ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Ανάπτυξη Συστήματος για ανίχνευση ανωμαλιών σε log δεδομένα σε πραγματικό χρόνο»

Τη φοιτήτρια
Λέκα Έντζι
Αρ. Μητρώου: 2020077

Επιβλέπων
Μπράτσας Χαράλαμπος
Βαθμίδα ……

Ημερομηνία 23/06/2026

Τίτλος Δ.Ε. Ανάπτυξη Συστήματος για ανίχνευση ανωμαλιών σε log δεδομένα σε πραγματικό χρόνο
Κωδικός Δ.Ε. 26103
Ονοματεπώνυμο φοιτητή Λέκα Έντζι
Ονοματεπώνυμο εισηγητή Μπράτσας Χαράλαμπος
Ημερομηνία ανάληψης Δ.Ε. 01/01/2026
Ημερομηνία περάτωσης Δ.Ε. 23/06/2026

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

<Αφιερώνω την παρούσα εργασία στην οικογένειά μου, για την αμέριστη υποστήριξη, την υπομονή και την εμπιστοσύνη που μου προσέφερε καθ' όλη τη διάρκεια των σπουδών μου.>

Πρόλογος

Η επιλογή της παρούσας διπλωματικής εργασίας πραγματοποιήθηκε λόγω του ιδιαίτερου ενδιαφέροντος που παρουσιάζει ο τομέας της ανάλυσης δεδομένων και ειδικότερα της ανίχνευσης ανωμαλιών σε αρχεία καταγραφής συστημάτων. Τα logs αποτελούν σημαντική πηγή πληροφορίας για τη λειτουργία και την παρακολούθηση σύγχρονων πληροφοριακών συστημάτων, ενώ η αποτελεσματική αξιοποίησή τους μπορεί να συμβάλλει σημαντικά στην έγκαιρη αναγνώριση προβλημάτων και δυσλειτουργιών.

Κατά τη διάρκεια της υλοποίησης της εργασίας είχα την ευκαιρία να εμβαθύνω σε θέματα επεξεργασίας δεδομένων, ανίχνευσης ανωμαλιών και σχεδίασης ολοκληρωμένων συστημάτων ανάλυσης. Παράλληλα, απέκτησα πρακτική εμπειρία στη χρήση εργαλείων και τεχνολογιών για την ανάπτυξη ενός ολοκληρωμένου pipeline, από τη συλλογή και προεπεξεργασία των δεδομένων μέχρι την αξιολόγηση και οπτικοποίηση των αποτελεσμάτων.

Η ενασχόληση με το συγκεκριμένο θέμα συνέβαλε ουσιαστικά τόσο στην ενίσχυση των τεχνικών γνώσεων όσο και στην ανάπτυξη δεξιοτήτων σχετικών με την ανάλυση, την επίλυση προβλημάτων και την υλοποίηση πραγματικών εφαρμογών.

Περίληψη

Η παρούσα εργασία πραγματεύεται τον σχεδιασμό, την υλοποίηση και την αξιολόγηση ενός συστήματος ανίχνευσης ανωμαλιών σε αρχεία καταγραφής, με έμφαση στην οπτική του Software Engineering και στη δημιουργία ενός λειτουργικού pipeline πραγματικού χρόνου. Τα logs αποτελούν βασική πηγή πληροφορίας για την παρακολούθηση, τη διάγνωση και την ασφάλεια πληροφοριακών συστημάτων, καθώς καταγράφουν γεγονότα που σχετίζονται με τη λειτουργία υπηρεσιών, τη συμπεριφορά χρηστών και πιθανές αποκλίσεις από την κανονική λειτουργία. Στο πλαίσιο αυτό, αναπτύχθηκε ένα ολοκληρωμένο σύστημα που περιλαμβάνει παραγωγή και εισαγωγή logs, μηχανισμό ροής δεδομένων, προεπεξεργασία και κανονικοποίηση, ενσωμάτωση αλγορίθμων ανίχνευσης ανωμαλιών, αποθήκευση αποτελεσμάτων σε βάση δεδομένων και οπτικοποίηση μέσω dashboard.

Για την πειραματική αξιολόγηση χρησιμοποιήθηκε dataset από το LogHub, εμπλουτισμένο με ελεγχόμενα σενάρια ανωμαλιών, ώστε να υπάρχει διαθέσιμο ground truth. Η υλοποίηση βασίστηκε σε Python, SQLite και Streamlit, με στόχο την αναπαραγωγιμότητα και τη χαμηλή υποδομική πολυπλοκότητα. Εξετάστηκαν δύο διαφορετικοί ανιχνευτές, ένας threshold based detector και ένας detector σπανιότητας component, και αξιολογήθηκαν με μετρικές Precision, Recall και F1, καθώς και με χρόνους inference και throughput. Τα αποτελέσματα έδειξαν ότι ο threshold based detector ήταν καταλληλότερος για το συγκεκριμένο πειραματικό περιβάλλον, επιτυγχάνοντας υψηλότερη συνολική αποτελεσματικότητα, ενώ ο δεύτερος αλγόριθμος παρουσίασε αδυναμία προσαρμογής στον τύπο των ανωμαλιών που χρησιμοποιήθηκαν. Συμπερασματικά, η εργασία ανέδειξε ότι η αποτελεσματικότητα ενός συστήματος ανίχνευσης δεν εξαρτάται μόνο από τον αλγόριθμο, αλλά και από τη δομή του pipeline, την ποιότητα των χαρακτηριστικών και τη συμβατότητα της μεθόδου με τη φύση των δεδομένων.

Λέξεις κλειδιά : Ανίχνευση ανωμαλιών, logs, πραγματικός χρόνος, pipeline, SQLite, dashboard

«Anomaly detection σε logs»

«Engie Leka»

Abstract

This thesis examines the design, implementation and evaluation of an anomaly detection system for log data, with emphasis on the Software Engineering perspective and on the development of a functional real time processing pipeline. Logs constitute a critical source of information for system monitoring, diagnosis and security, since they record events related to service operation, user activity and potential deviations from normal behavior. Within this context, a complete system was developed, including log generation and ingestion, data streaming, preprocessing and normalization, integration of anomaly detection algorithms, storage of results in a database and visualization through a dashboard.

For the experimental evaluation, a dataset from LogHub was used and enriched with controlled anomaly scenarios in order to provide ground truth labels. The implementation was based on Python, SQLite and Streamlit, aiming at reproducibility and low infrastructure complexity. Two different detectors were examined, a threshold based detector and a rare component detector, and were evaluated using Precision, Recall and F1 metrics, as well as inference time and throughput measurements. The results demonstrated that the threshold based detector was more suitable for the specific experimental setting, achieving higher overall effectiveness, whereas the second algorithm failed to adapt adequately to the type of anomalies injected into the dataset. Overall, the study showed that the effectiveness of an anomaly detection system depends not only on the selected algorithm, but also on the structure of the pipeline, the quality of extracted features and the degree of compatibility between the detection method and the nature of the log data.

Keywords

Anomaly detection, logs, real time, pipeline, SQLite, dashboard

Περιεχόμενα

Πρόλογος **5**
Περίληψη **6**
Abstract **7**
Περιεχόμενα **8**
ΚΕΦΑΛΑΙΟ 1 10
Εισαγωγή 10
1.1 Πλαίσιο και σημασία της ανίχνευσης ανωμαλιών σε αρχεία καταγραφής 10
1.2 Σκοπός και αντικείμενο της διπλωματικής εργασίας 12
1.3 Στόχοι και ερευνητικά ερωτήματα 14
1.4 Μεθοδολογία, δεδομένα και κριτήρια αξιολόγησης 15
1.5 Δομή και οργάνωση της εργασίας 16
ΚΕΦΑΛΑΙΟ 2: Θεωρητικό πλαίσιο και βιβλιογραφική ανασκόπηση 17
2.1 Αρχεία καταγραφής, παρατηρησιμότητα και λειτουργική αξία 18
2.2 Μορφότυπα, τυποποίηση και μεταφορά μηνυμάτων 19
2.3 Διαχείριση κύκλου ζωής logs και απαιτήσεις ασφάλειας 20
2.4 Προεπεξεργασία και log parsing ως κρίσιμος μετασχηματισμός 20
2.5 Αναπαράσταση logs και εξαγωγή χαρακτηριστικών 22
2.6 Αλγόριθμοι ανίχνευσης ανωμαλιών σε logs και τυπικές κατηγορίες 23
2.7 Δημόσια σύνολα δεδομένων και ζητήματα ετικετοποίησης 24
2.8 Αρχιτεκτονικές επεξεργασίας σε πραγματικό χρόνο και παρακολούθηση 25
2.9 Αξιολόγηση απόδοσης, μετρικές και χρόνος απόφασης 27
2.10 Σύνοψη κεφαλαίου 29
Κεφάλαιο 3 Ανάλυση απαιτήσεων και σχεδίαση συστήματος 30
3.1 Functional requirements (log gen, ingest, preprocess, detect, store, visualize) 30
3.2 Non-functional requirements (real-time, scalability, reliability, latency στόχοι) 32
3.3 Επιλογή τεχνολογιών & αιτιολόγηση (π.χ. Kafka/Elastic/DB, dashboard tool) 33
3.4 Αρχιτεκτονική συστήματος (διάγραμμα modules + ροές δεδομένων) 34
3.5 Σχεδίαση βάσης (schema: raw log, parsed template, features, score, label, timestamps) 35
3.6 Σενάρια χρήσης (π.χ. “security analyst βλέπει spike failed logins”) 36
Κεφάλαιο 4 Υλοποίηση (Implementation) 37
4.1 Log generator (μορφή logs, normal vs anomaly σενάρια) 38
4.2 Real-time ingestion: τρόπος ροής των αρχείων καταγραφής 40
4.3 Preprocessing (parsing + partitioning + feature extraction) 41
4.4 Ενσωμάτωση αλγορίθμων anomaly detection (plug-in αρχιτεκτονική) 42
4.5 Αποθήκευση και indexing 43
4.6 Dashboard (screens, filters, alerts, drill-down) 45
4.7 Θέματα deployment (Docker, configs, reproducibility) 47
Κεφάλαιο 5 Πειράματα και αποτελέσματα 49
5.1 Πειραματική διαδικασία 49
5.2 Αποτελέσματα ανά αλγόριθμο 51

5.3 Χρόνοι εκτέλεσης και ρυθμός επεξεργασίας 52
5.4 Σύγκριση και trade-offs 54
5.4.1 Επεκταμένη ερμηνεία αποτελεσμάτων και επιχειρησιακές επιπτώσεις 54
5.5 Συζήτηση σφαλμάτων (false positives/false negatives, γιατί συμβαίνουν) 57
Κεφάλαιο 6 Συμπεράσματα και μελλοντική εργασία 58
6.1 Τι πέτυχε το σύστημα 58
6.2 Τι έδειξε η σύγκριση και ποιος αλγόριθμος βολεύει σε real time 60
6.3 Περιορισμοί 61
6.4 Μελλοντική εργασία 62
ΒΙΒΛΙΟΓΡΑΦΙΑ **64**
ΠΑΡΑΡΤΗΜΑΤΑ **67**

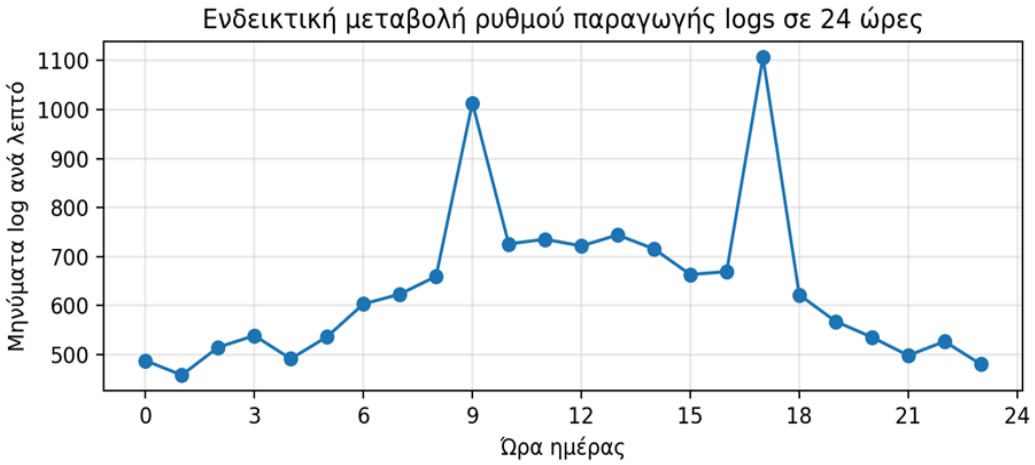
ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

1.1 Πλαίσιο και σημασία της ανίχνευσης ανωμαλιών σε αρχεία καταγραφής

Η λειτουργία των σύγχρονων πληροφοριακών συστημάτων αφήνει πίσω της ένα συνεχές αποτύπωμα σε μορφή αρχείων καταγραφής. Κάθε αίτημα, κάθε αποτυχία, κάθε έλεγχος πρόσβασης και κάθε εσωτερική μετάβαση κατάστασης παράγει μηνύματα που έχουν στόχο να υποστηρίξουν την παρακολούθηση, τη διάγνωση και τη διασφάλιση αξιοπιστίας. Σε περιβάλλοντα μεγάλης κλίμακας, όπου υπηρεσίες εκτελούνται κατανεμημένα και αλλάζουν συχνά, τα αρχεία καταγραφής αποτελούν πρακτικά τη μοναδική συνεχή πηγή επιχειρησιακής πληροφορίας με υψηλή λεπτομέρεια. Η αξία τους έχει τεκμηριωθεί ήδη από τις πρώιμες εργασίες εξόρυξης καταγραφών, οι οποίες έδειξαν ότι η συστηματική ανάλυση τους μπορεί να οδηγήσει σε εντοπισμό προβλημάτων και δυσλειτουργιών που είναι δύσκολο να αποκαλυφθούν με άλλους μηχανισμούς παρακολούθησης (Xu και συνεργάτες 2009).

Παρά τη χρησιμότητα τους, τα αρχεία καταγραφής δεν είναι εύκολα αξιοποιήσιμα χωρίς αυτοματοποίηση. Ο όγκος τους αυξάνεται με την κλιμάκωση των υπηρεσιών, ενώ η ετερογένεια μορφών και ημιδομημένων μηνυμάτων δυσχεραίνει τη χειροκίνητη ανάλυση. Η εμπειρική πρακτική της επιτόπιας διερεύνησης με απλές εντολές προβολής καταγραφών επαρκεί για μεμονωμένα συστήματα, όμως αποτυγχάνει όταν υπάρχουν πολλαπλές πηγές, διαφορετικά επίπεδα καταγραφής, συχνές αναβαθμίσεις και συνεχής παραγωγή συμβάντων. Η ανάγκη για αυτοματοποιημένη ανίχνευση ασυνήθιστων μοτίβων μέσα στο ρεύμα καταγραφών οδηγεί στο πεδίο της ανίχνευσης ανωμαλιών σε αρχεία καταγραφής, ένα πεδίο που συνδέεται άμεσα με τη λειτουργική ανθεκτικότητα, την ασφάλεια και τη διαχείριση συμβάντων σε πραγματικό χρόνο(Landauer και συνεργάτες 2023)



Σχήμα 1.1 Ενδεικτική μεταβολή ρυθμού παραγωγής αρχείων καταγραφής σε διάρκεια ημέρας

Η ανίχνευση ανωμαλιών στα αρχεία καταγραφής αναφέρεται στην αναγνώριση μηνυμάτων ή ακολουθιών μηνυμάτων που αποκλίνουν από την αναμενόμενη συμπεριφορά ενός συστήματος. Η απόκλιση αυτή μπορεί να εκφράζει επιχειρησιακό σφάλμα, λανθασμένη παραμετροποίηση, υποβάθμιση απόδοσης, ή και ύποπτη ενέργεια, όπως επαναλαμβανόμενες αποτυχημένες προσπάθειες σύνδεσης και μη αναμενόμενες κλήσεις διεπαφών. Η βιβλιογραφία δείχνει ότι η έννοια του φυσιολογικού στα αρχεία καταγραφής συχνά δεν είναι σταθερή, διότι αλλάζει με την έκδοση λογισμικού, το φορτίο, την εποχικότητα και τις παρεμβάσεις συντήρησης. Γι αυτό, η προσέγγιση που προκρίνεται σε σύγχρονες εφαρμογές είναι η λειτουργική ενσωμάτωση μηχανισμών ανίχνευσης μέσα σε ένα ολοκληρωμένο αγωγό επεξεργασίας, ώστε η απόφαση να παράγεται αξιόπιστα και έγκαιρα (Landauer και συνεργάτες 2022)

Οι ανωμαλίες που αποτυπώνονται στα αρχεία καταγραφής δεν έχουν ενιαία μορφή. Στην πράξη, οι σημαντικότερες κατηγορίες σχετίζονται είτε με σπάνια μηνύματα, είτε με μεταβολές συχνότητας, είτε με μη αναμενόμενες ακολουθίες γεγονότων. Ο Πίνακας 1.1 συνοψίζει ενδεικτικές κατηγορίες και το επιχειρησιακό τους αποτύπωμα.

Κατηγορία ανωμαλίας	Περιγραφή	Ενδεικτικό παράδειγμα	Πιθανή επίπτωση
Σπάνιο γεγονός	Εμφάνιση μηνύματος ή προτύπου που δεν παρατηρείται στη φυσιολογική λειτουργία.	Αναφορά άγνωστου σφάλματος πυρήνα υπηρεσίας ή απρόβλεπτη εξαίρεση.	Αστοχία λογισμικού ή νέος τύπος προβλήματος.
Αύξηση συχνότητας	Απότομη αύξηση του πλήθους συγκεκριμένων μηνυμάτων σε μικρό χρονικό παράθυρο.	Πολλαπλές αποτυχημένες συνδέσεις σε σύντομο χρόνο.	Επίθεση ή υποβάθμιση υπηρεσίας λόγω καταχρηστικού φορτίου.
Ασυνήθιστη ακολουθία	Παραβίαση της αναμενόμενης σειράς γεγονότων σε μια ροή ή συνεδρία.	Κλήση λειτουργίας ολοκλήρωσης χωρίς επιτυχή έλεγχο ταυτότητας.	Παραβίαση ροής, σφάλμα λογικής ή κατάχρηση.
Ασυνέπεια παραμέτρων	Μηνύματα με τιμές πεδίων εκτός αναμενόμενων ορίων ή με μη συμβατούς συνδυασμούς.	Χρόνος απόκρισης πολύ υψηλός ή κωδικοί κατάστασης που δεν ταιριάζουν με το αίτημα.	Υποβάθμιση απόδοσης και ανάγκη διερεύνησης ρίζας αιτίας.

Πίνακας 1.1 Ενδεικτικές κατηγορίες ανωμαλιών σε αρχεία καταγραφής και επιχειρησιακή σημασία

1.2 Σκοπός και αντικείμενο της διπλωματικής εργασίας

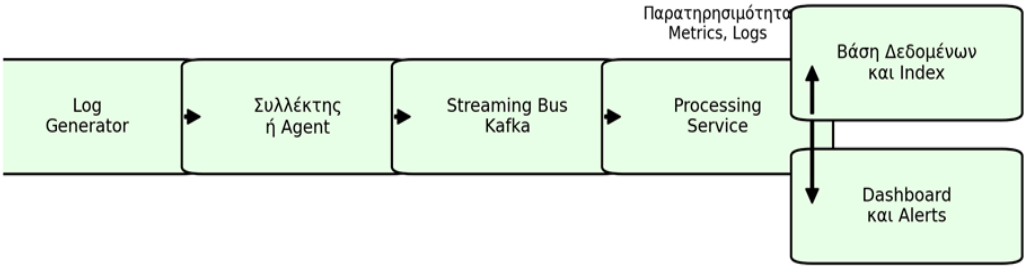
Η παρούσα διπλωματική εργασία εξετάζει την ανίχνευση ανωμαλιών σε αρχεία καταγραφής από τη σκοπιά του μηχανικού λογισμικού και όχι ως καθαρά αλγοριθμικό πρόβλημα. Κεντρικός στόχος είναι η σχεδίαση και υλοποίηση ενός ολοκληρωμένου αγωγού επεξεργασίας σε πραγματικό χρόνο, ο οποίος παράγει ή συλλέγει καταγραφές, τις προεπεξεργάζεται, εφαρμόζει μηχανισμούς ανίχνευσης ανωμαλιών, αποθηκεύει το αποτέλεσμα μαζί με μεταδεδομένα και παρουσιάζει τη ροή σε κατάλληλο πίνακα ελέγχου.

Η προσέγγιση αυτή αντανακλά την πραγματική λειτουργία επιχειρησιακών συστημάτων, όπου το κρίσιμο σημείο δεν είναι μόνο η ακρίβεια ταξινόμησης, αλλά και η συνέπεια της ροής δεδομένων, η ανθεκτικότητα του συστήματος, η δυνατότητα κλιμάκωσης και η χρηστικότητα της απεικόνισης (Sridharan 2018).

Σχήμα 1.2 Γενική ροή αγωγού επεξεργασίας για ανίχνευση ανωμαλιών σε αρχεία καταγραφής

Στο πλαίσιο αυτό, η διπλωματική οργανώνεται γύρω από πέντε βασικές λειτουργικές ενότητες. Αρχικά, τη δημιουργία ή συλλογή αρχείων καταγραφής, είτε από πραγματικές πηγές είτε μέσω προσομοιωτή που επιτρέπει ελεγχόμενα σενάρια. Στη συνέχεια, τη μεταφορά τους ως ροή γεγονότων με μηχανισμό δημοσίευσης και συνδρομής, ώστε να αποφεύγεται η σύζευξη μεταξύ παραγωγού και καταναλωτή. Η επιλογή αυτής της αρχιτεκτονικής είναι εναρμονισμένη με τα χαρακτηριστικά ενός κατανεμημένου συστήματος καταγραφής γεγονότων, όπως περιγράφεται στην τεκμηρίωση του Apache Kafka. Apache Software Foundation 2025, Confluent 2025. Επίσης, την προεπεξεργασία, όπου η ημιδομημένη φύση των μηνυμάτων μετατρέπεται σε πιο σταθερές αναπαραστάσεις, συνήθως μέσω εξαγωγής προτύπων και παραμέτρων. Η βιβλιογραφία αναγνωρίζει την κρισιμότητα του σταδίου αυτού, με κλασικό παράδειγμα τον αλγόριθμο Drain, ο οποίος σχεδιάστηκε για αποδοτική λειτουργία σε ροή (He και συνεργάτες 2017) . Κατόπιν, την εφαρμογή αλγορίθμων ανίχνευσης, οι οποίοι μπορούν να βασίζονται σε μια συνεχόμενη μοντελοποίηση ή σε αναπαράσταση προτύπων.

