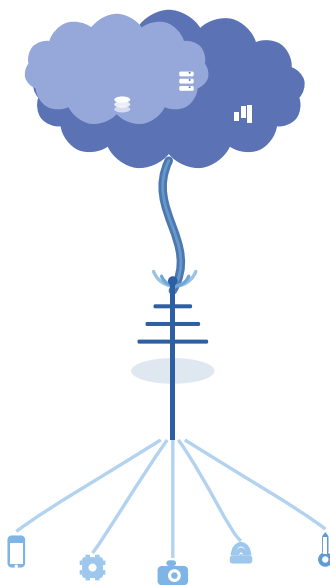


Διεθνές Πανεπιστήμιο Ελλάδος
Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων
ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

**Cloud-Edge-IoT continuum: Επισκόπηση
Τεχνολογιών και Ανάπτυξη Πιλοτικού
Συστήματος**



cloud · edge · iot

Φοιτητής:

Ευάγγελος Αγγέλης
Αριθμός Μητρώου: 518003

Επιβλέπων:

Μάριος Αγγελόπουλος

25 Σεπτεμβρίου 2026

Title of Dissertation Cloud-Edge-IoT continuum: Επισκόπηση Τεχνολογιών και Ανάπτυξη

Πιλοτικού Συστήματος

Code of Dissertation 25360

Student's full name Euaggelos Angelis

Supervisor's full name Marios Angelopoulos

Date of undertaking 1-6-2026

Date of completion 3-9-2026

We hereby affirm the authorship of this paper as well as the acknowledgement and credit of whichever assistance We received in its composition. We have, furthermore, noted the various sources from which We extracted data, ideas, visual or written material, in paraphrase or exact quotation. Moreover, we affirm the exclusive composition of this paper by myself only, for the purpose of it being a dissertation, in the Department of Information and Electronic Engineering of the I.H.U.

This paper constitutes the intellectual property of Euaggelos Angelis the student that composed it. According to the open-access policy, the author/composer offers the International Hellenic University authorisation to use the right to reproduce, borrow, publicly present and digitally distribute the paper globally, in electronic form and media of all kinds, for teaching or research purposes, voluntarily. Open access to the full text, by no means grants the right to trespass the intellectual property of the author/composer, nor does it authorise the reproduction, republication, duplication, selling, commercial use, distribution, publication, downloading, uploading, translation, modification of any kind, in part or summary of the paper, without the explicit written consent of the authors.

The approval of this dissertation by the Department of Information and Electronic Engineering of the International Hellenic University, does not necessarily entail the adoption of the author's views, on behalf of the Department.

Αφιέρωση

Θα ήθελα να ευχαριστήσω τον καθηγητή μου, κ. Μάριο Αγγελόπουλο. Χάρη σε εκείνον μου δόθηκε η δυνατότητα να δουλέψω πάνω σε αυτό το θέμα και να ανακαλύψω νέες πτυχές του εαυτού μου μέσα από την παρούσα διπλωματική εργασία. Θα ήθελα επίσης να τον ευχαριστήσω για τη βοήθεια, την καθοδήγηση αλλά και τη διαθεσιμότητά του καθ' όλη τη διάρκεια αυτής της προσπάθειας, μέσα από τις εβδομαδιαίες μας συναντήσεις.

Παράλληλα, θα ήθελα να ευχαριστήσω την οικογένειά μου, που όλα αυτά τα χρόνια των σπουδών μου βρισκόταν δίπλα μου και με στήριζε αδιάκοπα. Πιο συγκεκριμένα, αφιερώνω αυτή τη διπλωματική εργασία στον αδερφό μου, που ήταν πάντα δίπλα μου και με καθοδηγούσε.

Τέλος, η εργασία αυτή είναι αφιερωμένη σε όλους όσους πάλεψαν και δεν τα παράτησαν στις δύσκολες στιγμές, μέχρι να βρουν αυτό που πραγματικά δίνει νόημα στη ζωή τους.

Πρόλογος

Λόγω της ολοένα και αυξανόμενης τεχνολογίας του Διαδικτύου των Πραγμάτων και της διάδοσης της με ταχύτατους ρυθμούς, έχουν σαν αποτέλεσμα δισεκατομμύρια συσκευές να συνδέονται πλέον σε καθημερινή βάση στο διαδίκτυο, παράγοντας τεράστιους όγκους δεδομένων, αναδεικνύοντας έτσι τα όρια της παραδοσιακής, κεντροποιημένης αρχιτεκτονικής του Cloud Computing. Η υψηλή καθυστέρηση απόκρισης που εισάγει η μεταφορά δεδομένων προς απομακρυσμένα κέντρα δεδομένων, οδήγησε στην ανάπτυξη εναλλακτικών προσεγγίσεων, με κυριότερη αυτή του Edge Computing, το οποίο έχει σαν στόχο να φέρει την επεξεργασία των δεδομένων όσο το δυνατόν πιο κοντά στον τελικό χρήστη.

Η παρούσα διπλωματική εργασία έχει σαν κύριο στόχο την κατανόηση του τρόπου με τον οποίο οι θεωρητικές αρχές του Cloud-Edge-IoT continuum μπορούν να μεταφραστούν σε ένα πραγματικό, λειτουργικό σύστημα. Η προσέγγιση αυτή προκύπτει από το γεγονός ότι δεν περιορίζεται σε θεωρητική ανάλυση, αλλά επιτρέπει την πρακτική επαλήθευση του τρόπου με τον οποίο αλληλεπιδρούν οι επιμέρους τεχνολογίες σε πραγματικές συνθήκες λειτουργίας.

Τα κεφάλαια που ακολουθούν αναπτύσσουν σε βάθος αυτό το θέμα. Στο Κεφάλαιο 1 γίνεται αναφορά στο θεωρητικό υπόβαθρο του Cloud και Edge Computing, καθώς και στις τεχνολογίες που έπαιξαν καθοριστικό ρόλο στην εξέλιξη τους. Στο Κεφάλαιο 2 γίνεται περιγραφή της αρχιτεκτονικής αλλά και της υλοποίησης ενός συστήματος τριών επίπεδων Cloud-Edge-IoT: Το επίπεδο IoT που περιλαμβάνει αισθητήρες για την συλλογή μετρήσεων, το επίπεδο Edge υπεύθυνο για την τοπική επεξεργασία μέσω μοντέλων μηχανικής μάθησης και το επίπεδο Cloud για την αποθήκευση και οπτικοποίηση των δεδομένων σε πραγματικό χρόνο. Το κεφάλαιο 3 επικεντρώνεται στο κομμάτι της μηχανικής μάθησης και διασύνδεσης των επιμέρους στοιχείων του συστήματος. Στο Κεφάλαιο 4 παρουσιάζονται τα αποτελέσματα από τις δοκιμές, ενώ στο Κεφάλαιο 5 συνοψίζονται τα συμπεράσματα και διατυπώνονται προτάσεις για μελλοντική επέκταση του συστήματος.

