekleştirilmiştir. AES şifreleme algoritmasının kullanılabilmesi adına, *Fernet* adlı python kütüphanesinden [178] faydalanılmıştır. Şifreleme ve şifre çözme işlemleri, sunucu ve istemci yazılımlarındaki, *other.py* adlı dosyada bulunan, *encrypt_file* ve *decrypt_file* fonksiyonları ile gerçekleştirilmiştir. Bu fonksiyonlara ait sözde (pseudo) kodlar Tablo 3.11'de verilmiştir.

Tablo 3.11. Dosya şifreleme ve çözme için kullanılan fonksiyonların sözde kodları.

```
BEGIN FUNCTION encrypt_file(input_file, output_file):
    CALL OPEN 'secrets/cipher.key' IN read-binary MODE AS key_file
    CALL READ key_file INTO loaded_key
    CALL CLOSE key_file
    CALL CREATE cipher OBJECT USING loaded_key
    CALL OPEN input_file IN read-binary MODE AS f
    CALL READ f INTO file_data AND CLOSE f
    SET encrypted_data = CALL ENCRYPT file_data USING cipher
    CALL OPEN output_file IN write-binary MODE AS f
    CALL WRITE encrypted_data INTO f
    CALL CLOSE f
END FUNCTION
```

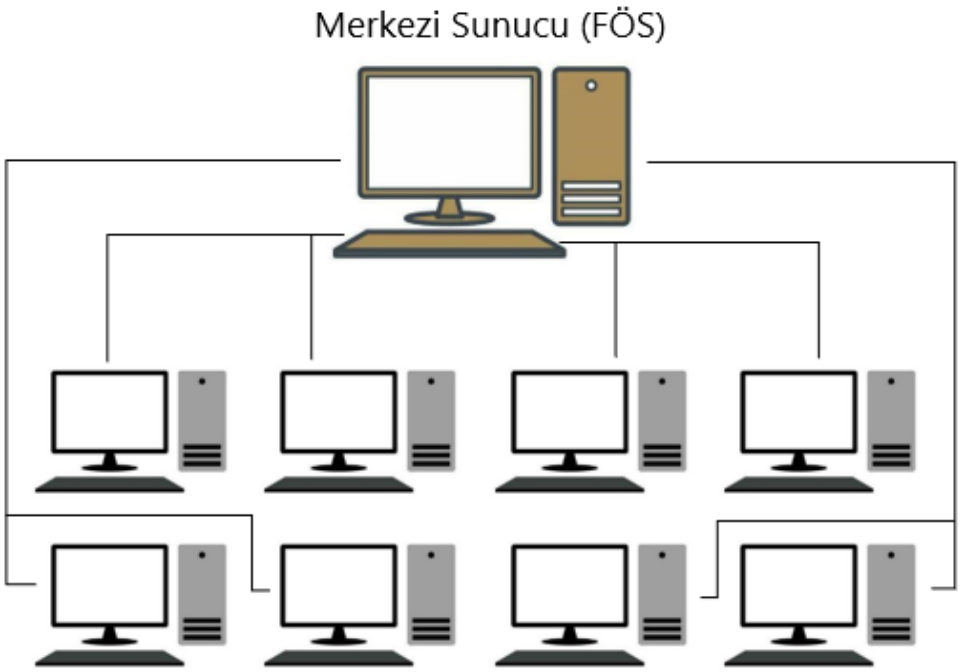
```
BEGIN FUNCTION decrypt_file(input_file, output_file):
    CALL OPEN 'secrets/cipher.key' IN read-binary MODE AS key_file
    CALL READ key_file INTO loaded_key
    CALL CLOSE key_file
    CALL CREATE cipher OBJECT USING loaded_key
    CALL OPEN input_file IN read-binary MODE AS f
    CALL READ f INTO encrypted_data AND CLOSE f
    SET decrypted_data = CALL DECRYPT encrypted_data USING cipher
    CALL OPEN output_file IN write-binary MODE AS f
    CALL WRITE decrypted_data INTO f
    CALL CLOSE f
    RETURN decrypted_data
END FUNCTION
```

Bu fonksiyonlarda, AES şifreleme algoritmasının çalışabilmesi için gizli bir anahtara ihtiyaç vardır. Bu anahtar, Fernet kütüphanesi ile üretilip, hem sunucu yazılımında hem de istemci yazılımında bulunan *secrets* klasöründeki *cipher.key* dosyasına kaydedilmiştir.

3.5. Simülasyon Ortamının Ayrıntıları

Simülasyon ortamı hazırlanırken, olabildiğince gerçek dünya uygulamasına yakınlaşılmaya özen gösterilmiştir. Gerçek bir merkezi sunucu FÖS ve farklı yerel cihazlar (gerçek bilgisayarlar) ile simülasyon desteklenmiştir. Yerel cihazlar ürettikleri modelleri, gerçek bir ağ bağlantısı üzerinden ve şifreli bir şekilde merkezi sunucuya göndermişlerdir. Bu anlamda, sunucu-istemci paradigması gerçek anlamda test edilmiş olup, karşılaşılan tüm problemler, sunucu veya istemci yazılımının güncellenmesine zemin hazırlamıştır.

Çalışmada iki, dört ve sekiz kullanıcılı federe sistemlerin, derin öğrenme modelleri test edilmiştir. Deneysel sonuçların tümü, laboratuvar ortamında elde edilmiştir. Bu amaçla, *İstanbul Gedik Üniversitesi*, *Gedik Meslek Yüksekokulu* laboratuvarındaki bilgisayarlar kullanılmıştır. Şekil 3.5'te simülasyon ortamının temsili bir görseli bulunmaktadır.



Şekil 3.5. Simülasyon ortamının temsili resmi.

Çalışmada kullanılan (merkezi sunucu dahil), tüm bilgisayarların teknik özellikleri aynı olup, Tablo 3.12'de verildiği gibidir.

Tablo 3.12. Tez çalışmasındaki deneylerde kullanılan bilgisayarların teknik özellikleri.

Özellik	**Açıklaması**
İşlemci (CPU)	12th Gen Intel Core i7-12700 2.10 GHz
RAM	16 GB
Dahili Bellek	500 GB SSD
İşletim Sistemi	Windows 10 Pro

Simülasyon ortamının inşası için birçok Python kütüphanesinden faydalanılmıştır. Python kütüphanelerinin isimleri ve versiyon bilgileri, daha sonra yapılacak kurulumları kolaylaştırmak amacıyla *requirements.txt* dosyasına kaydedilmiştir. Federe sisteme eklenen her yeni bilgisayarda Python kurulu olduğu sürece, tüm kütüphaneler, komut istemine girilen aşağıdaki komut yardımıyla, otomatik olarak ilgili bilgisayara sonradan yüklenebilmiştir.

```
pip install -r requirements.txt
```

Simülasyon ortamını oluştururken karşılaşılan güçlüklerden biri, yüksek boyutlu dosyalarla çalışma zorluğudur. Bu zorluk, *requirements.txt* dosyası kullanılarak aşılmıştır. Böylece her iki yazılımın da yeni bir bilgisayara taşınması, tüm kütüphane dosyalarını taşımadan, daha küçük dosya boyutuyla mümkün hale gelmiştir.

3.6. Federe Sistem Başlatılmadan Önce Yapılması Gerekenler

Federe sistem başlatılmadan önce hem sunucu hem de istemci yazılımlarındaki, ayarların bulunduğu JSON dosyalarında bir takım düzenlemeler gerçekleştirilmelidir.

Öncelikle federe sistem için planlanan kullanıcı sayısı kadar yerel cihaza istemci yazılımının kurulumu yapılmış olmalıdır. Sunucu ile istemci cihazların iletişim kurabildiği teyit edilmelidir. Bu cihazların hepsi aynı yerel ağda veya internet üzerinden birbirleriyle iletişim gerçekleştirebilir halde bulunmalıdır.

Bu anlamda her iki yazılımda da *server_host* ve *server_port* ayarları, ilgili JSON dosyalarında düzenlenmelidir.

Sırasıyla 3.6.1 ve 3.6.2 başlıklarında sunucu ve istemci yazılımlarında gerçekleştirilmesi gereken bazı önemli hazırlıklardan bahsedilmiştir. Burada bahsedilen ayarlamaların dışındaki ayarlar için 3.2.1 ve 3.2.2 başlıklarında yapılan açıklamalardan faydalanılabilir.

3.6.1. Sunucu Yazılımında Gerçekleştirilmesi Gereken Önemli Hazırlıklar

Veri seti dosyalarının *datasets* klasöründe, API anahtarlarının bulunduğu *users.json* dosyasının *apikeys* klasöründe ve şifreleme işlemleri ile ilgili dosyaların (*server.key*, *server.crt* ve *cipher.key*) *secrets* klasöründe hazır olduğu varsayımından hareketle, *server_main.json* dosyasında aşağıdaki değişiklerin yapılması gerekmektedir.

Federe sistemin kaç kullanıcıyla çalıştırılacağı ve hangi DÖ mimarisinin kullanılacağına bağlı olarak, ilgili JSON dosyasında *users*, *arc_type* ve *class_type* ayarları düzenlenmelidir.

Hedef doğruluğu ölçen yazılım, sunucu yazılımı olduğu için ve farklı DÖ mimarilerine ait farklı hedef doğruluk oranları olabileceğinden, doğruluk oranı da ilgili JSON ayarında yerine yazılmalıdır. Bu amaçla *hedef_acc* ayarı düzenlenmelidir.

3.6.2. İstemci Yazılımında Gerçekleştirilmesi Gereken Önemli Hazırlıklar

Bir önceki başlıkta bahsedilen malum klasörlerdeki dosyaların hazır olduğu varsayımından hareketle, sadece *client_main.json* dosyasında aşağıdaki değişiklerin yapılması gerekmektedir.

İstemci yazılımlarındaki *send_time* ayarı önemli bir parametredir. Çünkü hangi zaman aralıkları ile sunucuya yeniden model dosyası gönderileceği bilgisi, bu ayardan sağlanmaktadır. Bu değerin, çok küçük veya çok büyük olmadan, orta bir değer olması önerilmektedir. Bu değer büyük seçilirse, federe sistemin tamamlanması gereksiz yere uzayabilir. Küçük seçilirse de fazladan gereksizce sunucu-istemci iletişimi yaşanmasına ve dolayısıyla sistemin kasılmasına sebebiyet verilebilmektedir. Bu tez çalışmasında bir dizi denemeler sonrasında *send_time* değerinin 10 olarak ayarlanmasına karar verilmiştir.

Federe DÖ modellerinin eğitimlerinin geliştirileceği yazılım, istemci yazılımı olduğu için, model hiperparametrelerinin de bu yazılımdaki ilgili JSON dosyasında ayarlanması

gerekmektedir. Bu anlamda *epochs*, *batch_size*, *learning_rate* ayarları ve Adam optimizasyon algoritmasıyla ilgili olan; *beta_1*, *beta_2* ve *epsilon* ayarları da federe sistem başlatılmadan önce gerekli görüldüğü takdirde düzenlenmelidir.