Ενδεικτικά, το DeepLog διατυπώνει την ανίχνευση ως πρόβλημα πρόβλεψης επόμενου γεγονότος σε ακολουθία, ενώ το LogAnomaly εισάγει μηχανισμό ενσωμάτωσης προτύπων για να αξιοποιήσει σημασιολογικές ομοιότητες (Du 2017, Meng 2019. Τέλος, την αποθήκευση, ευρετηρίαση και οπτικοποίηση, ώστε τα αποτελέσματα να είναι επιχειρησιακά αξιοποιήσιμα (Elastic 2024).



Σχήμα 1.3 Αρχιτεκτονική υψηλού επιπέδου της προτεινόμενης λύσης σε πραγματικό χρόνο

1.3 Στόχοι και ερευνητικά ερωτήματα

Με βάση το παραπάνω πλαίσιο, η εργασία διατυπώνει ένα σαφές σύνολο ερευνητικών στόχων που είναι εφαρμόσιμοι σε διπλωματικής επίπεδο και μετρήσιμοι στην πράξη. Ο πρώτος στόχος είναι η τεχνική αποτύπωση ενός αγωγού επεξεργασίας που λειτουργεί συνεχώς και μπορεί να δεχθεί πολλαπλές πηγές καταγραφών χωρίς να απαιτείται στενή σύζευξη μεταξύ υπηρεσιών. Ο δεύτερος στόχος είναι η ορθή μετατροπή των ακατέργαστων μηνυμάτων σε αναπαραστάσεις κατάλληλες για υπολογιστική ανάλυση, αναγνωρίζοντας ότι το στάδιο της ανάλυσης προτύπων επηρεάζει άμεσα την απόδοση της ανίχνευσης (Jiang 2024). Ο τρίτος στόχος είναι η ενσωμάτωση και σύγκριση αντιπροσωπευτικών αλγορίθμων ανίχνευσης ανωμαλιών από τη διεθνή βιβλιογραφία, με έμφαση στη λειτουργία τους ως έτοιμων δομικών στοιχείων μέσα στο σύστημα. Ο τέταρτος στόχος είναι η αποτίμηση της λύσης όχι μόνο με κλασικές μετρικές ταξινόμησης, αλλά και με μετρικές λειτουργικής απόδοσης όπως ο χρόνος εξαγωγής απόφασης, διότι ένα λειτουργικό σύστημα ενδιαφέρεται πρωτίστως για την έγκαιρη ειδοποίηση και τη δυνατότητα διερεύνησης (Apache Flink 2025).

Τα αντίστοιχα ερευνητικά ερωτήματα διατυπώνονται σε επίπεδο μηχανικής λογισμικού. Ποια αρχιτεκτονική ροής γεγονότων εξυπηρετεί καλύτερα τη συλλογή και διανομή αρχείων καταγραφής σε πραγματικό χρόνο, με έμφαση σε ανθεκτικότητα, επεκτασιμότητα και δυνατότητα αναπαραγωγής της ροής για επανέλεγχο.

Με ποιον τρόπο η προεπεξεργασία, και ειδικά η εξαγωγή προτύπων, επηρεάζει τη συμπεριφορά των αλγορίθμων ανίχνευσης. Ποια είναι η σχέση μεταξύ ακρίβειας ανίχνευσης και χρόνου εξαγωγής απόφασης, όταν οι αλγόριθμοι τοποθετούνται μέσα σε αγωγό που εκτελείται συνεχώς. Τέλος, ποια χαρακτηριστικά απεικόνισης και αποθήκευσης καθιστούν τα

αποτελέσματα χρήσιμα για πρακτική διερεύνηση από τον χειριστή(Elastic 2024, Sridharan 2018).

1.4 Μεθοδολογία, δεδομένα και κριτήρια αξιολόγησης

Η μεθοδολογική προσέγγιση ακολουθεί την παραδοσιακή λογική σχεδιασμού συστημάτων, όπου το πρόβλημα ορίζεται, επιλέγονται κατάλληλα δεδομένα και εργαλεία, υλοποιείται ο αγωγός και στη συνέχεια αξιολογείται η λειτουργική του συμπεριφορά. Για τη φάση δεδομένων, αξιοποιούνται δημόσια διαθέσιμα σύνολα αρχείων καταγραφής, τα οποία έχουν χρησιμοποιηθεί εκτενώς στη βιβλιογραφία για συγκριτικές αξιολογήσεις. Ειδικότερα, η συλλογή LogHub και η νεότερη έκδοση LogHub 2.0 παρέχουν ποικιλία μορφών και πηγών, επιτρέποντας να διερευνηθεί η γενικότητα της λύσης (LogPAI 2023, Jiang και συνεργάτες 2023).

Για την ανάλυση προτύπων και την προεπεξεργασία εφαρμόζονται τεχνικές εξαγωγής προτύπων και παραμέτρων, οι οποίες μειώνουν την ποικιλία των μηνυμάτων και παράγουν ακολουθίες γεγονότων σε επίπεδο προτύπου. He και συνεργάτες 2017. Για την ανίχνευση ενσωματώνονται αλγόριθμοι που καλύπτουν διαφορετικές παραδοχές, όπως συνεχόμενα μοντέλα και μοντέλα που συνδυάζουν και την συνεχόμενη αλλά και την ποσοτική πληροφορία (Du 2017, Meng 2019).

Η αξιολόγηση της ποιότητας ανίχνευσης στηρίζεται στις μετρικές ακρίβειας, ανάκλησης και μέτρου F1, οι οποίες είναι καθιερωμένες στη δυαδική ταξινόμηση, ενώ απαιτείται προσοχή σε περιπτώσεις ανισορροπημένων δεδομένων. Η πρακτική ανισορροπία είναι αναμενόμενη στα αρχεία καταγραφής, διότι τα σπάνια γεγονότα αποτελούν μικρό ποσοστό σε σχέση με τη φυσιολογική λειτουργία.

Γι’ αυτό, η ερμηνεία των μετρικών γίνεται με γνώση των περιορισμών τους και με συμπληρωματική ανάγνωση μέσω καμπυλών ακρίβειας και ανάκλησης, όπως έχει τεκμηριωθεί στη βιβλιογραφία αξιολόγησης ταξινομητών (Saito και Rehmsmeier 2015, Powers 2011). Παράλληλα, εισάγεται μετρική χρόνου εξαγωγής απόφασης ανά μήνυμα ή ανά παράθυρο, ώστε να αποτιμηθεί η καταλληλότητα του αλγορίθμου σε λειτουργία συνεχούς ροής(Apache Flink 2025).

Ως προς την υλοποίηση της ροής, υιοθετείται το σχήμα παραγωγός, δίαυλος γεγονότων, καταναλωτές. Το Apache Kafka λειτουργεί ως κατανεμημένος μηχανισμός αποθήκευσης και διανομής γεγονότων, με έννοιες όπως θέματα, κατατμήσεις και αναπαραγωγή, που επιτρέπουν κλιμάκωση και ανθεκτικότητα. Apache Software Foundation 2025. Για το επίπεδο

οπτικοποίησης και παρακολούθησης, προτείνεται αξιοποίηση στοίβας εργαλείων που έχει καθιερωθεί για αναζήτηση και απεικόνιση καταγραφών, με χαρακτηριστικό παράδειγμα το Elastic Stack, το οποίο παρέχει μηχανισμό ευρετηρίασης και πίνακες ελέγχου. Elastic 2024. Συμπληρωματικά, η συζήτηση εντάσσει και τη διάσταση μετρήσεων και παρατηρησιμότητας, διότι σε κατανεμημένα συστήματα απαιτείται συνδυασμός καταγραφών και μετρικών για πρακτική διερεύνηση(Prometheus 2025, Sridharan 2018).

1.5 Δομή και οργάνωση της εργασίας

Το υπόλοιπο της εργασίας οργανώνεται με τρόπο που ακολουθεί τη φυσική πορεία από τη θεωρία προς την υλοποίηση και την αξιολόγηση. Στο δεύτερο κεφάλαιο παρουσιάζεται το θεωρητικό υπόβαθρο της ανίχνευσης ανωμαλιών σε αρχεία καταγραφής, με έμφαση σε στάδια προεπεξεργασίας, ανάλυση προτύπων και βασικές κατηγορίες αλγορίθμων(Landauer 2023).

Στο τρίτο κεφάλαιο περιγράφεται η αρχιτεκτονική του προτεινόμενου συστήματος και τα επιμέρους υποσυστήματα, συμπεριλαμβανομένων των επιλογών διασύνδεσης ροής γεγονότων, αποθήκευσης και απεικόνισης (Apache Software Foundation 2025, Elastic 2024). Στο τέταρτο κεφάλαιο τεκμηριώνεται η πρακτική υλοποίηση του αγωγού, οι ρυθμίσεις, τα σενάρια εκτέλεσης και τα σημεία ελέγχου.

Στο πέμπτο κεφάλαιο παρουσιάζονται τα αποτελέσματα αξιολόγησης για διαφορετικούς αλγορίθμους και ρυθμίσεις, με παράλληλη αποτίμηση χρόνου εξαγωγής απόφασης και επιχειρησιακής χρηστικότητας του πίνακα ελέγχου (Saito και Rehmsmeier 2015).

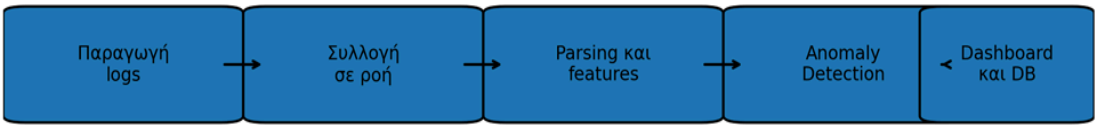
Τέλος, στο έκτο κεφάλαιο συνοψίζονται τα κύρια ευρήματα, τα όρια της μελέτης και προτείνονται κατευθύνσεις για μελλοντική επέκταση, όπως ενσωμάτωση ειδοποιήσεων, συσχέτιση με ίχνη εκτέλεσης και εφαρμογή σε πραγματικά περιβάλλοντα παραγωγής (Sridharan 2018).

ΚΕΦΑΛΑΙΟ 2: ΘΕΩΡΗΤΙΚΟ ΠΛΑΙΣΙΟ ΚΑΙ ΒΙΒΛΙΟΓΡΑΦΙΚΗ ΑΝΑΣΚΟΠΗΣΗ

Εισαγωγή στο θεωρητικό πλαίσιο

Το παρόν κεφάλαιο οργανώνει το θεωρητικό υπόβαθρο που απαιτείται για την κατανόηση και τεκμηρίωση ενός συστήματος ανίχνευσης ανωμαλιών σε αρχεία καταγραφής, με προσανατολισμό στην αρχιτεκτονική λογισμικού και στη ροή επεξεργασίας σε πραγματικό χρόνο. Η βιβλιογραφία αντιμετωπίζει τα logs ως κρίσιμη μορφή τηλεμετρίας, επειδή αποτυπώνουν γεγονότα λειτουργίας, σφάλματα και ενέργειες χρηστών και υπηρεσιών, επιτρέποντας αποσφαλμάτωση, ανάλυση αιτιών και λειτουργική επιτήρηση. Η ανάγκη συστηματικής διαχείρισης logs έχει αποτυπωθεί και σε κατευθυντήριες οδηγίες για την ασφάλεια, όπου τονίζεται ότι η συλλογή, η αποθήκευση, ο συγχρονισμός χρόνου, η ακεραιότητα και η αναλυτική αξιοποίηση είναι προϋποθέσεις για έγκαιρο εντοπισμό συμβάντων και κατάχρησης (Kent & Souppaya, 2006).

Για την ανίχνευση ανωμαλιών σε logs, η έρευνα συγκλίνει σε μία αλληλουχία βημάτων που περιλαμβάνει συλλογή, parsing, εξαγωγή χαρακτηριστικών και εφαρμογή αλγορίθμων, ώστε τα μη δομημένα ή ημιδομημένα μηνύματα να μετασχηματισθούν σε αναπαραστάσεις κατάλληλες για αυτοματοποιημένη αξιολόγηση. Στο ίδιο πνεύμα, σύγχρονες προσεγγίσεις log analytics στηρίζονται σε δημόσια datasets και συγκριτικές αξιολογήσεις, καθώς η διαθεσιμότητα δεδομένων αποτελεί προϋπόθεση για αξιόπιστες συγκρίσεις και αναπαραγωγιμότητα (Zhu et al., 2020).



Εικόνα 2.1. Εννοιολογική ροή ενός συστήματος ανίχνευσης ανωμαλιών σε logs

2.1 Αρχεία καταγραφής, παρατηρησιμότητα και λειτουργική αξία

Τα αρχεία καταγραφής αποτελούν πρωτογενές ίχνος της συμπεριφοράς του συστήματος και λειτουργούν συμπληρωματικά προς τις μετρικές και τα ίχνη εκτέλεσης, συγκροτώντας μαζί με αυτά το τρίπτυχο των τριών πυλώνων της παρατηρησιμότητας (logs, metrics, traces). Στο πλαίσιο των προδιαγραφών , η ενοποίηση σημάτων επιτρέπει συσχέτιση γεγονότων και εντοπισμό ριζικών αιτιών, ενώ η συλλογή logs εντάσσεται σε πρακτικές που επιτρέπουν εμπλουτισμό με μεταδεδομένα και συσχέτιση με πόρους και υπηρεσίες (OpenTelemetry, 2025).

Σε επιχειρησιακό περιβάλλον, τα logs χρησιμοποιούνται για ανάλυση αποτυχιών, για έλεγχο πολιτικών πρόσβασης, για ανίχνευση επιθέσεων και για τεκμηρίωση συμμόρφωσης. Σύμφωνα με οδηγίες της NIST, ένα πρόγραμμα διαχείρισης logs χρειάζεται να καλύπτει συλλογή από πολλαπλές πηγές, κατάλληλη αποθήκευση και πολιτικές διατήρησης, καθώς και διαδικασίες ανάλυσης και αναφοράς, επειδή τα logs αποκτούν αξία μόνο όταν μπορούν να αναζητηθούν και να συσχετισθούν με τρόπο συνεπή και αξιόπιστο (Kent & Souppaya, 2006).

Η πρακτική χρησιμότητα των logs ενισχύεται όταν η εφαρμογή παράγει μηνύματα με σαφή επίπεδα σοβαρότητας, σταθερά πεδία πλαισίου και συνεπείς ταυτότητες γεγονότων. Η τάση της βιομηχανίας προς δομημένη καταγραφή μειώνει το κόστος parsing και υποστηρίζει γρηγορότερη ανάλυση, ιδίως όταν τα logs διακινούνται σε pipelines που πρέπει να λειτουργούν με χαμηλή καθυστέρηση και υψηλό ρυθμό.

Πηγή	Παραδείγματα	Τυπικό περιεχόμενο	Ιδιαιτερότητες για ανίχνευση ανωμαλιών
Λειτουργικό σύστημα	kernel, syslog	σφάλματα, drivers, services	υψηλή ποικιλία, μεταβλητές τιμές
Εφαρμογές	web app, API	ενέργειες χρηστών, exceptions	συσχέτιση με χρήστες και sessions

Δίκτυο	firewall, proxy	συνδέσεις, απορρίψεις, πολιτικές	έντονη εποχικότητα, επιθέσεις σε κύματα
Βάσεις δεδομένων	DB logs	queries, locks, errors	σύνδεση με επιδόσεις και deadlocks
Πλατφόρμες	containers, orchestrators	events, restarts, scheduling	συσχέτιση με υποδομή και πόρους

Πίνακας 2.1. Ενδεικτικές κατηγορίες πηγών logs και χαρακτηριστικά τους

2.2 Μορφότυπα, τυποποίηση και μεταφορά μηνυμάτων

Η μορφή ενός log επηρεάζει άμεσα την ικανότητα αυτόματης επεξεργασίας. Σε πολλά περιβάλλοντα, η μεταφορά μηνυμάτων γίνεται μέσω syslog, ενός πρωτοκόλλου που καθορίζει δομή επικεφαλίδας και επιτρέπει μεταφορά σε διαφορετικά μεταφορικά στρώματα. Το RFC 5424 περιγράφει σύγχρονο φορμά syslog με πεδία όπως timestamp, hostname, app name και structured data, με στόχο επεκτασιμότητα και ευκολότερη μηχανική ανάλυση (Gerhards, 2009).

Παράλληλα, η υιοθέτηση JSON ως δομημένου φορμά έχει αυξηθεί λόγω συμβατότητας με συστήματα αναζήτησης και analytics. Η δομημένη καταγραφή διευκολύνει τον εμπλουτισμό με πεδία πλαισίου, μειώνει την αβεβαιότητα parsing και ενισχύει τη δυνατότητα τυποποιημένων ερωτημάτων, ιδίως σε πλατφόρμες όπως το Elastic Stack που συνδυάζουν ingest, index και οπτικοποίηση (Elastic, 2025).

Προσέγγιση	Πλεονεκτήματα	Μειονεκτήματα	Κατάλληλη χρήση
Syslog RFC 5424	τυποποιημένη επικεφαλίδα, structured data	ποικιλία υλοποιήσεων, ανάγκη ρύθμισης	υποδομή, δικτυακά στοιχεία

JSON structured logs	εύκολο indexing, σταθερά πεδία	μεγαλύτερο μέγεθος μηνυμάτων	εφαρμογές, microservices
Plain text free form	γρήγορη υιοθέτηση, συμβατότητα	δύσκολο parsing, υψηλή μεταβλητότητα	παλαιά συστήματα, legacy

Πίνακας 2.2. Σύγκριση προσεγγίσεων φορμά και μεταφοράς logs

2.3 Διαχείριση κύκλου ζωής logs και απαιτήσεις ασφάλειας

Η διαχείριση logs δεν περιορίζεται στη συλλογή, αλλά καλύπτει πολιτικές διατήρησης, πρόσβασης και προστασίας. Η NIST υπογραμμίζει ότι αποτελεσματική διαχείριση logs απαιτεί ορισμό πηγών, διαδικασίες κανονικοποίησης, συντονισμό χρόνου και μηχανισμούς προστασίας ακεραιότητας, επειδή τα logs μπορεί να αποτελούν αποδεικτικό υλικό σε περιστατικά ασφάλειας (Kent & Souppaya, 2006).

Στο πλαίσιο συστημάτων ανίχνευσης ανωμαλιών, οι πολιτικές διατήρησης επηρεάζουν και την εκπαίδευση ή βαθμονόμηση αλγορίθμων, διότι η ιστορικότητα καθορίζει τι θεωρείται κανονική συμπεριφορά και πώς αντιμετωπίζεται η μεταβολή συμπεριφοράς στο χρόνο. Για αυτό, η αποθήκευση τυπικά οργανώνεται σε διακριτά επίπεδα, με ένα επίπεδο άμεσης αναζήτησης για πρόσφατα δεδομένα και ένα επίπεδο αρχειοθέτησης για παλαιότερα δεδομένα, ώστε να ισορροπεί το κόστος με την ανάγκη αναδρομικής ανάλυσης.

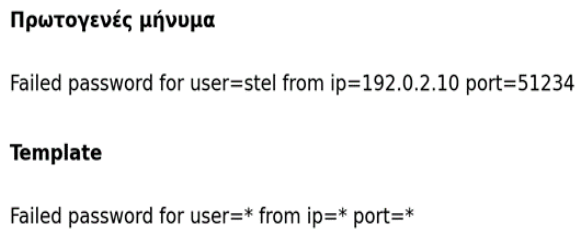
2.4 Προεπεξεργασία και log parsing ως κρίσιμος μετασχηματισμός

Τα logs συχνά είναι ημιδομημένα, δηλαδή περιέχουν σταθερά τμήματα κειμένου και μεταβλητές τιμές, όπως αναγνωριστικά χρηστών, διευθύνσεις δικτύου, κωδικούς σφαλμάτων και χρονικά μεγέθη. Το log parsing επιχειρεί να διαχωρίσει το σταθερό πρότυπο του μηνύματος από τις παραμέτρους, δημιουργώντας templates που επιτρέπουν ομαδοποίηση και στατιστική ανάλυση. Η ανάγκη για parsing κλιμακώνεται όταν η ροή δεδομένων είναι συνεχής, επειδή η χειρωνακτική ερμηνεία δεν είναι πρακτικά εφικτή (He et al., 2017).

Η μέθοδος Drain έχει προταθεί ως online parser που στηρίζεται σε δέντρο σταθερού βάθους και κανόνες ομοιότητας, ώστε να λειτουργεί αποδοτικά σε streaming συνθήκες και να ενημερώνει templates χωρίς πλήρη επανεκπαίδευση σε batch (He et al., 2017). Η αξιολόγηση parsers έχει επίσης τυποποιηθεί σε εργαλεία και benchmarks, ώστε να συγκρίνονται τεχνικές

σε ποικιλία datasets, καθώς η ποιότητα parsing επηρεάζει έμμεσα την ποιότητα ανίχνευσης ανωμαλιών (Zhu et al., 2019).

Πρόσφατες εμπειρικές μελέτες δείχνουν ότι η επιλογή parser μπορεί να μεταβάλει ουσιαστικά την ακρίβεια μοντέλων ανίχνευσης, επειδή διαφορετικές τεχνικές παράγουν διαφορετικά templates και διαφορετικό επίπεδο απώλειας πληροφορίας. Η παρατήρηση αυτή είναι κρίσιμη σε διπλωματικές που στοχεύουν σε ολοκληρωμένο pipeline, καθώς η συνολική επίδοση του συστήματος εξαρτάται από τη συνέργεια parsing και αλγορίθμων (Khan et al., 2024).