Ελπίζω η παρούσα διπλωματική εργασία να αποτελέσει μια χρήσιμη αλλά και πρακτική αναφορά για όποιον ενδιαφέρεται να μελετήσει η ακόμη και να υλοποίηση αντίστοιχα συστήματα, τονίζοντας την αξία της πρακτικής εφαρμογής των θεωρητικών αρχών του Cloud-Edge-IoT Continuum.

Abstract

This thesis presents the evolution of Cloud and Edge Computing technologies, driven by the rapid growth and widespread adoption of IoT. The introduction of Cloud Computing enabled users and organizations alike to access resources on demand, at any time, without the need to maintain their own data center. High response latency, along with several other limitations, led to the reasons behind the development of Edge Computing technology. The objective of this thesis is to implement and validate the principles of the Cloud-Edge-IoT Continuum through a fully functional system. To achieve this objective, an architecture based on these three layers was designed and implemented: sensors at the IoT layer collect measurements, which are then processed locally at the Edge layer through machine learning models, before being stored and visualized at the Cloud layer in real time. In conclusion, this thesis confirms that the Cloud-Edge-IoT continuum architecture can be implemented using low-cost equipment, providing a foundation for the development of similar systems.

Keywords: Cloud Computing, Edge Computing, IoT, Machine Learning, Cloud-Edge-IoT Continuum.

Περίληψη

Η παρούσα διπλωματική εργασία παρουσιάζει την εξέλιξη των τεχνολογιών του Cloud και Edge Computing λόγω της ραγδαίας ανάπτυξης και της ευρείας διάδοσης του IoT. Η εισαγωγή του Cloud Computing έδωσε την δυνατότητα στον χρήστη αλλά και σε οποιαδήποτε οργανισμό να έχει πρόσβαση σε πόρους οποιαδήποτε στιγμή επιθυμεί, χωρίς να είναι απαραίτητο να διαθέτει δικό του κέντρο δεδομένων. Η υψηλή καθυστέρηση απόκρισης καθώς και πολλοί άλλοι περιορισμοί, οδήγησαν στους λόγους για τους οποίους αναπτύχθηκε η τεχνολογία του Edge Computing. Στόχος της εργασίας είναι να υλοποιηθούν και να επαληθευτούν οι αρχές του Cloud-Edge-IoT Continuum μέσω ενός πλήρως λειτουργικού συστήματος. Για την επίτευξη αυτού του στόχου σχεδιάστηκε και υλοποιήθηκε μια αρχιτεκτονική βασισμένη στα τρία αυτά επίπεδα, καθώς οι αισθητήρες που βρίσκονται στο επίπεδο IoT συλλέγουν μετρήσεις, όπου και γίνεται η επεξεργασία τους τοπικά στο επίπεδο Edge μέσω μοντέλων μηχανικής μάθησης, όπου και ύστερα αποθηκεύονται αλλά και απεικονίζονται στο επίπεδο Cloud σε πραγματικό χρόνο.

Συμπερασματικά, η παρούσα εργασία επιβεβαιώνει ότι η αρχιτεκτονική του Cloud-Edge-IoT continuum μπορεί να υλοποιηθεί με εξοπλισμό χαμηλού κόστους και δίνει την βάση για την υλοποίηση αντίστοιχων συστημάτων.

Λέξεις-κλειδιά: Cloud Computing, Edge Computing, IoT, Μηχανική Μάθηση, Cloud-Edge-IoT Continuum.

Περιεχόμενα

1	Εισαγωγή	1
1.1	Cloud computing	1
1.1.1	Ορισμός	1
1.2	Ιστορική Αναδρομή	2
1.2.1	Χαρακτηριστικά του Cloud Computing	4
1.3	Μοντέλα υπηρεσιών Cloud Computing	4
1.3.1	Υποδομή ως Υπηρεσία (IaaS)	5
1.3.2	Πλατφόρμα ως Υπηρεσία (PaaS)	6
1.3.3	Λογισμικό ως Υπηρεσία (SaaS)	6
1.3.4	Σύγκριση των υπηρεσιών Cloud	6
1.3.5	Παρόχους Υπηρεσιών Cloud	7
1.4	Cloud Deployment Models	7
1.4.1	Public Cloud	8
1.4.2	Private Cloud	8
1.4.3	Community Cloud	9
1.4.4	Hybrid Cloud	10
1.4.5	Σύγκριση των μοντέλων υλοποίησης Cloud	10
1.4.6	Σχέσεις μεταξύ των μοντέλων υλοποίησης και των μοντέλων υπηρεσιών cloud	11
1.5	Θετικά και Αρνητικά του Cloud Computing	12
1.5.1	Πλεονεκτήματα του Cloud Computing	13
1.5.2	Αδυναμίες του Cloud Computing	14
1.6	Edge Computing	14
1.6.1	Ορισμός	14
1.6.2	Ιστορική Αναδρομή	15
1.7	Η Αναγκαιότητα του Edge Computing	17
1.8	Σχετικές Υπολογιστικές Τεχνολογίες και Αρχιτεκτονικές	18
1.8.1	Κινητό Υπολογιστικό Νέφος	18
1.8.2	Cloudlet	19
1.8.3	Fog Computing	20
1.8.4	Υπολογιστική Άκρου Πολλαπλής Πρόσβασης	21
1.8.5	Πλεονεκτήματα & Μειονεκτήματα	22
2	Αρχιτεκτονική Και Υλοποίηση	23
2.1	Επισκόπηση Αρχιτεκτονικής του Συστήματος	23

