Research article

Journal of Smart Systems Research (JOINSSR) 7(1), 16-29, 2026
Received: 17-Feb-2026 Accepted: 17-May-2026
https://doi.org/10.58769/joinssr.1891109

Merkezi İzleme Yazılımı ile Anomali Tespiti ve Yapay Zekâ Destekli Kapasite Planlama: AIOps Odaklı Bir Yaklaşım

Ali YAZICI[1*], Ahmet ZENGİN [2]

[1] Bilgisayar Ve Bilişim Mühendisliği Anabilim Dalı, Fen Bilimleri Enstitüsü, Sakarya Üniversitesi, Türkiye.
yaziciali@gmail.com
[2] Bilgisayar Ve Bilişim Mühendisliği Anabilim Dalı, Fen Bilimleri Enstitüsü, Sakarya Üniversitesi, Türkiye.
azengin@sakarya.edu.tr

ÖZ

Modern bilgi teknolojileri (BT) altyapılarının artan karmaşıklığı ve veri hacmindeki üstel artış, operasyonel sürekliliği sağlamakla yükümlü ekipler için geleneksel izleme yöntemlerini yetersiz kılmaktadır. Statik eşik değerlere dayalı reaktif sistemler, dinamik yük değişimlerine uyum sağlayamamakta; bu durum kritik performans sorunlarının gözden kaçırılmasına veya alarm yorgunluğuna yol açan yanlış uyarılara neden olmaktadır. Bu çalışmada, endüstri standardı Zabbix izleme yazılımı, TimescaleDB zaman serisi veritabanı ve gelişmiş derin öğrenme algoritmalarını entegre eden hibrit bir BT Operasyonları için Yapay Zeka (Artificial Intelligence for IT Operations - AIOps) mimarisi önerilmiştir. Çalışma kapsamında, gerçek dünya sunucu verileri üzerinde istatistiksel (Z-Score), makine öğrenimi (Isolation Forest, Local Outlier Factor - LOF) ve derin öğrenme (Long Short-Term Memory - LSTM, Gated Recurrent Unit - GRU) tabanlı anomali tespit modelleri karşılaştırılmış ve alan uzmanları tarafından doğrulanan bir zemin doğrusu referans alınarak karşılaştırmalı olarak analiz edilmiştir. Deneysel sonuçlar, GRU modelinin %76,92 $F_{0,5}$ Skoru ile yanlış alarmları minimize etmede en başarılı yöntem olduğunu göstermiştir. Kapasite planlama tarafında ise Prophet algoritması kullanılarak, disk doluluk oranları ve mevsimsel kaynak tükenişleri yüksek doğrulukla öngörülmüştür. Önerilen sistem, reaktif yönetimden proaktif ve öngörüsel yönetime geçişi sağlayarak operasyonel verimliliği artırmaktadır.

Anahtar Kelimeler: AIOps, anomali tespiti, kapasite planlama, Zabbix, derin öğrenme.

[*] Sorumlu yazarın e-posta adresi: yaziciali@gmail.com

Cite as: Yazıcı, A. & Zengin, A. (2026). Merkezi İzleme Yazılımı ile Anomali Tespiti ve Yapay Zekâ Destekli Kapasite Planlama: AIOps Odaklı Bir Yaklaşım, *Journal of Smart Systems Research, 7*(1), 16-29. https://doi.org/10.58769/joinssr.1891109

Anomaly Detection and AI-Supported Capacity Planning with Centralized Monitoring Software: An AIOps Focused Approach

ABSTRACT

The increasing complexity of modern Information Technology (IT) infrastructures and the exponential growth in data volume render traditional monitoring methods insufficient for ensuring operational continuity. Reactive systems based on static thresholds fail to adapt to dynamic load changes, leading to missed critical performance issues or false alerts causing alert fatigue. This study proposes a hybrid Artificial Intelligence for IT Operations (AIOps) architecture integrating Zabbix monitoring software, TimescaleDB time-series database, and advanced deep learning algorithms. Unsupervised statistical (Z-Score), machine learning (Isolation Forest, Local Outlier Factor - LOF), and deep learning (Long Short-Term Memory - LSTM, Gated Recurrent Unit - GRU) anomaly detection models were developed using real-world server data and comparatively analyzed against an expert-validated ground truth. Experimental results demonstrated that the GRU model was the most successful method in minimizing false alarms with an $F_{0.5}$-Score of 76.92%. For capacity planning, the Prophet algorithm accurately predicted disk usage rates and seasonal resource exhaustion. The proposed system enhances operational efficiency by enabling a transition from reactive to proactive and predictive management.

Keywords: AIOps, anomaly detection, capacity planning, Zabbix, deep learning.

1 Giriş

Dijital dönüşüm süreçlerinin hızlanmasıyla birlikte, BT altyapıları kurumların operasyonel sürekliliği için hayati bir "sinir sistemi" haline gelmiştir. Büyük ölçekli veri merkezlerinde binlerce sunucu ve ağ cihazı, saniyede milyonlarca veri noktası üretmektedir. Geleneksel izleme araçları, genellikle sistem yöneticilerinin deneyimlerine dayalı statik eşik değerlerine (örneğin "Kaynak kullanım oranı > %90") dayalı reaktif bir yaklaşım sergiler. Ancak bu deterministik yaklaşım, modern sistemlerin dinamik doğası karşısında yetersiz kalmakta ve iki temel soruna yol açmaktadır: Yanlış pozitifler ve yanlış negatifler.