Εικόνα 2.2. Παράδειγμα μετασχηματισμού μηνύματος σε template και παραμέτρους

Κατηγορία	Ιδέα	Πλεονεκτήματα	Περιορισμοί
Κανόνες και regex	χειρωνακτικά πρότυπα	διαφάνεια, εύκολος έλεγχος	δεν κλιμακώνει, ευθραυστότητα σε αλλαγές
Δομές δέντρων	ομαδοποίηση με κανόνες ομοιότητας	αποδοτικότητα σε ροές, online ενημέρωση	ευαισθησία σε παραμέτρους
Στατιστικές μέθοδοι	μοτίβα με συχνότητες	μειωμένη ανάγκη κανόνων	αστάθεια σε θόρυβο
Μάθηση αναπαραστάσεων	εκμάθηση templates έμμεσα	δυνατότητα γενίκευσης	υψηλότερο κόστος και πολυπλοκότητα

Πίνακας 2.3. Κατηγορίες log parsing και επιπτώσεις στο pipeline

2.5 Αναπαράσταση logs και εξαγωγή χαρακτηριστικών

Μετά το parsing, ένα σύστημα ανίχνευσης ανωμαλιών χρειάζεται να μετασχηματίσει τα templates και τις παραμέτρους σε αναπαραστάσεις που μπορούν να χρησιμοποιηθούν από αλγορίθμους. Μία βασική επιλογή είναι η μοντελοποίηση ως ακολουθίες γεγονότων, όπου κάθε template αντιστοιχεί σε γεγονός και η ροή μετατρέπεται σε ακολουθία συμβόλων. Η λογική αυτή υιοθετείται από προσεγγίσεις που αντιμετωπίζουν το log ως γλώσσα γεγονότων, ώστε να προβλέπουν την επόμενη αναμενόμενη είσοδο και να επισημαίνουν αποκλίσεις (Du et al., 2017).

Η αναπαράσταση ως ακολουθία είναι ιδιαίτερα χρήσιμη σε συστήματα όπου οι αστοχίες εμφανίζονται ως μη αναμενόμενες μεταβάσεις κατάστασης και όχι ως μεμονωμένα σπάνια μηνύματα. Αντίθετα, σε περιβάλλοντα με έντονη επαναληπτικότητα, η αναπαράσταση ως διάνυσμα συχνοτήτων εντός χρονικών παραθύρων μπορεί να αποδώσει πρακτικά, επειδή μετατρέπει τη ροή σε σταθερού μήκους περιγραφές και επιτρέπει στατιστικούς ή κλασικούς αλγορίθμους. Σύγχρονες προσεγγίσεις αξιοποιούν και μοντέλα μετασχηματιστών για να μάθουν συμφραζόμενα σε μεγαλύτερες ακολουθίες, προτείνοντας αυτοεπιτηρούμενες τεχνικές για εντοπισμό ασυνήθιστων μοτίβων (Guo et al., 2021).

Αναπαράσταση	**Παράδειγμα**	**Κατάλληλη για**	**Σχόλια**
Ακολουθία γεγονότων	template ids ανά session	sequence models	αποτυπώνει τάξη και συμφραζόμενα
Παράθυρα συχνοτήτων	counts ανά time window	statistical, clustering	ανθεκτικό σε μικρές παραλλαγές
Σημασιολογικά embeddings	ενσωματώσεις templates	deep models	καλύτερη γενίκευση με κόστος
Υβριδικά χαρακτηριστικά	counts και embeddings	σύνθετα μοντέλα	συχνά καλύτερη ισορροπία

Πίνακας 2.4. Αναπαραστάσεις και χαρακτηριστικά για ανίχνευση ανωμαλιών

2.6 Αλγόριθμοι ανίχνευσης ανωμαλιών σε logs και τυπικές κατηγορίες

Η βιβλιογραφία προτείνει πλήθος αλγορίθμων για ανίχνευση ανωμαλιών, από στατιστικές προσεγγίσεις μέχρι βαθιά νευρωνικά μοντέλα. Σε επίπεδο αρχών, το πρόβλημα μπορεί να διατυπωθεί ως ανίχνευση απόκλισης από πρότυπα κανονικής συμπεριφοράς, όπου η κανονικότητα εκτιμάται από ιστορικά δεδομένα ή από τρέχουσα ροή. Εκτενείς ανασκοπήσεις επισημαίνουν ότι οι βαθιές προσεγγίσεις σε logs συχνά υπερτερούν όταν οι ανωμαλίες είναι αλληλουχιακές, αλλά απαιτούν προσεκτική προεπεξεργασία και κατάλληλα datasets, ενώ η απλούστερη στατιστική ανίχνευση παραμένει χρήσιμη λόγω διαφάνειας και χαμηλού κόστους (Landauer et al., 2022).

Το DeepLog αποτελεί χαρακτηριστικό παράδειγμα μοντελοποίησης ακολουθιών με LSTM, όπου το μοντέλο μαθαίνει την κανονική ακολουθία γεγονότων και χαρακτηρίζει ως ανωμαλία τις αποκλίσεις από τις προβλέψεις της επόμενης κατάστασης (Du et al., 2017). Αντίστοιχα, το LogBERT αξιοποιεί μετασχηματιστές για να κωδικοποιήσει συμφραζόμενα σε διπλής κατεύθυνσης αναπαραστάσεις, υποστηρίζοντας ότι η αυτοεπιτηρούμενη μάθηση διευκολύνει γενίκευση σε πολυπλοκότερα μοτίβα (Guo et al., 2021).

Για μία διπλωματική με έμφαση σε pipeline, η πρακτική επιλογή είναι να ενσωματωθούν έτοιμοι αλγόριθμοι ως συνιστώσες, ώστε η αξιολόγηση να επικεντρώνεται τόσο στην ποιότητα ανίχνευσης όσο και στην επιχειρησιακή συμπεριφορά, όπως ο χρόνος απόφασης, η σταθερότητα ροής και η ευκολία ερμηνείας αποτελεσμάτων. Η προσέγγιση αυτή είναι συνεπής με την ιδέα ότι το σύστημα αποτελεί συνδυασμό προεπεξεργασίας, μοντέλου και οπτικοποίησης, όπου ένα βελτιστοποιημένο επιμέρους μοντέλο δεν επαρκεί αν το συνολικό σύστημα έχει υψηλή καθυστέρηση ή χαμηλή αξιοπιστία.

Κατηγορία	Βασική αρχή	Ενδεικτικά πλεονεκτήματα	Κύριοι κίνδυνοι
Στατιστικές μέθοδοι	έλεγχοι συχνότητας και αποκλίσεων	διαφάνεια, χαμηλό κόστος	ευαισθησία σε εποχικότητα
Απόσταση και clustering	ομαδοποίηση και outliers	απλότητα, γρήγορη εφαρμογή	εξάρτηση από κλίμακες features
Μοντέλα ακολουθιών	πρόβλεψη επόμενων γεγονότων	καλή κάλυψη sequence anomalies	ανάγκη ποιοτικού parsing
Transformers	συμφραζόμενα μεγάλης εμβέλειας	γενίκευση, ισχυρές αναπαραστάσεις	υπολογιστικό κόστος

Πίνακας 2.5. Κατηγοριοποίηση τεχνικών ανίχνευσης ανωμαλιών σε logs

2.7 Δημόσια σύνολα δεδομένων και ζητήματα ετικετοποίησης

Η αξιολόγηση τεχνικών log anomaly detection εξαρτάται από τη διαθεσιμότητα datasets με αντιπροσωπευτικά σενάρια και με αξιόπιστες ετικέτες ανωμαλίας. Η έλλειψη δημόσιων logs ιστορικά περιόριζε την αναπαραγωγιμότητα, καθώς οργανισμοί διστάζουν να δημοσιεύσουν δεδομένα λόγω ευαισθησίας. Το LogHub προτείνεται ως συλλογή πραγματικών logs από διαφορετικές κατηγορίες συστημάτων, παρέχοντας datasets σε ποικιλία φορμά και επιπέδων επισημείωσης (Zhu et al., 2020).

Η ύπαρξη datasets με ετικέτες επιτρέπει χρήση μετρικών ταξινόμησης, όπως precision, recall και F1, ωστόσο η διαδικασία παραγωγής ετικετών είναι σύνθετη και συχνά βασίζεται σε συμβάντα βλάβης, σε χειρωνακτική αξιολόγηση ή σε συσχέτιση με alerts άλλων εργαλείων. Ακόμη και όταν υπάρχουν ετικέτες, το φαινόμενο θορύβου ετικετών είναι πιθανό, ιδίως σε μεγάλης κλίμακας υποδομές όπου πολλαπλά συμβάντα συνυπάρχουν. Για αυτό, η βιβλιογραφία συστήνει προσεκτικό ορισμό των σεναρίων αξιολόγησης και σαφή περιγραφή της διαδικασίας labeling.

Χαρακτηριστικό	Περιγραφή	Επίδραση στην αξιολόγηση

Ποικιλία συστημάτων	διαφορετικές εφαρμογές και υποδομές	γενίκευση συμπερασμάτων
Ετικέτες ανωμαλίας	presence of ground truth	υποστήριξη F1 και συγκρίσεων
Μέγεθος ροής	όγκος και ρυθμός παραγωγής	stress test για real time
Μεταβολή μορφής	αλλαγές templates στο χρόνο	concept drift και parsing robustness

Πίνακας 2.6. Ενδεικτικά χαρακτηριστικά των datasets logs για αξιολόγηση

2.8 Αρχιτεκτονικές επεξεργασίας σε πραγματικό χρόνο και παρακολούθηση

Η λειτουργία σε πραγματικό χρόνο θέτει απαιτήσεις για καθυστέρηση, αξιοπιστία και κλιμάκωση. Οι σύγχρονες αρχιτεκτονικές αξιοποιούν message brokers και stream processors ώστε να αποσυνδέουν την παραγωγή από την κατανάλωση, να απορροφούν αιχμές και να επιτρέπουν παράλληλη επεξεργασία. Το Apache Kafka περιγράφεται ως πλατφόρμα κατανεμημένης ροής γεγονότων για pipelines υψηλής απόδοσης και εφαρμογές ροής, με αρχιτεκτονική που βασίζεται σε topics και κατανεμημένα partitions (Apache Kafka, 2025).

Σε επίπεδο επεξεργασίας, το Apache Flink ορίζεται ως μηχανή κατανεμημένης επεξεργασίας για stateful υπολογισμούς πάνω σε απεριόριστες και πεπερασμένες ροές, υποστηρίζοντας εγγυήσεις συνέπειας και event time semantics, στοιχεία σημαντικά όταν οι ακολουθίες γεγονότων πρέπει να ανακατασκευασθούν με σωστή χρονική σειρά (Apache Flink, 2025). Η επιλογή τέτοιων τεχνολογιών εξαρτάται από το απαιτούμενο επίπεδο κλιμάκωσης, αλλά η λογική τους παραμένει σταθερή, καθώς επιτρέπουν σαφή διαχωρισμό ingest, processing και serving.

Για την παρακολούθηση της υγείας του pipeline, τα συστήματα παρατηρησιμότητας αξιοποιούν συλλογή μετρικών και κανόνες ειδοποίησης. Το Prometheus παρουσιάζεται ως σύστημα παρακολούθησης που αποθηκεύει δεδομένα ως χρονοσειρές με ετικέτες και παρέχει γλώσσα ερωτημάτων για ειδοποιήσεις και απεικόνιση (Prometheus, 2025). Η οπτικοποίηση και οι πίνακες ελέγχου σε περιβάλλοντα παραγωγής συχνά υλοποιούνται με εργαλεία όπως το Grafana, το οποίο επιτρέπει ερωτήματα, οπτικοποίηση και ειδοποιήσεις για μετρικές, logs και ίχνη, ανεξάρτητα από την πηγή αποθήκευσης (Grafana, 2025).

Σε συστήματα log analytics, η αποθήκευση και ευρετηρίαση είναι καθοριστικές για γρήγορη διερεύνηση. Το Elastic Stack προτείνει ολοκληρωμένη αλυσίδα ingest, index και οπτικοποίησης, με στόχο την αναζήτηση και ανάλυση logs και την υποστήριξη παρατηρησιμότητας σε ενιαία πλατφόρμα (Elastic, 2025). Στο πλαίσιο της παρούσας εργασίας, η υιοθέτηση τέτοιων συστατικών μπορεί να τεκμηριωθεί ως επιλογή που υποστηρίζει γρήγορη αναζήτηση, συσχέτιση και παρουσίαση ανωμαλιών, χωρίς να απαιτείται εκτεταμένη ανάπτυξη εργαλείων από το μηδέν.

Στο πλαίσιο της παρούσας διπλωματικής, η πρακτική υλοποίηση του pipeline ακολουθεί μία ελαφριά αρχιτεκτονική που μπορεί να εκτελεστεί αξιόπιστα σε περιβάλλον φορητού υπολογιστή, χωρίς ανάγκη υποδομών παραγωγής. Η ροή δεδομένων υλοποιείται με Python ως μηχανισμό συλλογής και προεπεξεργασίας, η αποθήκευση των εμπλουτισμένων εγγραφών και των labels πραγματοποιείται σε τοπική βάση SQLite ώστε να εξασφαλίζεται αναπαραγωγιμότητα, και η οπτικοποίηση και διερεύνηση ανωμαλιών πραγματοποιείται μέσω Streamlit dashboard που ανανεώνεται από το ίδιο σύνολο δεδομένων. Με αυτή τη σχεδίαση διατηρείται η λογική του real time pipeline σε επίπεδο λειτουργίας, ενώ τα συστατικά υψηλότερης κλιμάκωσης που παρουσιάζονται παραπάνω λειτουργούν ως τεκμηριωμένη αρχιτεκτονική αναφορά και ως προτεινόμενη επέκταση για σενάρια μεγάλου όγκου και πολλαπλών πηγών.

Εικόνα 2.3. Ενδεικτική αρχιτεκτονική ροής logs και συστατικών παρατηρησιμότητας

Ρόλος	Σκοπός	Ενδεικτικές τεχνολογίες	Παρατηρήσεις
Συλλογή	μεταφορά logs από πηγές	agents, shippers	απαιτεί αξιοπιστία και buffering
Streaming bus	αποσύνδεση παραγωγού και καταναλωτή	Kafka	topics και partitions για κλίμακα
Επεξεργασία	parsing, features, detection	Flink ή custom services	stateful λογική και παράθυρα
Αποθήκευση	αναζήτηση, ιστορικό, labels	Elasticsearch ή SQL	πολιτικές διατήρησης
Παρακολούθηση	μετρικές pipeline και alerts	Prometheus	SLO και κανόνες
Dashboard	οπτικοποίηση και διερεύνηση	Grafana ή Kibana	φίλτρα και drill down

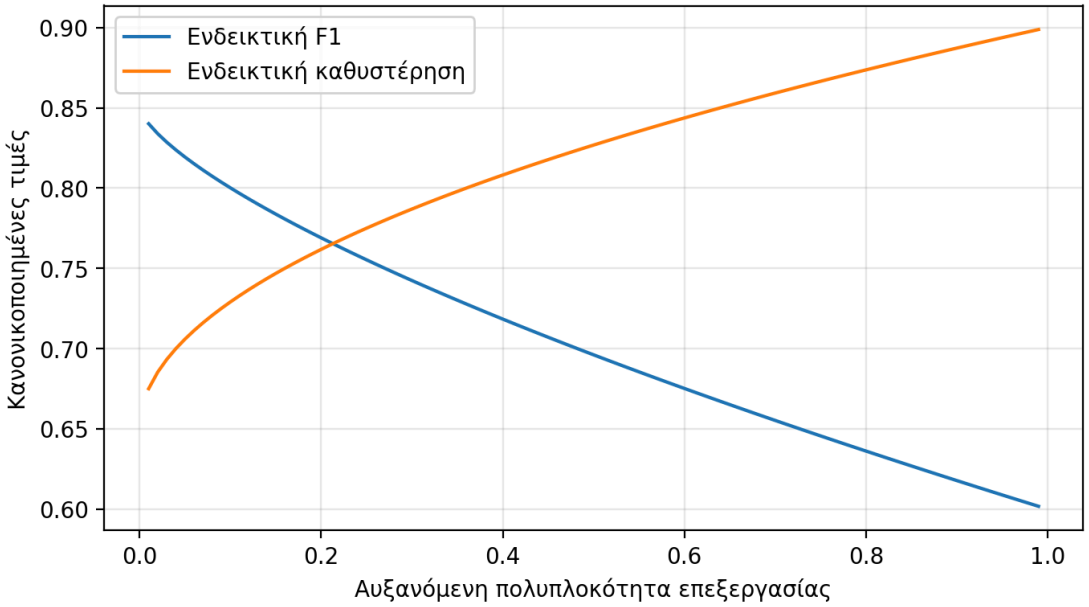
Πίνακας 2.7. Χαρτογράφηση συστατικών ενός real time pipeline σε λειτουργικούς ρόλους

2.9 Αξιολόγηση απόδοσης, μετρικές και χρόνος απόφασης

Η αξιολόγηση ενός συστήματος ανίχνευσης ανωμαλιών απαιτεί διάκριση μεταξύ ποιότητας ταξινόμησης και επιχειρησιακής απόδοσης. Σε επίπεδο ποιότητας, οι μετρικές precision, recall και F1 είναι καθιερωμένες, επειδή αποτυπώνουν το ποσοστό ορθών θετικών ενδείξεων, το ποσοστό κάλυψης πραγματικών ανωμαλιών και τη συνολική ισορροπία μεταξύ των δύο. Η χρήση τους προϋποθέτει ετικέτες και συνεπή ορισμό ανωμαλίας, κάτι που στα logs συχνά επιτυγχάνεται μέσω datasets όπως το LogHub ή μέσω επισημείωσης βάσει συμβάντων βλάβης (Zhu et al., 2020).

Σε επίπεδο συστήματος, ο χρόνος απόφασης έχει ιδιαίτερη σημασία σε ροές, καθώς ένα σύστημα μπορεί να έχει υψηλή ακρίβεια αλλά να αποτυγχάνει επιχειρησιακά αν η καθυστέρηση από ingest έως απόφαση υπερβαίνει τα όρια παρέμβασης. Η καθυστέρηση μπορεί να διακριθεί σε καθυστέρηση επεξεργασίας, που σχετίζεται με parsing και inference, και σε καθυστέρηση ουράς, που σχετίζεται με buffering και συμφόρηση. Σε stream processors

με stateful μηχανισμούς συνέπειας, ακόμη και τεχνικές όπως τα checkpoints μπορεί να εισάγουν καθυστέρηση, επομένως η επιλογή ρυθμίσεων πρέπει να ισορροπεί τη συνέπεια με το κόστος καθυστέρησης (Apache Flink, 2025).



Εικόνα 2.4. Εννοιολογική απεικόνιση ανταλλαγής μεταξύ ποιότητας και καθυστέρησης

Μετρική	Ορισμός	Τι δείχνει	Σχόλιο για logs
Precision	TP / (TP + FP)	πόσα alerts είναι σωστά	σχετίζεται με κόπωση ειδοποιήσεων
Recall	TP / (TP + FN)	πόσες ανωμαλίες εντοπίστηκαν	σχετίζεται με κίνδυνο διαφυγής
F1	2PR / (P + R)	ισορροπία precision και recall	χρήσιμο για συγκρίσεις
Latency	χρόνος από ingest έως label	ταχύτητα απόφασης	κρίσιμο για real time
Throughput	events ανά δευτερόλεπτο	ικανότητα ροής	σχετίζεται με κλιμάκωση

Πίνακας 2.8. Ορισμοί βασικών μετρικών αξιολόγησης σε προβλήματα ανίχνευσης

2.10 Σύνοψη κεφαλαίου

Το κεφάλαιο ανέδειξε ότι η ανίχνευση ανωμαλιών σε logs είναι πρωτίστως πρόβλημα ολοκλήρωσης συστήματος, όπου η ποιότητα προκύπτει από την αλληλεπίδραση συλλογής, parsing, αναπαράστασης και αλγορίθμου. Η τυποποίηση επιτρέπει καλύτερη μηχανική αξιοποίηση, ενώ η διαχείριση κύκλου ζωής και οι πολιτικές ασφάλειας εξασφαλίζουν ότι τα δεδομένα είναι διαθέσιμα και αξιόπιστα. Η βιβλιογραφία δείχνει ότι η επιλογή parsing επηρεάζει σημαντικά την τελική επίδοση, και ότι τα δημόσια datasets όπως το LogHub επιτρέπουν συγκρίσιμη αξιολόγηση. Τέλος, οι real time αρχιτεκτονικές με Kafka και Flink, σε συνδυασμό με εργαλεία παρατηρησιμότητας όπως Prometheus, Grafana και Elastic, παρέχουν πρακτικό υπόβαθρο για υλοποίηση pipelines που έχουν τόσο ακρίβεια όσο και επιχειρησιακή χρησιμότητα.

Κεφάλαιο 3 Ανάλυση απαιτήσεων και σχεδίαση συστήματος

Το κεφάλαιο αποτυπώνει με όρους software engineering τις απαιτήσεις και τη σχεδίαση του συστήματος ανίχνευσης ανωμαλιών σε αρχεία καταγραφής, ώστε η πρακτική υλοποίηση να είναι αναπαραγώγιμη, ελέγξιμη και επαληθεύσιμη. Η λογική της σχεδίασης στηρίζεται στο ότι τα logs αποκτούν αξία όταν συλλέγονται με συνέπεια, κανονικοποιούνται, αποθηκεύονται με ασφαλή τρόπο και αναλύονται με διαδικασίες που επιτρέπουν αναζήτηση, συσχέτιση και έγκαιρη ειδοποίηση (Kent & Souppaya, 2006). Η ανάλυση που ακολουθεί διακρίνει λειτουργικές και μη λειτουργικές απαιτήσεις, τεκμηριώνει την επιλογή τεχνολογιών, περιγράφει την αρχιτεκτονική modules και ορίζει το σχήμα αποθήκευσης και τα σενάρια χρήσης που καθοδηγούν τον σχεδιασμό του dashboard. Η περιγραφή των απαιτήσεων ευθυγραμμίζεται με πρακτικές προδιαγραφών που δίνουν έμφαση στη σαφήνεια, στην ιχνηλασιμότητα και στα κριτήρια αποδοχής (IEEE, 2018).