3.7. Yazılımların Çalıştırılması

Hazırlıklar tamamlandıktan sonra, yazılımların çalıştırılması aşamasına geçilmiştir. Her iki yazılım için de Windows ortamında otomatik çalıştırma işlemleri için, **.BAT** uzantılı script dosyaları kullanılmıştır. Bu dosyalar kullanılmadan önce her cihazda (cihazın görevine göre) sunucu veya istemci yazılımı kurulumlarının yapılmış olması beklenmektedir. Bu amaçla, laboratuvar ortamındaki bir bilgisayar, ana bilgisayar olarak belirlenip, o bilgisayara sunucu yazılımı kurulmuştur. Sunucu yazılımının ana bilgisayara kurulumu için *install_server.bat* dosyası oluşturulmuş olup, içeriğindeki kodlar, Tablo 3.13'te gösterilmiştir.

Tablo 3.13. Sunucu yazılımı için kullanılan *install_server.bat* dosyasının kodları.

```
@echo off
echo server_01 venv kuruluyor...
call python -m venv server_01
echo Sanal Ortam aktif ediliyor...
call .\server_01\Scripts\activate
echo Kutuphaneler yukleniyor...
pip install -r requirements.txt
echo Server basliyor...
python server_main.py
pause
```

Sunucu yazılımının kurulumu tamamlandıktan sonra, sekiz adet bilgisayara istemci yazılımı kurulmuştur. İstemci yazılımları kurulurken, her bir istemciye *client_01*, *client_02*, … şeklinde bir etiket ataması yapılmıştır. Bu etiketteki numara, her bir istemci için *client_main.json* dosyasında belirtilecek olan *user_no* ayarında da aynı şekilde

belirtilmiştir. Python Environment[3] isimleri de *client_01*, *client_02*, … olacak şekilde yapılandırılmıştır.

İstemci yazılımının kurulumu için de *install_client.bat* dosyası oluşturulmuştur. Bu dosya içeriğinde yer alan client (istemci) numarası, her farklı bilgisayar için değiştirilerek çalıştırılmıştır. Tipik bir *install_client.bat* dosyasının içeriği Tablo 3.14'te verilmiştir.

Tablo 3.14. İstemci yazılımı için kullanılan *install_client.bat* dosyasının kodları.

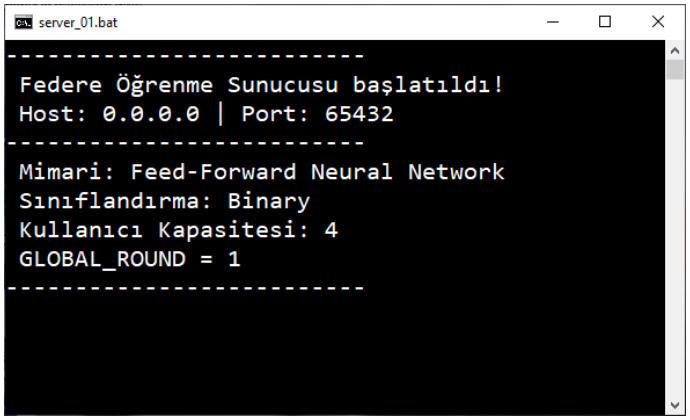
```
@echo off
echo client_01 venv kuruluyor...
call python -m venv client_01
echo Sanal Ortam aktif ediliyor...
call .\client_01\Scripts\activate
echo Kutuphaneler yukleniyor...
pip install -r requirements.txt
echo Client_01 basliyor...
python client_main.py
pause
```

Kurulumlar tamamlandıktan sonra, sunucu yazılımının çalıştırılmasını sağlamak üzere *server_01.bat* dosyası oluşturulmuştur. Bu dosya içeriğinde sadece, *server_main.py* dosyasının tetiklenmesini sağlayan aşağıdaki kod bulunmaktadır.

```
python server_main.py
```

Bu kodun tetiklenmesi ile sunucu yazılımı başlatılmış olur. Şekil 3.6'da, dört kullanıcılı FNN mimarisine sahip ve ikili sınıflandırma için çalıştırılan bir sunucu yazılımının başlatıldığı anda komut isteminde oluşan görüntü yer almaktadır.

[3] https://docs.python.org/3.11/library/venv.html



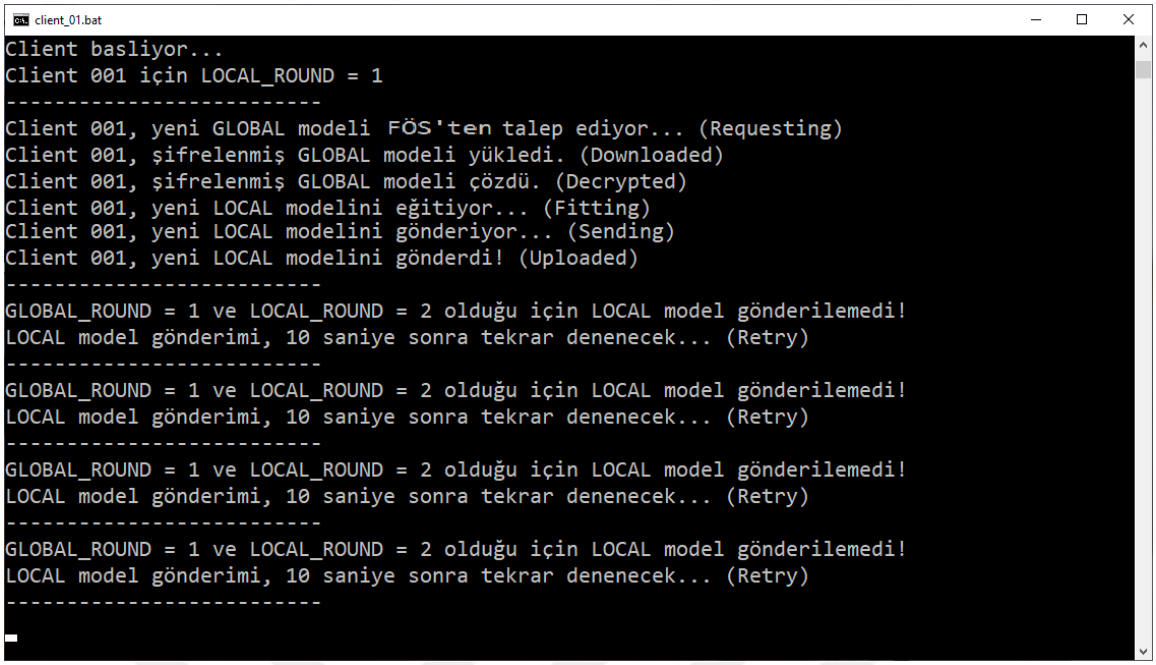
Şekil 3.6. Sunucu yazılımı başlatıldığında komut isteminde oluşan görüntü.

Sunucu yazılımı çalıştırıldıktan sonra, sunucuda ayarlanan kullanıcı sayısı kadar (bu tez çalışmasında; 2, 4 veya 8'dir) bilgisayarda istemci yazılımı çalıştırılmıştır. Bir istemci yazılımı çalıştırıldığında, *round_num* değerinin başlangıç değeri "1" olarak kabul edilir ve her zaman bir istemcide *round_num* değerine özel olarak bir model eğitimi gerçekleştirilir.

Sunucu yazılımı, tüm istemcilerin kendi modellerini gönderdiğini belirlediği zaman, gönderilen yerel modelleri "*belli bir round için*" birleştirmektedir. Sunucu, tüm istemciler yerel modellerini göndermeden asla birleştirme işlemi yapmamaktadır. Bu yüzden, istemcilerden birinin gönderim işlemini geciktirmesi durumunda, global modelin birleştirilmesi işlemi de gecikmiş olacaktır.

İstemciler, *send_time* ayarı ile belirlenen bir süre kadar, sürekli olarak round sorgulaması yaparlar. Sunucudan gelen round bilgisi (*GLOBAL_ROUND*) ile kendi saydıkları round bilgisi (*LOCAL_ROUND*) eşleşiyorsa ve daha önce o round için, ilgili istemciden yerel model gönderimi yapılmamışsa, o durumda yerel model gönderimi gerçekleştirilir.

Eğer sunucu bir sonraki round'a geçmişse, istemciler yeni round için yerel modellerini oluşturup, gönderim işlemi gerçekleştirebilirler. Aksi durumda, eski round için yeniden model oluşturma ve gönderme işlemi gerçekleştirmezler. Kısaca, her round için sadece bir kez yerel model gönderimi yapılmaktadır. Ardından her istemci eğittiği modeli merkezi sunucuya gönderir.



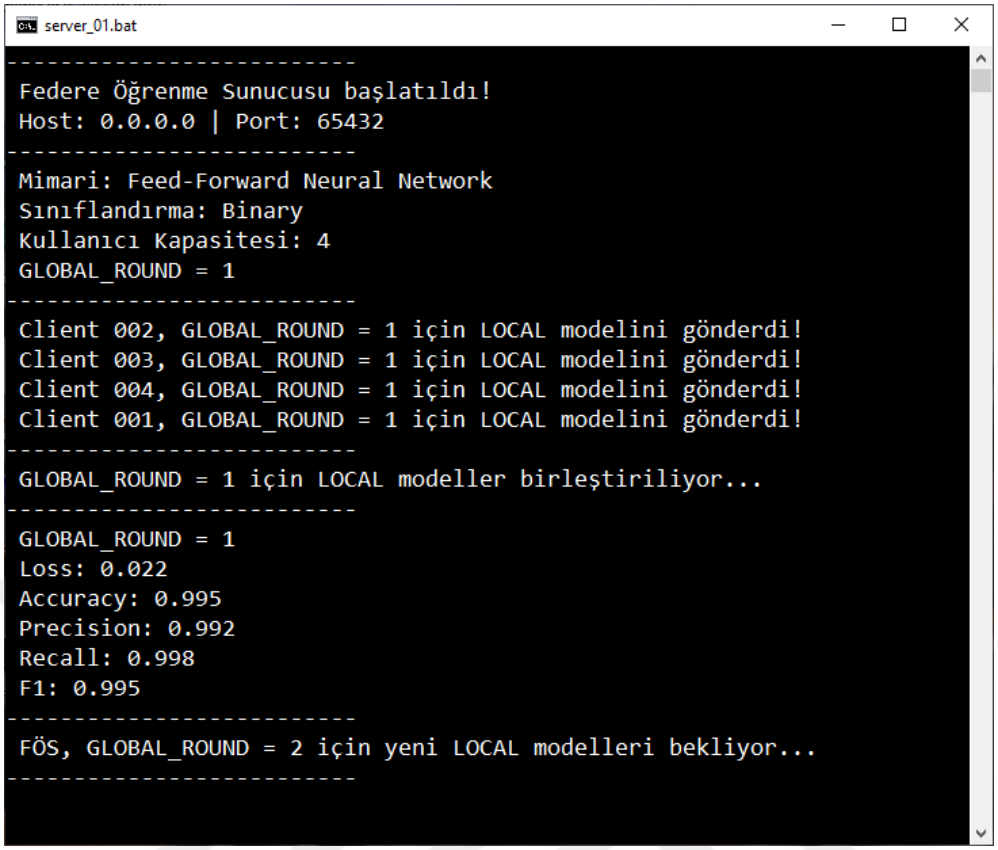
Şekil 3.7. İstemci yazılımı yüklü olan bir yerel cihazdan model gönderimleri.