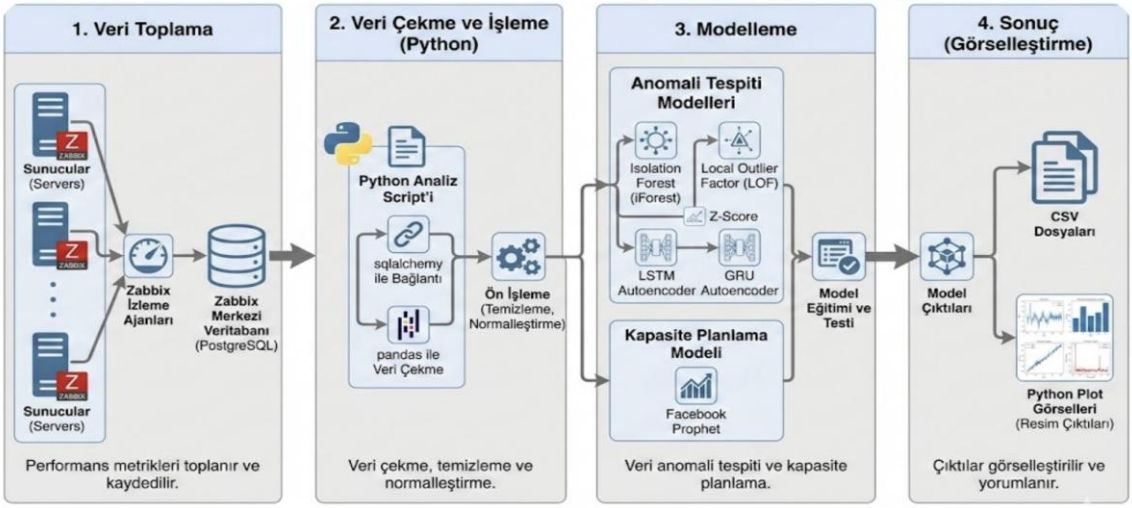
Yanlış pozitifler, meşru iş süreçleri(örneğin gece yedeklemeleri) nedeniyle oluşan geçici yük artışlarının arıza sanılması sonucu ortaya çıkar ve operasyon ekiplerinde "alarm yorgunluğuna" neden olur. Yanlış negatifler ise, statik eşiklerin altında kalan ancak normal davranıştan sapan "sinsi hataların" gözden kaçmasına yol açar (Mendoza & Amistadi, 2018). Bu kısıtlamaları aşmak için, yapay zeka ve makine öğrenimi tekniklerinin kullanıldığı AIOps yaklaşımı bir zorunluluk haline gelmiştir. Son yıllarda yapılan çalışmalar, yapay zekanın BT operasyonlarında ve bulut bilişim altyapılarında (Islam vd., 2024), karmaşık veri setlerinden anlamlı örüntüler çıkarmada kritik bir rol oynadığını göstermektedir. Özellikle Şahin ve Karayel (2024), üretken yapay zekanın iş süreçlerindeki dönüşüm potansiyelini vurgulayarak, manuel operasyonların otonom sistemlere evrilmesinin önemine dikkat çekmiştir. Ayrıca, akıllı kampüs ve ağ altyapılarının enerji verimliliği ile izlenmesinde makine öğrenimi yaklaşımlarının operasyonel maliyetleri düşürdüğü kanıtlanmıştır (Manassra & Işık, 2025). Büyük ölçekli bulut sistemlerinde anomali tespiti üzerine yapılan güncel çalışmalar (Islam vd., 2024) ve AIOps'un olay yönetimindeki rolünü inceleyen araştırmalar (Dang vd., 2021), geleneksel eşik tabanlı sistemlerin yetersiz kaldığı durumlarda otonom sistemlere evrilen izleme çözümlerinin operasyonel yükü hafiflettiğini vurgulamaktadır.

Bu çalışmanın amacı, açık kaynaklı teknolojiler ve yapay zeka algoritmaları kullanılarak maliyet etkin, ölçeklenebilir ve yüksek doğruluklu bir AIOps platformu geliştirmektir. Çalışma, özellikle yanlış

alarmların minimize edilmesi ve kaynakların tükenmeden önce öngörülmesi problemlerine odaklanmaktadır.

2 Metodoloji

Çalışmada geliştirilen sistem, veri toplama, depolama, ön işleme ve analiz katmanlarından oluşan modüler bir mimariye sahiptir. Önerilen platformun genel veri akışı ve temel mimari bileşenleri Şekil 1'de sunulmaktadır. Sistemin tasarımında, donanım seviyesinden uygulama katmanına kadar uzanan geniş bir gözlemlenebilirlik sağlanması ve toplanan verilerin makine öğrenimi modelleriyle entegre edilmesi hedeflenmiştir.



Şekil 1: Veri Akışı ve Mimarisi

2.1 Veri toplama ve depolama altyapısı

Şekil 1'de gösterildiği gibi, veri toplama katmanında heterojen kaynaklardan veri almak için Zabbix izleme yazılımı kullanılmıştır (Zabbix, t.y.). Linux ve Windows sunuculara zabbix ajanlar kurulmuştur. Toplanan merkezi işlem birimi (Central Processing Unit - CPU), rastgele erişimli bellek (Random Access Memory - RAM) ve disk gibi yüksek hacimli zaman serisi verilerinin depolanması için, PostgreSQL veritabanı üzerine kurulu TimescaleDB eklentisi kullanılmıştır. TimescaleDB'nin mimarisi, veriyi zamansal parçalara bölerek yüksek yazma hızı sağlamaktadır (Freedman, 2017). Bu çalışma için 1 aylık bir dönemde, 1 dakikalık frekanslarla toplanan 185 sunucuya ait toplanan veri noktaları analiz edilmiştir.

2.2 Veri Ön İşleme ve Zamansal Veri Dönüşüm Süreçleri

Ham veriler, makine öğrenimi modellerine beslenmeden önce veri kalitesini artırmak amacıyla çeşitli ön işleme adımlarından geçirilmiştir. Öncelikle, sensör hataları veya ağdaki anlık iletim sorunları nedeniyle ortaya çıkan aykırı değerleri temizlemek için sinyalin genel yapısını bozmayan medyan filtresi uygulanmıştır. Modelleme aşamasında, farklı ölçeklere sahip olan metriklerin model ağırlıklarını orantısız etkilememesi ve gradyan inişi optimizasyonunun kararlı çalışması için minimum-maksimum ölçekleyici (Min-Max Scaler) kullanılarak tüm veriler sıfır ile bir aralığına normalize edilmiştir. Zaman serisi verilerindeki ardışık ilişkilerin modellenebilmesi için veriler, yirmi dört adımlık kayan pencerelere bölünmüştür.