2.2	IoT Layer	24
2.2.1	Αισθητήρες και Συνδεσμολογία	24
2.2.2	Arduino Uno	27
2.2.3	Δικτύωση: XBee S6B (ρύθμιση μέσω XCTU)	29
2.3	Edge Layer	31
2.3.1	Khadas VIM3	31
2.3.1.1	Διαμόρφωση Ασύρματου Δικτύου (Hotspot) και Δρομολόγησης	31
2.3.2	Λήψη Δεδομένων (UDP)	33
2.4	Cloud Layer	33
2.4.1	Azure VM — Υποδομή και Εικονικοποίηση	33
2.4.2	Docker Containers (FastAPI, PostgreSQL, Grafana, Nginx)	34
2.4.3	Ασφάλεια & Πρόσβαση (JWT, DuckDNS, SSL)	36
2.4.4	CI/CD Pipeline	37
3	Μηχανική Μάθηση Και Διασύνδεση	38
3.1	Δεδομένα	38
3.2	Scikit-Learn	39
3.2.1	Μεθοδολογία για το Occupancy Detection Dataset	40
3.2.2	Μεθοδολογία για το Smoke Detection Dataset	43
3.2.3	Εξαγωγή τελικών μοντέλων	45
3.3	Εκτέλεση Μοντέλων σε Πραγματικό Χρόνο & Ανάλυση Κώδικα Εκτέλεσης Μο- ντέλων	46
3.3.1	Ενοποιημένος Κώδικας Εκτέλεσης Μοντέλων	46
3.4	Cloud: Docker Ορχήστρωση και Διασύνδεση	47
3.4.1	Ανάλυση των Services	47
3.4.2	Cloud: Αποθήκευση Δεδομένων στη Βάση Δεδομένων	48
3.4.3	Παρακολούθηση σε Πραγματικό Χρόνο (Grafana)	49
4	Αποτελέσματα	53
4.1	Αξιολόγηση Μοντέλων Μηχανικής Μάθησης	53
4.1.1	Occupancy Detection	53
4.1.2	Smoke Detection	56
4.2	Μοντέλο Ανίχνευσης Κατάληψης Χώρου (Occupancy Detection)	58
4.2.1	Κατάσταση Ηρεμίας	58
4.2.2	Κατάσταση Ενεργοποίησης	59
4.3	Μοντέλο Ανίχνευσης Καπνού (Smoke Detection)	60
4.3.1	Κατάσταση Ηρεμίας	60
4.3.2	Κατάσταση Ενεργοποίησης	61
5	Συμπεράσματα	64
	Βιβλιογραφία	66
	Α' Παρατήματα Κώδικα	69
A'1	Κώδικας για τους Αισθητήρες	69

A'.1.1	MQ-135	69
A'.1.1.1	Βαθμονόμηση	69
A'.1.1.2	Υπολογισμός συγκέντρωσης CO2	69
A'.1.2	Πλήρες Sketch Καταγραφής Μετρήσεων	70
A'.2	Machine Learnig	72
A'.2.1	Ocupancy ML	72
A'.2.1.1	See if the test and traing data are the same	74
A'.2.1.2	Coeficientes	75
A'.2.1.3	Split the dataset to the features and the labes	77
A'.2.1.4	Preprosering the data	77
A'.2.1.5	Feature elimination	79
A'.2.1.6	Model Training	81
A'.2.1.7	Compare the results for the two algorithms	81
A'.2.1.8	Tunig params	83
A'.2.1.9	Use the best params and train the final model	85
A'.2.2	Smoke ML	88
A'.2.2.1	Drop features	90
A'.2.2.2	Split the dataset	91
A'.2.2.3	Run algortithms	92
A'.2.2.4	Tuning Params	96
A'.3	Data Collection	101
A'.4	Data Presentation	107

Κατάλογος σχημάτων

1.1	Cloud[2].	1
1.2	Πυραμίδα μοντέλων υπηρεσιών Cloud.	5
1.3	Τα τέσσερα βασικά μοντέλα Cloud (Cloud Deployment Models).	7
1.4	Χρήση υπηρεσιών Cloud από διαφορετικούς παρόχους.	8
1.5	Κοινή χρήση πόρων από συνεργαζόμενους οργανισμούς.	9
1.6	Δομή Hybrid Cloud με συνδυασμό ιδιωτικού και δημόσιου περιβάλλοντος.	10
1.7	Σχέσεις μεταξύ των μοντέλων υλοποίησης και των μοντέλων υπηρεσιών cloud.	12
1.8	Υπολογιστικό μοντέλο Edge Computing.	15
1.9	Αρχιτεκτονική συστήματος Cloudlet	20
2.1	Αρχιτεκτονική Συστήματος σε Τρία Επίπεδα (IoT, Edge, Cloud).	23
2.2	Διάταξη αισθητήρων πάνω στο Arduino Uno με το Wireless SD Shield και το XBee S6B.	25
2.3	Ο αισθητήρας MQ-135 [21].	25
2.4	Ο αισθητήρας BH1750 [22].	26
2.5	Ο αισθητήρας AM2302 [23].	26
2.6	Ο αισθητήρας SGP40 [24].	27
2.7	Arduino Uno Rev3 [25].	27
2.8	XBee S6B WiFi Module [26].	30
2.9	Το Khadas VIM3 [27].	31
2.10	Επισκόπηση της εικονικής μηχανής (VM) στο Microsoft Azure Portal.	34
2.11	Επισκόπηση των endpoints του API μέσω Swagger UI.	35
2.12	Επιτυχής απάντηση (200 OK) του endpoint /occupancy, με πραγματικά δεδομένα από τη βάση.	35
2.13	Το καταχωρημένο domain στο DuckDNS.	36
2.14	Επιτυχής εκτέλεση του CI/CD pipeline μέσω GitHub Actions.	37
3.1	Χάρτης επιλογής αλγορίθμων της βιβλιοθήκης Scikit-Learn.	39
3.2	Αποτελέσματα του RFECV.	41
3.3	Αποτελέσματα του Confusion Matrix με τις βέλτιστες παραμέτρους	43
3.4	Το τελικό Grafana <i>dashboard</i> με τα οκτώ <i>panels</i> παρακολούθησης σε πραγματικό χρόνο.	50
4.1	Σύγκριση των αλγορίθμων χρησιμοποιώντας το Confusion matrix	54
4.2	Αποτελέσματα του Confusion Matrix με τις βέλτιστες παραμέτρους	54

4.3	Αποτελέσματα από τον Confusion Matrix.	55
4.4	Αποτελέσματα των αλγορίθμων χρησιμοποιώντας το Confusion Matrix	56
4.5	Σύγκριση των αλγορίθμων χρησιμοποιώντας το Confusion matrix στο trainigset .	57
4.6	Αποτελέσματα του RandomForest στο testset	57
4.7	Αποτελέσματα του μοντέλου occupancy detection σε κατάσταση ηρεμίας (κενός χώρος, φώτα κλειστά).	58
4.8	Αποτελέσματα του μοντέλου occupancy detection σε κατάσταση ενεργοποίησης (έκθεση του αισθητήρα φωτός σε φακό κινητού τηλεφώνου).	59
4.9	Αποτελέσματα του μοντέλου smoke detection σε κατάσταση ηρεμίας (απουσία καπνού/φωτιάς), με τον ενοποιημένο κώδικα εκτέλεσης.	60
4.10	Αποτελέσματα του μοντέλου smoke detection σε κατάσταση ενεργοποίησης (ανίχνευση φωτιάς μέσω υπερθέρμανσης αισθητήρα).	61
4.11	Ειδοποίηση (alert) ενεργοποίησης (<i>firing</i>) που στάλθηκε μέσω <i>email</i> κατά την ανίχνευση φωτιάς.	62
4.12	Ειδοποίηση (alert) επίλυσης (<i>resolved</i>) που στάλθηκε μέσω <i>email</i> μετά την επιστροφή σε κανονική κατάσταση.	63