Şekil 3.7'de *client_01* etiketli bir istemcinin merkezi sunucuya ***ilk yerel modelini gönderdikten sonra***, *send_time* süresi boyunca (bu tez çalışmasında 10 saniye) sürekli model gönderme denemeleri yaptığı, fakat sunucu henüz bir sonraki round'a geçmediği için modelini gönderemediği görülmektedir. Aynı şekilde yerel cihazın gerçekleştirdiği tüm işlemleri görebilmek de mümkündür.

Merkezi sunucudan talep edilen model bilgileri, SSL bir bağlantı üzerinden AES şifreleme algoritması ile şifrelenmiş bir dosya şeklinde geldiği için, öncelikle yüklenen dosyanın şifresinin çözülmesi gerekmektedir. Daha sonra, yüklenen güncel global model üzerinde, yerel cihazın kendi verileriyle bir eğitim işlemi başlatılmaktadır.

Bu işlem, gerçek dünya senaryosunda yerel bir cihazın kendi ürettiği veri ile gerçekleştirilecektir. Bu tez çalışmasındaki simülasyon ortamında ise hazır veri setinden faydalanılmıştır.

Daha önce içeriği açıklanan *client_main.json* dosyasında belirtilen, *batch_size* değerine göre belirli bir miktar veri, rastgele olarak seçilir ve yerel cihazın ürettiği veriymiş gibi kabul edilerek, bir model oluşturulur. Böylece her model oluşumunda, yerel cihazın veri üretmesi sağlanmış olur.



Şekil 3.8. Yerel cihazların modellerinin birleştirildiği sırada, sunucu yazılımının konsolunda görüntülenen raporlamalar.

Yerel cihazlar, modellerini merkezi sunucuya gönderdiği sırada, sunucu yazılımının ekranında da raporlama yapılmaktadır. Model birleştirme sonrasında elde edilen başarı metrikleri hem konsola yansıtılmakta hem de daha önce bahsi geçen *logs* klasöründeki ilgili JSON dosyasına sonuç olarak eklenmektedir. Burada anlatılan tüm işlemlere dair sunucu yazılımının konsoluna yansıyan raporlamalar, temsili bir görsel ile Şekil 3.8'de gösterilmektedir.

Ayrıca sistemin nasıl çalıştığının daha iyi anlaşılması adına, sekiz kullanıcılı bir federe öğrenme modelinin eğitildiği, sunucu-istemci yazılımlarının ve simülasyon ortamının genel çalışma düzenini gösteren, örnek bir video, YouTube adlı siteden[4] izlenebilmektedir.

[4] https://youtu.be/1_dtkFgn3vE

4. BULGULAR VE TARTIŞMA

Bu tez çalışmasında, dört farklı derin öğrenme mimarisinde yapılandırılan, FÖ yaklaşımlı ve FÖ yaklaşımlı olmayan temel anlamda 16 model oluşturulmuştur. Varyant modeller de ilave edildiğinde toplam model sayısı 28'e çıkmıştır.

Bu bölümde, temel modellerin performansları detaylı bir şekilde analiz edilmiştir. Ardından varyant modellerin çalışmaya olan etkileri hakkında ek değerlendirmelerde bulunulmuştur.

Çalışmanın sonuçlarının sunulduğu Tablo 4.1'de, her bir temel modelin hangi mimariye sahip olduğu, modellerin kayıp ve doğruluk değerleri, FÖ yaklaşımına sahip olup olmadığı, federe bir model ise kaç kullanıcılı olduğu ve herhangi bir model eğitiminin kaç tur (round) veya dönem (epoch) sürdüğü gibi bilgiler yer almaktadır.

Tablo 4.1. Çalışmada önerilen tüm modellerin metrikleri.

Model	Tür	Mimari	Kullanıcı	Tur	Dönem	Kayıp	Doğruluk
1	Federe Olmayan	FNN-BC	1	-	5	0.004	0.999
2	Federe	FNN-BC	2	2	1	0.008	0.999
3	Federe	FNN-BC	4	2	1	0.008	0.999
4	Federe	FNN-BC	8	2	1	0.006	0.999
5	Federe Olmayan	FNN-MC	1	-	330	0.361	0.839
6	Federe	FNN-MC	2	109	1	0.365	0.839
6A	Federe	FNN-MC	2	30	5	0.359	0.839
6B	Federe	FNN-MC	2	13	10	0.360	0.839
7	Federe	FNN-MC	4	109	1	0.357	0.841
7A	Federe	FNN-MC	4	23	5	0.363	0.841
7B	Federe	FNN-MC	4	13	10	0.359	0.840
8	Federe	FNN-MC	8	104	1	0.361	0.839

8A	Federe	FNN-MC	8	22	5	0.367	0.839
8B	Federe	FNN-MC	8	11	10	0.364	0.841
9	Federe Olmayan	LSTM-BC	1	-	5	0.002	0.999
10	Federe	LSTM-BC	2	3	1	0.004	0.999
11	Federe	LSTM-BC	4	2	1	0.015	0.999
12	Federe	LSTM-BC	8	2	1	0.004	0.999
13	Federe Olmayan	LSTM-MC	1	-	370	0.346	0.843
14	Federe	LSTM-MC	2	125	1	0.351	0.843
14A	Federe	LSTM-MC	2	25	5	0.352	0.844
14B	Federe	LSTM-MC	2	14	10	0.349	0.843
15	Federe	LSTM-MC	4	122	1	0.355	0.843
15A	Federe	LSTM-MC	4	27	5	0.346	0.845
15B	Federe	LSTM-MC	4	14	10	0.351	0.843
16	Federe	LSTM-MC	8	114	1	0.348	0.844
16A	Federe	LSTM-MC	8	25	5	0.348	0.846
16B	Federe	LSTM-MC	8	15	10	0.342	0.845

Tablo 4.1 incelendiğinde, oluşturulan 28 modelin dört tanesinin (model 1, 5, 9 ve 13), federe olmayan, klasik anlayışla eğitilen modellerden oluştuğu görülmektedir. Çalışmaya öncelikle bu modellerin eğitimleri ile başlanmıştır. Çünkü bu modellerin doğruluk oranları baz alınmıştır ve federe modellerin bu doğruluk oranlarına ulaşıp ulaşamayacağı veya kaç tur (round) sonrasında bu başarıyı elde edecekleri test edilmiştir.

Bu amaçla; dört farklı federe mimarideki modellerin her biri için, federe olmayan bir modelin doğruluk oranı belirleyici olmuştur. FNN-BC mimarisindeki federe modeller için model 1, FNN-MC için model 5, LSTM-BC için model 9 ve LSTM-MC için model 13'ün doğruluk oranları baz alınmış ve federe sistemin durdurma kriteri olarak bu oranlar kullanılmıştır.

Federe olmayan modellerin tur bilgileri bulunmamaktadır. Çünkü tur bilgisi sadece federe yaklaşımlı modellerde bir anlam taşımaktadır.

Model numarasının yanında A veya B şeklinde harf ile isimlendirilmiş modeller, *Varyant Model* olarak adlandırılmaktadır. Örneğin 6A ve 6B modelleri, model 6'nın varyantları olarak sayılmaktadır. Varyant modeli temel bir modelden ayıran ölçüt, dönem (epoch) sayısıdır. Varyant modellerde dönem sayısı arttırılmıştır. Bu değerin arttırılmasının tur sayısına etkisi test edilmiştir. Varyant modeller sadece çoklu sınıflandırma amacıyla oluşturulan federe yaklaşımlı modeller için üretilmiştir. Çünkü federe yaklaşımlı ikili sınıflandırma modellerinin dönem sayıları "1" olup, her modelde sabittir. Federe yaklaşımlı ikili sınıflandırma modellerinde tur sayıları zaten düşük olduğu için (genellikle iki veya üç) varyant model oluşturmaya gerek duyulmamıştır.

Genel anlamda, federe ve federe olmayan modellerden elde edilen bulgular arasında bir karşılaştırma yapıldığında; federe modellerin, federe olmayanların doğruluk oranlarına ulaştığı görülebilmektedir. Mimariler ise ikili ve çoklu sınıflandırma modelleri açısından karşılaştırıldığında, ikili sınıflandırma için kullanılan modellerin, çoklu sınıflandırma için kullanılan modellerden daha başarılı sonuçlar verdiği anlaşılmıştır. İkili sınıflandırma modelleri her mimaride **0.999** doğruluk oranına ulaşmıştır. Federe olmayan ve baz alınma amaçlı kullanılan çoklu sınıflandırma modellerinde ise FNN-MC (model 5) için **0.839**, LSTM-MC (model 13) için **0.843** doğruluk değerlerine ulaşılmıştır.

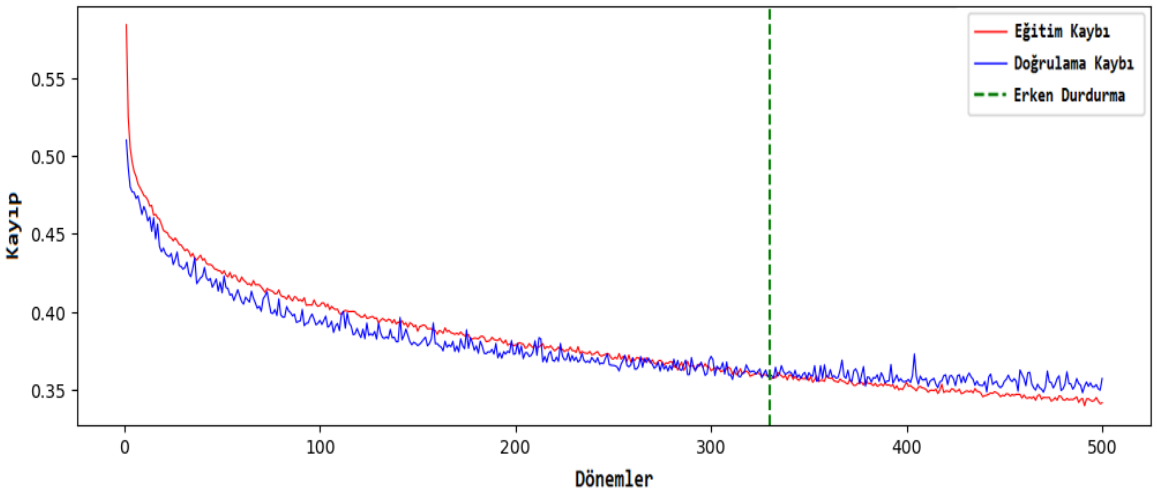
Oluşturulan tüm varyant modeller arasında doğruluk açısından bir değerlendirme yapıldığında ise en yüksek doğruluk oranına sahip modelin, 0.846 doğruluk oranıyla, 16A adlı varyant model olduğu görülmektedir. Fakat bu sonuç yanıltıcı olabilir. Çünkü federe sistemde durdurma kriteri olarak "*hedef bir doğruluk değeri*" kullanıldığı için, bazı federe modeller, federe olmayan versiyonlarından daha büyük doğruluk değerlerine ulaşabilmektedir. Örneğin, model 13 federe olmayan LSTM-MC mimarisine sahip bir modeldir ve bu modelin federe olan versiyonlarına bakıldığında (14, 14A, 14B, 15, 15A, 15B, 16, 16A, 16B), model 13'e göre daha büyük doğruluk değerlerine rastlanabilir. Böyle daha büyük değerlerin ortaya çıkması, o federe modellerin, federe olmayan versiyonundan daha başarılı olduğu anlamına gelmemektedir.