Çalışma kapsamında genel mimari ve kapasite planlama süreçleri 185 sunucu üzerinden yürütülmüş olsa da, anomali tespit modellerinin eğitimi ve hassas performans değerlendirmesi için temsil kabiliyeti yüksek, kritik bir sunucu üzerinden derinlemesine bir vaka analizi yürütülmüştür. Bu spesifik sunucuya ait bir aylık veri setinin ayrıştırılmasında kronolojik sıraya sadık kalınmıştır. Veri seti matematiksel olarak şu şekilde bölümlendirilmiştir: ilk on dört günlük kısım modellerin eğitimi için, takip eden dokuz günlük kısım hiperparametre optimizasyonu ve dinamik eşik değerlerinin belirlenmesi için, son yedi günlük kısım ise nihai değerlendirme verisi olarak ayrılmıştır. Eğitim aşamasında kullanılan ilk on dört günlük veri setinde herhangi bir anomali etiketlemesi veya veriyi normalleştirmek adına bir temizleme işlemi yapılmamıştır. Otokodlayıcı ve diğer makine öğrenimi modelleri, ilgili Python kütüphanelerinin algoritmik özellikleri kullanılarak doğrudan ham eğitim verisi üzerinde tamamen denetimsiz olarak eğitilmiştir. Anomalilerin genel veri seti içerisindeki oranının son derece düşük olması nedeniyle, modeller herhangi bir dış müdahaleye ihtiyaç duymadan baskın olan normal sistem davranışını otonom olarak öğrenmişlerdir.

Ancak, modellerin test veri seti üzerindeki tespit performanslarını değerlendirebilmek ve kesinlik ile duyarlılık gibi metrikleri hesaplayabilmek için bir zemin doğrusuna ihtiyaç duyulmuştur. Literatürde denetimsiz modellerin doğruluk metriklerinin hesaplanabilmesi için test verilerinin bu şekilde uzmanlar tarafından sonradan etiketlenmesi kabul gören bir yaklaşımdır (Audibert vd., 2020; Perez & Luckner, 2025). Temsili sunucuya ait 1 dakikalık frekansla toplanan 7 günlük test periyodu tam 10.080 veri noktasından oluşmaktadır. Bu optimize edilmiş hacim, verilerin uzmanlar tarafından görsel olarak doğrulanmasını bilimsel olarak uygulanabilir kılmıştır. Test periyoduna ait zaman serisi grafikleri ve olay kayıtları alan uzmanları tarafından manuel olarak incelenmiş; sistemin normal çalışma davranışından belirgin şekilde sapan, açıklanamayan ani sıçramalar gerçek anomali şeklinde etiketlenmiştir. Oluşturulan bu zemin doğrusu, eğitim sürecinde değil, yalnızca modellerin test seti üzerinde ürettiği sonuçların formülize edilerek doğru tespit ve yanlış alarm oranlarının hesaplanması amacıyla kullanılmıştır. Büyük ölçekli BT sistemlerinde kesin bir zemin doğrusu belirlemenin zorlukları göz önüne alındığında, bu tür daraltılmış vaka analizleri ve uzman onaylı etiketlemeler gerçek dünya verileriyle çalışmanın temel bir gerekliliğidir (Islam vd., 2024).

2.3 Algoritmik çerçeve ve analiz yöntemleri

Anomali tespiti için istatistiksel (Z-Score (Grubbs, 1969)), makine öğrenimi (Isolation Forest (Liu vd., 2008), LOF (Breunig vd., 2000)) ve derin öğrenme (LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho vd., 2014)) modelleri karşılaştırmalı olarak kullanılmıştır. Kapasite planlama için ise mevsimsellik ve trend bileşenlerini ayrıştırabilen Prophet algoritması tercih edilmiştir.

Çalışmada, tekil modellerin (ISO, LOF, Z-Score) yanı sıra, bu modellerin sonuçlarını birleştiren bir hibrit model (ISLOZS: Isolation Forest + LOF + Z-Score) tasarlandı. Bu hibrit yapı ile az kaynak tüketen ve daha hızlı anomali tespiti yapabilmek hedeflendi, fakat yine çok fazla anomali noktası tespit ettiği görülerek başarısız olduğu görülmüştür.

3 Teori, Matematiksel İfadeler ve Hesaplamalar

Bu bölümde, sistemin temelini oluşturan derin öğrenme ve katkısal regresyon algoritmalarının matematiksel altyapısı ile tercih gerekçeleri sunulmaktadır.

3.1 Zaman serilerinde anomali tespiti: GRU tabanlı otokodlayıcı mimarisi

Zaman serisi verilerindeki ardışık bağımlılıkları ve karmaşık örüntüleri modellemek için literatürde sıklıkla LSTM ağları kullanılmaktadır. Ancak yüksek frekanslı BT metriklerinin analizinde LSTM'in

yüksek parametre sayısı, eğitim süresini uzatmakta ve aşırı öğrenme riskini artırmaktadır. Bu sorunu aşmak ve hesaplama karmaşıklığını azaltmak amacıyla geliştirilmiş daha sade bir mimari olan GRU modeli de test edilmiştir (Cho vd., 2014). GRU, hücre durumu ile gizli durumu birleştirerek üç yerine yalnızca iki kapı (güncelleme ve sıfırlama) kullanır. Bir GRU hücresinin anlık aktivasyonu Denklem (1)'de gösterilmektedir:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \qquad (1)$$

Burada z_t güncelleme kapısını, h_{t-1} önceki gizli durumu, $\tilde{h}_t$ aday aktivasyonu ve $\odot$ işareti öğe bazında Hadamard çarpımını temsil eder. Güncelleme kapısı, geçmiş bilginin ne kadarının geleceğe aktarılacağını belirlerken; sıfırlama kapısı geçmiş bilginin ne kadarının unutulacağına karar verir. Bu çalışmada GRU, bir otokodlayıcı mimarisi içerisinde kullanılmıştır. Otokodlayıcı, yalnızca normal çalışma verilerini alarak düşük boyutlu bir gizli uzaya sıkıştırır ve ardından orijinal veriyi yeniden oluşturmaya çalışır. Sisteme yeni bir veri noktası (x_i) geldiğinde, modelin ürettiği çıktı ($\hat{x}_i$) ile orijinal veri arasındaki fark, Ortalama Kare Hata (Mean Squared Error - MSE) tabanlı bir yeniden oluşturma hatası olarak Denklem (2)'deki gibi hesaplanır:

$$RE = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 \qquad (2)$$

Model yalnızca normal davranışları öğrendiği için, sisteme anormal bir veri (örneğin donanım arızası kaynaklı ani yük artışı) girdiğinde yeniden oluşturma hatası belirgin şekilde artmaktadır. Hata değerinin, dinamik bir eşiği aşması durumunda ilgili veri noktası anomali olarak etiketlenmektedir.