Κατάλογος πινάκων

1.1	Cloud Computing [3]	2
1.2	Ιστορικά ορόσημα του Cloud Computing.	3
1.3	Σύγκριση Υπηρεσιών Cloud [6].	6
1.4	Παροχή Υπηρεσιών Cloud Comptunig [7].	7
1.5	Εσωτερική χρήση υπηρεσιών μέσω ιδιωτικού cloud.	9
1.6	Κύρια πλεονεκτήματα των μοντέλων διάθεσης cloud [9].	11
1.7	Σύγκριση των μοντέλων διάθεσης cloud [9].	11
1.8	Πλεονεκτήματα και μειονεκτήματα του υπολογιστικού νέφους.	13
1.9	Ιστορική αναδρομή του Edge Computing	16
1.10	Στόχοι Ανάπτυξης και Λειτουργικά Οφέλη του MCC	19
1.11	Πλεονεκτήματα και Μειονεκτήματα του Edge Computing	22
2.1	Αισθητήρες της τερματικής συσκευής IoT.	24
2.2	Συνδεσμολογία αισθητήρων στο Arduino Uno.	24
2.3	Βασικές παράμετροι ρυθμίσεων του XBee S6B στο XCTU.	30
3.1	Συσχέτιση των χαρακτηριστικών ως προς Occurancy χρησιμοποιώντας τους Pearson και Kendall correlation.	40
3.2	Μέσος όρος (Mean) και Τυπική Απόκλιση (Standard Deviation) των χαρακτηριστικών πριν και μετά την εφαρμογή της κλιμάκωσης (scaling).	41
3.3	Ο χώρος αναζήτησης (Parameter Grid) για τη βελτιστοποίηση των υπερπαραμέτρων του αλγορίθμου LinearSVC.	42
3.4	Ο χώρος αναζήτησης (Parameter Grid) για τη βελτιστοποίηση των υπερπαραμέτρων του αλγορίθμου KNeighborsClassifier.	44
3.5	Ο χώρος αναζήτησης (Parameter Grid) για τη βελτιστοποίηση των υπερπαραμέτρων του αλγορίθμου RandomForestClassifier.	44
3.6	Οι βέλτιστες τιμές των υπερπαραμέτρων που προέκυψαν για τους αλγόριθμους KNeighborsClassifier και RandomForestClassifier.	45
3.7	Δομή του πίνακα occupancy στη βάση δεδομένων iotdb.	48
3.8	Δομή του πίνακα smoke στη βάση δεδομένων iotdb.	49
4.1	Αποτελέσματα των αλγορίθμων ανά Fold.	53
4.2	Αναφορά ταξινόμησης (Classification Report) για την αξιολόγηση του μοντέλου.	55
4.3	Αναφορά ταξινόμησης (Classification Report) για την αξιολόγηση του μοντέλου.	57

Κεφάλαιο 1

Εισαγωγή

1.1 Cloud computing

Σε αυτό το κεφάλαιο, θα παρουσιαστεί μια εκτενής ανάλυση σχετικά με τον ορισμό του Υπολογιστικού Νέφους (Cloud Computing) καθώς και τα βασικά χαρακτηριστικά του. Επιπρόσθετα, θα παρατεθεί μια ιστορική αναδρομή καθώς επίσης θα γίνει μια ανάλυση τόσο των μοντέλων υπηρεσιών Cloud (Cloud service models) όσο και τα μοντέλα ανάπτυξης Cloud (Cloud deployment models) και, τέλος, θα επισημανθούν τα πλεονεκτήματα και τα μειονεκτήματα του Cloud Computing.

1.1.1 Ορισμός

Ο ορισμός του Cloud Computing δεν είναι μονοσήμαντος. Σύμφωνα με το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας (NIST): Το Cloud Computing είναι ένα μοντέλο που επιτρέπει την πανταχού παρούσα, βολική και κατ'απαιτηση δικτυακή πρόσβαση σε μια κοινόχρηστη δεξαμενή διαμορφώσιμων υπολογιστικών πόρων (π.χ. δίκτυα, εξυπηρετητές, αποθηκευτικοί χώροι, εφαρμογές και υπηρεσίες), οι οποίοι μπορούν να διατεθούν και να αποδεσμευτούν ταχύτατα με ελάχιστη προσπάθεια διαχείρισης ή αλληλεπίδραση με τον πάροχο της υπηρεσίας [1].



Σχήμα 1.1: Cloud[2].

Ως διακριτό περιβάλλον πληροφορικής, το Cloud διαφοροποιείται από το Δημόσιο Διαδίκτυο, καθώς διαθέτει ένα πεπερασμένο αρχιτεκτονικό όριο, το οποίο βρίσκεται συνήθως υπό ιδιωτική ιδιοκτησία και προσφέρει πρόσβαση στους πόρους ανάλογα την χρήση. Επιπρόσθετα, σε αντίθεση με τον Παγκόσμιο Ιστό (World Wide Web) που εστιάζει στην κατανάλωση περιεχόμενου, οι υποδομές του Cloud είναι αποκλειστικά προσανατολισμένες στην παροχή υπολογιστικών δυνατοτήτων στο παρασκήνιο, λειτουργώντας πάνω σε οποιαδήποτε δομημένο πρωτόκολλο επικοινωνίας όπου επιτρέπει την εξ αποστάσεως διαχείριση τους [3].

Μέσα από την τεχνολογία αυτή, ο χρήστης δεν χρειάζεται να διαθέτει δική του υπολογιστική ισχύ και πόρους, παρά μόνο μια ευρυζωνική σύνδεση προκειμένου να μπορεί να συνδεθεί με τον πάροχο και να αποκτήσει πρόσβαση στις προσφερόμενες υπηρεσίες υπολογιστικού νέφους. Επίσης, ο χρήστης δεν απαιτείται να έχει εξειδικευμένες γνώσεις, καθώς η διαχείριση του περιβάλλοντος και του συστήματος που χρησιμοποιεί αναλαμβάνεται πλήρως από τον πάροχο. Ακόμη, ο ορισμός του NIST αναφέρει ότι: Αυτό το μοντέλο υπολογιστικού νέφους αποτελείται από πέντε βασικά χαρακτηριστικά, τρία μοντέλα υπηρεσιών και τέσσερα μοντέλα ανάπτυξης [3], τα οποία απεικονίζονται στον Πίνακα 1.1.