Model 13'ün doğruluk değeri olan 0.843 değeri, federe versiyon modellerinde, bir durdurma kriteri olarak kullanılmıştır. Bu durdurma, bazen 0.843'ün üzerinde

gerçekleşebilmektedir. Örneğin bir federe modelin doğruluk oranı, önceki turda 0.841 iken bir sonraki turda, bir anda 0.845'e çıkabilmekte ve federe sistem durdurulmaktadır. Dolayısıyla, bir yandan 0.843 değeri aşıldığı için federe sistem durdurulurken, diğer yandan doğruluk oranı açısından 0.843 değeri biraz aşılmış durumda olmaktadır. Bu yüzden değer olarak, diğer federe modellerden yüksek gibi gözükse de model başarısı açısından daha başarılı olduğunu kesin olarak söylemek mümkün değildir.

Federe olmayan klasik MÖ modellerinde aşırı öğrenme (overfitting), tespit edilebilen ve önlem alınabilen bir problemdir [179]. Bu yüzden, tez çalışmasında federe olmayan, klasik MÖ yaklaşımıyla oluşturulan modellerin (model 1, 5, 9 ve 13) ulaştığı doğruluk değeri baz alınarak, daha fazlası aşırı öğrenmeye yol açabilir düşüncesiyle, FÖ sistemi durdurulmaktadır. Bu anlamda, "hedef bir doğruluk değerine göre federe sistemin durdurulması" kararının, aynı zamanda aşırı öğrenmeyi de engelleyen bir faydası olduğunu söyleyebilmek mümkündür.

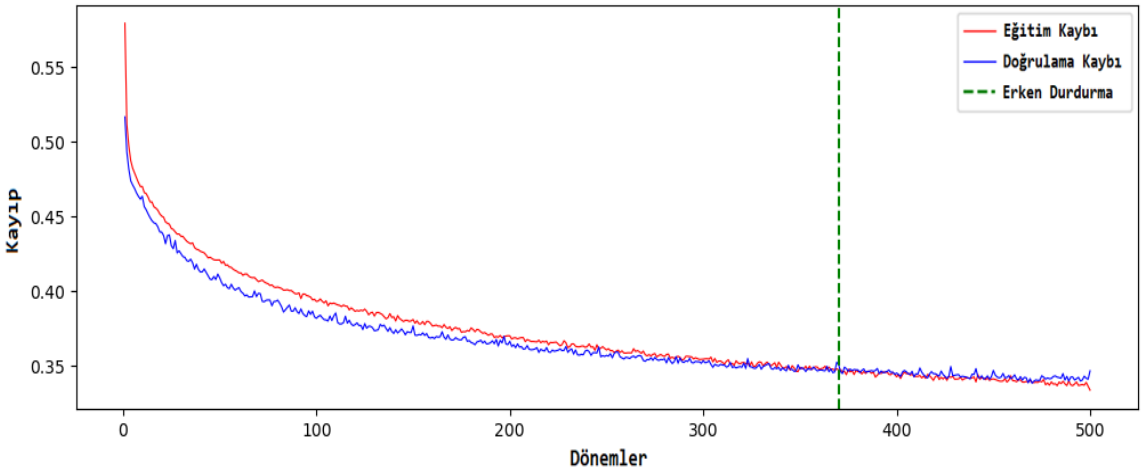
FNN-MC mimarisindeki tüm federe modeller için, model 5'in doğruluk oranı olan 0.839 değeri kullanılmıştır. Federe olmayan model 5'in ulaştığı en büyük doğruluk oranının aşırı öğrenmeye maruz kalmaması adına, klasik MÖ modeli eğitiminde *Erken Durdurma* (Early Stopping) tekniği uygulanmıştır [180, 181]. Erken durdurmanın günümüzde birçok farklı tekniği bulunmaktadır. Bu çalışmada erken durdurmanın en yaygın olarak bilinen tekniği kullanılmıştır. Eğitim sürecinin en uygun anında, yani validasyon kaybının (loss) eğitim kaybını aşmaya başladığı ilk noktada, erken durdurma gerçekleştirilmiştir.



Şekil 4.1. Model 5'e ait Kayıp (Loss) – Dönemler (Epochs) grafiği.

Model 5'in eğitimi, doğrulama kayıp değerinin (Validation Loss – VL), eğitim kayıp değerini (Training Loss – TL) aştığı 330. dönem değerinde durdurularak aşırı öğrenme engellenmiştir. Şekil 4.1'de model 5'e ait Kayıp-Dönemler grafiği görülmektedir.

LSTM-MC mimarisindeki tüm federe modeller için, model 13'ün doğruluk oranı olan 0.843 kullanılmıştır. Model 5'te olduğu gibi, model 13 için de erken durdurma tekniği uygulanmıştır. Model 13'ün eğitimi, eğitim kayıp değerinin, doğrulama kayıp değerini aştığı 370. dönem değerinde durdurularak aşırı öğrenme engellenmiştir. Şekil 4.2'de model 13'e ait Kayıp-Dönemler grafiği görülmektedir.



Şekil 4.2. Model 13'e ait Kayıp (Loss) – Dönemler (Epochs) grafiği.

Varyant modeller (6A, 6B, 7A, 7B, 8A, 8B, 14A, 14B, 15A, 15B, 16A ve 16B), temel versiyonları ile karşılaştırıldığında, dönem sayısının etkilerini daha net görebilmek mümkündür. Varyant modellerin dönem sayısının artışı, yerel cihazlardaki eğitimin biraz daha uzatılması anlamına gelmektedir. Yerel cihazlarda, dönem sayısı normalde "1" değerine sahip iken, varyant modellerde 5 veya 10 şeklinde değişiklik göstermektedir.

Deneysel sonuçlar, yerel cihazlarda gerçekleştirilen eğitimin arttırılmasının, doğruluğu çok etkilemediğini göstermektedir. Kayıp değerleri açısında incelendiğinde ise genelde azalma görülmektedir. Örneğin model 6 incelendiğinde kayıp değeri 0.365 iken, varyant modellerinde bu değer sırasıyla 6A ve 6B için 0.359 ve 0.360 olmuştur. Özellikle dönem sayısının 10 olduğu varyant modellerde, temel modele göre kayıp değerlerinde düşüşler saptanmıştır. Örneğin, model 14 ve 14B sırasıyla, 0.351 ve 0.349; model 15 ve 15B sırasıyla 0.355 ve 0.351; model 16 ve 16B sırasıyla 0.348 ve 0.342 kayıp değerlerine sahiptir.

Mimarilerin kayıp değerleri incelendiğinde, FNN ve LSTM mimarileri arasında belirgin farklılıklar gözlemlenmektedir. FNN mimarilerinde kayıp değerleri genellikle 0.004 ile 0.367 arasında değişirken, LSTM mimarilerinde bu değerler 0.002 ile 0.355 arasında değişmektedir. FNN-BC ve LSTM-BC mimarilerinde kayıp değerleri oldukça düşük ve doğruluk oranları oldukça yüksekken, FNN-MC ve LSTM-MC mimarilerinde kayıp değerleri biraz daha yüksek ve doğruluk oranları biraz daha düşüktür. Bu durum, ikili sınıflandırma problemlerinin daha düşük kayıp değerleri ve daha yüksek doğruluk oranları ile sonuçlandığını göstermektedir.

Tur sayıları açısından varyant modeller incelendiğinde, tüm varyant modellerdeki dönem sayısının arttıkça, tur sayısının düştüğü görülmüştür. Dolayısıyla, federe modelin tamamlanması daha kısa sürmüştür. Örneğin federe yaklaşımlı model 6, 109 turda biterken; 6A ve 6B varyant modelleri sırasıyla 30 ve 13 turda tamamlanmıştır. Çoklu sınıflandırmalı federe modellerde tur düşüşünü daha net görebilmek mümkündür. Örneğin model 14, 125 turda tamamlanırken 14A ve 14B varyant modelleri sırasıyla, 25 ve 14 turda tamamlanmışlardır. Varyant modellerdeki değişimlere genel olarak bakıldığında, federe sistemin tur sayısını etkili bir şekilde düşürdüğü görülebilmektedir. Doğruluk değeri açısından da bazı varyant modeller daha iyi sonuçlara sahiptir ama meydana gelen etki sınırlı düzeyde kalmıştır.

Tablo 4.1'deki sonuçlardan hareketle; federe modellerdeki kullanıcı sayıları baz alınarak da bir değerlendirme yapılabilir. FNN-BC ve LSTM-BC gibi, ikili sınıflandırma amaçlı oluşturulan modeller incelendiğinde, kullanıcı sayısı artışının doğruluk artışına sebep olmadığı görülmektedir. Ayrıca tur sayısında da ciddi bir düşüş görülmemiştir. Federe yaklaşımlı FNN-BC modellerde tur sayısı kullanıcı sayısına göre değişmez iken, LSTM-BC modellerde ise üçten ikiye sınırlı bir düşüş olmuştur. Federe yaklaşımlı FNN-MC ve LSTM-BC modelleri incelendiğinde de kullanıcı sayısının artmasının, önemli bir artışa sebep olacak şekilde doğruluk oranını etkilemediğini söyleyebilmek mümkündür.

Araştırmadaki tüm bulgular incelendiğinde, yapılan çalışma ile ilgili büyük miktarda önemli bilgiler elde edilmektedir. Ulaşılan en önemli sonuç, aynı mimarideki tüm federe öğrenme modellerinin, federe olmayan tüm modeller ile aynı doğruluk değerlerine ulaşabilmiş olmasıdır. Bu sonuç, FÖ paradigmasının etkinliğini ve uygulanabilirliğini kanıtlayan, deneysel bir sonuçtur. Klasik makine öğreniminde sıklıkla kullanılan, tüm

verinin merkezi bir sunucuda toplanması gerekliliğini ortadan kaldırmaktadır. Özellikle veri mahremiyetinin kritik olduğu uygulamalar için önemli bir avantajdır. Bu bağlamda, FÖ yaklaşımı, bir yandan veri mahremiyetini sağlarken, öte yandan model performansından ödün vermeyen, kesin bir çözüm sunmaktadır.