3.2 Kapasite planlama için katkısal regresyon modellemesi

BT altyapılarında disk doluluk oranları veya sunucu kaynak tüketimleri, doğrusal bir artıştan ziyade mesai saatleri, hafta sonu yedeklemeleri veya yıl sonu kapanış işlemleri gibi güçlü periyodik döngüler barındırır. Bu nedenle, kapasite planlama problemlerinde geleneksel otoregresif (ARIMA) veya basit doğrusal regresyon modelleri, verilerdeki yapısal kırılmaları ve çoklu mevsimsellikleri yakalamada yetersiz kalmaktadır. Bu çalışmada, zaman serisi verilerindeki doğrusal olmayan eğilimleri ve periyodik değişimleri modellemek için esnek bir katkısal regresyon algoritması kullanılmıştır. Modelin matematiksel ifadesi Denklem (3)'te verilmiştir:

$$y(t) = g(t) + s(t) + h(t) + \mathcal{E}_t \qquad (3)$$

Burada $g(t)$ trend fonksiyonunu (genellikle parçalı lineer veya lojistik büyüme), $s(t)$ periyodik değişiklikleri (haftalık/yıllık mevsimsellik), $h(t)$ tatil etkilerini ve $\mathcal{E}_t$ hata terimini ifade eder (Taylor & Letham, 2018). Trend fonksiyonu $g(t)$, sistemin fiziksel kapasite sınırlarına yaklaşmasını modelleyebilmek için lojistik büyüme veya parçalı doğrusal büyüme şeklinde uyarlanabilmektedir. Periyodik değişiklik bileşeni $s(t)$ ise, haftalık ve yıllık mevsimsellikleri modellemek amacıyla Fourier serilerinden yararlanır. Sistem bakım günleri gibi tekrarlayan döngüler bu bileşen sayesinde başarıyla izole edilir. Hata terimi $\mathcal{E}_t$ ise normal dağılıma sahip olduğu varsayılan rastgele gürültüyü temsil eder. Bu katkısal yaklaşım, aşırı öğrenmeye karşı dirençli olup, operasyonel tahminlerde yüksek doğruluk ve yorumlanabilirlik sağlamaktadır.

4 Bulgular ve Tartışma

Bu bölümde, modellerin gerçek dünya verileri üzerindeki performansı, $F_{0,5}$-Skor, Kesinlik, Duyarlılık ve Yanlış Alarm metrikleri üzerinden değerlendirilmiştir.

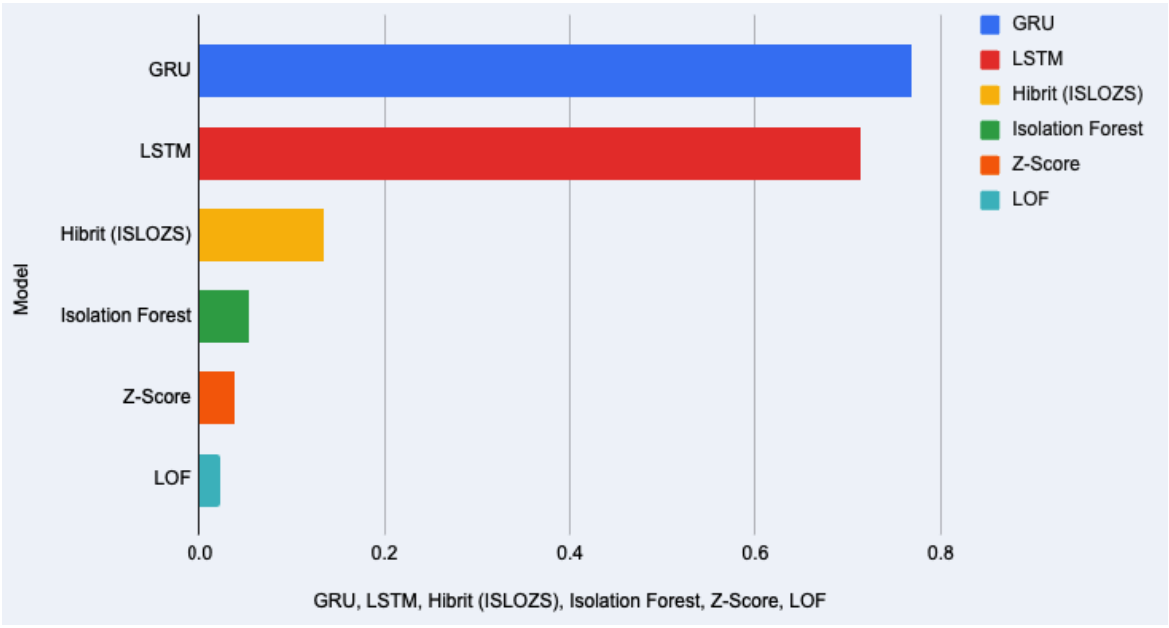
4.1 Anomali tespit modellerinin performans analizi

Modellerin başarısı, özellikle dengesiz veri setlerinde kritik olan ve yanlış alarmlara karşı duyarlılığı ölçen $F_{0,5}$ Skoru ile değerlendirilmiştir. $F_{0,5}$ Skoru, kesinlik değerine duyarlılık değerinden daha fazla ağırlık verir. Tablo 1'de modellerin performans karşılaştırması sunulmuştur.