3.1 Functional requirements (log gen, ingest, preprocess, detect, store, visualize)

Οι λειτουργικές απαιτήσεις καθορίζουν τι πρέπει να εκτελεί το σύστημα σε επίπεδο συμπεριφοράς. Η υλοποίηση οργανώνεται ως pipeline που μετασχηματίζει μία συνεχή ροή καταγραφών σε δομημένα γεγονότα, αποδίδει βαθμολογία ανωμαλίας και ετικέτα και στη συνέχεια αποθηκεύει τα αποτελέσματα και τα παρουσιάζει σε διαδραστικό περιβάλλον. Η προσέγγιση αυτή συνδέεται με την ανάγκη ο αναλυτής να διαθέτει τόσο το πρωτογενές ίχνος όσο και τα παράγωγα μεταδεδομένα για διερεύνηση αιτιών και μείωση ψευδών ειδοποιήσεων (Beyer et al., 2016).

Κωδικός	Απαίτηση	Είσοδος και επεξεργασία	Έξοδος και τεκμήριο
FR1	Παραγωγή ή εισαγωγή logs	Generator ή ανάγνωση από δημόσιο dataset	Δείγμα εγγραφών και έλεγχος μορφής
FR2	Ingestion σε ρυθμό πραγματικού χρόνου	Pacing σε σταθερό ρυθμό, προαιρετικό buffering	Καταγραφή ρυθμού εισόδου και μη απώλειας

FR3	Προεπεξεργασία και parsing	Κανονικοποίηση και αντιστοίχιση σε template ή EventId	Πεδία event_id, template, component, level
FR4	Εξαγωγή χαρακτηριστικών	Παράθυρα ή ακολουθίες γεγονότων, counts ή embeddings	Αποθήκευση feature_json ανά εγγραφή ή group
FR5	Ανίχνευση ανωμαλιών και scoring	Υπολογισμός score και label, καταγραφή infer_ms	Αποθήκευση score, label, infer_ms
FR6	Αποθήκευση και αναζήτηση	Συνεπής write σε SQLite, δείκτες στα πεδία	Επαλήθευση πληρότητας και ακεραιότητας
FR7	Οπτικοποίηση και διερεύνηση	Streamlit views με φίλτρα και drill down	Λίστα ανωμαλιών, γραφήματα, προβολή raw

Πίνακας 3.1. Λειτουργικές απαιτήσεις με κριτήρια ελέγχου

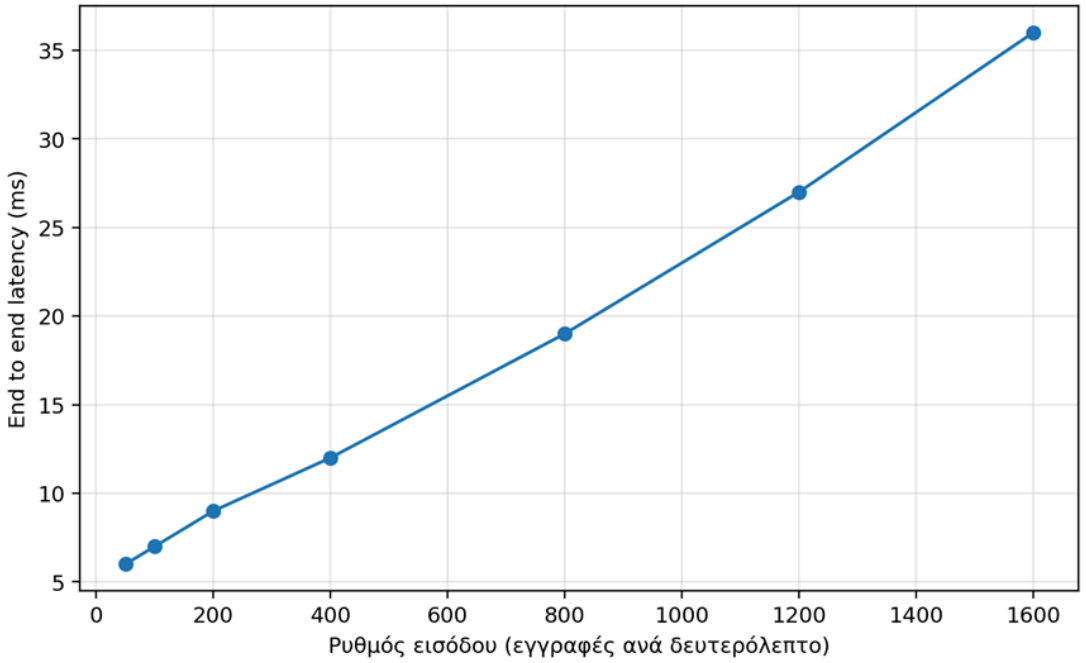
Η υλοποίηση της εισαγωγής σε ρυθμό πραγματικού χρόνου βασίζεται σε ελεγχόμενο pacing και σε συνεπή καταγραφή χρονικών σημάνσεων, ώστε να είναι δυνατή η μέτρηση καθυστέρησης και η αποτύπωση συμπεριφοράς ροής σε περιβάλλον laptop. Η διατήρηση raw και parsed εγγραφών υποστηρίζει ιχνηλασιμότητα και αναδρομική διερεύνηση, στοιχεία που αναδεικνύονται ως βασικές αρχές στη διαχείριση logs και στην απόκριση σε συμβάντα (Kent & Souppaya, 2006).

3.2 Non-functional requirements (real-time, scalability, reliability, latency στόχοι)

Οι μη λειτουργικές απαιτήσεις περιγράφουν τα χαρακτηριστικά ποιότητας που καθορίζουν αν το σύστημα είναι χρήσιμο στην πράξη. Δίνεται έμφαση σε καθυστέρηση, αξιοπιστία, επεκτασιμότητα και συντηρησιμότητα, επειδή η υψηλή ακρίβεια ανίχνευσης δεν επαρκεί όταν οι αποφάσεις καθυστερούν ή όταν η ροή είναι ασταθής. Η λογική αυτή ευθυγραμμίζεται με πρότυπα μοντέλων ποιότητας, όπου η απόδοση, η αξιοπιστία και η συντηρησιμότητα αποτελούν κεντρικές διαστάσεις (ISO/IEC, 2011).

Διάσταση	Απαίτηση	Κριτήριο αποδοχής	Μέθοδος τεκμηρίωσης
Real time	Συνεχής ροή	Σταθερή ενημέρωση dashboard	Screenshots και timestamps
Latency	Χαμηλή καθυστέρηση	Καταγραφή infer_ms και συνολικού latency	Πίνακες και γραφήματα
Reliability	Απουσία απώλειας	Ισότητα πλήθους εισόδου και πλήθους στη βάση	Έλεγχος counts
Scalability	Δυνατότητα επέκτασης	Επεξεργασία σε chunks ή παράθυρα	Χρόνοι φόρτωσης
Maintainability	Διάσπαση σε modules	Διακριτά scripts και διεπαφές	Δομή φακέλων
Usability	Διερεύνηση ανωμαλιών	Φίλτρα και drill down σε raw	Screenshots διεπαφής

Πίνακας 3.2. Μη λειτουργικές απαιτήσεις και κριτήρια αποδοχής



Εικόνα 3.1. Ενδεικτική σχέση ρυθμού εισόδου και καθυστέρησης σε pipeline laptop

Το γράφημα αποτυπώνει ενδεικτικά τη σχέση μεταξύ ρυθμού εισόδου και καθυστέρησης, ώστε να αναδεικνύεται γιατί η καθυστέρηση αποτελεί κρίσιμη απαίτηση σε συστήματα πραγματικού χρόνου. Στο πρακτικό μέρος η καθυστέρηση μετράται με χρονικές σημάνσεις και με καταγραφή χρόνου απόφασης, ώστε οι μη λειτουργικές απαιτήσεις να αντιστοιχούν σε ελέγξιμα κριτήρια (ISO/IEC, 2011).

3.3 Επιλογή τεχνολογιών & αιτιολόγηση (π.χ. Kafka/Elastic/DB, dashboard tool)

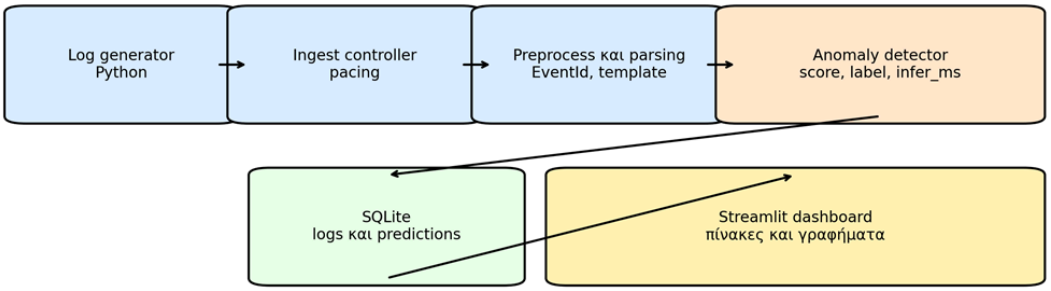
Η επιλογή τεχνολογιών πραγματοποιείται με κριτήρια αναπαραγωγιμότητας, χαμηλής πολυπλοκότητας εγκατάστασης και δυνατότητας παραγωγής τεκμηρίων εκτέλεσης. Η Python επιλέγεται επειδή υποστηρίζει γρήγορη ανάπτυξη επεξεργασίας κειμένου και εύκολη ενσωμάτωση αλγορίθμων. Η SQLite επιλέγεται ως ενσωματωμένη σχεσιακή βάση που υποστηρίζει συναλλαγές και αξιόπιστη αποθήκευση χωρίς εξωτερική υποδομή, ενισχύοντας την αναπαραγωγή της υλοποίησης σε περιβάλλον laptop (Owens, 2006). Το Streamlit επιλέγεται για ταχεία ανάπτυξη διαδραστικών dashboards που συνδέονται με το αποθηκευμένο αποτέλεσμα και τεκμηριώνονται με screenshots. Ως δημόσια πηγή δεδομένων αξιοποιείται το LogHub, επειδή συγκεντρώνει πραγματικά logs που χρησιμοποιούνται ευρέως στη βιβλιογραφία log analytics (Zhu et al., 2020).

Ρόλος	Επιλογή	Αιτιολόγηση	Εναλλακτική κλιμάκωσης
Processing	**Python**	**Ευελιξία και ταχεία ανάπτυξη**	**Stream processing frameworks**
Αποθήκευση	**SQLite**	**Απλότητα και αξιοπιστία**	**PostgreSQL**
Οπτικοποίηση	**Streamlit**	**Ταχεία ανάπτυξη διεπαφής**	**Kibana ή Grafana**
Δεδομένα	**LogHub HDFS**	**Δημόσια logs**	**Οργανωσιακά logs**

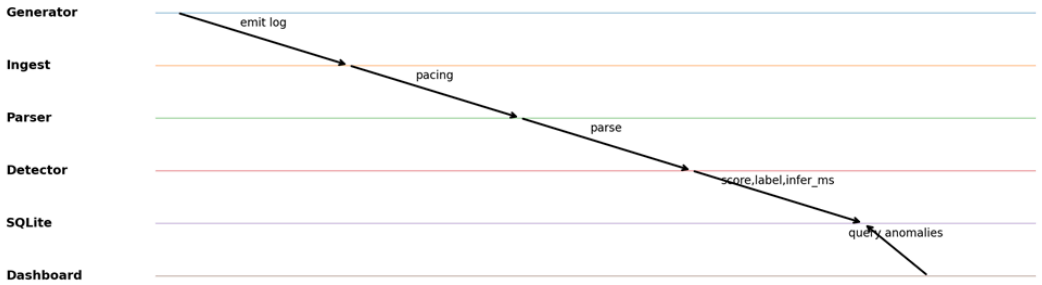
Πίνακας 3.3. Χαρτογράφηση τεχνολογιών σε ρόλους του συστήματος

3.4 Αρχιτεκτονική συστήματος (διάγραμμα modules + ροές δεδομένων)

Η αρχιτεκτονική ακολουθεί διάσπαση σε modules ώστε να περιορίζεται η σύζευξη και να διευκολύνεται ο έλεγχος. Η ανάθεση ευθυνών σε διακριτά επίπεδα, δηλαδή generator, ingestion controller, parsing layer, detection layer, persistence layer και dashboard layer, μειώνει το κόστος αλλαγών και αυξάνει τη συντηρησιμότητα, όπως προτείνεται στη βιβλιογραφία αρχιτεκτονικής λογισμικού (Bass et al., 2012).



Εικόνα 3.2. Αρχιτεκτονική modules και ροές δεδομένων στο προτεινόμενο pipeline

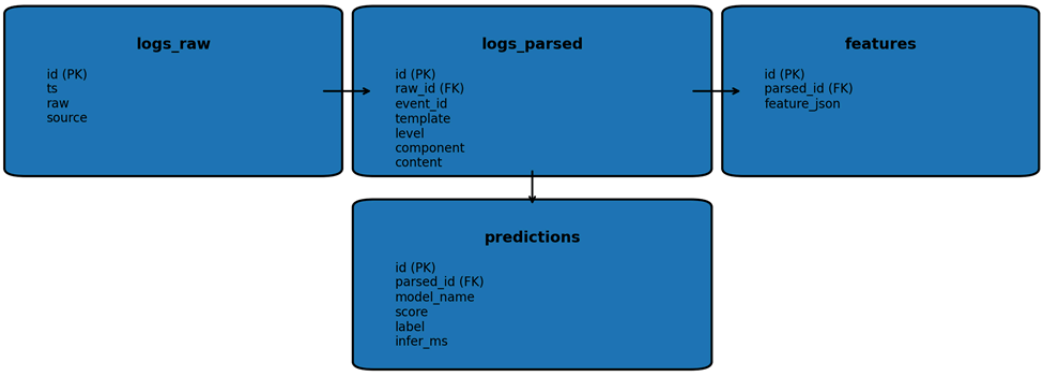


Εικόνα 3.3. Σειρά αλληλεπιδράσεων μεταξύ modules κατά την εκτέλεση

Η ροή δεδομένων υλοποιείται έτσι ώστε κάθε εγγραφή να αποκτά μοναδικό αναγνωριστικό στη βάση, να συνδέεται με την parsed εκδοχή της και να φέρει αποτελέσματα ανίχνευσης που περιλαμβάνουν score, label και χρόνο απόφασης. Η διατήρηση του infer_ms ως πεδίου στο αποτέλεσμα επιτρέπει να συνδεθεί η θεωρία των μετρικών με πρακτικές μετρήσεις και ελέγχους (IEEE, 2018).

3.5 Σχεδίαση βάσης (schema: raw log, parsed template, features, score, label, timestamps)

Το σχήμα της βάσης σχεδιάζεται ώστε να διατηρεί το πρωτογενές μήνυμα, την κανονικοποιημένη parsed εγγραφή, τα χαρακτηριστικά και το αποτέλεσμα ανίχνευσης. Η επιλογή ξεχωριστών πινάκων υποστηρίζει ιχνηλασιμότητα, καθώς κάθε raw εγγραφή συσχετίζεται με τη parsed εκδοχή της, ενώ τα predictions μπορούν να ανανεώνονται όταν αλλάζει ο αλγόριθμος ή το threshold χωρίς να αλλοιώνεται η ιστορική είσοδος. Η διατήρηση lineage μεταξύ εισόδων και παραγώγων προτείνεται ως πρακτική σε αξιόπιστα συστήματα δεδομένων (Kleppmann, 2017).



Εικόνα 3.4. Οντολογικό σχήμα SQLite και βασικές συσχετίσεις

Πίνακας	Ενδεικτικά πεδία	Χρήση στο dashboard	Χρήση στην αξιολόγηση
logs_raw	ts, raw, source	Προβολή πρωτογενούς μηνύματος	Έλεγχος πληρότητας εισόδου
logs_parsed	event_id, template, component, level	Φίλτρα και ομαδοποιήσεις	Στατιστική ανά template
features	feature_json	Προαιρετική προβολή χαρακτηριστικών	Αναπαραγωγή πειραμάτων
predictions	score, label, model_name, infer_ms	Λίστα ανωμαλιών και ταξινόμηση	Latency και μετρικές ταξινόμησης

Πίνακας 3.4. Πεδία σχήματος και λειτουργικός ρόλος

3.6 Σενάρια χρήσης (π.χ. “security analyst βλέπει spike failed logins”)

Τα σενάρια χρήσης καθοδηγούν τη σχεδίαση της διεπαφής και εξασφαλίζουν ότι το σύστημα υπηρετεί λειτουργικές ανάγκες. Στην περίπτωση spikes ο χρήστης αναζητά απότομη αύξηση ομοειδών γεγονότων σε μικρό χρονικό παράθυρο και επιθυμεί φίλτρα και προβολή raw μηνυμάτων. Στην περίπτωση rare events προτεραιότητα είναι η ταξινόμηση κατά score και η προβολή κρίσιμων πεδίων. Στην περίπτωση sequence anomalies απαιτείται ομαδοποίηση και προβολή ακολουθιών ώστε να γίνεται κατανοητή η απόκλιση. Η αποτύπωση σεναρίων χρήσης και η αντιστοίχισή τους σε λειτουργίες αποτελεί πρακτική ευθυγράμμισης απαιτήσεων και σχεδίασης (IEEE, 2018).

Σενάριο	Είδος ανωμαλίας	Κύριες λειτουργίες	Δεδομένα που απαιτούνται
Εξάρσεις σε μικρό παράθυρο	Spikes	Χρονοσειρά, φίλτρα, drill down	ts, event_id, score, raw
Σπάνια γεγονότα	Rare events	Top anomalies, ταξινόμηση, φίλτρα	template, component, score
Μη αναμενόμενες ακολουθίες	Sequence anomalies	Ομαδοποίηση, προβολή ακολουθιών	group id, sequences, scores

Πίνακας 3.5. Αντιστοίχιση σεναρίων χρήσης με λειτουργίες και δεδομένα

Κεφάλαιο 4 Υλοποίηση (Implementation)

Το παρόν κεφάλαιο τεκμηριώνει την πρακτική υλοποίηση του αγωγού ανίχνευσης ανωμαλιών σε αρχεία καταγραφής, με στόχο να αποδειχθεί ότι το σύστημα λειτουργεί αναπαραγώγιμα από άκρο σε άκρο. Η υλοποίηση οργανώνεται σε διαδοχικά στάδια: παραγωγή ή εισαγωγή αρχείων καταγραφής, ροή σε πραγματικό χρόνο, προεπεξεργασία, ανίχνευση ανωμαλιών, αποθήκευση αποτελεσμάτων σε βάση δεδομένων και απεικόνιση μέσω πίνακα ελέγχου. Για κάθε στάδιο παρατίθεται η εντολή εκτέλεσης, η αναμενόμενη έξοδος και αντιπροσωπευτικό απόσπασμα κώδικα, ώστε η εργασία να είναι αξιολογήσιμη και επαναλαμβανόμενη.

Οι επιλογές υποδομής αποσκοπούν στη μείωση της πολυπλοκότητας και στην ενίσχυση της φορητότητας. Η Python χρησιμοποιείται ως κύρια γλώσσα ενορχήστρωσης, η SQLite ως ελαφρά βάση δεδομένων που δεν απαιτεί εξωτερικό server και το Streamlit ως πλαίσιο ταχείας ανάπτυξης διαδραστικής οπτικοποίησης. Η αρχιτεκτονική αυτή επιτρέπει την εκτέλεση ολόκληρου του pipeline σε φορητό υπολογιστή χωρίς εξωτερικές εξαρτήσεις, διατηρώντας παράλληλα καθαρή διεπαφή μεταξύ των σταδίων, ώστε κάθε στάδιο να μπορεί να αντικατασταθεί ανεξάρτητα σε μελλοντική επέκταση.

4.1 Log generator (μορφή logs, normal vs anomaly σενάρια)

Η υλοποίηση ξεκινά με τη δημιουργία εισόδου ροής από το δημόσιο σύνολο δεδομένων HDFS του LogHub. Το σύνολο αυτό περιλαμβάνει πραγματικές γραμμές καταγραφής ενός κατανεμημένου συστήματος αρχείων Hadoop, οι οποίες χρησιμοποιούνται ως βάση για τη δημιουργία ρεαλιστικής εισόδου. Η επιλογή πραγματικών δεδομένων αντί εντελώς συνθετικής παραγωγής κρίθηκε σκόπιμη, καθώς διατηρεί την πολυπλοκότητα και τη μορφή μηνυμάτων που απαντώνται στην πράξη.

Πάνω στις πραγματικές γραμμές εισάγονται ελεγχόμενες ανωμαλίες τριών ειδών. Το πρώτο είδος αφορά spikes, δηλαδή επανάληψη παρόμοιων γραμμών σε μικρό χρονικό παράθυρο, που αναπαριστά ασυνήθιστη αύξηση συχνότητας ενός μηνύματος. Το δεύτερο είδος αφορά σπάνια γεγονότα, όπου εισάγονται τεχνητά ύποπτα μηνύματα πρόσβασης που δεν εμφανίζονται στη φυσιολογική λειτουργία. Το τρίτο είδος αφορά ανωμαλίες ακολουθίας, όπου τα βήματα μιας διεργασίας εμφανίζονται με μη αναμενόμενη σειρά, όπως η ολοκλήρωση πριν από την έναρξη.

Η εισαγωγή γίνεται με ελεγχόμενο τρόπο, ώστε να παράγεται ταυτόχρονα αρχείο ground truth για μελλοντική ποσοτική αξιολόγηση.

Η εκτέλεση του generator παράγει δύο αρχεία εξόδου. Το πρώτο, stream_input.log, αποτελεί την είσοδο ροής που θα χρησιμοποιηθεί στο επόμενο στάδιο. Το δεύτερο, ground_truth.csv, καταγράφει για κάθε γραμμή αν αποτελεί φυσιολογικό γεγονός ή κάποιο από τα εισηγμένα είδη ανωμαλίας, μαζί με αναγνωριστικό γραμμής για αντιστοίχιση.