CLOUD COMPUTING		
Χαρακτηριστικά	DEPLOYMENT MODELS	SERVICE MODELS
Καθολική πρόσβαση μέσω δικτύου	(Private)	Υποδομή ως Υπηρεσία (IaaS)
Μέτρηση υπηρεσίας	(Hybrid)	Λογισμικό ως Υπηρεσία (SaaS)
Κοινή χρήση και ομαδοποίηση πόρων	(Community)	Πλατφόρμα ως Υπηρεσία (PaaS)
Αυτόνομη χρήση πόρων	(Public)	-
Άμεση προσαρμοστικότητα και ελαστικότητα	-	-

Πίνακας 1.1: Cloud Computing [3]

1.2 Ιστορική Αναδρομή

Η τεχνολογία του cloud computing δεν είναι μια πρόσφατη τεχνολογική ανακάλυψη. Οι ρίζες της εντοπίζονται, στην δεκαετία του 1950 και 1960, όταν έκαναν την εμφάνισή τους τα πρώτα κεντρικά υπολογιστικά συστήματα. Αυτή η ιστορική εξέλιξη οδήγησε στην πρώτη επίσημη εισαγωγή του όρου "cloud computing" το 1997. Με το πέρασμα του χρόνου, η τεχνολογία αυτή εξελίχθηκε σε βαθμό τέτοιο, ώστε άρχισε να αξιοποιείται εμπορικά από μεγάλες εταιρίες της τεχνολογίας, όπως η Amazon και η Google, με αποτέλεσμα η παγκόσμια αγορά που σχετίζεται με το Cloud Computing να παρουσιάζει τεράστια ανάπτυξη, ξεπερνώντας τα \$370 δισεκατομμύρια κατά τα προηγούμενα έτη. Στον παρακάτω πίνακα απεικονίζονται τα σημαντικότερα ορόσημα στην εξέλιξη του cloud computing, με την πάροδο του χρόνου, τα οποία προήλθαν από την πηγή. [4]

Έτος	Τεχνολογικό Ορόσημο
1950s	Ο John McCarthy εισάγει τον καταμερισμό χρόνου στα κεντρικά υπολογιστικά συστήματα
1969	Ανάπτυξη του ARPANET
1972	IBM λανσάρει την εικονικοποίηση (virtualization)
1997	Ο όρος Cloud Computing διατυπώθηκε από τον καθηγητή Chellappa
1999	Λανσάρισμα της Salesforce
2006	Λανσάρισμα του Amazon AWS
2008	Κυκλοφορία του GCP και του APP Engine από την Google
2013	Το Docker κάνει την εμφάνισή του
2014	Κυκλοφορία των Kubernetes και AWS Lambda
2020	Η υιοθέτηση του cloud επιταχύνθηκε στη μετα-COVID εποχή
2022	Κυκλοφόρησε το ChatGPT
2025	Ενσωμάτωση AI και Edge στο cloud

Πίνακας 1.2: Ιστορικά ορόσημα του Cloud Computing.

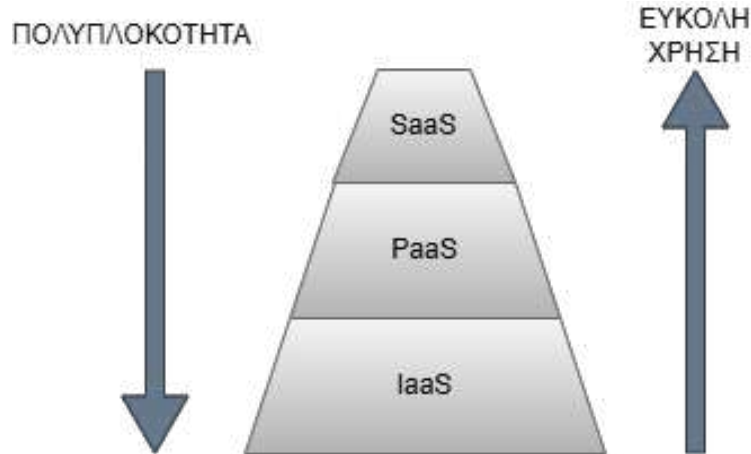
1.2.1 Χαρακτηριστικά του Cloud Computing

Σύμφωνα με το NIST, τα χαρακτηριστικά του Cloud Computing παρατίθενται και περιγράφονται παρακάτω:

- **Αυτόνομη χρήση πόρων:** Σημαίνει ότι ο οποιοσδήποτε που χρησιμοποιεί το Cloud μπορεί να <<δεσμεύσει>> Servers ή αποθηκευτικό χώρο τη στιγμή που τα χρειάζεται, χωρίς να ζητήσει έγκριση από κανέναν. Το σύστημα ανταποκρίνεται αυτόματα στις ανάγκες του, χωρίς να απαιτείται καμιά αλληλεπίδραση με υπαλλήλους της υπηρεσίας [1].
- **Καθολική πρόσβαση μέσω δικτύου:** Το χαρακτηριστικό αυτό εξασφαλίζει ότι το Cloud είναι διαθέσιμο σε κάθε δίκτυο και προσβάσιμο από μια μεγάλη ποικιλία διαφορετικών συσκευών. Είτε πρόκειται για συσκευές χαμηλών απαιτήσεων είτε για υπολογιστές με μεγάλη επεξεργαστική ισχύ, όπου η πρόσβαση γίνεται χωρίς κανένα πρόβλημα, χάρη σε κοινά πρωτοκολλά σύνδεσης.
- **Κοινή χρήση και ομαδοποίηση πόρων:** Στο μοντέλο αυτό, οι υπολογιστικοί πόροι του παρόχου μοιράζονται σε πολλούς πελάτες ταυτόχρονα, χωρίς όμως ο ένας να επηρεάζει τον άλλον. Επιπρόσθετα, το σύστημα μοιράζει αυτόματα τη μνήμη, το Bandwidth και την επεξεργαστική ισχύ ανάλογα με τη ζήτηση της στιγμής. Επίσης, ένα βασικό χαρακτηριστικό είναι η ανεξαρτησία της τοποθεσίας, όπου ο πελάτης δεν είναι υποχρεωμένος να γνωρίζει σε ποιον Server βρίσκονται τα δεδομένα του, άλλα όμως έχει την δυνατότητα να ορίσει την τοποθεσία σε γενικότερο επίπεδο, επιλέγοντας για παράδειγμα μια συγκεκριμένη χωρά ή περιφέρεια.
- **Άμεση προσαρμοστικότητα και ελαστικότητα:** Το σύστημα έχει την δυνατότητα να αυξάνει ή να μειώνει τις υπολογιστικές του ικανότητες πολύ γρήγορα και συχνά εντελώς αυτόματα ανάλογα με τις εκάστοτε ανάγκες. Με αυτόν τον τρόπο, δίνεται στον χρήστη η αίσθηση ότι οι διαθέσιμοι πόροι είναι πρακτικά ανεξάντλητοι, επιτρέποντας του να αποκτά όση ισχύ ή αποθηκευτικό χώρο χρειάζεται, ακριβώς τη στιγμή που την θέλει, χωρίς κανέναν περιορισμό στην ποσότητα.
- **Μέτρηση υπηρεσίας:** Το χαρακτηριστικό αυτό αφορά την αυτόματη μέτρηση και διαχείριση των πόρων που χρησιμοποιούνται. Ανάλογα με την υπηρεσία (π.χ. Bandwidth, μνήμη ή λογαριασμοί χρηστών), το σύστημα καταγράφει την ακριβή δραστηριότητα. Με αυτόν τον τρόπο δημιουργούνται σαφείς αναφορές χρήσης, διασφαλίζοντας απολυτή διαφάνεια στις χρεώσεις και στη διαχείριση τόσο για την ίδια την εταιρεία όσο και για τον τελικό χρήστη.