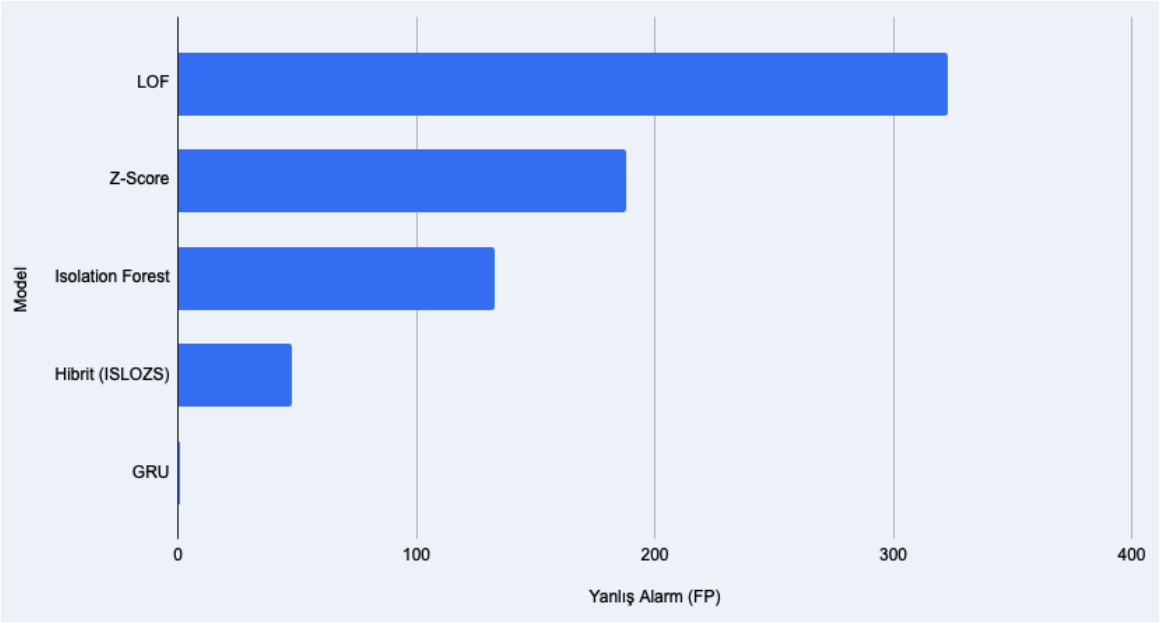
Tablo 1 ile modellerin performansını özetlemektedir:

Tablo 1: *Modellerin Performans Karşılaştırması*

Model	**$F_{0.5}$ Skor**	**Kesinlik (Precision)**	**Duyarlılık (Recall)**	**Yanlış Alarm (FP)**
GRU	**%76,92**	**%80,00**	%66,67	1
LSTM	**%71,43**	**%100,00**	**%33,33**	**0**
Hibrit (ISLOZS)	**%13,51**	**%11,11**	**%100,00**	**48**
Isolation Forest	**%5,34**	**%4,32**	**%100,00**	**133**
Z-Score	**%3,84**	**%3,09**	**%100,00**	**188**
LOF	**%2,27**	**%1,82**	**%100,00**	**323**



Şekil 2: Modellerin $F_{0.5}$-Skor Karşılaştırması



Şekil 3: Model Bazında Yanlış Alarm Yükü (AIOps Verimliliğinin Somutlaştırılması)

Tablo 1'de görüldüğü üzere klasik yöntemlerin (LOF, iForest, Z-Score) yüksek duyarlılığa sahip olmalarına rağmen Şekil 3'te göründüğü gibi ürettikleri yüzlerce yanlış alarm nedeniyle operasyonel kullanım için elverişsiz olduğunu göstermektedir. Buna karşılık Şekil 2'de görüldüğü gibi GRU, sadece 1 yanlış alarm üreterek %80 kesinlik oranına ulaşmış, güvenilir bir "danışman" rolü üstlenmiştir ve hız kazanmak ve daha az kaynak tüketimi için oluşturulan Hibrit modelin düşük kesinlik değerine (%11,11) bakıldığında yetersiz kaldığı görülmüştür. Bu durum, zaman serisi tabanlı izleme verilerindeki ardışık ilişkileri ve bağlamsal özellikleri kavramada otokodlayıcı temelli derin öğrenme yaklaşımlarının, verinin yalnızca dağılımına veya mekansal yoğunluğuna odaklanan istatistiksel yöntemlerden belirgin ölçüde üstün olduğunu kanıtlamaktadır. Derin öğrenme tabanlı bu başarım, literatürde büyük ölçekli bulut sistemlerindeki anomali tespiti çalışmalarıyla ve operasyonel verimlilik hedefleriyle de doğrudan örtüşmektedir.

Aynı tabloda LSTM modelinin GRU'ya kıyasla oldukça düşük bir duyarlılık oranı (%33,33) sergilediği görülmektedir. Veriye ve mimariye dayalı analizler, bu düşüklüğün temel nedeninin LSTM'in karmaşık üç kapılı (özellikle unutma kapısı) yapısından kaynaklandığını ortaya koymuştur. Yüksek frekanslı ve gürültülü sunucu telemetri verilerinde LSTM, eğitim setindeki normal volatiliteyi fazlasıyla ezberleyerek aşırı uyum (overfitting) göstermiştir. Model, küçük gürültüleri normal bir kalıp olarak öğrendiği için, test esnasındaki bazı sinsi anomalileri de "öğrenilmiş gürültü" olarak değerlendirip gözden kaçırmış ve bu durum kaçırılan hata oranını artırarak duyarlılığı düşürmüştür. Daha az parametreye sahip olan iki kapılı GRU ise bu veri setinde daha iyi bir genelleme yaparak gerçek sapmalara karşı hassasiyetini koruyabilmiştir.