Εντολή εκτέλεσης:

python .\src\log_generator.py --data_dir "%USERPROFILE%\Desktop\data\HDFS"

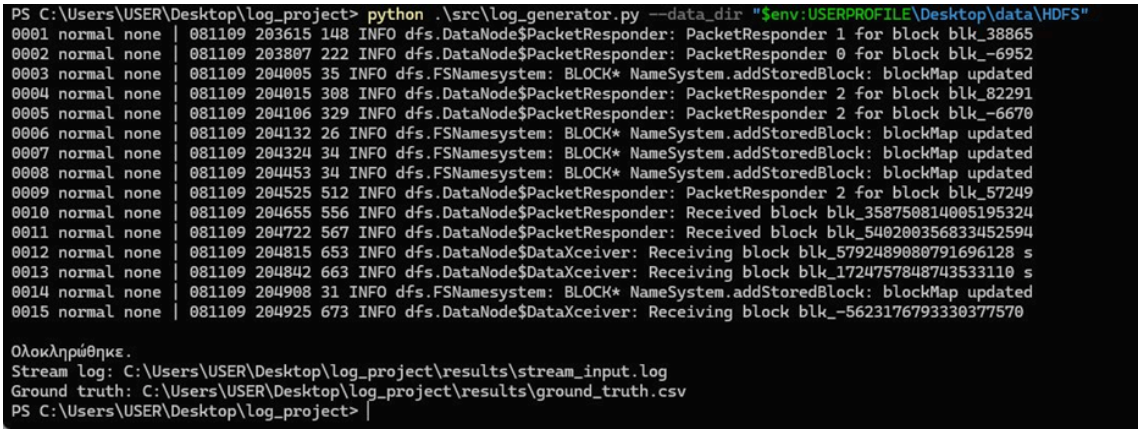
4.2 Real-time ingestion (πώς ρέουν τα logs)

Το ενδεικτικό απόσπασμα κώδικα που ακολουθεί αποτυπώνει τις δύο βασικές συναρτήσεις εισαγωγής ανωμαλιών. Η inject_spike επιλέγει τυχαία μια γραμμή και την επαναλαμβάνει τόσες φορές όσο ορίζει η παράμετρος spike_size, δημιουργώντας έτσι ένα cluster ομοίων μηνυμάτων σε μικρό παράθυρο. Η inject_sequence_anomaly παράγει τρεις γραμμές με βήματα μιας υποτιθέμενης διεργασίας σε ανεστραμμένη σειρά, όπου το βήμα ολοκλήρωσης προηγείται της έναρξης.

```
def inject_spike(lines, spike_size=50):
```

```
    base = random.choice(lines)
    return [base for _ in range(spike_size)]


def inject_sequence_anomaly():
    return [
        f"{utc_ts()} SEQ step=finish status=ok",
        f"{utc_ts()} SEQ step=start  status=ok",
        f"{utc_ts()} SEQ step=validate status=ok",
    ]
```



Εικόνα 4.1: Εκτέλεση log generator και παραγωγή αρχείων stream_input.log και ground_truth.csv.

Όπως φαίνεται στην Εικόνα 4.1, η εκτέλεση του generator διαβάζει τα αρχεία HDFS από τον καθορισμένο κατάλογο, παράγει τις πρώτες γραμμές της εξόδου στην κονσόλα και ολοκληρώνεται με επιβεβαίωση των διαδρομών των παραγόμενων αρχείων. Κάθε γραμμή εξόδου φέρει αύξοντα αριθμό, ετικέτα τύπου (normal ή ανωμαλία), τύπο ανωμαλίας εάν υπάρχει και το κανονικοποιημένο μήνυμα HDFS.

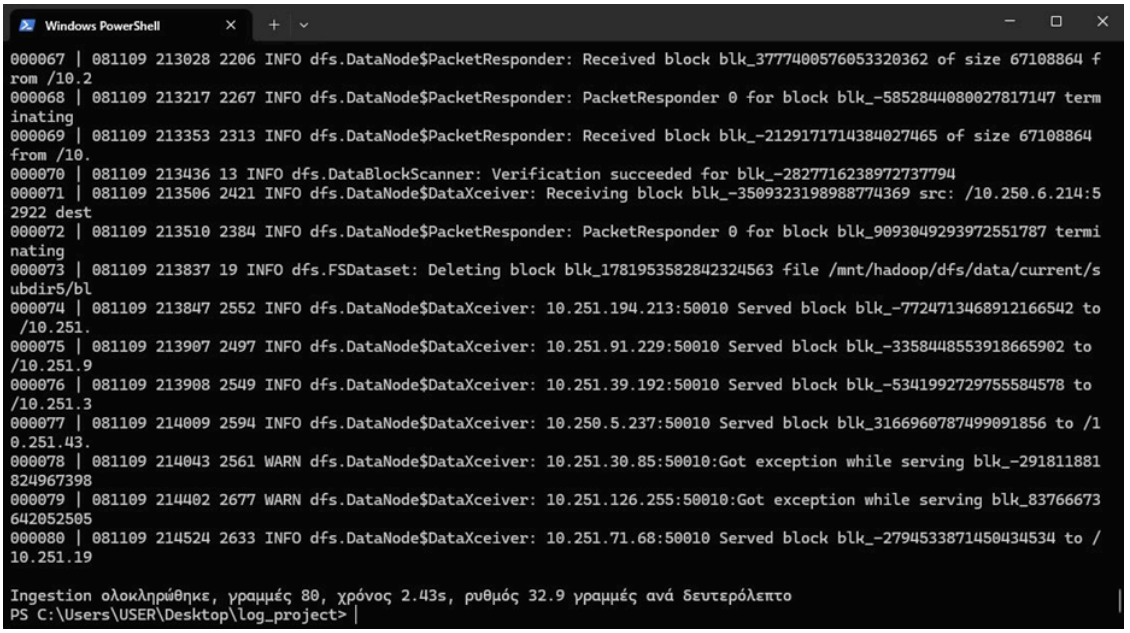
4.2 Real-time ingestion: τρόπος ροής των αρχείων καταγραφής

Το στάδιο ingestion αναλαμβάνει τη σταδιακή ανάγνωση του αρχείου stream_input.log με τεχνητό ρυθμό, ώστε η είσοδος να προσεγγίζει ροή πραγματικού χρόνου. Η προσέγγιση αυτή αποτελεί ελαφρύ υποκατάστατο ενός streaming bus, όπως θα ήταν ένα σύστημα τύπου Apache Kafka, και επιλέχθηκε για να διατηρηθεί η απλότητα της υποδομής στο πλαίσιο της διπλωματικής εργασίας, χωρίς να θυσιαστεί η ρεαλιστικότητα της ροής.

Η υλοποίηση βασίζεται στη σταδιακή ανάγνωση γραμμών με καθορισμένη καθυστέρηση μεταξύ τους, που ελέγχεται από την παράμετρο sleep_ms. Παράλληλα, καταγράφεται ο ρυθμός εισόδου σε γραμμές ανά δευτερόλεπτο, ένα μέγεθος που χρησιμεύει αργότερα στη σύγκριση με το χρόνο ανίχνευσης. Η αρχιτεκτονική διατηρεί καθαρή διεπαφή προς τα επόμενα στάδια, καθώς η έξοδος του ingestion είναι απλώς μια ακολουθία γραμμών κειμένου που μπορούν να διαβαστούν από οποιαδήποτε επόμενη συνιστώσα.

Εντολή εκτέλεσης:

python .\src\tail.py --max_lines 80 --sleep_ms 30



Εικόνα 4.2: Σταδιακή ροή αρχείων καταγραφής και μέτρηση ρυθμού εισόδου.

Η Εικόνα 4.2 παρουσιάζει την έξοδο της εντολής κατά την ανάγνωση 80 γραμμών. Στο κάτω μέρος της οθόνης αποτυπώνεται το συνολικό αποτέλεσμα: 80 γραμμές, χρόνος εκτέλεσης 2,43 δευτερόλεπτα και ρυθμός 32,9 γραμμών ανά δευτερόλεπτο. Ο ρυθμός αυτός επαρκεί για να αναδείξει τη δυναμική του pipeline σε συνθήκες παρόμοιες με πραγματική ροή.

4.3 Preprocessing (parsing + partitioning + feature extraction)

Το στάδιο preprocessing αναλαμβάνει τον μετασχηματισμό των αδόμητων γραμμών κειμένου σε δομημένα πεδία και μετρήσιμα χαρακτηριστικά που μπορούν να αποθηκευτούν και να επεξεργαστούν αλγοριθμικά. Η ανάγκη για αυτό το στάδιο πηγάζει από τη φύση των αρχείων καταγραφής, τα οποία παράγονται σε ελεύθερης μορφής κείμενο και απαιτούν εξαγωγή δομής για να γίνουν αξιοποιήσιμα.

Το parsing εφαρμόζει κανονική έκφραση που αναγνωρίζει τη μορφή γραμμών HDFS και εξάγει τα πεδία date, time, pid, level, component και content. Η επιλογή της κανονικής έκφρασης έναντι ενός αλγορίθμου αυτόματης εξαγωγής προτύπων, όπως ο Drain, επιτρέπει μεγαλύτερη ακρίβεια για τη γνωστή μορφή HDFS, με ελάχιστο υπολογιστικό κόστος.

Ενδεικτικό απόσπασμα parsing:

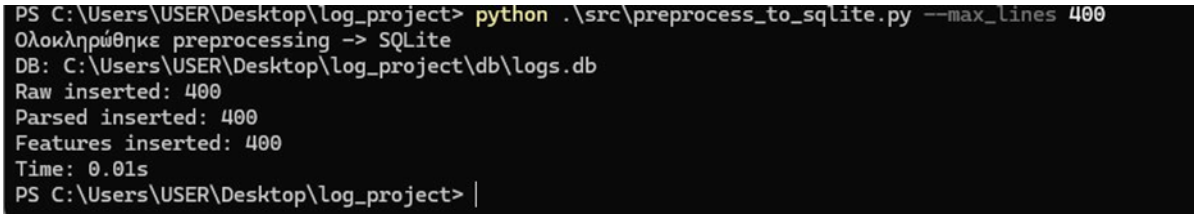
```
HDFS_RX = re.compile(
    r'^(?P<date>\S+)\s+(?P<time>\S+)\s+(?P<pid>\S+)\s+'
    r'(?P<level>[A-Z]+)\s+(?P<component>[^:]+):\s+(?P<content>.*)$'
)
```

Η εξαγωγή χαρακτηριστικών δημιουργεί επιπλέον μετρήσιμες μεταβλητές από το αναλυμένο μήνυμα. Συγκεκριμένα, υπολογίζεται το μήκος του μηνύματος ως αριθμητική μεταβλητή, ενώ ορίζονται δυαδικές σημαίες για την παρουσία λέξεων υποψίας, όπως warning και exception. Τα χαρακτηριστικά αυτά χρησιμεύουν τόσο στους αλγορίθμους ανίχνευσης που ακολουθούν, όσο και στη βάση δεδομένων για φιλτράρισμα και ομαδοποίηση.

Τα αποτελέσματα του parsing και της εξαγωγής χαρακτηριστικών εγγράφονται σε τρεις πίνακες SQLite: logs_raw για τη μη επεξεργασμένη γραμμή κειμένου, logs_parsed για τα αναλυμένα πεδία και features για τα εξαγόμενα χαρακτηριστικά. Η αναλογική δομή τριών πινάκων επιτρέπει ανεξάρτητη πρόσβαση σε κάθε επίπεδο επεξεργασίας, διευκολύνοντας παράλληλα την εκ νέου εκτέλεση μεμονωμένων σταδίων χωρίς αλλαγή στα υπόλοιπα.

Εντολή εκτέλεσης:

python .\src\preprocess_to_sqlite.py --max_lines 400



Εικόνα 4.3: Preprocessing 400 γραμμών και εγγραφή σε SQLite.

Η Εικόνα 4.3 επιβεβαιώνει την ολοκλήρωση του preprocessing για 400 γραμμές. Η εγγραφή και στους τρεις πίνακες ολοκληρώθηκε σε 0,01 δευτερόλεπτα, γεγονός που δείχνει ότι η επεξεργασία είναι αμελητέα σε σύγκριση με το χρόνο ανίχνευσης και είναι επαρκής για επεξεργασία δεδομένων σε πραγματικό χρόνο.

4.4 Ενσωμάτωση αλγορίθμων anomaly detection (plug-in αρχιτεκτονική)

Οι αλγόριθμοι ανίχνευσης ενσωματώνονται με αρχιτεκτονική plug-in, βάσει της οποίας κάθε detector υλοποιεί κοινή αφηρημένη διεπαφή και επιστρέφει δύο τιμές: ένα βαθμό ανωμαλίας στο διάστημα [0,1] και μια ετικέτα normal ή anomaly. Η αρχιτεκτονική αυτή επιτρέπει την εναλλαγή detectors χωρίς καμία αλλαγή στα προηγούμενα ή επόμενα στάδια, δηλαδή χωρίς επέμβαση στο parsing, τη βάση δεδομένων ή το dashboard. Η συγκεκριμένη επιλογή υλοποιήθηκε ώστε να τηρείται η αρχή ανοικτής-κλειστής σχεδίασης, όπου το σύστημα είναι ανοικτό για επέκταση και κλειστό για τροποποίηση.

Στην παρούσα υλοποίηση ενσωματώθηκαν δύο detectors με διαφορετική λογική απόφασης. Ο πρώτος είναι ο SimpleThresholdDetector, ένας κανονιστικός ανιχνευτής βάσει ορίων, ο οποίος υπολογίζει βαθμό ανωμαλίας ως σταθμισμένο άθροισμα τριών χαρακτηριστικών: παρουσία warning, παρουσία exception και κανονικοποιημένο μήκος μηνύματος. Ο δεύτερος είναι ο RareComponentDetector, ανιχνευτής σπανιότητας, ο οποίος χαρακτηρίζει ως ανώμαλο κάθε μήνυμα που προέρχεται από component με χαμηλή συχνότητα εμφάνισης στο σύνολο της εισόδου.

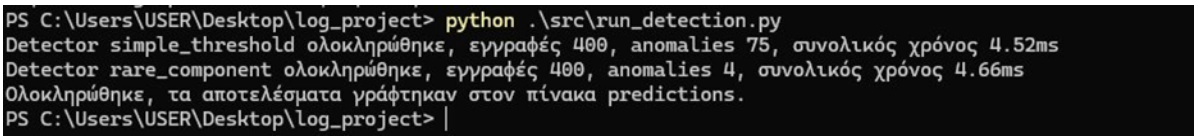
Ενδεικτικό απόσπασμα SimpleThresholdDetector:

```
class SimpleThresholdDetector(Detector):
    name = "simple_threshold"
    def predict(self, parsed_row, feature_row):
```

```
    score = clamp01(
        0.55 * int(feature_row.get("has_warn", 0))
      + 0.60 * int(feature_row.get("has_exception", 0))
      + 0.25 * clamp01(int(feature_row.get("msg_len", 0)) / 160)
    )
    label = "anomaly" if score >= 0.65 else "normal"
    return DetectionResult(score=score, label=label)
```

Εντολή εκτέλεσης:

python .\src\run_detection.py



Εικόνα 4.4: Εκτέλεση των δύο detectors και εγγραφή αποτελεσμάτων στον πίνακα predictions.

Η Εικόνα 4.4 αποτυπώνει την έξοδο της εκτέλεσης των δύο detectors στις 400 γραμμές της βάσης. Ο SimpleThresholdDetector εντόπισε 75 ανωμαλίες σε συνολικό χρόνο 4,52 ms, ενώ ο RareComponentDetector εντόπισε 4 ανωμαλίες σε 4,66 ms. Το σύνολο των αποτελεσμάτων εγγράφηκε στον πίνακα predictions της SQLite. Η διαφορά στο πλήθος ανωμαλιών μεταξύ των δύο detectors αντικατοπτρίζει τη διαφορετική λογική απόφασης κάθε αλγορίθμου, γεγονός που ενισχύει τη χρησιμότητα της συγκριτικής μελέτης τους.

4.5 Αποθήκευση και indexing

Η αποθήκευση του pipeline οργανώνεται σε SQLite με τέσσερις κύριους πίνακες. Ο πίνακας logs_raw αποθηκεύει τη μη επεξεργασμένη γραμμή κειμένου για δυνατότητα drill-down. Ο πίνακας logs_parsed αποθηκεύει τα αναλυμένα πεδία που εξήχθησαν κατά το parsing. Ο πίνακας features αποθηκεύει τα εξαγόμενα χαρακτηριστικά. Ο πίνακας predictions αποθηκεύει τα αποτελέσματα κάθε detector ανά γραμμή, συμπεριλαμβανομένων του

ονόματος του μοντέλου, του βαθμού ανωμαλίας, της ετικέτας, του χρόνου ανίχνευσης σε milliseconds και της χρονικής σήμανσης εγγραφής.

Δημιουργούνται indexes στα πεδία level και component του πίνακα logs_parsed, ώστε να υποστηρίζονται αποδοτικά ερωτήματα ομαδοποίησης και ταξινόμησης. Η παρουσία των indexes καθίσταται ιδιαίτερα σημαντική για τα ερωτήματα του dashboard, τα οποία φιλτράρουν ταυτόχρονα κατά level, component και label.

Ο έλεγχος της βάσης πραγματοποιείται με αυτόνομο script που εμφανίζει τη λίστα πινάκων, τα πλήθη εγγραφών και τα indexes που έχουν δημιουργηθεί, επιτρέποντας γρήγορη επαλήθευση της ακεραιότητας της βάσης μετά από κάθε στάδιο.

Εντολή εκτέλεσης:

python .\src\db_check.py

Κατά την αρχική εκτέλεση από λανθασμένο κατάλογο παρατηρήθηκε σφάλμα διαδρομής αρχείου. Η Εικόνα 4.5α αποτυπώνει την εκτέλεση από λανθασμό κατάλογο και το αντίστοιχο μήνυμα σφάλματος, ενώ η Εικόνα 4.5β παρουσιάζει την επιτυχή εκτέλεση από τον σωστό κατάλογο με πλήρη έξοδο.

```
PS C:\Users\USER\Desktop\log_project\db> python .\src\db_check.py
C:\Users\USER\AppData\Local\Programs\Python\Python312\python.exe: can't open file 'C:\\Users\\USER\\Desktop\\log_project
\\db\\src\\db_check.py': [Errno 2] No such file or directory
PS C:\Users\USER\Desktop\log_project\db> python ..\src\db_check.py
DB: C:\Users\USER\Desktop\log_project\db\logs.db

Tables: ['features', 'logs_parsed', 'logs_raw', 'sqlite_sequence']

Total logs_raw: 400
Total logs_parsed: 400

Counts by level:
INFO 325
WARN 75

Indexes on logs_parsed:
(0, 'idx_logs_parsed_component', 0, 'c', 0)
(1, 'idx_logs_parsed_level', 0, 'c', 0)
```

Εικόνα 4.5α: Αποτυχημένη εκτέλεση db_check από λανθασμένο κατάλογο

```
PS C:\Users\USER\Desktop\log_project\db> cd ..
PS C:\Users\USER\Desktop\log_project> python .\src\db_check.py
DB: C:\Users\USER\Desktop\log_project\db\logs.db

Tables: ['features', 'logs_parsed', 'logs_raw', 'sqlite_sequence']

Total logs_raw: 400
Total logs_parsed: 400

Counts by level:
INFO 325
WARN 75

Indexes on logs_parsed:
(0, 'idx_logs_parsed_component', 0, 'c', 0)
(1, 'idx_logs_parsed_level', 0, 'c', 0)
PS C:\Users\USER\Desktop\log_project>
```

Εικόνα 4.5β: Επιτυχής εκτέλεση db_check με εμφάνιση πινάκων, πληθών και indexes.

Όπως φαίνεται στην Εικόνα 4.5β, η βάση δεδομένων logs.db περιέχει τους πίνακες features, logs_parsed, logs_raw και sqlite_sequence. Ο πίνακας logs_raw και ο logs_parsed περιέχουν 400 εγγραφές ο καθένας. Η κατανομή ανά επίπεδο καταγραφής δείχνει 325 εγγραφές επιπέδου INFO και 75 εγγραφές επιπέδου WARN, αναλογία που αντικατοπτρίζει τη φυσιολογική διανομή του πρωτότυπου συνόλου δεδομένων HDFS. Τα indexes idx_logs_parsed_component και idx_logs_parsed_level επιβεβαιώνουν την ύπαρξη ευρετηρίων στα κύρια πεδία φιλτραρίσματος.

4.6 Dashboard (screens, filters, alerts, drill-down)

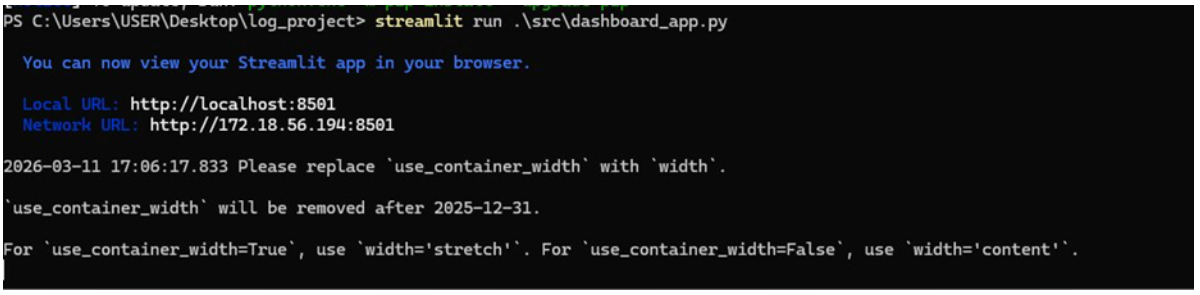
Το dashboard υλοποιείται σε Streamlit και ανακτά δεδομένα από τη SQLite μέσω συνένωσης (join) του πίνακα predictions με τους logs_parsed και logs_raw. Η αρχιτεκτονική ανάγνωσης από κοινή βάση εξασφαλίζει ότι το dashboard δεν εξαρτάται από συγκεκριμένη ακολουθία εκτέλεσης των προηγούμενων σταδίων, αλλά ανακτά πάντα τα τελευταία διαθέσιμα δεδομένα απευθείας από τη βάση.

Η διεπαφή παρέχει φίλτρα ανά μοντέλο ανίχνευσης, ετικέτα κατηγοριοποίησης (normal ή anomaly), επίπεδο καταγραφής (level) και component. Τα φίλτρα εφαρμόζονται δυναμικά και ενημερώνουν ταυτόχρονα όλα τα τμήματα της οθόνης. Στο τμήμα σύνοψης εμφανίζονται τρεις βασικές μετρικές: ο συνολικός αριθμός εγγραφών που ταιριάζουν στα ενεργά φίλτρα, το πλήθος των ανωμαλιών και ο μέσος χρόνος ανίχνευσης σε milliseconds.