Bu tez çalışmasında elde edilen bulgular, literatürde yayınlanmış olan diğer çalışmalarla mukayese edildiğinde, tez çalışmasının birçok açıdan daha iyi sonuçlar verdiğini söyleyebilmek mümkündür. Karşılaştırmalar iki şekilde gerçekleştirilebilir. Birinci karşılaştırma, CIC-MalMem-2022 veri setini kullanan ama FÖ yaklaşımını kullanmadan malware tespiti ve malware tür tespiti yapan çalışmalarla yapılan karşılaştırmadır. İkinci karşılaştırma ise FÖ yaklaşımı ile farklı veri setlerini kullanarak malware tespiti yapan çalışmalarla gerçekleştirilmiştir. Tablo 4.2'de birinci tür karşılaştırmaya yönelik bilgiler bulunmaktadır.

Tablo 4.2. FÖ yaklaşımı olmadan CIC-MalMem-2022 veri setini kullanarak malware tespiti ve malware tür tespiti yapan çalışmalarla kıyaslama tablosu.

Çalışma	**Sınıflandırma**	**Mimari**	**Doğruluk**
Tezde Önerilen Çalışma	***İkili (Binary)***	***FNN-BC***	***0.9999***
		LSTM-BC	***0.9999***
Maya H.L.L. ve ark. [114]		RF	1.0000
Talukder ve ark. [115]		XGBoost & RF	1.0000
Daryle Smith ve ark. [117]		Adaboost	0.9999
Mezina ve ark. [118]		RF	0.9900
M. Dener ve ark. [119]		LR	0.9997
		GBT	0.9994
		NB	0.9841
Tezde Önerilen Çalışma	***Çoklu (Multiclass)***	***FNN-MC***	***0.8390***
		LSTM-MC	***0.8430***
Mezina ve ark. [118]		CNN	0.8353

Bu tez çalışmasının literatürdeki diğer çalışmalara göre önemli bir üstünlüğü vardır. Önerilen çalışma, ikili sınıflandırma açısından incelendiğinde, [114] ve [115] numaralı çalışmalar haricinde, diğer tüm çalışmalardan daha iyi bir performans gösterilmiştir. [114] ve [115] numaralı çalışmalar da bu çalışmaya göre sadece 0.0001 civarında daha

iyi bir doğruluk değerine erişmiştir. Tez çalışmasının yukarıdaki bölümlerde anlatılmış olan diğer kazanımları da ele alındığında, bu denli az olan bir doğruluk kaybı göz ardı edilebilir düzeydedir. Çünkü Tablo 4.2'deki hiçbir çalışma FÖ yaklaşımını kullanmamıştır. Bu çalışmada ise bir yandan diğer çalışmalar gibi klasik MÖ yaklaşımı kullanılmış, diğer yandan FÖ yaklaşımı ile araştırmacılara daha geniş bir perspektif sunulmuştur. Federe olmayan modellerin ulaştığı tüm doğruluk değerlerine, federe modeller de ulaşmıştır. Tablo 4.2'deki diğer tüm çalışmalar, veri mahremiyetini dikkate almamıştır. Bu tez çalışmasında önerilen federe modeller ise bir yandan kullanıcıların hassas sayılabilecek verilerinin korunmasını sağlanırken, diğer yandan literatürdeki çalışmalara doğruluk açısından üstünlük sağlamıştır. Ayrıca bu çalışma, bahsi geçen ilgili çalışmalara göre daha çok farklı model test edilen, daha farklı alanlarda, daha fazla bilgi içeren bir çalışma olmuştur.

Çoklu sınıflandırma açısından değerlendirildiğinde, daha önce yapılan pek fazla çalışma bulunmamaktadır. CIC-MalMem-2022 veri setiyle gerçekleştirilen malware tespit çalışmalarının hemen hemen tamamı sadece ikili sınıflandırma içermektedir. Sadece [118] numaralı çalışma çoklu bir sınıflandırma örneği içermektedir ve 0.8353 doğruluk değerine sahiptir. Bu çalışmada ise hem FNN hem LSTM tabanlı modellerde bu doğruluk değeri aşılmıştır. Üstelik hem federe olmayan hem de federe öğrenme tabanlı modeller, sırasıyla 0.839 ve 0.843 doğruluk değerine ulaşarak, çoklu sınıflandırmada da daha iyi performans göstermişlerdir. FÖ yaklaşımı, klasik MÖ yöntemlerine kıyasla, farklı veri kaynaklarından gelen bilgileri bir araya getirerek daha kapsamlı ve güvenilir modeller oluşturulmasını mümkün kılmıştır. Böylece araştırmacılara ve uygulama geliştiricilere yeni fırsatlar sunarak, veri mahremiyetinden ödün vermeden yüksek performanslı modeller geliştirilmesini sağlamaktadır.

Bu tez çalışmasının diğer önemli katkılarından biri de, çeşitli modeller üzerinde geniş kapsamlı testler yapılmasıdır. Çalışmada kullanılan CIC-MalMem-2022 veri seti, siber güvenlik alanında yaygın olarak kullanılan bir veri seti olup, farklı malware türlerinin tespiti için çok uygun bir örnek teşkil etmektedir. Bu veri setiyle yapılan deneyler, çoklu sınıflandırma görevlerinde FÖ tabanlı modellerin üstün performansını göstermiştir. Ayrıca bu çalışma, literatürdeki diğer çalışmalardan farklı olarak hem FNN hem de LSTM tabanlı modellerin performansını karşılaştırmalı olarak ele almış ve FÖ yönteminin bu modellerde de başarılı olduğunu ortaya koymuştur.

CIC-MalMem-2022 veri setiyle gerçekleştirilen çalışmaların dışında, farklı veri setleri ile gerçekleştirilmiş malware tespit çalışmaları da bulunmaktadır. Bu tez çalışması, daha farklı veri seti kullanan çalışmalar ile karşılaştırıldığında da önemli üstünlüklere sahip olmuştur. Tablo 4.3'te CIC-MalMem-2022 dışındaki farklı veri setleri ile gerçekleştirilen çalışmalar verilmiştir.

Tablo 4.3. FÖ yaklaşımı ve ikili sınıflandırma ile malware tespiti yapan ve farklı veri setleri kullanan diğer çalışmalar ile kıyaslama tablosu.

Çalışma	Veri Seti	Mimari	Ölçüt
Tezde Önerilen Çalışma	***CIC-MalMem-2022***	***FNN-BC LSTM-BC***	***Doğruluk: 0.9999 F1: 0.9999***
Kuang-Yao Lin ve ark. [120]	VirusTotal	SVM	*Doğruluk: 0.9167*
V. Rey ve ark. [121]	N-BaIoT	SVM	*Doğruluk > 0.9700*
Ruei-Hau Hsu ve ark. [122]	Opera Mobile Store	SVM	*Doğruluk: 0.9345*
Rafa Gálvez ve ark. [123]	MaMaDroid	LiM	*F1: 0.9500*
Changnan Jiang ve ark. [124]	Androzoo	FedHGCDroid	*Doğruluk: 0.913*
Taheri ve ark. [125]	Genome	FedGAN-BK	*Doğruluk: 0.9324*

Tablo 4.3'te verilen çalışmalar, FÖ yaklaşımı kullanılarak gerçekleştirilen çalışmalardır. Bazı çalışmaların sonuçlarında sadece doğruluk veya sadece F1 skor sonucu verildiği için, sadece bu ölçütler üzerinden bir karşılaştırma yapılmıştır. Bu karşılaştırmaya göre, bu çalışmanın tabloda verilen diğer tüm çalışmalardan, her iki ölçüt açısından da daha başarılı olduğu görülmektedir. Ayrıca ilgili tabloda verilen diğer çalışmaların hepsi sadece ikili sınıflandırma gerçekleştirmişlerdir. Bu tez çalışmasında ise hem ikili hem de çoklu malware sınıflandırması gerçekleştirildiği için, bu yönüyle de diğer çalışmalardan üstün olduğunu söylemek mümkündür.

5. SONUÇLAR

Bu tez çalışmasında, DÖ mimarileri ile oluşturulan FÖ tabanlı modeller ile federe olmayan modellerin sınıflandırma sonuçları çeşitli yönlerden mukayese edilmiştir. *Kanada Siber Güvenlik Enstitüsü* tarafından oluşturulan *CIC-MalMem-2022* veri seti kullanılarak malware tespiti ve sınıflandırması yapılmıştır.

FNN ve LSTM gibi derin öğrenme algoritmaları kullanılmış, ikili ve çoklu sınıflandırmalar için farklı özelliklere sahip, dört tanesi federe olmayan olan, toplamda 28 adet model test edilmiştir.

Gerçekleştirilen çalışmada, ikili sınıflandırma için en başarılı federe olmayan model 0.999 doğruluk değerine; çoklu sınıflandırmada ise en başarılı federe olmayan model 0.843 doğruluk değerine ulaşmıştır. Ayrıca ikili ve çoklu sınıflandırma için aynı derin öğrenme mimarisini kullanan, farklı kullanıcı sayılarına sahip, FÖ tabanlı modeller de geliştirilmiş olup, bu modellerin de aynı doğruluk değerlerine ulaştığı tespit edilmiştir.

Bu çalışmanın en yenilikçi yönü, CIC-MalMem-2022 gibi güncel bir veri setiyle hem geleneksel hem federe yaklaşımlı bir malware tespiti ve sınıflandırması yapılmış olmasıdır. Ayrıca literatürde bulunan önceki çalışmalara göre; daha fazla bilgi içeren, daha yeni, daha farklı sonuçlar içermesi de çalışmanın diğer yenilikçi yönleridir.

Federe öğrenme, günümüzde gelişmeye devam eden yeni sayılabilecek teknik bir alandır. Bu alanda yapılacak farklı çalışmalar, bu alanın gelişmesine katkı sunacaktır. Bu tez çalışması da FÖ alanında araştırmalar yapmak isteyen araştırmacılara rehberlik edecek düzeydedir.

Sonuç olarak, bu tez çalışması, farklı veri setleri ve sınıflandırma görevlerinde gösterdiği üstün performans ile literatürde önemli bir yer edinmiştir. FÖ yaklaşımının etkin kullanımı ile veri güvenliğini sağlamakta ve yüksek doğruluk oranlarına ulaşmaktadır. Bu çalışmanın bulguları, gelecekteki araştırmalar için sağlam bir temel oluşturarak, federe öğrenmenin siber güvenlik uygulamalarında etkin bir şekilde kullanılabileceğini göstermektedir. FÖ yöntemlerinin gelişimi, veri mahremiyeti ve güvenliği konularında yeni fırsatlar sunarak, daha güvenilir ve etkili modellerin geliştirilmesine olanak tanıyacaktır.