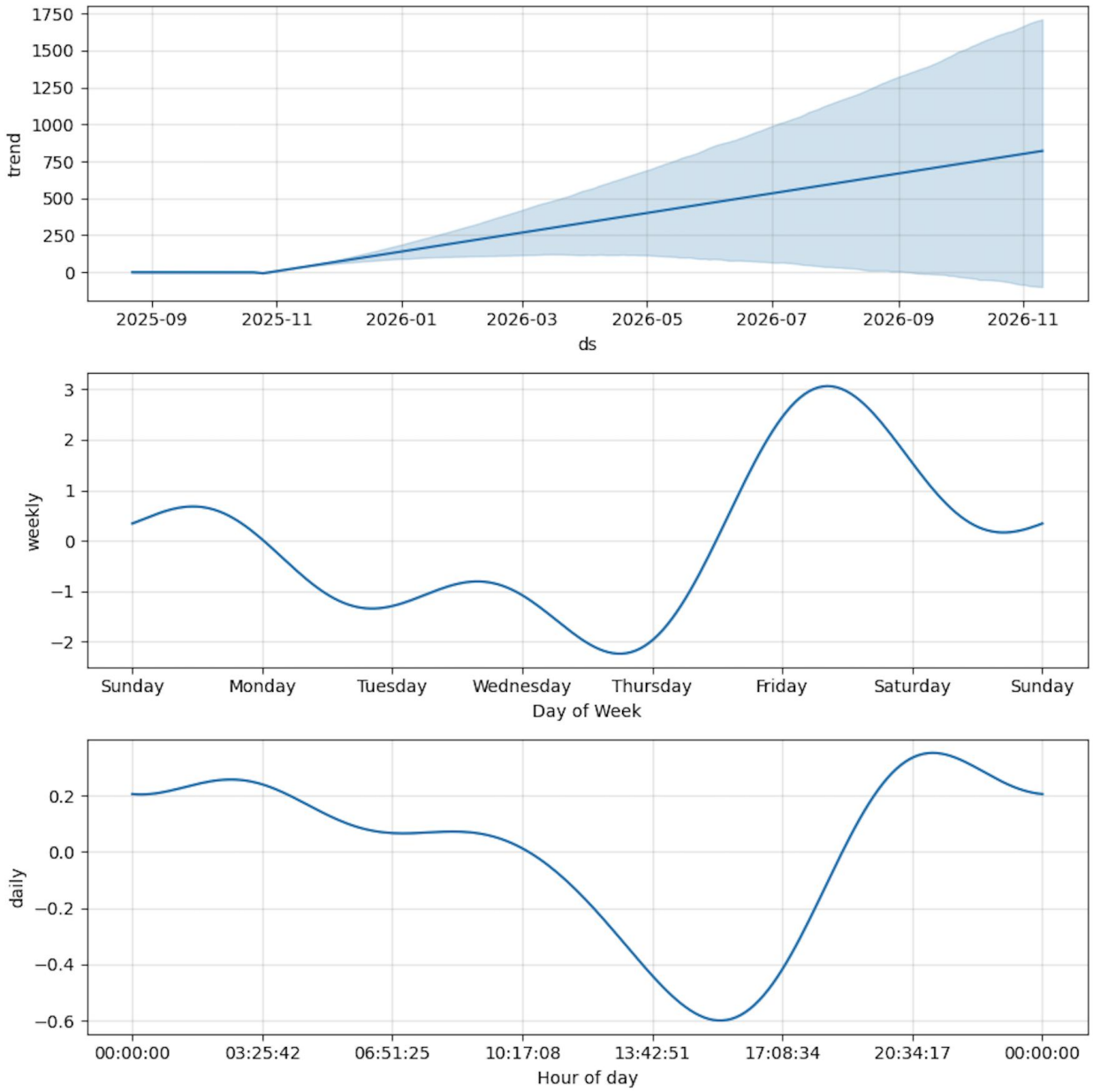
4.2 Kapasite tahminleme ve stratejik planlama bulguları

Katkısal regresyon algoritması ile gerçekleştirilen kapasite tahmin analizleri, sunucu disklerinin doluluk tarihlerini öngörmede istatistiksel ve operasyonel açıdan dikkate değer sonuçlar sunmuştur. Veriler incelendiğinde, bazı kritik sunucuların acil müdahale gerektiren risk sınırlarına yaklaştığı tespit edilmiştir. Örneğin, belirli bir veritabanı sunucusuna ait sürücünün 26 Kasım 2025 tarihinde, güvenlik loglarını tutan bir diğer sunucu sürücüsünün ise 1 Aralık 2025 tarihinde %100 doluluk oranına ulaşacağı

öngörülmüştür. Bu tarihler, ilgili sistemlerde proaktif depolama genişletmesi yapılmadığı takdirde operasyonel kesintilerin kaçınılmaz olacağını matematiksel olarak kanıtlamaktadır.

Bunun yanı sıra, analizlerin en belirgin çıktılarından biri küme bazlı senkronize tükeniş paterninin keşfedilmesidir. Merkezi log yönetimi kümesini oluşturan üç farklı sunucunun veri disklerinin 30 Ocak ile 16 Şubat 2026 tarihleri arasında dar bir zaman penceresinde sırayla dolacağı tespit edilmiştir. Bu bulgu, sorunun tekil bir sunucu üzerindeki geçici bir anormallikten ziyade, toplam mimari kapasitenin yetersizliğinden kaynaklandığını açıkça göstermektedir. Ayrıca, veritabanı ve uygulama sunucuları gibi kritik sistemlerin dolum tarihlerinin yılın ilk çeyreğine yığılması, kullanılan modelin yıl sonu kapanış işlemleri ve log rotasyonlarından kaynaklanan mevsimsel yük artışlarını başarıyla ayrıştırabildiğini kanıtladı. Lineer regresyon modellerinin gürültü veya kalıcı bir trend artışı sanarak yanılacağı bu mevsimsel dalgalanmalar, katkısal modelleme sayesinde izole edilerek kapasite yönetiminde yüksek hassasiyet sağlanmıştır.

Ayrıca Lineer Regresyon modelleri, disk kullanımlarındaki geçici artışları (örneğin haftalık yedeklemelerden kaynaklanan sıçramalar) yanlış yorumlar. Lineer model, geçici bir sıçramayı kalıcı bir trend artışı sanarak, diskin dolum tarihini olduğundan çok daha erken (örneğin 2 ay yerine 2 hafta sonra) tahmin etmekte ve gereksiz panik ve maliyet oluşturur. Buna karşılık Prophet; haftalık döngüleri, yıllık döngüleri ve tatilleri modelin içine dahil etmiştir. Prophet, veritabanı sunucusun disklerinde (%91,87 doluluk tahmini) haftalık yedekleme operasyonlarını mevsimsel olarak ayıklamış ve ana trendi doğru tespit etmiştir. Bu sayede, mevsimsel dalgalanmaların oluşturduğu gürültüden arındırılmış, son derece kararlı ve güvenilir kapasite doluluk tarihleri üretilmiştir. Prophet modelinin hata oranlarının, lineer modellere göre belirgin şekilde düşük olması, kapasite planlamada mevsimselliğin göz ardı edilemez bir faktör olduğunu kanıtlamaktadır.