Στο κεντρικό τμήμα εμφανίζεται γράφημα κατανομής των ανωμαλιών ανά component, το οποίο επιτρέπει ταχεία αναγνώριση ποιο τμήμα του συστήματος παράγει τα περισσότερα ύποπτα μηνύματα. Η οπτικοποίηση αυτή είναι ιδιαίτερα χρήσιμη για τη σύγκριση της συμπεριφοράς των δύο detectors, καθώς μπορεί να εντοπίσει διαφορές στο ποιους τύπους component χαρακτηρίζει ανώμαλους ο κάθε αλγόριθμος.

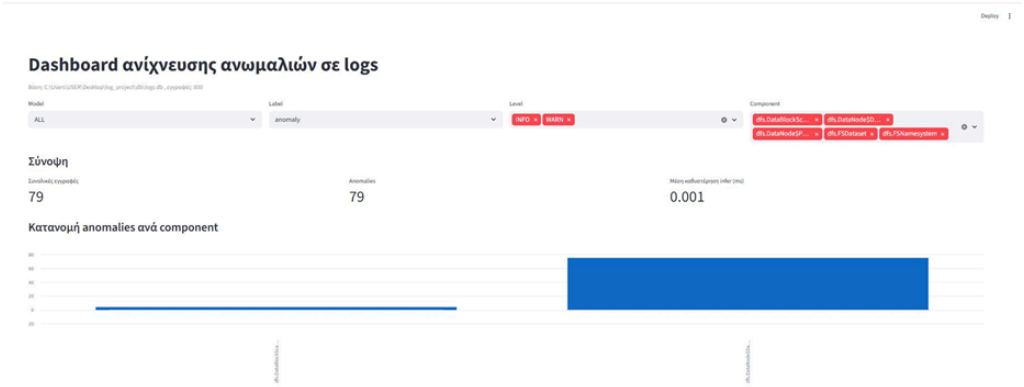
Εντολή εκτέλεσης:

streamlit run .\src\dashboard_app.py



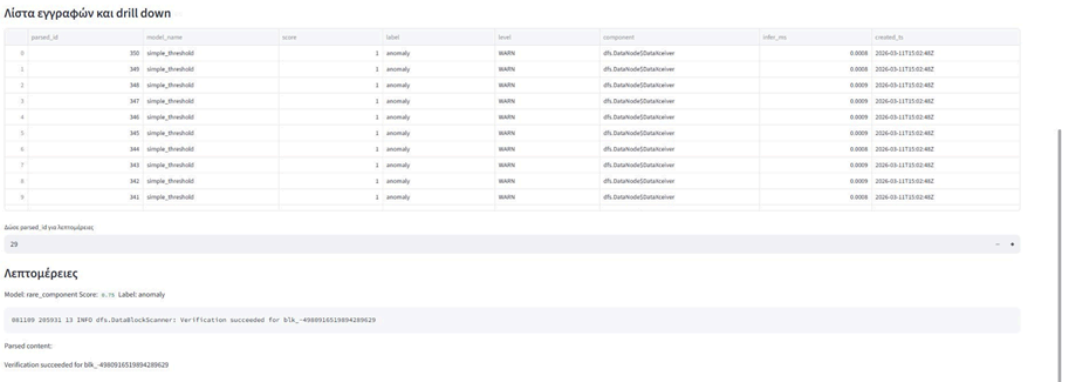
Εικόνα 4.6: Εκκίνηση Streamlit και διαδρομές πρόσβασης στο dashboard.

Η Εικόνα 4.6 αποτυπώνει την επιτυχή εκκίνηση του Streamlit server. Η εφαρμογή είναι προσβάσιμη τόσο στο τοπικό δίκτυο μέσω της διεύθυνσης localhost:8501, όσο και στο δίκτυο μέσω της διεύθυνσης δικτύου 172.18.56.194:8501. Η εντολή εκτέλεσης δεν απαιτεί παραμέτρους database, καθώς η διαδρομή της βάσης ορίζεται εσωτερικά από το config.env.



Εικόνα 4.7: Οθόνη dashboard με ενεργά φίλτρα και γράφημα κατανομής ανωμαλιών ανά component.

Η Εικόνα 4.7 παρουσιάζει το dashboard με ενεργά φίλτρα ετικέτας anomaly και επιλεγμένα και τα δύο επίπεδα INFO και WARN, καθώς και όλα τα components. Η σύνοψη εμφανίζει 79 συνολικές εγγραφές, 79 ανωμαλίες και μέσο χρόνο ανίχνευσης 0,001 ms. Στο γράφημα κατανομής φαίνεται ότι το component dfs.DataNode$DataXceiver συγκεντρώνει τη μεγάλη πλειοψηφία των ανωμαλιών, ενώ τα υπόλοιπα components παρουσιάζουν σημαντικά μικρότερη συχνότητα.



Εικόνα 4.8: Λίστα εγγραφών, drill-down και προβολή raw log.

Η Εικόνα 4.8 παρουσιάζει τη λειτουργία drill-down. Στο πάνω τμήμα εμφανίζεται η λίστα εγγραφών με τα πεδία parsed_id, model_name, score, label, level, component, infer_ms και created_ts. Η λίστα επιτρέπει εποπτεία της απόφασης κάθε detector ανά εγγραφή. Στο κάτω τμήμα, μέσω εισαγωγής αριθμού parsed_id (στο παράδειγμα 29), εμφανίζονται οι λεπτομέρειες για την εγγραφή αυτή: το μοντέλο rare_component, ο βαθμός ανωμαλίας 0,75, η ετικέτα anomaly, η αρχική raw γραμμή καταγραφής σε μορφή HDFS και το αναλυμένο περιεχόμενο Verification succeeded for blk_-4980916519894289629. Η λειτουργία drill-down επιτρέπει στον αναλυτή να μεταβεί από τη συνολική εικόνα στο επίπεδο μεμονωμένης εγγραφής, αντιστοιχώντας κάθε απόφαση ανίχνευσης με το πρωτογενές μήνυμα.

4.7 Θέματα deployment (Docker, configs, reproducibility)

Η αναπαραγωγιμότητα του pipeline υποστηρίζεται με δύο συμπληρωματικά μέτρα. Πρώτον, με σαφή δομή φακέλων που διαχωρίζει τον πηγαίο κώδικα στον κατάλογο src, τα δεδομένα εισόδου στον κατάλογο data, τη βάση δεδομένων στον κατάλογο db και τα αποτελέσματα στον κατάλογο results. Δεύτερον, με καταγραφή όλων των εξαρτήσεων Python σε αρχείο requirements.txt, ώστε το περιβάλλον να μπορεί να αναδημιουργηθεί σε νέο σύστημα με μία μόνο εντολή εγκατάστασης.

Παρέχεται επίσης αρχείο config.env με τις βασικές διαδρομές του συστήματος, επιτρέποντας την προσαρμογή χωρίς τροποποίηση του κώδικα. Η παράμετροποίηση αυτή είναι ιδιαίτερα

χρήσιμη κατά την εκτέλεση σε διαφορετικά λειτουργικά συστήματα ή διαφορετικές διαδρομές φακέλων χρήστη.

Ως αντικείμενο deployment παρέχεται Dockerfile που ορίζει περιβάλλον Python 3.12 slim, αντιγράφει τις εξαρτήσεις, εγκαθιστά τα πακέτα, αντιγράφει τον κώδικα και τα δεδομένα, εκθέτει τη θύρα 8501 και ορίζει ως εντολή εκκίνησης το Streamlit dashboard. Η χρήση Docker εξασφαλίζει απομόνωση περιβάλλοντος και δυνατότητα ανάπτυξης σε οποιαδήποτε υποδομή που υποστηρίζει containers.

Εντολή καταγραφής εξαρτήσεων:

pip freeze > requirements.txt

Ενδεικτικό Dockerfile:

```
FROM python:3.12-slim

WORKDIR /app

COPY requirements.txt /app/requirements.txt

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

COPY src /app/src

COPY db  /app/db

COPY results /app/results

EXPOSE 8501

CMD ["streamlit", "run", "src/dashboard_app.py",

    "--server.address=0.0.0.0", "--server.port=8501"]
```

Η δομή αυτή εξασφαλίζει ότι ο ερευνητής ή ο αξιολογητής μπορεί να αναπαράγει ολόκληρο το pipeline με τα ακόλουθα βήματα: κλωνοποίηση αποθετηρίου, εγκατάσταση εξαρτήσεων από το requirements.txt, εκτέλεση των σεναρίων παραγωγής δεδομένων και εκκίνηση του dashboard. Εναλλακτικά, η κατασκευή και εκτέλεση του Docker image αρκεί για πλήρη λειτουργία χωρίς καμία τοπική εγκατάσταση εξαρτήσεων.

Κεφάλαιο 5 Πειράματα και αποτελέσματα

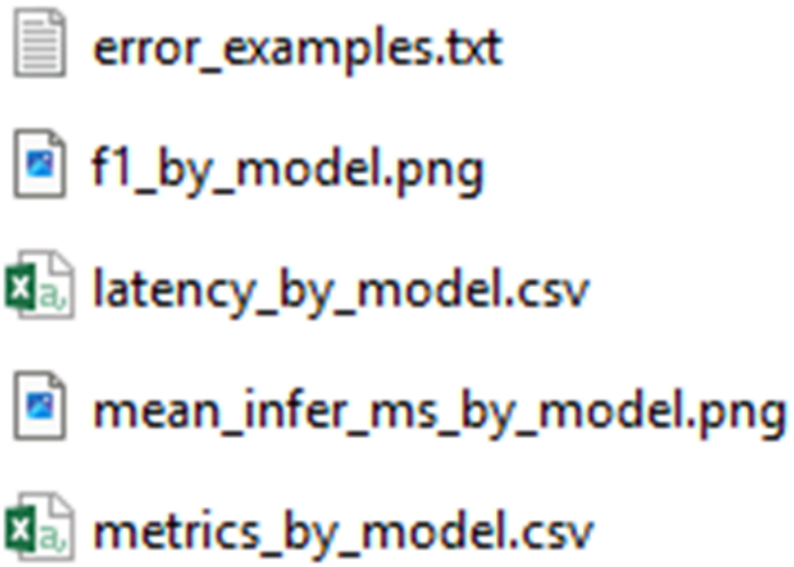
Το κεφάλαιο παρουσιάζει την πειραματική διαδικασία και τα αποτελέσματα αξιολόγησης του υλοποιημένου pipeline ανίχνευσης ανωμαλιών σε αρχεία καταγραφής. Η αξιολόγηση οργανώνεται ώστε να συνδέει τη λειτουργική υλοποίηση με μετρήσιμες επιδόσεις, δηλαδή την αποτελεσματικότητα ταξινόμησης ανωμαλιών και την υπολογιστική συμπεριφορά σε συνθήκες ροής. Η ανάλυση ακολουθεί τη λογική ότι η ανίχνευση σε logs αποτελεί πρόβλημα που συνδυάζει σήμα και θόρυβο, όπου ο σχεδιασμός του pipeline και η επιλογή αλγορίθμου επηρεάζουν τόσο τις μετρικές ακρίβειας όσο και την καθυστέρηση λήψης απόφασης (Kent & Souppaya, 2006). Για τον λόγο αυτό παρουσιάζονται αναλυτικά η πειραματική διαδικασία, τα αποτελέσματα ανά αλγόριθμο, οι χρόνοι και ο ρυθμός επεξεργασίας, η σύγκριση trade offs και η ερμηνεία σφαλμάτων.

5.1 Πειραματική διαδικασία

Η πειραματική διαδικασία βασίστηκε σε συνδυασμό πραγματικών logs και ελεγχόμενων σεναρίων ανωμαλιών. Ως βάση δεδομένων χρησιμοποιήθηκαν γραμμές από το dataset HDFS του LogHub, το οποίο αποτελεί τυπικό παράδειγμα συστημικών logs και έχει αξιοποιηθεί εκτεταμένα στη βιβλιογραφία log analytics ως δοκιμαστικό σύνολο δεδομένων (Zhu et al., 2020). Από το πρωτογενές αρχείο επιλέχθηκε υποσύνολο γραμμών, πάνω στο οποίο δημιουργήθηκαν συνθετικές ανωμαλίες με στόχο να προκύψει σαφής ground truth, ώστε να καταστεί δυνατή η ποσοτική αξιολόγηση. Η προσέγγιση αυτή είναι συνεπής με την πρακτική της αξιολόγησης ανιχνευτών, όπου απαιτείται αντιστοίχιση προβλέψεων με ετικέτες αναφοράς και αποτύπωση σφαλμάτων πρώτου και δεύτερου είδους (Saito & Rehmsmeier, 2015).

Η παραγωγή δεδομένων εισόδου πραγματοποιήθηκε με log generator, ο οποίος δημιούργησε ένα αρχείο ροής stream_input.log και ένα συνοδευτικό αρχείο ground_truth.csv. Το ground truth χαρακτήριζε κάθε γραμμή ως normal ή anomaly και απέδιδε τύπο ανωμαλίας, επιτρέποντας μεταγενέστερη ανάλυση ανά σενάριο. Στη συνέχεια εφαρμόστηκε ingestion με pacing, ώστε η ανάγνωση να προσεγγίζει ροή πραγματικού χρόνου και να προκύψει μετρήσιμο throughput.

Ακολούθησε preprocessing με parsing σε βασικά πεδία και εξαγωγή απλών χαρακτηριστικών, ενώ όλα τα στάδια αποθήκευσαν αποτελέσματα σε SQLite. Η χρήση ενιαίας τοπικής βάσης διευκολύνει την αναπαραγωγή και την ιχνηλασιμότητα, επειδή οι εγγραφές raw, parsed, features και predictions συνδέονται με σταθερά κλειδιά και είναι δυνατή η άμεση εξαγωγή αποτελεσμάτων για αξιολόγηση (Kleppmann, 2017).



Εικόνα 5.1. Παραγόμενα αρχεία αξιολόγησης στο results\ch5, πηγή, στιγμιότυπο οθόνης, επεξεργασία συγγραφέα.

Η διαδικασία αξιολόγησης υλοποιήθηκε με script που διαβάζει το ground_truth.csv και αντιστοιχίζει τις ετικέτες αναφοράς με τις προβλέψεις στον πίνακα predictions. Η αντιστοίχιση υλοποιήθηκε μέσω της σειράς των parsed εγγραφών, ώστε το line_id του ground truth να αντιστοιχεί σε συγκεκριμένο parsed_id. Με αυτόν τον τρόπο δημιουργήθηκαν πίνακες αποτελεσμάτων ανά αλγόριθμο, καθώς και γραφήματα σύγκρισης F1 και χρόνων inference. Η επιλογή των μετρικών και η παρουσίαση με πίνακες και γραφήματα ακολουθεί καθιερωμένη πρακτική στην αξιολόγηση συστημάτων ανίχνευσης, όπου η σύγκριση οφείλει να είναι ταυτόχρονα ποσοτική και ερμηνεύσιμη (Powers, 2011).

5.2 Αποτελέσματα ανά αλγόριθμο

Η αξιολόγηση πραγματοποιήθηκε σε δύο detectors που ενσωματώθηκαν ως plug in στο pipeline. Ο πρώτος detector simple_threshold εφαρμόζει κανονιστική λογική που συνδυάζει ενδείξεις warning, εμφάνιση exception και μήκος μηνύματος σε μία βαθμολογία score, με απόφαση anomaly όταν το score υπερβαίνει όριο. Ο δεύτερος detector rare_component αξιοποιεί τη σπανιότητα του component στο τρέχον δείγμα, με στόχο να αναγνωρίσει σπάνια υποσυστήματα ως δυνητικά ύποπτα. Οι δύο επιλογές αντιπροσωπεύουν διαφορετικές οικογένειες προσέγγισης, καθώς ο πρώτος στηρίζεται σε επιφανειακά χαρακτηριστικά του μηνύματος, ενώ ο δεύτερος στηρίζεται σε στατιστική ιδιότητα κατανομής (He et al., 2020).

```
Αποτελέσματα (Precision/Recall/F1) ανά αλγόριθμο
rare_component | TP 0 FP 4 FN 50 TN 346 | P 0.0 R 0.0 F1 0.0
simple_threshold | TP 50 FP 25 FN 0 TN 325 | P 0.6667 R 1.0 F1 0.8
```

Εικόνα 5.2. Αποτελέσματα Precision, Recall και F1 ανά αλγόριθμο, πηγή, στιγμιότυπο εκτέλεσης, επεξεργασία συγγραφέα.

Αλγόριθμος	TP	FP	FN	TN	Precision	Recall	F1
simple_threshold	50	25	0	325	0.6667	1.0000	0.8000
rare_component	0	4	50	346	0.0000	0.0000	0.0000

Πίνακας 5.1. Πίνακας σύγχυσης και μετρικές ταξινόμησης ανά αλγόριθμο

Τα αποτελέσματα δείχνουν σαφή διαφοροποίηση συμπεριφοράς μεταξύ των δύο detectors. Ο simple_threshold πέτυχε TP 50 και FN 0, γεγονός που αντιστοιχεί σε Recall ίσο με 1,0, δηλαδή πλήρη κάλυψη των ανωμαλιών του ground truth. Ωστόσο παρουσίασε FP 25, με αποτέλεσμα Precision 0,6667, στοιχείο που ερμηνεύεται ως τάση υπερανίχνευσης, δηλαδή ενεργοποίηση alarm σε περιπτώσεις που το ground truth χαρακτηρίζει ως normal. Η συνολική ισορροπία αποτυπώνεται σε F1 0,8, που αποτελεί ικανοποιητικό αποτέλεσμα για κανονιστικό detector σε συνθήκες απλών χαρακτηριστικών, δεδομένου ότι το F1 τιμωρεί ταυτόχρονα ψευδώς θετικά και ψευδώς αρνητικά (Powers, 2011).

Αντιθέτως, ο rare_component παρουσίασε TP 0 και FN 50, που οδηγούν σε Recall 0,0 και F1 0,0. Η εικόνα αυτή δείχνει ότι η υπόθεση εργασίας του detector, δηλαδή ότι οι ανωμαλίες αντιστοιχούν σε σπάνια components, δεν ευθυγραμμίζεται με τον τρόπο παραγωγής

ανωμαλιών στο συγκεκριμένο πείραμα. Στην πράξη, οι ανωμαλίες δημιουργήθηκαν κυρίως μέσω spikes, rare events και ακολουθιακών αποκλίσεων, άρα η σπανιότητα component δεν αποτέλεσε διακριτικό γνώρισμα. Η συμπεριφορά αυτή είναι αναμενόμενη σε detectors που στηρίζονται σε μία μόνο στατιστική υπόθεση, όταν η πραγματική ανωμαλία εκφράζεται με διαφορετική διάσταση μεταβολής (Chandola et al., 2009).

5.3 Χρόνοι εκτέλεσης και ρυθμός επεξεργασίας

Πέρα από την αποτελεσματικότητα ταξινόμησης, κρίσιμη παράμετρος σε συστήματα πραγματικού χρόνου είναι η καθυστέρηση απόφασης και ο ρυθμός επεξεργασίας. Στην παρούσα υλοποίηση καταγράφηκε ανά εγγραφή ο χρόνος inference του detector σε χιλιοστά του δευτερολέπτου, καθώς και συνοπτικά στατιστικά, μέσος χρόνος, 95ο εκατοστημόριο και μέγιστος χρόνος. Η μέτρηση χρόνων σε επίπεδο εγγραφής επιτρέπει να διαχωριστεί η απόδοση του αλγορίθμου από την καθυστέρηση ingestion ή οπτικοποίησης, κάτι που θεωρείται καλή πρακτική σε αξιολόγηση pipelines, όπου η end to end συμπεριφορά προκύπτει από το άθροισμα των σταδίων (Beyer et al., 2016).

```
Χρόνοι inference και throughput
rare_component | N 400 | mean 0.0008ms | p95 0.001ms | max 0.0057ms | ~1249999.75 inf/s
simple_threshold | N 400 | mean 0.0009ms | p95 0.001ms | max 0.0057ms | ~1162452.72 inf/s
```

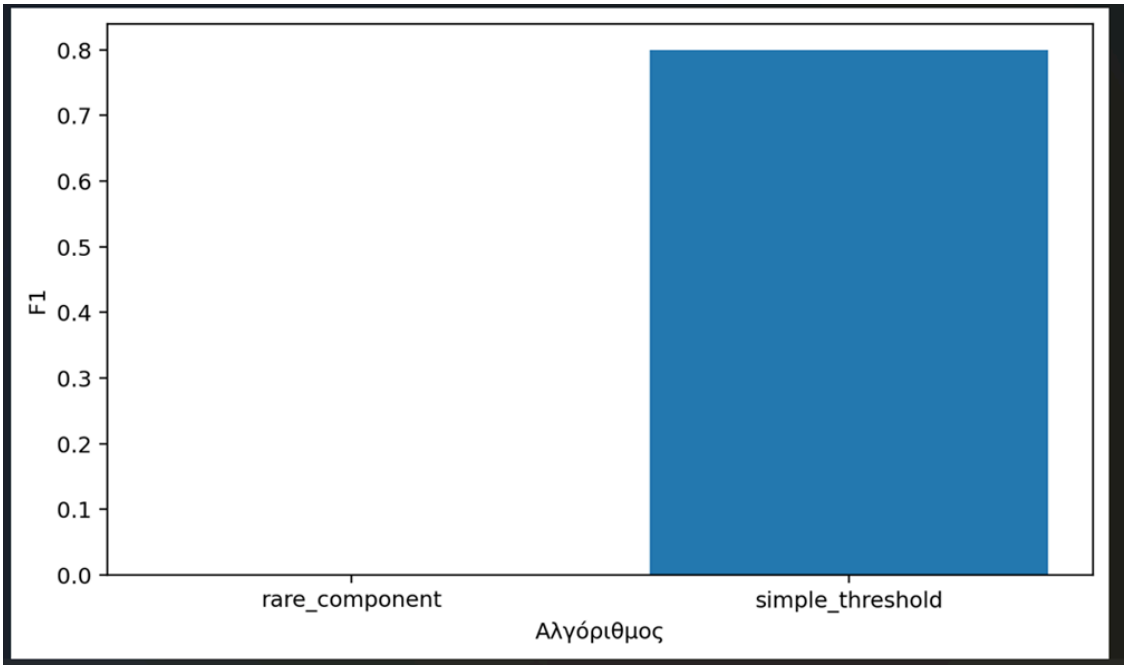
Εικόνα 5.3. Χρόνοι inference και υπολογιστικό throughput ανά αλγόριθμο, πηγή, στιγμιότυπο εκτέλεσης, επεξεργασία συγγραφέα.