1.3 Μοντέλα υπηρεσιών Cloud Computing

Τα μοντέλα υπηρεσιών του Cloud διαμορφώνονται από τον συνδυασμό των ψηφιακών πόρων που διαθέτει κάθε πάροχος, προσφέροντας πρόσβαση μέσω ίντερνετ σε τρεις βασικές και ευρέως διαδεδομένες κατηγορίες. Το κόστος αυτόν των υπηρεσιών καθορίζεται με βάση τις ανάγκες και την συμφωνία που γίνεται ανάμεσα στον πελάτη και την εταιρεία. Οι κατηγορίες αυτές είναι το Λογισμικό ως υπηρεσία (Software as a Service), Πλατφόρμα ως υπηρεσία (Platform as a Service) και η Υποδομή ως Υπηρεσία (Infrastructure as a Service).



Σχήμα 1.2: Πυραμίδα μοντέλων υπηρεσιών Cloud.

Όπως φαίνεται και στο σχήμα 1.2, η πολυπλοκότητα των συστημάτων μεγαλώνει, ενώ όσο ανεβαίνουμε προς την κορυφή, οι υπηρεσίες γίνονται πιο άπλες στη χρήση [5]. Δομικά, το SaaS στηρίζεται πάνω στο PaaS, το οποίο με την σειρά του λειτουργεί πάνω από το IaaS. Παρόλα αυτά, η εξέλιξη της τεχνολογίας και η μαζική στρόφη των οργανισμών προς το Cloud οδήγησαν στην δημιουργία νέων παραλλαγών. Μερικά χαρακτηριστικά παραδείγματα είναι:

- Αποθήκευση ως Υπηρεσία (Storage as a Service)
- Υποδομές Πληροφορικών Συστημάτων ως Υπηρεσία (Data Center as a Service)
- Ασφάλεια Πληροφοριακών Συστημάτων ως Υπηρεσία (Security as a Service)
- Λειτουργίες ως Υπηρεσία (Function as a Service)

1.3.1 Υποδομή ως Υπηρεσία (IaaS)

Στο μοντέλο IaaS, ο πάροχος προσφέρει στους χρήστες πρόσβαση σε εικονικούς υπολογιστικούς πόρους, δημιουργώντας ένα ανεξάρτητο περιβάλλον που είναι άμεσα διαθέσιμο για χρήση. Το μεγάλο πλεονέκτημα για τις επιχειρήσεις είναι ότι μπορούν να στήσουν και να τρέξουν τις εφαρμογές τους χωρίς να χρειάζεται να επενδύσουν στην αγορά ακριβών Servers ή στη δημιουργία δικών τους υποδομών και κέντρα δεδομένων. Αντίθετα, νοικιάζουν τις υποδομές του παρόχου και πληρώνουν μόνο για το χρονικό διάστημα που τις χρησιμοποιούν, μετατρέποντας τα μεγάλα αρχικά έξοδα αγοράς σε τρέχοντα λειτουργικά έξοδα. Τέλος, επειδή οι πόροι παραδίδονται σε <<πρωτογενή>> μορφή δηλαδή χωρίς προεγκατεστημένα προγράμματα ή ρυθμίσεις, δίνουν την δυνατότητα στο χρήστη να διατηρεί την πλήρη ελευθερία, αλλά και την ευθύνη, να τους διαμορφώσει και να τους διαχειριστεί όπως ο ίδιος επιθυμεί.

1.3.2 Πλατφόρμα ως Υπηρεσία (PaaS)

Χτισμένη απευθείας πάνω στις εικονικές υποδομές του IaaS, η συγκεκριμένη κατηγορία πηγαίνει την παροχή υπηρεσιών ένα βήμα παραπέρα. Στην περίπτωση αυτή, ο πάροχος εφοδιάζει τον χρήστη με ένα πλήρες πακέτο από γλώσσες προγραμματισμού, έτοιμα εργαλεία λογισμικού, την Διεπαφή Προγραμματισμού Εφαρμογών (APIs) και έναν ενδιάμεσο λογισμικό ανάπτυξης. Αυτό απαλλάσσει τον προγραμματιστή ή την επιχείρηση από το βάρος της διαχείρισης των πόρων και της αρχικής παραμετροποίησης του περιβάλλοντος. Αντίθετα, η προσοχή στρέφεται αποκλειστικά στη σχεδίαση και την υλοποίηση εξατομικευμένων εφαρμογών, καθώς ο πάροχος αναλαμβάνει πλήρως τη συντήρηση του λειτουργικού συστήματος, των Servers και των βάσεων δεδομένων.

1.3.3 Λογισμικό ως Υπηρεσία (SaaS)

Στο υψηλότερο επίπεδο της αρχιτεκτονικής του Cloud, οι χρήστες έχουν άμεση πρόσβαση σε έτοιμες εφαρμογές λογισμικού, χωρίς να χρειάζεται να τις εγκαταστήσουν τοπικά, καθώς η σύνδεση γίνεται απλώς μέσω ενός προγράμματος περιήγησης, μιας ειδικής εφαρμογής ή κάποιου API. Σε αυτό το μοντέλο, ο καταναλωτής δεν έχει σχεδόν κανέναν έλεγχο στις υποδομές ή στους υπολογιστικούς πόρους. Το ψηφιακό περιβάλλον παραδίδεται πλήρως διαμορφωμένο και προκαθορισμένο, γεγονός που σημαίνει ότι ο πελάτης δεν μπορεί να κάνει τεχνικές παραμετροποιήσεις στο σύστημα. Αντίθετα, ο πάροχος του Cloud φέρει την αποκλειστική και πλήρη ευθύνη για την ομαλή λειτουργία, τη συντήρηση και την ασφάλεια ολόκληρης της υποδομής.

1.3.4 Σύγκριση των υπηρεσιών Cloud

Ο Πίνακας 1.3 μας δείχνει ποια τμήματα της υποδομής μπορεί να διαχειρίζεται ο πάροχος και ποια ο πελάτης, ανάλογα με την εκάστοτε υπηρεσία Cloud. Τα μπλε κελιά του πίνακα αφορούν τον πελάτη, ενώ τα πορτοκαλί τον πάροχο.

ON-PREMISES	IaaS	PaaS	SaaS
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
Operation System	Operation System	Operation System	Operation System
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Πίνακας 1.3: Σύγκριση Υπηρεσιών Cloud [6].