Şekil 4: Prophet algoritması tarafından oluşturulan zaman serisi bileşenleri analizi. Üst panel: Genel büyüme trendi ve belirsizlik aralığı. Orta panel: Haftalık mevsimsellik döngüsü. Alt panel: Günlük mevsimsellik döngüsü

Şekil 4'te sunulan ayrıştırma grafikleri, Prophet modelinin ham veriyi Trend ve Mevsimsellik olarak nasıl katmanlarına ayırdığını detaylandırmaktadır.

Prophet ile yapılan doğru kapasite planlaması, kurumların BT bütçelerini optimize etmelerini sağlar. Kapasite dolum tarihlerinin hassas bir şekilde bilinmesi, donanım veya bulut kaynağı artırımlarının zamanında yapılmasını sağlar. Bu, gereksiz donanım yatırımını önleyerek hem yatırım maliyetlerini hem de bakım maliyetlerini düşürür.

5 Sonuçlar

Bu çalışmada, büyük ölçekli bilgi teknolojileri altyapılarının kesintisiz ve verimli bir şekilde yönetilebilmesi amacıyla, açık kaynaklı izleme araçları ile derin öğrenme ve zaman serisi tahmin algoritmalarını entegre eden hibrit bir sistem geliştirilmiş ve performansı değerlendirilmiştir. Gerçekleştirilen deneysel analizler, veriye dayalı karar destek mekanizmalarının geleneksel kural tabanlı sistemlere kıyasla belirgin üstünlükler sunduğunu ortaya koymuştur.

Anomali tespit sürecinde, zaman serisi verilerindeki ardışık ilişkileri modelleyebilen kapılı tekrarlayan birim tabanlı otokodlayıcı mimarisinin kullanılması, makine öğrenimi modellerinin kronik problemi olan yüksek yanlış alarm oranını neredeyse tamamen ortadan kaldırmıştır. Sistem yöneticileri için en büyük operasyonel engellerden biri olan alarm yorgunluğu bu sayede minimize edilmiş ve yüksek kesinlik oranına ulaşılmıştır.

Kapasite planlama ve kaynak yönetimi perspektifinden bakıldığında ise, katkısal regresyon tabanlı tahmin algoritmalarının, donanım kaynaklarının gelecekteki kullanım eğilimlerini öngörmede yüksek bir başarı sergilediği görülmüştür. Sistemlerin doğasında var olan günlük ve yıllık mevsimsel dalgalanmaların matematiksel olarak doğru modellenmesi, kaynak tükeniş tarihlerinin aylar öncesinden tespit edilebilmesine olanak tanımıştır. Bu proaktif yaklaşım, kurumlara kaynak artırımlarını tam zamanında yapma esnekliği sunarak olası hizmet kesintilerini engellemekte ve gereksiz donanım yatırımlarından kaynaklanan âtıl maliyetleri düşürmektedir.

Sonuç olarak, önerilen bu yenilikçi mimari, operasyonların otonomlaşması yolunda güçlü bir temel oluşturmaktadır. Gelecek çalışmalarda, tespit edilen anomalilerin kök nedenlerinin otonom olarak belirlenmesi, doğal dil işleme destekli asistanlar ile sistem loglarının birleştirilmesi ve kendi kendini iyileştiren ağ sistemlerinin geliştirilmesi hedeflenmektedir.

6 Beyanname

6.1 Çalışmanın Sınırları

Bu çalışma, tek bir kurumun verileriyle sınırlıdır.

6.2 Rakip Çıkarlar

Bu çalışmada herhangi bir çıkar çatışması yoktur.

6.3 Yazarların Katkıları

Sorumlu Yazar Ali YAZICI: Araştırma ve makale için fikir ya da hipotezin oluşturulması, sonuçlara ulaşmak için gereç ve yöntemlerin planlanması, deneylerin yapılması, verilerin düzenlenmesi ve bildirilmesi için sorumluluk almak, bulguların mantıklı açıklanması ve sunumu için sorumluluk almak, araştırma sırasında literatür taraması ile ilgili sorumluluk almak, yazının tümü veya asıl bölümün oluşturulması için sorumluluk almak, makaleyi teslim etmeden önce sadece imla ve dil bilgisi açısından değil aynı zamanda entelektüel içerik açısından yeniden çalışma yapmak.

2. Yazar Prof. Dr. Ahmet ZENGİN: Araştırma fikri ve hipotezinin akademik derinlik kazanmasına katkıda bulunmuş; sonuçlara ulaşmak için kullanılan gereç ve yöntemlerin bilimsel geçerliliğini denetlemiş; bulguların analizi, yorumlanması ve makalenin entelektüel içeriğinin uluslararası yayın standartlarına uygun şekilde yapılandırılması süreçlerinde rehberlik ve sorumluluk almıştır.

Kaynakça

Audibert, J., Michau, P., Campardelli, A., & Boutin, M. (2020). USAD: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 3395-3404).

Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). LOF: identifying density-based local outliers. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 93–104.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Dang, Y., Lin, Q., & Huang, P. (2021). AIOps: Real-world challenges and research innovations. *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 4–5.

Freedman, M. J. (2017). TimescaleDB: SQL made scalable for time-series data. *Timescale Technical Report*. https://www.timescale.com/papers/timescaledb.pdf

Grubbs, F. E. (1969). Procedures for detecting outlying observations in samples. *Technometrics*, 11(1), 1–21.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.

Islam, M. S., Rakha, M. S., Pourmajidi, W., Sivaloganathan, J., Steinbacher, J., & Miranskyy, A. (2024). Anomaly detection in large-scale cloud systems: An industry case and dataset. *arXiv preprint arXiv:2411.09047*.

Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. *2008 Eighth IEEE International Conference on Data Mining*, 413–422.

Manassra, A., & Işık, G. (2025). A model for artificial intelligence supported energy management in smart campuses. *Journal of Smart Systems Research*, 6(2), 74-92. https://doi.org/10.58769/joinssr.1677699

Mendoza, M.-A., & Amistadi, H. R. (2018). *Machine learning for anomaly detection on VM and host performance metrics*. The MITRE Corporation.