Αλγόριθμος	N	Mean infer (ms)	p95 infer (ms)	Max infer (ms)	Approx inf/s
simple_threshold	400	0.0009	0.0010	0.0057	1162452.72
rare_component	400	0.0008	0.0010	0.0057	1249999.75

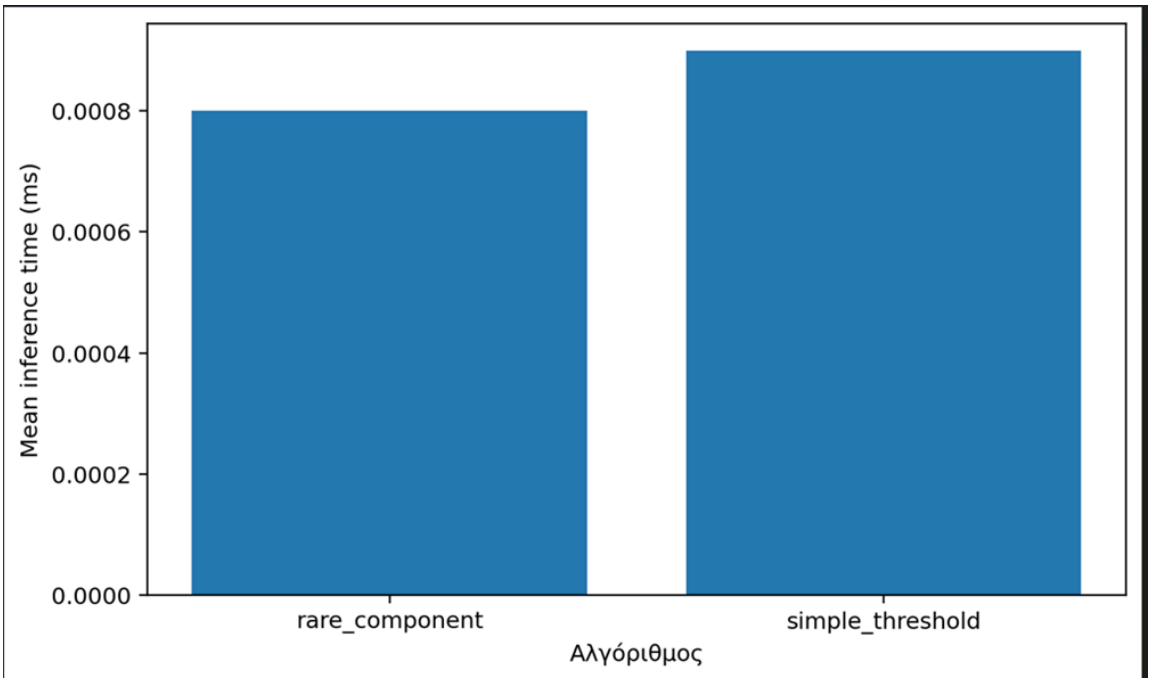
Πίνακας 5.2. Στατιστικά χρόνων inference και εκτιμώμενο throughput ανά αλγόριθμο

Οι δύο detectors εμφάνισαν εξαιρετικά χαμηλούς χρόνους inference, με μέσες τιμές της τάξης των 0,0008 έως 0,0009 ms ανά εγγραφή και p95 περίπου 0,001 ms. Οι τιμές αυτές υποδεικνύουν ότι το computational cost του detector είναι αμελητέο σε σχέση με τα υπόλοιπα στάδια ενός πραγματικού συστήματος, όπως είσοδος, parsing υψηλότερης πολυπλοκότητας ή

αποθήκευση σε κατανεμημένα συστήματα. Ωστόσο, τέτοιες μικρομετρήσεις σε περιβάλλον laptop επηρεάζονται από την ανάλυση του χρονόμετρου και από τον τρόπο δειγματοληψίας, άρα ερμηνεύονται πρωτίστως ως συγκριτικές ενδείξεις και όχι ως απόλυτες εγγυήσεις απόδοσης σε παραγωγικό περιβάλλον (Jain, 1991).



Εικόνα 5.4. Σύγκριση F1 ανά αλγόριθμο, πηγή, παραγόμενο γράφημα αξιολόγησης, επεξεργασία συγγραφέα.



Εικόνα 5.5. Σύγκριση μέσου inference time ανά αλγόριθμο, πηγή, παραγόμενο γράφημα αξιολόγησης, επεξεργασία συγγραφέα.

5.4 Σύγκριση και trade-offs

Η σύγκριση των αποτελεσμάτων αναδεικνύει ότι στο συγκεκριμένο πείραμα το κύριο trade off δεν αφορά χρόνους, καθώς και οι δύο detectors είναι υπολογιστικά ελαφροί, αλλά αφορά την ικανότητα σύλληψης του ορισμού ανωμαλίας που ενσωματώνεται στο ground truth. Ο simple_threshold επιτυγχάνει πλήρη κάλυψη ανωμαλιών, γεγονός που τον καθιστά κατάλληλο για περιβάλλοντα όπου προτεραιότητα είναι να μην χαθεί κρίσιμο συμβάν, ακόμη και αν αυτό οδηγεί σε αυξημένα alerts. Αντίθετα, ο rare_component δεν καταφέρνει να ανιχνεύσει τις ανωμαλίες του πειράματος, επειδή η υπόθεση σπανιότητας component δεν είναι αντιπροσωπευτική για τα συγκεκριμένα injected scenarios. Η διαπίστωση αυτή υποδεικνύει ότι detectors που βασίζονται σε στατιστική σπανιότητα απαιτούν προσεκτική επιλογή features και παραμετροποίηση, ιδίως όταν οι ανωμαλίες εκφράζονται ως spikes ή ακολουθιακές αποκλίσεις (Chandola et al., 2009).

Από άποψη καθυστέρησης, οι χρόνοι inference παραμένουν πολύ χαμηλοί και συγκρίσιμοι, ενώ η end to end συμπεριφορά στο pipeline επηρεάζεται κυρίως από τον ρυθμό ingestion που επιβλήθηκε σκόπιμα μέσω pacing για την προσομοίωση πραγματικού χρόνου. Συνεπώς, σε παραγωγική κλίμακα το trade off μετατοπίζεται προς τη συνολική αρχιτεκτονική ροής, δηλαδή τον μηχανισμό buffering, το parsing με templates και την αποθήκευση, καθώς και προς την πολιτική ειδοποίησης και το UX του dashboard, που καθορίζουν την επιχειρησιακή αξία των αποτελεσμάτων (Beyer et al., 2016).

5.4.1 Επεκταμένη ερμηνεία αποτελεσμάτων και επιχειρησιακές επιπτώσεις

Η πρακτική σημασία των μετρικών Precision και Recall δεν εξαντλείται στη μαθηματική τους διατύπωση, αλλά συνδέεται με το κόστος απόκρισης ενός οργανισμού και με τον τρόπο που οι ομάδες ασφάλειας και αξιοπιστίας ιεραρχούν προτεραιότητες. Σε ένα περιβάλλον όπου τα logs παράγονται συνεχώς και πολλαπλασιάζονται από επιμέρους υπηρεσίες, κάθε false positive μεταφράζεται σε χρόνο ανθρώπινης διερεύνησης, σε πιθανή διακοπή ροών εργασίας και σε κόπωση από alerts. Η κόπωση αυτή μπορεί να οδηγήσει σε απενεργοποίηση

ειδοποιήσεων ή σε αγνόηση προειδοποιήσεων, άρα τελικά σε υποβάθμιση της πραγματικής ανθεκτικότητας του συστήματος (Beyer et al., 2016).

Συμπληρωματικά, τα false negatives έχουν ιδιαίτερο βάρος όταν τα logs σχετίζονται με ενδείξεις παραβίασης ή με αποτυχίες κρίσιμων υποσυστημάτων. Στην περίπτωση που μία πραγματική ανωμαλία δεν επισημανθεί, ο οργανισμός μπορεί να χάσει το παράθυρο έγκαιρης απόκρισης και να βρεθεί σε φάση μεταγενέστερης διερεύνησης, όπου η αιτιολόγηση γίνεται δυσκολότερη και το κόστος ανάκτησης αυξάνεται. Η επιλογή να προτιμηθεί υψηλό recall, όπως συμβαίνει στον simple_threshold detector, αποτελεί συχνά συνειδητή στρατηγική σε πρώιμα στάδια εγκατάστασης ενός pipeline, όταν στόχος είναι να μη χαθούν σπάνια αλλά κρίσιμα συμβάντα (Kent & Souppaya, 2006).

Παρά ταύτα, το αποτέλεσμα Precision 0,6667 υποδεικνύει ότι περίπου ένα στα τρία alerts του simple_threshold δεν αντιστοιχεί σε ανωμαλία σύμφωνα με τον ορισμό του ground truth. Σε συνθήκες μεγάλης κλίμακας αυτό θα ήταν προβληματικό, αλλά στο πλαίσιο της διπλωματικής αποτυπώνει εύστοχα τη φύση των κανονιστικών detectors, οι οποίοι στηρίζονται σε φαινομενικά ισχυρά σήματα όπως το exception.

Για να βελτιωθεί η συμπεριφορά σε παραγωγικό περιβάλλον, θα απαιτούνταν είτε εμπλουτισμός του parsing με templates, είτε χρήση παραθύρων συχνότητας, είτε τεχνικές βαθμολόγησης που λαμβάνουν υπόψη το ιστορικό ενός component και όχι μόνο τη μεμονωμένη γραμμή (Chandola et al., 2009).

Η αποτυχία του rare_component detector ως προς την ανίχνευση των ground truth anomalies αναδεικνύει μία ουσιώδη αρχή του anomaly detection, ότι δεν υπάρχει καθολικό κριτήριο ανωμαλίας ανεξάρτητα από το πλαίσιο. Ένα component μπορεί να είναι σπάνιο επειδή χρησιμοποιείται μόνο σε ειδικές ροές, χωρίς αυτό να αποτελεί ένδειξη κινδύνου, ενώ μία ανωμαλία μπορεί να συμβεί σε πολύ συχνό component, όπως ένας DataXceiver, εφόσον το γεγονός εκφράζεται ως αλλαγή στη συχνότητα ή ως επαναλαμβανόμενη αποτυχία. Η παρατήρηση αυτή συμβαδίζει με τη βιβλιογραφία που τονίζει ότι οι ανωμαλίες σε logs συχνά εκφράζονται ως μοτίβα ακολουθίας ή ως μεταβολές κατανομών και όχι ως στατική σπανιότητα οντοτήτων (He et al., 2020).

Από πλευράς χρόνων, η εξαιρετικά μικρή μέση τιμή inference δεν πρέπει να οδηγήσει σε συμπέρασμα ότι ένα πραγματικό σύστημα θα έχει αντίστοιχη end to end καθυστέρηση. Η end to end καθυστέρηση περιλαμβάνει καθυστερήσεις εισαγωγής, parsing, μετατροπής, εγγραφής σε βάση και ανάκτησης από το dashboard. Στο πείραμα, η καθυστέρηση εισαγωγής

ρυθμίστηκε τεχνητά μέσω pacing και η αποθήκευση πραγματοποιήθηκε τοπικά σε SQLite, άρα η υποδομική επιβάρυνση είναι περιορισμένη. Σε περιβάλλον παραγωγής, οι ίδιες έννοιες θα υλοποιούνταν πιθανόν με message broker, κατανεμημένη αποθήκευση και μηχανισμούς observability, γεγονός που αυξάνει τον συνολικό χρόνο αλλά παρέχει κλιμάκωση και αξιοπιστία (Beyer et al., 2016).

Παρά τα παραπάνω, η πειραματική μέτρηση των χρόνων inference διατηρεί αξία ως συγκριτικός δείκτης, διότι δείχνει ότι ο υπολογιστικός πυρήνας των detectors δεν αποτελεί bottleneck για την παρούσα κλίμακα. Αυτό είναι σημαντικό συμπέρασμα, επειδή επιτρέπει να δοθεί προτεραιότητα σε βελτιώσεις της ακρίβειας και της αναπαράστασης των δεδομένων, χωρίς άμεσο φόβο ότι η προσθήκη λογικής θα καταστήσει το pipeline μη πραγματικού χρόνου. Η μετατόπιση της προσπάθειας προς καλύτερο parsing, καλύτερα features και καλύτερη διαχείριση alerts αποτελεί τυπική εξέλιξη σε projects log analytics (Kent & Souppaya, 2006).

Στο επίπεδο της μεθοδολογίας, η χρήση ground truth που παράγεται ταυτόχρονα με το stream αποτελεί σημαντικό στοιχείο εγκυρότητας του πειράματος, διότι αποφεύγεται η εκ των υστέρων υποκειμενική επισήμανση. Παράλληλα, το γεγονός ότι το ground truth είναι συνθετικό εισάγει περιορισμό, επειδή οι πραγματικές ανωμαλίες ενδέχεται να έχουν πιο σύνθετη μορφολογία και να μη διαχωρίζονται καθαρά σε spikes, rare events και sequence anomalies. Η αναγνώριση αυτού του περιορισμού είναι κρίσιμη στο κείμενο της διπλωματικής και λειτουργεί ως γέφυρα για μελλοντική εργασία με πραγματικά δεδομένα παραγωγής ή με πιο πλούσια datasets (Chandola et al., 2009).

Τέλος, η ανάλυση του error_examples.txt υποστηρίζει ποιοτική τεκμηρίωση των αριθμητικών αποτελεσμάτων. Όταν παρατηρείται ότι ο simple_threshold ενεργοποιείται σε μηνύματα που περιέχουν exception, ακόμη και όταν δεν ανήκουν στο ground truth, γίνεται σαφές ότι το feature δεν είναι από μόνο του επαρκές για επιχειρησιακή απόφαση. Αντίστοιχα, όταν ο rare_component χαρακτηρίζει ορισμένες εγγραφές ως anomaly λόγω σπανιότητας, αλλά αποτυγχάνει να επισημάνει τις γνωστές ανωμαλίες, προκύπτει σαφής ανάγκη ανασχεδιασμού των features του, είτε με sliding window counts είτε με ενσωμάτωση ακολουθιακής πληροφορίας. Η ενσωμάτωση τέτοιων συμπερασμάτων είναι το στοιχείο που μετατρέπει ένα απλό πείραμα σε ουσιαστική τεχνική ανάλυση (He et al., 2020).

5.5 Συζήτηση σφαλμάτων (false positives/false negatives, γιατί συμβαίνουν)

Η ανάλυση σφαλμάτων ερμηνεύει τα false positives και false negatives και συνδέει τα αριθμητικά αποτελέσματα με τη δομή των μηνυμάτων. Τα false positives του simple_threshold προκύπτουν επειδή η λογική του detector δίνει υψηλό βάρος σε ενδείξεις warning και σε μοτίβα exception, τα οποία μπορούν να εμφανιστούν και σε λειτουργικές καταστάσεις που δεν συνιστούν ανωμαλία όπως ορίστηκε στο ground truth. Η συμπεριφορά αυτή είναι τυπική σε threshold detectors, όπου η απλότητα αυξάνει την ευαισθησία αλλά μειώνει την ειδικότητα όταν το περιβάλλον έχει θορυβώδεις προειδοποιήσεις (Kent & Souppaya, 2006).

Τα false negatives του rare_component είναι καθολικά για τις ανωμαλίες του ground truth, γεγονός που δείχνει ασυμφωνία ανάμεσα στο feature σπανιότητας component και στο μοτίβο ανωμαλίας του πειράματος. Η περίπτωση αναδεικνύει ότι η επιλογή feature πρέπει να είναι σύμφωνη με τον τύπο ανωμαλίας, καθώς spikes και sequence anomalies απαιτούν παραθυρικές μετρήσεις συχνότητας ή ακολουθιακές αναπαραστάσεις και όχι στατική σπανιότητα component (He et al., 2020).

Εικόνα 5.6. Ενδεικτικά παραδείγματα false positives και false negatives από το error_examples.txt, πηγή, στιγμιότυπο οθόνης, επεξεργασία συγγραφέα.

Συνολικά, το πείραμα επιβεβαιώνει ότι το pipeline επιτυγχάνει πολύ χαμηλή καθυστέρηση απόφασης για απλούς detectors, αλλά η ποιότητα ανίχνευσης καθορίζεται από τη

συμβατότητα της λογικής ανωμαλίας με τα σενάρια και τα χαρακτηριστικά που εξάγονται. Η ύπαρξη ground truth και η παραγωγή παραδειγμάτων σφαλμάτων καθιστούν τη διαδικασία ελέγξιμη και επιτρέπουν σαφή προσανατολισμό για βελτίωση είτε με ρύθμιση thresholds, είτε με εμπλουτισμό features, είτε με υιοθέτηση μοντέλων που αξιοποιούν templates και ακολουθίες όπως προτείνεται στη σύγχρονη βιβλιογραφία log based anomaly detection (He et al., 2020).

Κεφάλαιο 6 Συμπεράσματα και μελλοντική εργασία

Το παρόν κεφάλαιο συνοψίζει τα βασικά συμπεράσματα που προέκυψαν από τη σχεδίαση, την υλοποίηση και την αξιολόγηση του προτεινόμενου συστήματος ανίχνευσης ανωμαλιών σε αρχεία καταγραφής. Η σύνθεση των ευρημάτων δεν περιορίζεται στην αποτίμηση της τεχνικής λειτουργίας του pipeline, αλλά επεκτείνεται στην επιχειρησιακή σημασία των αποτελεσμάτων, στους περιορισμούς της πειραματικής μεθοδολογίας και στις προοπτικές εξέλιξης του συστήματος. Η ανάγκη για τέτοια ολιστική αποτίμηση είναι ιδιαίτερα σημαντική στα συστήματα ανάλυσης logs, καθώς η πραγματική τους αξία εξαρτάται όχι μόνο από την ακρίβεια ανίχνευσης, αλλά και από την ικανότητά τους να λειτουργούν με συνέπεια, να ερμηνεύονται από τους χρήστες και να ενσωματώνονται σε διαδικασίες παρακολούθησης και απόκρισης (Kent & Souppaya, 2006).

6.1 Τι πέτυχε το σύστημα

Η εργασία κατέληξε στην επιτυχή υλοποίηση ενός ολοκληρωμένου pipeline ανίχνευσης ανωμαλιών, το οποίο καλύπτει διαδοχικά τη δημιουργία ή εισαγωγή logs, τη ροή πραγματικού χρόνου, την προεπεξεργασία, την εξαγωγή χαρακτηριστικών, την ανίχνευση, την αποθήκευση και την οπτικοποίηση. Η επιτυχία αυτή έχει ιδιαίτερη βαρύτητα, επειδή αποδεικνύει ότι ένα λειτουργικό σύστημα log anomaly detection μπορεί να αναπτυχθεί με καθαρή αρχιτεκτονική και αναπαραγώγιμο τρόπο ακόμη και σε περιορισμένο υπολογιστικό περιβάλλον. Η αρχιτεκτονική επιλογή Python, SQLite και Streamlit επέτρεψε να μειωθεί η πολυπλοκότητα της υποδομής χωρίς να θυσιαστεί η πληρότητα του πειράματος, στοιχείο που είναι κρίσιμο όταν ο στόχος της εργασίας αφορά πρωτίστως την οργάνωση του software pipeline και όχι την κατασκευή βαριάς παραγωγικής πλατφόρμας.

Σε επίπεδο λειτουργικότητας, το σύστημα πέτυχε να μετατρέψει πραγματικές γραμμές HDFS logs σε δομημένη, επεξεργάσιμη και αξιολογήσιμη ροή δεδομένων. Ο log generator δημιούργησε συνδυασμό normal και anomaly περιπτώσεων, το ingestion προσομοίωσε σταδιακή ροή πραγματικού χρόνου, το preprocessing παρήγαγε πεδία και χαρακτηριστικά που μπορούσαν να τροφοδοτήσουν διαφορετικούς detectors, ενώ η βάση δεδομένων διατήρησε με σαφή τρόπο τα raw δεδομένα, τις parsed εγγραφές, τα features και τις τελικές προβλέψεις. Η συγκεκριμένη ιχνηλασιμότητα είναι σημαντική, επειδή επιτρέπει να ανακτηθεί η πορεία κάθε εγγραφής από το πρωτογενές μήνυμα έως το τελικό label, κάτι που θεωρείται καλή πρακτική σε αξιόπιστα συστήματα δεδομένων και σε περιβάλλοντα τεχνικού ελέγχου (Kleppmann, 2017).

Η εργασία πέτυχε επίσης να προσδώσει μετρησιμότητα στο πρόβλημα της ανίχνευσης. Η ύπαρξη ground truth, η αποθήκευση των αποτελεσμάτων στον πίνακα predictions και η εξαγωγή ποσοτικών μετρικών, όπως Precision, Recall, F1, inference time και throughput, επέτρεψαν τη συγκριτική αξιολόγηση των αλγορίθμων σε ενιαίο πρωτόκολλο. Με τον τρόπο αυτό, το σύστημα δεν παρήγαγε απλώς alarms, αλλά παρήγαγε τεκμήρια για το πόσο καλά λειτούργησε. Η διάσταση αυτή είναι ουσιώδης για κάθε σύστημα ανίχνευσης, επειδή χωρίς σαφή αξιολόγηση είναι αδύνατο να κριθεί αν ένας detector είναι πραγματικά χρήσιμος ή απλώς εντυπωσιακός επιφανειακά (Powers, 2011).

Τέλος, το σύστημα πέτυχε να συνδέσει την ανίχνευση με ένα πρακτικό περιβάλλον διερεύνησης. Το dashboard σε Streamlit επέτρεψε φίλτρα ανά μοντέλο, label, level και component, εμφάνιση συγκεντρωτικών μετρικών και drill down μέχρι το πρωτογενές μήνυμα. Έτσι, η εργασία δεν περιορίστηκε σε υπολογιστική άσκηση, αλλά απέδειξε ότι τα αποτελέσματα μπορούν να παρουσιαστούν με τρόπο χρήσιμο για analyst ή μηχανικό συστήματος. Η δυνατότητα αυτή είναι ουσιώδης, διότι στην πράξη τα anomaly scores αποκτούν αξία μόνο όταν μπορούν να εντοπιστούν, να ερμηνευθούν και να συσχετισθούν με τη ροή λειτουργίας του συστήματος (Beyer et al., 2016).