Bu nedenle, bu çalışmanın bulguları ve önerileri, sadece mevcut siber güvenlik sorunlarının çözümüne katkı sağlamakla kalmayacak, aynı zamanda bu alanda yeni araştırma ve geliştirme projelerine de ilham verecektir.

Bu çalışma, FÖ ve DÖ algoritmalarının birleştirilmesiyle elde edilen başarının bir göstergesi olup, bu tekniklerin daha geniş bir uygulama yelpazesinde kullanılmasının önünü açacaktır.

KAYNAKLAR

[1] Bettany, A. and M. Halsey, *Windows virus and malware troubleshooting*. 2017: Springer.

[2] McMahan, B., et al. *Communication-efficient learning of deep networks from decentralized data*. in *Artificial intelligence and statistics*. 2017. PMLR.

[3] Taghdiri, E., *Distributed Edge Computing in SDN-IoT Network.*

[4] Noul, D. and T. Jameel, *IoT Safety Frameworks: Managing Knowledge for Sustainable Societal Impact.*

[5] Akhmetova, S. and A. Ibrayeva, *The EUROPEAN DATA PROTECTION MODEL AS A GLOBAL PRIVACY STANDARD.* Farabi Journal of Social Sciences, 2023. **9**(1): p. 79-86.

[6] Goddard, M., *The EU General Data Protection Regulation (GDPR): European regulation that has a global impact.* International Journal of Market Research, 2017. **59**(6): p. 703-705.

[7] Regulation, G.D.P., *Guide to the general data protection regulation (GDPR).* Official Journal of the European Union. https://gdpr-info.eu, 2018.

[8] Rustad, M.L. and T.H. Koenig, *Towards a global data privacy standard.* Fla. L. Rev., 2019. **71**: p. 365.

[9] Poniatowska-Jaksch, M., *Threat of ransomware attack in Europe.* Journal of Modern Science, 2023. **52**(3): p. 379-395.

[10] Sheppard, C., *The shadow of 2022 is still shaping cyberthreats.* Network Security, 2023. **2023**(8).

[11] Aslan, Ö.A. and R. Samet, *A comprehensive review on malware detection approaches.* IEEE access, 2020. **8**: p. 6249-6271.

[12] Cybersecurity, T.E.U.A.f. *ENISA Threat Landscape 2023*. [cited May, 2024; Available from: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023.

[13] Carrier, T., *Detecting obfuscated malware using memory feature engineering.* 2021.

[14] Bettany, A., et al., *What is malware?* Windows Virus and Malware Troubleshooting, 2017: p. 1-8.

[15] Khan, I., *An introduction to computer viruses: problems and solutions.* Library Hi Tech News, 2012. **29**(7): p. 8-12.

[16] Gladychev, N., *Computer Viruses: The Abstract Theory Revisited.* arXiv preprint arXiv:1912.06510, 2019.

[17] Nazario, J., *Defense and detection strategies against Internet worms*. 2004: Artech House.

[18] Holz, T., et al., *Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm.* Leet, 2008. **8**(1): p. 1-9.

[19] Weaver, N., et al. *A taxonomy of computer worms*. in *Proceedings of the 2003 ACM workshop on Rapid Malcode*. 2003.

[20] Arce, D.G., *Malware and market share.* Journal of Cybersecurity, 2018. **4**(1): p. tyy010.

[21] Li, C., et al., *Backdoor attack on machine learning based android malware detectors.* IEEE Transactions on dependable and secure computing, 2021. **19**(5): p. 3357-3370.

[22] Hossain, M.A. and M.S. Islam, *Enhanced detection of obfuscated malware in memory dumps: a machine learning approach for advanced cybersecurity.* Cybersecurity, 2024. **7**(1): p. 16.
[23] Leon, R.S., et al., *Hypervisor-assisted dynamic malware analysis.* Cybersecurity, 2021. **4**: p. 1-14.
[24] Jimmy, F., *Understanding Ransomware Attacks: Trends and Prevention Strategies.* DOI: https://doi.org/10.60087/jklst. vol2, (1): p. p214.
[25] Hull, G., H. John, and B. Arief, *Ransomware deployment methods and analysis: views from a predictive model and human responses.* Crime Science, 2019. **8**(1): p. 1-22.
[26] Benmalek, M., *Ransomware on cyber-physical systems: Taxonomies, case studies, security gaps, and open challenges.* Internet of Things and Cyber-Physical Systems, 2024. **4**: p. 186-202.
[27] Vehabovic, A., et al. *Ransomware detection and classification strategies*. in *2022 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. 2022. IEEE.
[28] Garrie, D.B. and L.R. Komagome, *The voyeur among us: Navigating around the global spyware epidemic.* Journal of International Technology and Information Management, 2008. **17**(2): p. 3.
[29] Hu, Q. and T. Dinev, *Is spyware an Internet nuisance or public menace?* Commun. ACM, 2005. **48**(8): p. 61–66.
[30] Erbschloe, M., *Trojans, worms, and spyware: a computer security professional's guide to malicious code*. 2004: Elsevier.
[31] Kim, W., et al., *The dark side of the Internet: Attacks, costs and responses.* Information Systems, 2011. **36**(3): p. 675-705.
[32] Uscilowski, B., *Mobile adware and malware analysis.* Symantec Corp, 2013. **1**.
[33] Chien, E. *Techniques of adware and spyware*. in *the Proceedings of the Fifteenth Virus Bulletin Conference, Dublin Ireland*. 2005. Citeseer.
[34] Erturk, E. *A case study in open source software security and privacy: Android adware*. in *World Congress on Internet Security (WorldCIS-2012)*. 2012.
[35] Park, J. and S. Jung, *Android adware detection using soot and CFG.* Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 2022. **13**(4): p. 94-104.
[36] Schultz, E.E. and E. Ray, *Rootkits: the ultimate malware threat.* Inf. Secur. Manag. Handb, 2008. **2**: p. 175.
[37] Hoglund, G. and J. Butler, *Rootkits: subverting the Windows kernel*. 2006: Addison-Wesley Professional.
[38] Monroe, J. and N.A.A.M.D.O.C. SCIENCE, *An Analysis of Root-Kit Technologies and Strategies.* 2010.
[39] Khattak, S., et al., *A Taxonomy of Botnet Behavior, Detection, and Defense.* IEEE Communications Surveys & Tutorials, 2014. **16**(2): p. 898-924.
[40] Silva, S.S.C., et al., *Botnets: A survey.* Computer Networks, 2013. **57**(2): p. 378-403.
[41] Feily, M., A. Shahrestani, and S. Ramadass. *A Survey of Botnet and Botnet Detection*. in *2009 Third International Conference on Emerging Security Information, Systems and Technologies*. 2009.
[42] Zhang, C., *Impact of Defending Strategy Decision on DDoS Attack.* Complexity, 2021. **2021**: p. 6694383.

[43] Cil, A.E., K. Yildiz, and A. Buldu, *Detection of DDoS attacks with feed forward based deep neural network model.* Expert Systems with Applications, 2021. **169**: p. 114520.
[44] Alazab, M., et al. *Malicious Spam Emails Developments and Authorship Attribution*. in *2013 Fourth Cybercrime and Trustworthy Computing Workshop*. 2013.
[45] Wen, S., et al., *Modeling and Analysis on the Propagation Dynamics of Modern Email Malware.* IEEE Transactions on Dependable and Secure Computing, 2014. **11**(4): p. 361-374.
[46] Perryman, S., *A Design-Science Approach to Nullifying Malicious Links and Attachments in Spam Email*. 2021, Colorado Technical University.
[47] Ndibwile, J.D., *Validation Agents and Persuasive Designs for Phishing Detection and Update Compliance on Smartphones.* 2019.
[48] El Hammouti, W., *THE USE OF SOFTWARE AND HARDWARE VULNERABILITIES BY LAW ENFORCEMENT.* 2019.
[49] Molotsi, K.N., *Developing a framework for secure use of portable storage devices in a corporate environment*. 2015: University of Johannesburg (South Africa).
[50] Al-Zarouni, M., *The reality of risks from consented use of USB devices.* 2006.
[51] Krombholz, K., et al., *Advanced social engineering attacks.* Journal of Information Security and Applications, 2015. **22**: p. 113-122.
[52] Longtchi, T.T., et al., *Internet-Based Social Engineering Psychology, Attacks, and Defenses: A Survey.* Proceedings of the IEEE, 2024. **112**(3): p. 210-246.
[53] Learning, F., *Collaborative machine learning without centralized training data.* Publication date: Thursday, April, 2017. **6**.
[54] Yang, Q., et al., *Federated machine learning: Concept and applications.* ACM Transactions on Intelligent Systems and Technology (TIST), 2019. **10**(2): p. 1-19.
[55] Zhang, J., et al. *Gpfl: Simultaneously learning global and personalized feature information for personalized federated learning*. in *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023.
[56] Gabrielli, E., G. Pica, and G. Tolomei, *A survey on decentralized federated learning.* arXiv preprint arXiv:2308.04604, 2023.
[57] Guan, H., et al., *Federated learning for medical image analysis: A survey.* Pattern Recognition, 2024: p. 110424.
[58] Zhang, X., et al., *Fedpd: A federated learning framework with adaptivity to non-iid data.* IEEE Transactions on Signal Processing, 2021. **69**: p. 6055-6070.
[59] Chowdhury, A., et al. *A review of medical federated learning: Applications in oncology and cancer research*. in *International MICCAI Brainlesion Workshop*. 2021. Springer.
[60] Yan, G., H. Wang, and J. Li. *Seizing critical learning periods in federated learning*. in *Proceedings of the AAAI Conference on Artificial Intelligence*. 2022.
[61] Tu, X., et al., *Incentive mechanisms for federated learning: From economic and game theoretic perspective.* IEEE transactions on cognitive communications and networking, 2022. **8**(3): p. 1566-1593.
[62] Li, Z., et al. *Federated learning with gan-based data synthesis for non-iid clients*. in *International Workshop on Trustworthy Federated Learning*. 2022. Springer.