Perez, A., & Luckner, M. (2025). A comparison of unsupervised machine learning techniques for anomaly detection in flight data monitoring. *Aerospace*, 12(2), 151. https://doi.org/10.3390/aerospace12020151

Şahin, O., & Karayel, D. (2024). Generative artificial intelligence (GenAI) in business: A systematic review on the threshold of transformation. *Journal of Smart Systems Research*, 5(2), 156-175. https://doi.org/10.58769/joinssr.1597110

Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician*, 72(1), 37–45.

Zabbix. (t.y.). *Zabbix documentation*. https://www.zabbix.com/documentation/current/en/manual

Ekler

Ek A. Model Eğitimi İçin Kullanılan Python Kod Blokları

```
def train_model(metric_keys, days, engine, contamination=0.01):
    """Trains a specified anomaly detection model and measures performance."""

    monitor = ResourceMonitor()
    monitor.start()
    start_time = time.time()

    try:
        df = get_history_data(metric_keys, days, from_start=True)
        if df.empty: return None, {}

        df['timestamp_hr'] = pd.to_datetime(df['clock'], unit='s')

        models = None
        if engine in ['iso', 'lof']:
            models = _train_classic_model(df, metric_keys, engine, contamination)
        elif engine == 'zss':
            models = _train_zss_model(df, metric_keys)
        elif engine in ['aen', 'lstm', 'gru']:
            models = _train_deep_model(df, metric_keys, engine)
        elif engine in ['isozs', 'lofzs', 'islozs']:
            if engine == 'islozs':
                _train_classic_model(df, metric_keys, 'iso', contamination)
                _train_classic_model(df, metric_keys, 'lof', contamination)
            else:
                _train_classic_model(df, metric_keys, engine.replace('zs', ''),
contamination)
            _train_zss_model(df, metric_keys)
            models = "Hybrid model training complete."
        else:
            raise ValueError(f"Unknown engine: {engine}")
    finally:
        monitor.stop()

    training_time = time.time() - start_time
    max_ram, avg_cpu, max_gpu_mem, avg_gpu_util = monitor.get_metrics()

    metrics = {
        "Training Time": f"{training_time:.2f} seconds",
        "Max RAM Usage": f"{max_ram / 1024**2:.2f} MB",
        "Average CPU": f"{avg_cpu:.1f}%"
    }
    if max_gpu_mem > 0:
        metrics["Max GPU Memory"] = f"{max_gpu_mem} MB"
    if avg_gpu_util > 0:
        metrics["Average GPU Usage"] = f"{avg_gpu_util:.1f}%"
```

```
    return models, metrics

def _train_classic_model(df, metric_keys, engine, contamination):
    models = {}
    features = ['value', 'value_diff_from_mean']

    hostnames = df['hostname'].unique()
    total_hosts = len(hostnames)
    for i, hostname in enumerate(hostnames, 1):
        t_start = time.time()
        host_df = df[df['hostname'] == hostname].copy()
        start_date = host_df['timestamp_hr'].min().strftime('%Y-%m-%d %H:%M')
        print(f"[{i}/{total_hosts}] Training {engine} for {hostname} (Start:
{start_date})...", end=' ', flush=True)
        if host_df.shape[0] < 2:
            print("Skipped (not enough data)")
            continue

        host_df = engineer_features(host_df)

        model = None
        if engine == 'iso':
            model = IsolationForest(contamination=contamination, random_state=42)
        elif engine == 'lof':
            n_neighbors = min(50, len(host_df) - 1)
            if n_neighbors > 0:
                model = LocalOutlierFactor(n_neighbors=n_neighbors, novelty=True,
contamination=contamination)

        if model is not None:
            model.fit(host_df[features].values)
            _save_model(model, hostname, metric_keys[0], engine)
            models[hostname] = model
            print(f"Done ({time.time() - t_start:.2f}s)")
        else:
            print(f"Skipped ({time.time() - t_start:.2f}s)")

    return models if models else None

def _train_zss_model(df, metric_keys):
    stats = {}
    hostnames = df['hostname'].unique()
    total_hosts = len(hostnames)
    for i, hostname in enumerate(hostnames, 1):
        t_start = time.time()
        host_df = df[df['hostname'] == hostname].copy()
        start_date = host_df['timestamp_hr'].min().strftime('%Y-%m-%d %H:%M')
        print(f"[{i}/{total_hosts}] Training zss for {hostname} (Start:
{start_date})...", end=' ', flush=True)
        mean_val = host_df['value'].mean()
        std_val = host_df['value'].std()
```

```
            stat = {'mean': mean_val, 'std': std_val}
            _save_model(stat, hostname, metric_keys[0], 'zss')
            stats[hostname] = stat
            print(f"Done ({time.time() - t_start:.2f}s)")

    return stats

def _train_deep_model(df, metric_keys, engine):
    all_models = {}
    hostnames = df['hostname'].unique()
    total_hosts = len(hostnames)
    for i, hostname in enumerate(hostnames, 1):
        t_start = time.time()
        host_df = df[df['hostname'] == hostname].copy().sort_values('timestamp_hr')
        start_date = host_df['timestamp_hr'].min().strftime('%Y-%m-%d %H:%M')
        print(f"[{i}/{total_hosts}] Training {engine} for {hostname} (Start:
{start_date})...", end=' ', flush=True)
        if len(host_df) < SEQUENCE_LEN * 2:
            print("Skipped (not enough data)")
            continue

        scaler = MinMaxScaler()
        host_df['value_scaled'] = scaler.fit_transform(host_df[['value']])

        sequences = create_sequences(host_df['value_scaled'].values, SEQUENCE_LEN)
        if len(sequences) == 0:
            print("Skipped (no sequences)")
            continue
        sequences = sequences.reshape(-1, SEQUENCE_LEN, 1)

        model = _create_deep_model(engine, SEQUENCE_LEN, 1)
        model.fit(sequences, sequences, epochs=1, batch_size=32, verbose=0)

        train_mae_loss = np.mean(np.abs(model.predict(sequences, verbose=0) -
sequences), axis=1)
        threshold = np.max(train_mae_loss) * THRESHOLD_MULTIPLIER

        model_data = {'model': model, 'scaler': scaler, 'threshold': threshold}
        _save_model(model_data, hostname, metric_keys[0], engine)
        all_models[hostname] = model_data
        print(f"Done ({time.time() - t_start:.2f}s)")

    return all_models if all_models else None
```