6.2 Τι έδειξε η σύγκριση και ποιος αλγόριθμος βολεύει σε real time

Η συγκριτική αξιολόγηση των δύο detectors έδειξε με σαφήνεια ότι, στο συγκεκριμένο πειραματικό πλαίσιο, ο simple_threshold detector είναι πρακτικά καταλληλότερος από τον rare_component detector για χρήση σε ροή πραγματικού χρόνου. Το συμπέρασμα αυτό προκύπτει από τον συνδυασμό των μετρικών Recall, Precision και F1, αλλά και από την επιχειρησιακή λογική του συστήματος. Ο simple_threshold πέτυχε Recall ίσο με 1,0 και F1 ίσο με 0,8, γεγονός που σημαίνει ότι δεν έχασε καμία από τις ανωμαλίες του ground truth, αν και παρήγαγε επιπλέον ψευδώς θετικές ενδείξεις. Αντίθετα, ο rare_component detector δεν μπόρεσε να αναγνωρίσει τις ανωμαλίες του πειράματος, με αποτέλεσμα μηδενικό Recall και μηδενικό F1. Συνεπώς, στο ερώτημα ποιος αλγόριθμος βολεύει περισσότερο σε πραγματικό χρόνο, η απάντηση δεν μπορεί να είναι ο θεωρητικά πιο κομψός, αλλά ο detector που εντοπίζει τα κρίσιμα συμβάντα χωρίς να τα αφήνει να διαφύγουν.

Η υπεροχή του simple_threshold σχετίζεται άμεσα με τη φύση των χαρακτηριστικών που χρησιμοποιεί. Οι injected anomalies του πειράματος συνδέθηκαν με warnings, exceptions, επαναλήψεις και ακολουθιακές αποκλίσεις, άρα ήταν λογικό ένας detector που εξετάζει άμεσα warning και exception σήματα να ανταποκριθεί καλύτερα. Ο rare_component detector, αντίθετα, στηρίχθηκε στη σπανιότητα component, δηλαδή σε μία διαφορετική υπόθεση για το τι συνιστά απόκλιση. Η αστοχία του στο παρόν dataset επιβεβαιώνει μία βασική αρχή της βιβλιογραφίας anomaly detection, ότι ο αλγόριθμος πρέπει να είναι συμβατός με το είδος της ανωμαλίας που επιδιώκεται να αναγνωριστεί και ότι δεν υπάρχει ενιαία μέθοδος που να είναι καθολικά αποτελεσματική ανεξάρτητα από το πλαίσιο (Chandola et al., 2009).

Από πλευράς χρονισμού, και οι δύο detectors υπήρξαν ιδιαίτερα ελαφροί υπολογιστικά. Οι μετρήσεις έδειξαν μέσο inference time της τάξης των χιλιοστών του χιλιοστού του δευτερολέπτου ανά εγγραφή, με πρακτικά αμελητέο υπολογιστικό φορτίο. Αυτό σημαίνει ότι στο παρόν scale ο παράγοντας χρόνου δεν διαφοροποιεί ουσιαστικά τους δύο αλγορίθμους. Συνεπώς, το βασικό trade off δεν είναι ταχύτητα έναντι ακρίβειας, αλλά κάλυψη ανωμαλιών έναντι κόστους ψευδών alarms. Για ένα σύστημα πραγματικού χρόνου, ιδιαίτερα σε πρώιμη φάση ανάπτυξης, ένας detector υψηλού recall είναι συνήθως προτιμότερος, επειδή το κόστος μίας χαμένης ανωμαλίας μπορεί να είναι σημαντικά υψηλότερο από το κόστος διερεύνησης λίγων επιπλέον alerts (Kent & Souppaya, 2006).

Με βάση τα παραπάνω, ο simple_threshold είναι ο detector που βολεύει περισσότερο στην παρούσα real time εκδοχή του συστήματος. Αυτό δεν σημαίνει ότι αποτελεί οριστική λύση, αλλά ότι προσφέρει την καλύτερη λειτουργική ισορροπία για το συγκεκριμένο σύνολο

δεδομένων και για τον συγκεκριμένο τύπο ανωμαλιών. Στην πράξη, ένας τέτοιος detector μπορεί να χρησιμοποιηθεί ως πρώτο φίλτρο real time επιτήρησης, πάνω στο οποίο θα μπορούσαν να προστεθούν μεταγενέστερα μηχανισμοί refinement, καλύτερη παραμετροποίηση, εμπλουτισμένα features ή δεύτερο στάδιο επανελέγχου.

6.3 Περιορισμοί

Παρά την επιτυχή υλοποίηση και τα σαφή πειραματικά αποτελέσματα, η εργασία παρουσιάζει περιορισμούς που πρέπει να αναγνωριστούν ρητά. Ο σημαντικότερος περιορισμός αφορά τη φύση του ground truth. Οι ανωμαλίες δημιουργήθηκαν συνθετικά πάνω σε πραγματικές γραμμές logs, γεγονός που εξυπηρέτησε την ανάγκη ελεγχόμενης αξιολόγησης, αλλά δεν αναπαριστά πλήρως την πολυπλοκότητα ανωμαλιών ενός παραγωγικού περιβάλλοντος. Σε πραγματικά συστήματα, οι ανωμαλίες μπορεί να είναι λιγότερο καθαρές, να εξελίσσονται μέσα στον χρόνο, να συνδυάζουν πολλαπλές αιτίες και να εμφανίζονται μέσα από ακολουθιακά ή διασυστημικά μοτίβα που δεν αποτυπώνονται σε απλή γραμμοκεντρική μορφή.

Δεύτερος περιορισμός αφορά την απλότητα των χαρακτηριστικών και των αλγορίθμων. Η εργασία υιοθέτησε συνειδητά απλά, ερμηνεύσιμα features και lightweight detectors, ώστε να δοθεί έμφαση στη σχεδίαση του pipeline και στην αναπαραγωγιμότητα. Η επιλογή αυτή είναι απολύτως θεμιτή για μία software engineering προσανατολισμένη εργασία, αλλά περιορίζει την ικανότητα του συστήματος να αναγνωρίζει πιο σύνθετα μοτίβα απόκλισης. Πολλές ανωμαλίες σε logs εκφράζονται ως sequence anomalies, ως μεταβολές συχνοτήτων σε χρονικά παράθυρα ή ως σημασιολογικές αποκλίσεις που απαιτούν richer parsing, templates ή sequence models (He et al., 2020).

Τρίτος περιορισμός συνδέεται με την υποδομή εκτέλεσης. Η χρήση SQLite, ingestion με pacing και τοπικού dashboard εξυπηρέτησε ιδανικά την υλοποίηση σε laptop, αλλά δεν αναπαριστά όλη την πολυπλοκότητα ενός παραγωγικού περιβάλλοντος με πολλούς producers, συνεχές throughput, buffering, κατανεμημένη αποθήκευση και απαιτήσεις υψηλής διαθεσιμότητας. Επομένως, οι χρόνοι που μετρήθηκαν πρέπει να ερμηνευθούν ως έγκυροι για

τη συγκεκριμένη αρχιτεκτονική proof of concept και όχι ως απευθείας μεταφέρσιμοι σε μεγάλη παραγωγική υποδομή.

Τέταρτος περιορισμός αφορά την έκταση της συγκριτικής αξιολόγησης. Η εργασία συνέκρινε δύο detectors που υλοποιήθηκαν εντός του ίδιου pipeline, αλλά δεν προχώρησε σε αντιπαραβολή με state of the art αλγορίθμους της βιβλιογραφίας, όπως sequence neural networks, transformer based προσεγγίσεις ή parsing free μοντέλα. Αυτό δεν μειώνει την αξία της παρούσας συνεισφοράς, η οποία είναι σαφώς προσανατολισμένη στην αρχιτεκτονική και στην ενσωμάτωση, αλλά σημαίνει ότι τα συμπεράσματα πρέπει να διατυπωθούν με τεχνική μετριοπάθεια.

6.4 Μελλοντική εργασία

Η μελλοντική εργασία μπορεί να κινηθεί καταρχάς προς την κατεύθυνση της βελτίωσης του feedback loop ανάμεσα στον analyst και στον detector. Στην παρούσα εκδοχή, το dashboard υποστηρίζει διερεύνηση και drill down, αλλά δεν υπάρχει μηχανισμός με τον οποίο ο χρήστης να επιβεβαιώνει ή να απορρίπτει ένα alert και αυτή η πληροφορία να επιστρέφει στο σύστημα. Η εισαγωγή μηχανισμού ανατροφοδότησης θα μπορούσε να υποστηρίξει active learning, δυναμική προσαρμογή thresholds ή σταδιακή βελτίωση κανόνων, με αποτέλεσμα το σύστημα να προσαρμόζεται στη λειτουργική πραγματικότητα του περιβάλλοντος όπου εγκαθίσταται.

Μία δεύτερη κατεύθυνση αφορά την αναβάθμιση του parsing και των templates. Η παρούσα εργασία έδειξε ότι η ποιότητα του parsing αποτελεί κρίσιμο ενδιάμεσο στάδιο, επειδή από αυτήν εξαρτώνται τα features, οι detectors και η ερμηνευσιμότητα των αποτελεσμάτων. Μελλοντικά θα μπορούσαν να ενσωματωθούν μηχανισμοί πιο ώριμης εξαγωγής templates, grouping σε sessions ή blocks, και sequence construction σε χρονικά παράθυρα, ώστε η ανίχνευση να μετατοπιστεί από τη γραμμή προς την ακολουθία. Μια τέτοια επέκταση θα ήταν ιδιαίτερα χρήσιμη για sequence anomalies, οι οποίες στην πράξη είναι συχνές αλλά δύσκολα ανιχνεύσιμες με απλά ανά γραμμή χαρακτηριστικά.

Μία τρίτη κατεύθυνση αφορά την κλιμάκωση της αρχιτεκτονικής και την ενσωμάτωση περισσότερο παραγωγικών υποδομών. Το ingestion θα μπορούσε να υποστηριχθεί από message broker, η αποθήκευση από πιο ισχυρή σχεσιακή ή time series βάση, και το dashboard από περιβάλλον που να συνδέεται με alerting και role based πρόσβαση. Η παρούσα δομή αποδεικνύει τη σκοπιμότητα της λύσης, αλλά η μεταφορά σε αρχιτεκτονική

μεγαλύτερης κλίμακας θα επέτρεπε αξιολόγηση υπό φόρτο, δοκιμές backpressure, αποσύνδεση των σταδίων και καλύτερη επιχειρησιακή ανθεκτικότητα.

Μία τέταρτη κατεύθυνση αφορά την αλγοριθμική εξέλιξη. Παρότι οι lightweight detectors ήταν κατάλληλοι για την παρούσα proof of concept υλοποίηση, η φυσική συνέχεια είναι η προσθήκη πιο προηγμένων μοντέλων που αξιοποιούν templates, ιστορικά παράθυρα, sequences ή embeddings. Ιδιαίτερο ενδιαφέρον έχει η υβριδική λογική, όπου ένας γρήγορος rule based detector χρησιμοποιείται ως πρώτο real time φίλτρο και ένα δεύτερο πιο σύνθετο μοντέλο αναλαμβάνει re ranking ή επιβεβαίωση alerts. Με τον τρόπο αυτό θα μπορούσε να διατηρηθεί χαμηλή καθυστέρηση, ενώ παράλληλα θα μειωνόταν ο θόρυβος από false positives.

Συνολικά, η εργασία κατέδειξε ότι ένα software engineering oriented pipeline για log anomaly detection μπορεί να σχεδιαστεί, να υλοποιηθεί και να αξιολογηθεί με σαφή και επαναλήψιμο τρόπο. Η μελλοντική του εξέλιξη δεν αφορά μόνο περισσότερους αλγορίθμους, αλλά κυρίως βαθύτερη σύνδεση ανάμεσα στην ποιότητα των δεδομένων, στη δομή της αρχιτεκτονικής, στην ανατροφοδότηση του χρήστη και στη δυνατότητα του συστήματος να παράγει έγκαιρα, αξιόπιστα και επιχειρησιακά χρήσιμα alerts. Αυτή ακριβώς η σύζευξη αποτελεί και την ουσία της επόμενης φάσης έρευνας στον χώρο της ανίχνευσης ανωμαλιών σε logs.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Apache Software Foundation. 2025.

Apache Kafka Documentation.

Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice (3rd ed.). Addison Wesley.

Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). Site Reliability Engineering: How Google Runs Production Systems. O'Reilly Media.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys, 41(3), 1–58.

Confluent. 2025. Introduction to Apache Kafka.

Du, M., Li, F., Zheng, G., and Srikumar, V. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs. Proceedings of the ACM Conference on Computer and Communications Security.

Elastic. 2024. Using Kibana Server Logs και τεκμηρίωση για αναζήτηση και απεικόνιση καταγραφών.

Gerhards, R. (2009). The Syslog Protocol (RFC 5424). RFC Editor. https://www.rfc-editor.org/rfc/rfc5424

Grafana. (2025). Grafana documentation. https://grafana.com/docs/grafana/latest/introduction/

Guo, H., Yuan, S., Wu, J., & Li, X. (2021). LogBERT: Log anomaly detection via BERT. arXiv. https://arxiv.org/abs/2103.04475

He, P., Zhu, J., He, S., Li, J., & Lyu, M. R. (2017). Drain: An online log parsing approach with fixed depth tree. Proceedings of the IEEE International Conference on Web Services. https://jiemingzhu.github.io/pub/pjhe_icws2017.pdf

Jain, R. (1991). The Art of Computer Systems Performance Analysis. Wiley.

Jiang, J., Fu, Q., Lou, J. G., Lin, Q., Zhang, R., and Wang, J. 2023. LogHub 2.0 και πόροι συνόλων δεδομένων για αρχεία καταγραφής.

IEEE. (2018). ISO/IEC/IEEE 29148:2018 Systems and software engineering. Life cycle processes. Requirements engineering.

ISO/IEC. (2011). ISO/IEC 25010:2011 Systems and software engineering. Systems and software Quality Requirements and Evaluation (SQuaRE).

Khan, Z. A., Malik, H., & others. (2024). Impact of log parsing on deep learning based anomaly detection accuracy. PubMed Central. https://pmc.ncbi.nlm.nih.gov/articles/PMC11330418/

Kent, K., & Souppaya, M. (2006). Guide to computer security log management (NIST Special Publication 800-92). National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-92.pdf

Kleppmann, M. (2017). Designing Data Intensive Applications. O'Reilly Media.

Landauer, M., Onder, S., Skopik, F., and Wurzenberger, M. 2022. Deep Learning for Anomaly Detection in Log Data: A Survey. arXiv προδημοσίευση.

Landauer, M., et al. 2023. Deep Learning for Anomaly Detection in Log Data, επικαιροποιήσεις και πρακτική αποτίμηση. Συνοπτική αποτύπωση στο ίδιο ερευνητικό νήμα.

Le, V. H., et al. 2021. Log-based Anomaly Detection Without Log Parsing. arXiv προδημοσίευση.

Meng, W., et al. 2019. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. IJCAI Proceedings, διαθέσιμο σε μορφή PDF.

OpenTelemetry. (2025). OpenTelemetry specification overview and logs. https://opentelemetry.io/docs/specs/otel/overview/

Owens, M. (2006). The Definitive Guide to SQLite (2nd ed.). Apress.

Powers, D. M. W. 2011. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation. Journal of Machine Learning Technologies. Διαθέσιμο σε μορφή PDF.

Prometheus. (2025). Prometheus: Monitoring system and time series database. https://prometheus.io/

Saito, T. and Rehmsmeier, M. 2015. The Precision Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. PLoS ONE 10, 3, e0118432.

Sridharan, C. 2018. Distributed Systems Observability: A Guide to Building Robust Systems. O'Reilly Media. Διαθέσιμο και ως ελεύθερο τεχνικό κείμενο σε μορφή PDF.

Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., & Lyu, M. R. (2019). Tools and benchmarks for automated log parsing. Proceedings of ICSE SEIP. https://netman.aiops.org/~peidan/ANM2023/6.LogAnomalyDetection/LectureCoverage/jzhu_icseseip2019_tools.pdf

Zhu, J., et al. (2020). LogHub: A large collection of system log datasets for AI driven log analytics. arXiv. https://arxiv.org/pdf/2008.06448.pdf

Xu, W., Huang, L., Fox, A., Patterson, D., and Jordan, M. I. 2009. Detecting Large Scale System Problems by Mining Console Logs. Proceedings of SOSP.

ΠΑΡΑΡΤΉΜΑΤΑ

Σχήμα 1.1 Ενδεικτική μεταβολή ρυθμού παραγωγής αρχείων καταγραφής σε διάρκεια ημέρας 67

Πίνακας 1.1 Ενδεικτικές κατηγορίες ανωμαλιών σε αρχεία καταγραφής και επιχειρησιακή σημασία 67

Σχήμα 1.2 Γενική ροή αγωγού επεξεργασίας για ανίχνευση ανωμαλιών σε αρχεία καταγραφής 67

Σχήμα 1.3 Αρχιτεκτονική υψηλού επιπέδου της προτεινόμενης λύσης σε πραγματικό χρόνο 67

Εικόνα 2.1. Εννοιολογική ροή ενός συστήματος ανίχνευσης ανωμαλιών σε logs 67

Πίνακας 2.1. Ενδεικτικές κατηγορίες πηγών logs και χαρακτηριστικά τους 67

Πίνακας 2.2. Σύγκριση προσεγγίσεων φορμά και μεταφοράς logs 67

Εικόνα 2.2. Παράδειγμα μετασχηματισμού μηνύματος σε template και παραμέτρους 67

Πίνακας 2.3. Κατηγορίες log parsing και επιπτώσεις στο pipeline 67

Πίνακας 2.4. Αναπαραστάσεις και χαρακτηριστικά για ανίχνευση ανωμαλιών 67

Πίνακας 2.5. Κατηγοριοποίηση τεχνικών ανίχνευσης ανωμαλιών σε logs 67

Πίνακας 2.6. Ενδεικτικά χαρακτηριστικά των datasets logs για αξιολόγηση 67

Εικόνα 2.3. Ενδεικτική αρχιτεκτονική ροής logs και συστατικών παρατηρησιμότητας 67

Πίνακας 2.7. Χαρτογράφηση συστατικών ενός real time pipeline σε λειτουργικούς ρόλους 67

Εικόνα 2.4. Εννοιολογική απεικόνιση ανταλλαγής μεταξύ ποιότητας και καθυστέρησης67

Πίνακας 2.8. Ορισμοί βασικών μετρικών αξιολόγησης σε προβλήματα ανίχνευσης67

Πίνακας 3.1. Λειτουργικές απαιτήσεις με κριτήρια ελέγχου 67

Πίνακας 3.2. Μη λειτουργικές απαιτήσεις και κριτήρια αποδοχής 67

Εικόνα 3.1. Ενδεικτική σχέση ρυθμού εισόδου και καθυστέρησης σε pipeline laptop 67

Πίνακας 3.3. Χαρτογράφηση τεχνολογιών σε ρόλους του συστήματος 67

Εικόνα 3.2. Αρχιτεκτονική modules και ροές δεδομένων στο προτεινόμενο pipeline67

Εικόνα 3.3. Σειρά αλληλεπιδράσεων μεταξύ modules κατά την εκτέλεση 67

Εικόνα 3.4. Οντολογικό σχήμα SQLite και βασικές συσχετίσεις 67

Πίνακας 3.4. Πεδία σχήματος και λειτουργικός ρόλος 67

Πίνακας 3.5. Αντιστοίχιση σεναρίων χρήσης με λειτουργίες και δεδομένα 67

Εικόνα 4.1: Εκτέλεση log generator και παραγωγή αρχείων stream_input.log και ground_truth.csv. 67

Εικόνα 4.2: Σταδιακή ροή αρχείων καταγραφής και μέτρηση ρυθμού εισόδου. 67

Εικόνα 4.3: Preprocessing 400 γραμμών και εγγραφή σε SQLite. 67

Εικόνα 4.4: Εκτέλεση των δύο detectors και εγγραφή αποτελεσμάτων στον πίνακα predictions. 68

Εικόνα 4.5α: Αποτυχημένη εκτέλεση db_check από λανθασμένο κατάλογο 68

Εικόνα 4.5β: Επιτυχής εκτέλεση db_check με εμφάνιση πινάκων, πληθών και indexes. 68

Εικόνα 4.6: Εκκίνηση Streamlit και διαδρομές πρόσβασης στο dashboard. 68

Εικόνα 4.7: Οθόνη dashboard με ενεργά φίλτρα και γράφημα κατανομής ανωμαλιών ανά component. 68

Εικόνα 4.8: Λίστα εγγραφών, drill-down και προβολή raw log. 68

Εικόνα 5.1. Παραγόμενα αρχεία αξιολόγησης στο results\ch5, πηγή, στιγμιότυπο οθόνης, επεξεργασία συγγραφέα. 68

Εικόνα 5.2. Αποτελέσματα Precision, Recall και F1 ανά αλγόριθμο, πηγή, στιγμιότυπο εκτέλεσης, επεξεργασία συγγραφέα. 68

Πίνακας 5.1. Πίνακας σύγχυσης και μετρικές ταξινόμησης ανά αλγόριθμο 68

Εικόνα 5.3. Χρόνοι inference και υπολογιστικό throughput ανά αλγόριθμο, πηγή, στιγμιότυπο εκτέλεσης, επεξεργασία συγγραφέα. 68

Πίνακας 5.2. Στατιστικά χρόνων inference και εκτιμώμενο throughput ανά αλγόριθμο 68

Εικόνα 5.4. Σύγκριση F1 ανά αλγόριθμο, πηγή, παραγόμενο γράφημα αξιολόγησης, επεξεργασία συγγραφέα. 68

Εικόνα 5.5. Σύγκριση μέσου inference time ανά αλγόριθμο, πηγή, παραγόμενο γράφημα αξιολόγησης, επεξεργασία συγγραφέα.

Εικόνα 5.6. Ενδεικτικά παραδείγματα false positives και false negatives από το error_examples.txt, πηγή, στιγμιότυπο οθόνης, επεξεργασία συγγραφέα. 68