Παράλληλα, η κάθε κατηγορία έχει σχεδιαστεί για εντελώς διαφορετικούς χρήστες. Παραδείγματος χάρη, το IaaS το επιλέγουν κυρίως οι διαχειριστές συστημάτων πληροφορικής που θέλουν πλήρη έλεγχο του συστήματος, το PaaS απευθύνεται στους προγραμματιστές για να γράφουν τον κώδικά τους χωρίς να χάνουν χρόνο με τεχνικές ρυθμίσεις, ενώ το SaaS είναι έτοιμο για να το χρησιμοποιήσει αμέσως ο οποιοσδήποτε σαν τελικός χρήστης.

1.3.5 Παρόχους Υπηρεσιών Cloud

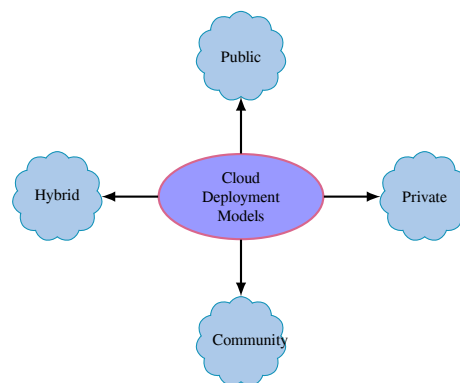
Στον πίνακα που ακολουθεί, παρουσιάζονται οι πιο γνώστες εταιρείες Cloud Computing, χωρισμένες με βάση το μοντέλο υπηρεσιών στο οποίο εξειδικεύεται η καθεμιά.

Πάρoχη Υπηρεσιών Cloud		
IaaS	PaaS	SaaS
Amazon EC2	Amazon Web Services	Amazon Web Services
Bare Metal Compute	Oracle Cloud Paas	Oracle SCM
Microsoft Private Cloud	Microsoft Azure	Microsoft Office 365
Compute Engine	Google App Engine	Google Applications
IBM SoftLayer	Smart Cloud Application Services	SaaS Products

Πίνακας 1.4: Παροχή Υπηρεσιών Cloud Computing [7].

1.4 Cloud Deployment Models

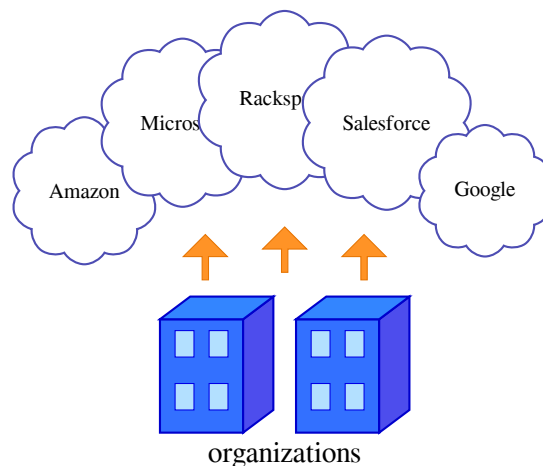
Στον χώρο του Cloud Computing υπάρχουν τέσσερα βασικά μοντέλα υλοποίησης, τα οποία είναι και τα πιο διαδεδομένα στον χώρο αυτό. Η επιλογή τους γίνεται ανάλογα με τις ανάγκες μιας επιχείρησης ή ακόμα και ενός ιδιώτη, τα οποία προσφέρουν και το ανάλογο επίπεδο ιδιωτικότητας, έλεγχο αλλά και ασφάλεια. Αυτές οι κατηγορίες είναι το δημόσιο (Public), το ιδιωτικό (private), το υβριδικό (hybrid) και το Community.



Σχήμα 1.3: Τα τέσσερα βασικά μοντέλα Cloud (Cloud Deployment Models).

1.4.1 Public Cloud

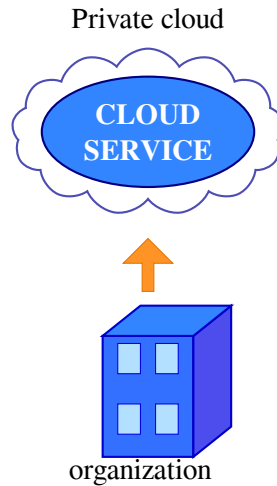
Το Public Cloud αποτελείται από μια σειρά υπηρεσιών που είναι προσβάσιμες μέσω ίντερνετ από τον οποιοδήποτε, από κάθε γωνιά του κόσμου ανά πάσα στιγμή. Η βασική φιλοσοφία στην οποία στηρίζεται, είναι στο ότι πολλοί διαφορετικοί χρήστες μοιράζονται το ίδιο hardware, που σημαίνει ότι είναι σχεδιασμένο για να εξυπηρετεί ταυτόχρονα ένα τεράστιο αριθμό χρηστών και όχι μόνο έναν μεμονωμένο πελάτη. Ο χρήστης μπορεί να δημιουργήσει έναν λογαριασμό στην εταιρεία, να εισάγει τα στοιχεία του και να ξεκινήσει αμέσως τη χρήση. Παρόλο, που οι πόροι μοιράζονται, ο κάθε χρήστης λειτουργεί μέσα σε ένα δικό του, απομονωμένο εικονικό περιβάλλον, εντελώς ξεχωριστό από τους υπόλοιπους. Ο διαχωρισμός αυτός είναι απαραίτητος, ώστε να καταφέρει ο πάροχος να ελέγξει σωστά την δραστηριότητα των χρηστών και να τους εγγυάται τις υψηλές επιδόσεις του συστήματος.



Σχήμα 1.4: Χρήση υπηρεσιών Cloud από διαφορετικούς παρόχους.

1.4.2 Private Cloud

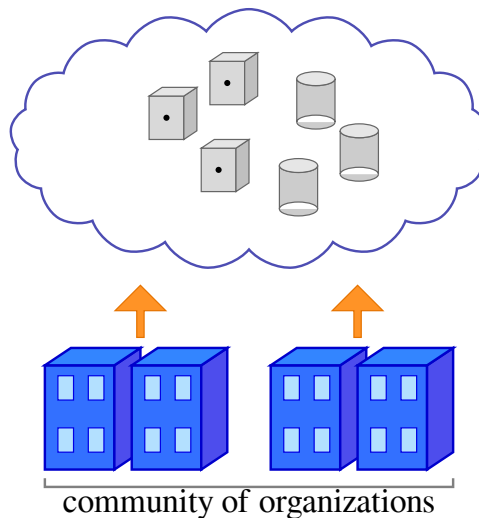
Αυτός ο τύπος Cloud δημιουργείται και χρησιμοποιείται αποκλειστικά από έναν και μόνο οργανισμό ή πελάτη. Πέρα από το ίδιο τον ιδιοκτήτη, εξωτερικοί συνεργάτες όπως για παράδειγμα τρίτες εταιρίες, έχουν πρόσβαση στις υποδομές με μονή προτεραιότητα τη συντήρηση και την ομαλή λειτουργία του συστήματος. Επειδή, η υποδομή αξιοποιείται από μια μόνο εταιρεία, είναι εφικτό να διατηρηθεί ένα εξαιρετικά υψηλό επίπεδο έλεγχου, ιδιωτικότητας άλλα και ασφάλειας, καθώς δικαίωμα στην πρόσβαση των δεδομένων έχουν αποκλειστικά και μόνο τα μέλη του συγκεκριμένου οργανισμού.