[63] BÜYÜKTANIR, B., et al., *du-CBA: Data-agnostic and incremental classification-based association rules extraction architecture du-CBA: Veriden habersiz ve artirimli siniflandirmaya dayali birliktelik kurallari çikarma mimarisi.* Journal of the Faculty of Engineering and Architecture of Gazi University, 2023. **38**(3).
[64] Konečný, J., et al., *Federated learning: Strategies for improving communication efficiency.* arXiv preprint arXiv:1610.05492, 2016.
[65] Beltrán, E.T.M., et al., *Decentralized Federated Learning: Fundamentals, State of the Art, Frameworks, Trends, and Challenges.* IEEE Communications Surveys & Tutorials, 2023. **25**(4): p. 2983-3013.
[66] Wang, Z., et al. *Beyond Inferring Class Representatives: User-Level Privacy Leakage From Federated Learning*. in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 2019.
[67] Mahmoudi, A., et al., *FedCau: A Proactive Stop Policy for Communication and Computation Efficient Federated Learning.* IEEE Transactions on Wireless Communications, 2024: p. 1-1.
[68] Jiao, S., et al., *Efficient DP-FL: Efficient Differential Privacy Federated Learning Based on Early Stopping Mechanism.* Computer Systems Science & Engineering, 2024. **48**(1).
[69] Qayyum, A., et al., *Collaborative Federated Learning for Healthcare: Multi-Modal COVID-19 Diagnosis at the Edge.* IEEE Open Journal of the Computer Society, 2022. **3**: p. 172-184.
[70] Sheller, M.J., et al., *Federated learning in medicine: facilitating multi-institutional collaborations without sharing patient data.* Scientific reports, 2020. **10**(1): p. 12598.
[71] Kairouz, P., et al., *Advances and open problems in federated learning.* Foundations and trends® in machine learning, 2021. **14**(1–2): p. 1-210.
[72] Li, T., et al., *Federated optimization in heterogeneous networks.* Proceedings of Machine learning and systems, 2020. **2**: p. 429-450.
[73] Bonawitz, K., et al., *Towards federated learning at scale: System design.* Proceedings of machine learning and systems, 2019. **1**: p. 374-388.
[74] Wang, H., et al., *Federated learning with matched averaging.* arXiv preprint arXiv:2002.06440, 2020.
[75] Chen, W., et al., *The need for energy efficiency and economic prosperity in a sustainable environment.* Gondwana Research, 2024. **127**: p. 22-35.
[76] Mishra, D.P., et al., *Integration of Electric Vehicles with Smart Grid*, in *Energy and Environmental Aspects of Emerging Technologies for Smart Grid*, S.R. Salkuti, Editor. 2024, Springer International Publishing: Cham. p. 269-290.
[77] Hanley, N., et al., *Do increases in energy efficiency improve environmental quality and sustainability?* Ecological Economics, 2009. **68**(3): p. 692-709.
[78] Hafez, F.S., et al., *Energy Efficiency in Sustainable Buildings: A Systematic Review with Taxonomy, Challenges, Motivations, Methodological Aspects, Recommendations, and Pathways for Future Research.* Energy Strategy Reviews, 2023. **45**: p. 101013.
[79] Rieke, N., et al., *The future of digital health with federated learning.* NPJ digital medicine, 2020. **3**(1): p. 1-7.
[80] Pandya, S., et al., *Federated learning for smart cities: A comprehensive survey.* Sustainable Energy Technologies and Assessments, 2023. **55**: p. 102987.

[81] Prayitno, et al., *A systematic review of federated learning in the healthcare area: From the perspective of data properties and applications.* Applied Sciences, 2021. **11**(23): p. 11191.
[82] Chaddad, A., Y. Wu, and C. Desrosiers, *Federated Learning for Healthcare Applications.* IEEE Internet of Things Journal, 2024. **11**(5): p. 7339-7358.
[83] Mishra, A., et al., *A federated learning approach for smart healthcare systems.* CSI Transactions on ICT, 2023. **11**(1): p. 39-44.
[84] Abdul Salam, M., et al., *Federated learning model for credit card fraud detection with data balancing techniques.* Neural Computing and Applications, 2024. **36**(11): p. 6231-6256.
[85] Luo, Y., et al. *Application Research of Vertical Federated Learning Technology in Banking Risk Control Model Strategy*. in *2023 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. 2023.
[86] Yuxin, M. and W. Honglin, *Federated Learning Based on Data Divergence and Differential Privacy in Financial Risk Control Research.* Computers, Materials \& Continua, 2023. **75**(1): p. 863--878.
[87] Xu, H., et al., *Multi-Level Split Federated Learning for Large-Scale AIoT System Based on Smart Cities.* Future Internet, 2024. **16**(3): p. 82.
[88] Priyadarshini, I., *Anomaly Detection of IoT Cyberattacks in Smart Cities Using Federated Learning and Split Learning.* Big Data and Cognitive Computing, 2024. **8**(3): p. 21.
[89] Zheng, Z., et al., *Applications of federated learning in smart cities: recent advances, taxonomy, and open challenges.* Connection Science, 2022. **34**(1): p. 1-28.
[90] Acar, B. and M. Sterling, *Ensuring Federated Learning Reliability for Infrastructure-Enhanced Autonomous Driving.* Journal of Intelligent and Connected Vehicles, 2023. **6**(3): p. 125-135.
[91] Xianjia, Y., et al., *Federated Learning in Robotic and Autonomous Systems.* Procedia Computer Science, 2021. **191**: p. 135-142.
[92] Tong, Z., et al., *UAV-Assisted Covert Federated Learning over mmWave Massive MIMO.* IEEE Transactions on Wireless Communications, 2024: p. 1-1.
[93] Byun, S., et al., *Secure Aggregation for Privacy-preserving Federated Learning in Vehicular Networks.* ACM J. Auton. Transport. Syst., 2024.
[94] N, M., et al., *Federated transfer learning for intrusion detection system in industrial iot 4.0.* Multimedia Tools and Applications, 2024. **83**(19): p. 57913-57941.
[95] Sudhina Kumar, G.K., K. Krishna Prakasha, and B. Muniyal. *Intrusion Detection Using Federated Learning*. in *Applications and Techniques in Information Security*. 2023. Singapore: Springer Nature Singapore.
[96] Driss, M., et al., *A federated learning framework for cyberattack detection in vehicular sensor networks.* Complex & Intelligent Systems, 2022. **8**(5): p. 4221-4235.
[97] Reisizadeh, A., et al., *Straggler-Resilient Federated Learning: Leveraging the Interplay Between Statistical Accuracy and System Heterogeneity.* IEEE Journal on Selected Areas in Information Theory, 2022. **3**(2): p. 197-205.

[98] Zawad, S., et al. *Curse or redemption? how data heterogeneity affects the robustness of federated learning*. in *Proceedings of the AAAI conference on artificial intelligence*. 2021.
[99] Ahmad, A., W. Luo, and A. Robles-Kelly, *Robust federated learning under statistical heterogeneity via hessian-weighted aggregation.* Machine Learning, 2023. **112**(2): p. 633-654.
[100] Zhao, H., et al., *Faster Rates for Compressed Federated Learning with Client-Variance Reduction.* SIAM Journal on Mathematics of Data Science, 2024. **6**(1): p. 154-175.
[101] Yu, X., et al. *FedSC: Compatible Gradient Compression for Communication-Efficient Federated Learning*. in *Algorithms and Architectures for Parallel Processing*. 2024. Singapore: Springer Nature Singapore.
[102] Ren, Y., et al., *Two-layer accumulated quantized compression for communication-efficient federated learning: TLAQC.* Scientific Reports, 2023. **13**(1): p. 11658.
[103] Jiang, Z., et al., *Computation and Communication Efficient Federated Learning With Adaptive Model Pruning.* IEEE Transactions on Mobile Computing, 2024. **23**(03): p. 2003-2021.
[104] Asad, M., et al. *Evaluating the Communication Efficiency in Federated Learning Algorithms*. in *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. 2021.
[105] Hua, M., et al. *Energy-Efficient Federated Transfer Learning in 6G Native AI Networks*. in *2023 IEEE Globecom Workshops (GC Wkshps)*. 2023.
[106] Lin, L., et al., *A Joint Gradient and Loss Based Clustered Federated Learning Design.* arXiv preprint arXiv:2311.13665, 2023.
[107] Pfeiffer, K., et al., *Federated learning for computationally constrained heterogeneous devices: A survey.* ACM Computing Surveys, 2023. **55**(14s): p. 1-27.
[108] Gupta, M., M. Kumar, and Y. Gupta, *A Blockchain-Empowered Federated Learning-based Framework for Data Privacy in Lung Disease Detection System.* Computers in Human Behavior, 2024: p. 108302.
[109] Hallaji, E., et al., *Decentralized Federated Learning: A Survey on Security and Privacy.* IEEE Transactions on Big Data, 2024.
[110] Hasan, J., *Security and privacy issues of federated learning.* arXiv preprint arXiv:2307.12181, 2023.
[111] Yu, S. and L. Cui, *Security and Privacy in Federated Learning*. 2023: Springer Nature.
[112] Ekmefjord, M., et al. *Scalable federated machine learning with fedn*. in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2022. IEEE.
[113] Staňo, M., et al. *Federated learning methods for analytics of big and sensitive distributed data and survey*. in *2023 IEEE 17th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. 2023. IEEE.
[114] Louk, M.H.L. and B.A. Tama, *Tree-based classifier ensembles for PE malware analysis: A performance revisit.* Algorithms, 2022. **15**(9): p. 332.
[115] Talukder, M.A., et al., *A dependable hybrid machine learning model for network intrusion detection.* Journal of Information Security and Applications, 2023. **72**: p. 103405.

[116] Tavallaee, M., et al. *A detailed analysis of the KDD CUP 99 data set*. in *2009 IEEE symposium on computational intelligence for security and defense applications*. 2009. Ieee.
[117] Smith, D., S. Khorsandroo, and K. Roy. *Supervised and unsupervised learning techniques utilizing malware datasets*. in *2023 IEEE 2nd International Conference on AI in Cybersecurity (ICAIC)*. 2023. IEEE.
[118] Mezina, A. and R. Burget. *Obfuscated malware detection using dilated convolutional network*. in *2022 14th international congress on ultra modern telecommunications and control systems and workshops (ICUMT)*. 2022. IEEE.
[119] Dener, M., G. Ok, and A. Orman, *Malware detection using memory analysis data in big data environment.* Applied Sciences, 2022. **12**(17): p. 8604.
[120] Lin, K.-Y. and W.-R. Huang. *Using federated learning on malware classification*. in *2020 22nd international conference on advanced communication technology (ICACT)*. 2020. IEEE.
[121] Rey, V., et al., *Federated learning for malware detection in IoT devices.* Computer Networks, 2022. **204**: p. 108693.
[122] Hsu, R.-H., et al. *A privacy-preserving federated learning system for android malware detection based on edge computing*. in *2020 15th Asia Joint Conference on Information Security (AsiaJCIS)*. 2020. IEEE.
[123] Gálvez, R., V. Moonsamy, and C. Diaz, *Less is More: A privacy-respecting Android malware classifier using federated learning.* arXiv preprint arXiv:2007.08319, 2020.
[124] Jiang, C., et al., *FedHGCDroid: An adaptive multi-dimensional federated learning for privacy-preserving android Malware classification.* Entropy, 2022. **24**(7): p. 919.
[125] Taheri, R., et al., *FED-IIoT: A robust federated malware detection architecture in industrial IoT.* IEEE transactions on industrial informatics, 2020. **17**(12): p. 8442-8452.
[126] POLAT, E.I., *Bizans saldırısına dayanıklı federe öğrenme.* Yüksek Lisans Tezi, Orta Doğu Teknik Üniversitesi Enformatik Enstitüsü, 2021.
[127] ÇAKIR, Z., *Hataya açık kanallar üzerinde enerji hasadı ile tarafsız federe öğrenme ve bilgi yaşı.* Yüksek Lisans Tezi, Orta Doğu Teknik Üniversitesi Fen Bilimleri Enstitüsü, 2022.
[128] AYGÜN, O., *Kablosuz kanallar üzerinden havada birleştirme ile federe öğrenme.* Yüksek Lisans Tezi, İhsan Doğramacı Bilkent Üniversitesi Mühendislik ve Fen Bilimleri Enstitüsü, 2022.
[129] OKTAY, E., *Genomik çalışmalarda kullanılan makine öğrenmesi modellerinin blokzincir temelli bir eko sistemde federe öğrenme yöntemiyle geliştirilmesi için kavramsal bir çalışma.* Yüksek Lisans Tezi, İstanbul Medeniyet Üniversitesi Lisansüstü Eğitim Enstitüsü, 2022.
[130] ERİŞ, M.C., *Kablosuz internet ağlarındaki kısıtların federe öğrenmeye olan etkilerinin ortaya çıkarılması.* Yüksek Lisans Tezi, İstanbul Teknik Üniversitesi Lisansüstü Eğitim Enstitüsü, 2022.
[131] ÇAĞLAYAN, C., *Federe öğrenmede zararlı modellerin tespiti için kümeleme tabanlı skorlama.* Yüksek Lisans Tezi, Boğaziçi Üniversitesi Fen Bilimleri Enstitüsü, 2023.
[132] Kiltz, S., A. Lang, and J. Dittmann, *Malware: specialized trojan horse*, in *Cyber Warfare and Cyber Terrorism*. 2007, IGI Global. p. 154-160.

[133] Stafford, T.F. and A. Urbaczewski, *Spyware: The ghost in the machine.* The Communications of the Association for Information Systems, 2004. **14**(1): p. 49.
[134] Richardson, R. and M.M. North, *Ransomware: Evolution, mitigation and prevention.* International Management Review, 2017. **13**(1): p. 10.
[135] O'Kane, P., S. Sezer, and D. Carlin, *Evolution of ransomware.* Iet Networks, 2018. **7**(5): p. 321-327.
[136] Li, J., *CS 332-006: Principles of Operating Systems.* 2020.
[137] Fox, R., *Linux with operating system concepts*. 2021: Chapman and Hall/CRC.
[138] Butwall, M., *Data Normalization and Standardization: Impacting Classification Model Accuracy.* International Journal of Computer Applications, 2021. **183**(35): p. 6-9.
[139] Haykin, S., *Neural networks: a comprehensive foundation*. 1998: Prentice Hall PTR.
[140] Goodfellow, I., Y. Bengio, and A. Courville, *Deep learning*. 2016: MIT press.
[141] Yildiz, K., *Identification of wool and mohair fibres with texture feature extraction and deep learning.* IET Image Processing, 2020. **14**(2): p. 348-353.
[142] ÜLKÜ, E.E. and K. YILDIZ, *Classification of hazelnuts with CNN based deep learning system.* 2022.
[143] Boyar, T. and K. Yıldız, *Powdery Mildew Detection in Hazelnut with Deep Learning.* Hittite Journal of Science and Engineering, 2022. **9**(3): p. 159-166.
[144] Dong, S., P. Wang, and K. Abbas, *A survey on deep learning and its applications.* Computer Science Review, 2021. **40**: p. 100379.
[145] Nwankpa, C., et al., *Activation functions: Comparison of trends in practice and research for deep learning.* arXiv preprint arXiv:1811.03378, 2018.
[146] Bebis, G. and M. Georgiopoulos, *Feed-forward neural networks.* IEEE Potentials, 1994. **13**(4): p. 27-31.
[147] Ilonen, J., J.-K. Kamarainen, and J. Lampinen, *Differential Evolution Training Algorithm for Feed-Forward Neural Networks.* Neural Processing Letters, 2003. **17**(1): p. 93-105.
[148] Svozil, D., V. Kvasnicka, and J.í. Pospichal, *Introduction to multi-layer feed-forward neural networks.* Chemometrics and Intelligent Laboratory Systems, 1997. **39**(1): p. 43-62.
[149] Hadjiprocopis, A. and P. Smith. *Feed forward neural network entities*. in *International Work-Conference on Artificial Neural Networks*. 1997. Springer.
[150] Rumelhart, D.E., G.E. Hinton, and R.J. Williams, *Learning representations by back-propagating errors.* nature, 1986. **323**(6088): p. 533-536.
[151] Hochreiter, S. and J. Schmidhuber, *Long short-term memory.* Neural computation, 1997. **9**(8): p. 1735-1780.
[152] Hochreiter, S., *The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions.* International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 1998. **06**(02): p. 107-116.
[153] Tan, H.H. and K.H. Lim. *Vanishing Gradient Mitigation with Deep Learning Neural Network Optimization*. in *2019 7th International Conference on Smart Computing & Communications (ICSCC)*. 2019.
[154] Hanin, B., *Which neural net architectures give rise to exploding and vanishing gradients?* Advances in neural information processing systems, 2018. **31**.

[155] Glorot, X. and Y. Bengio. *Understanding the difficulty of training deep feedforward neural networks*. in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010. JMLR Workshop and Conference Proceedings.
[156] Ioffe, S. and C. Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. in *International conference on machine learning*. 2015. pmlr.
[157] Gers, F.A. and J. Schmidhuber. *Recurrent nets that time and count*. in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*. 2000. IEEE.
[158] Sherstinsky, A., *Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network.* Physica D: Nonlinear Phenomena, 2020. **404**: p. 132306.
[159] Ramachandran, P., B. Zoph, and Q.V. Le, *Searching for activation functions.* arXiv preprint arXiv:1710.05941, 2017.
[160] Xu, S. and L. Chen, *A novel approach for determining the optimal number of hidden layer neurons for FNN's and its application in data mining.* 2008.
[161] Vujičić, T., et al. *Comparative analysis of methods for determining number of hidden neurons in artificial neural network*. in *Central european conference on information and intelligent systems*. 2016.
[162] Alvarez, J.M. and M. Salzmann, *Learning the number of neurons in deep networks.* Advances in neural information processing systems, 2016. **29**.
[163] Bostock, M., V. Ogievetsky, and J. Heer, *D³ data-driven documents.* IEEE transactions on visualization and computer graphics, 2011. **17**(12): p. 2301-2309.
[164] Baldi, P. and P.J. Sadowski, *Understanding dropout.* Advances in neural information processing systems, 2013. **26**.
[165] Srivastava, N., et al., *Dropout: a simple way to prevent neural networks from overfitting.* The journal of machine learning research, 2014. **15**(1): p. 1929-1958.
[166] Blanchard, P., D.J. Higham, and N.J. Higham, *Accurate computation of the log-sum-exp and softmax functions.* arXiv preprint arXiv:1909.03469, 2019.
[167] Krizhevsky, A., I. Sutskever, and G.E. Hinton, *ImageNet classification with deep convolutional neural networks.* Communications of the ACM, 2017. **60**(6): p. 84-90.
[168] Boncella, R.J., C. Authorities, and S. Channels, *Secure Sockets Layer (SSL).* The Internet Encyclopedia, 2004.
[169] Nechvatal, J., et al., *Report on the development of the Advanced Encryption Standard (AES).* Journal of research of the National Institute of Standards and Technology, 2001. **106**(3): p. 511.
[170] Pedregosa, F., et al., *Scikit-learn: Machine learning in Python.* the Journal of machine Learning research, 2011. **12**: p. 2825-2830.
[171] Idris, I., *NumPy: Beginner's Guide*. 2015: Packt Publishing Ltd.
[172] McKinney, W. and P. Team, *Pandas-Powerful python data analysis toolkit.* Pandas—Powerful Python Data Analysis Toolkit, 2015. **1625**.
[173] Abadi, M., et al. *{TensorFlow}: a system for {Large-Scale} machine learning*. in *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016.
[174] Gulli, A. and S. Pal, *Deep learning with Keras*. 2017: Packt Publishing Ltd.

[175] Rivest, R.L., A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems.* Communications of the ACM, 1978. **21**(2): p. 120-126.
[176] Project, O., *OpenSSL: The open source toolkit for SSL/TLS*. 2003.
[177] Malygin, E., *INVESTIGATION OF DIGITAL CERTIFICATES: Creation of self-signed certificate on Windows 8.* 2014.
[178] Chinni Prashanth, D. and V. Lavanya, *Securing the data in cloud using fernet.* 2022.
[179] Yeom, S., et al., *Overfitting, robustness, and malicious algorithms: A study of potential causes of privacy risk in machine learning.* Journal of Computer Security, 2020. **28**(1): p. 35-70.
[180] Ji, Z., J. Li, and M. Telgarsky, *Early-stopped neural networks are consistent.* Advances in Neural Information Processing Systems, 2021. **34**: p. 1805-1817.
[181] Prechelt, L., *Early stopping-but when?*, in *Neural Networks: Tricks of the trade*. 2002, Springer. p. 55-69.

ÖZGEÇMİŞ

Adı Soyadı : *Zeki ÇIPLAK*

Yabancı Dili : İngilizce

Öğrenim Durumu:

Derece	Bölüm/Program	Üniversite	Mezuniyet
Lisans	Fizik	Sakarya Üniversitesi	Ağustos 2006
Yüksek Lisans	Fizik	Sakarya Üniversitesi	Temmuz 2009
Lisans	Yönetim Bilişim Sistemleri	Anadolu Üniversitesi	Ocak 2022

İş Deneyimi:

Yıl	Firma/Kurum	Görev
2008-2019	Freelance (Serbest Geliştirici)	Yazılım Geliştirme
2019-2022	Udemy.com	Programlama Eğitmeni
2022-Halen	İstanbul Gedik Üniversitesi	Öğretim Görevlisi

Akademik Çalışmalar:

1. Çıplak, Z., Yıldız, S. G., & Doğan, B. (2024). Hibrit, Uzaktan ve Yerinde Çalışma Modellerinde, Çalışanların Görüşlerinin Değerlendirilmesi. *Bilgi Yönetimi*. https://doi.org/10.33721/by.1437637
2. Ciplak, Z., & Yildiz, K. (2024). Occupational groups prediction in Turkish Twitter data by using machine learning algorithms with multinomial approach. *Expert Systems with Applications*, 124175. https://dx.doi.org/10.1016/j.eswa.2024.124175
3. Çıplak, Z. (2009). Ni-Ag alaşımlarına paladyum katkısının termodinamik ve mekanik özelliklere etkisinin moleküler dinamik incelenmesi. *Yüksek Lisans Tezi*, Sakarya Üniversitesi Fen Bilimleri Enstitüsü, Sakarya, Türkiye. https://acikerisim.sakarya.edu.tr/handle/20.500.12619/80954