Πίνακας 1.5: Εσωτερική χρήση υπηρεσιών μέσω ιδιωτικού cloud.

1.4.3 Community Cloud

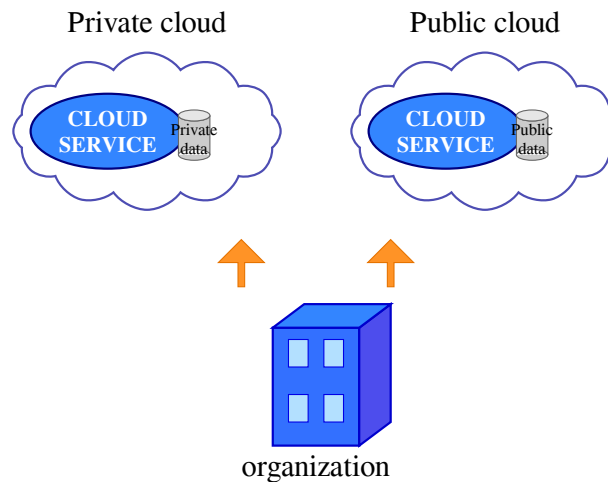
Ο τύπος αυτός αναφέρεται συχνά και ως semi-private cloud, καθώς δημιουργείται και χρησιμοποιείται από μια συγκεκριμένη και προκαθορισμένη ομάδα οργανισμών που μοιράζονται παρόμοιες ανάγκες, κοινούς στόχους και κοινές πολιτικές ασφαλείας (private, public, community) [8]. Στην πραγματικότητα, στήνεται ένα περιβάλλον που λειτουργεί σαν ιδιωτικό Cloud για τα μέλη της ομάδας, ενώ η διαχείριση της υποδομής μπορεί να γίνει είτε από κάποιον από τους ίδιους τους οργανισμούς που συμμετέχουν είτε από έναν εξωτερικό πάροχο.



Σχήμα 1.5: Κοινή χρήση πόρων από συνεργαζόμενους οργανισμούς.

1.4.4 Hybrid Cloud

Ο τύπος αυτός θεωρείται ο πιο σύνθετος, καθώς προκύπτει από τον συνδυασμό δυο ή περισσότερων μοντέλων υλοποίησης. Παρόλο που το κάθε περιβάλλον παραμένει ένα ξεχωριστό, ανεξάρτητο σύστημα, όλα μαζί ενώνονται μεταξύ τους μέσω εξειδικευμένων τεχνολογιών. Η δικτύωση αυτή επιτρέπει στα δεδομένα και στις εφαρμογές να μεταφέρονται με ευκολία από το ένα περιβάλλον στο άλλο.



Σχήμα 1.6: Δομή Hybrid Cloud με συνδυασμό ιδιωτικού και δημόσιου περιβάλλοντος.

Η επιλογή των μοντέλων που θα συνδυαστούν εξαρτάται αποκλειστικά από τις ανάγκες και τις απαιτήσεις του κάθε οργανισμού. Μια συνηθισμένη μορφή ενός Hybrid Cloud είναι ένα ιδιωτικό και ένα δημόσιο Cloud μαζί. Με αυτή την δομή, η επιχείρηση μπορεί να φιλοξενεί τις πιο σημαντικές δραστηριότητες της σε ένα ασφαλές ιδιωτικό Cloud, ενώ χρησιμοποιεί το δημόσιο Cloud για τις λιγότερο κρίσιμες υπηρεσίες ή για τις περιόδους όπου χρειάζεται επιπλέον υπολογιστική ισχύ.

1.4.5 Σύγκριση των μοντέλων υλοποίησης Cloud

Κάθε μοντέλο υλοποίησης Cloud έχει τα δικά του ξεχωριστά πλεονεκτήματα άλλα και χαρακτηριστικά. Για κάθε εταιρεία, η επιλογή του κατάλληλου μοντέλου είναι μια δύσκολη και απαιτητική απόφαση, καθώς η λύση που θα προκύξει πρέπει να καλύπτει απολυτά τις ανάγκες της. Στους παρακάτω πίνακες γίνεται μια αναλυτική σύγκριση, όπου ο πρώτος πίνακας παρουσιάζει τα βασικά πλεονεκτήματα του κάθε μοντέλου, ενώ ο δεύτερος τα βάζει δίπλα-δίπλα, ώστε να γίνει ξεκάθαρο ποιο από αυτά είναι το ιδανικό για την κάθε μια περίπτωση.

Public	Private	Community	Hybrid
Διαθέσιμο για όλους	Μόνο για ιδιωτική χρήση	Ιδανικό για συνεργασία	Εξαιρετικά ευέλικτο
Προσαρμοσμένο	Προσαρμοσμένο	Εξοικονόμηση κόστους	Προσαρμοσμένο
Αξιόπιστο	Αξιόπιστο	Ασφάλεια δεδομένων & ιδιωτικότητα πληροφοριών	Απαιτεί υψηλή τεχνογνωσία cloud
Οικονομικά αποδοτικό	Προσαρμοσμένο		

Πίνακας 1.6: Κύρια πλεονεκτήματα των μοντέλων διάθεσης cloud [9].

	Δημόσιο	Ιδιωτικό	Κοινοτικό	Υβριδικό
Εγκατάσταση & χρήση	Εξωτερική διαχείριση	Απαιτεί επαγγελματίες IT	Απαιτεί επαγγελματίες IT	Απαιτεί επαγγελματίες IT
Έλεγχος δεδομένων	Χαμηλός	Υψηλός	Μέτριος	Μέτριος
Ασφάλεια & ιδιωτικότητα	Χαμηλή	Υψηλή	Μέτρια	Ποικίλλει (Χαμηλή-Υψηλή)
Συνολική αξιοπιστία	Μέτρια	Υψηλή	Μέτρια	Μέτρια έως Υψηλή
Κόστος	Χαμηλό	Υψηλό	Μεταβλητό	Μεταβλητό
Ευελιξία & Προσαρμοστικότητα	Υψηλή	Μέτρια	Μέτρια	Εξαιρετικά Υψηλή
Υλικό (Hardware)	Τρίτου μέρους	Μεταβλητό	Μεταβλητό	Μεταβλητό

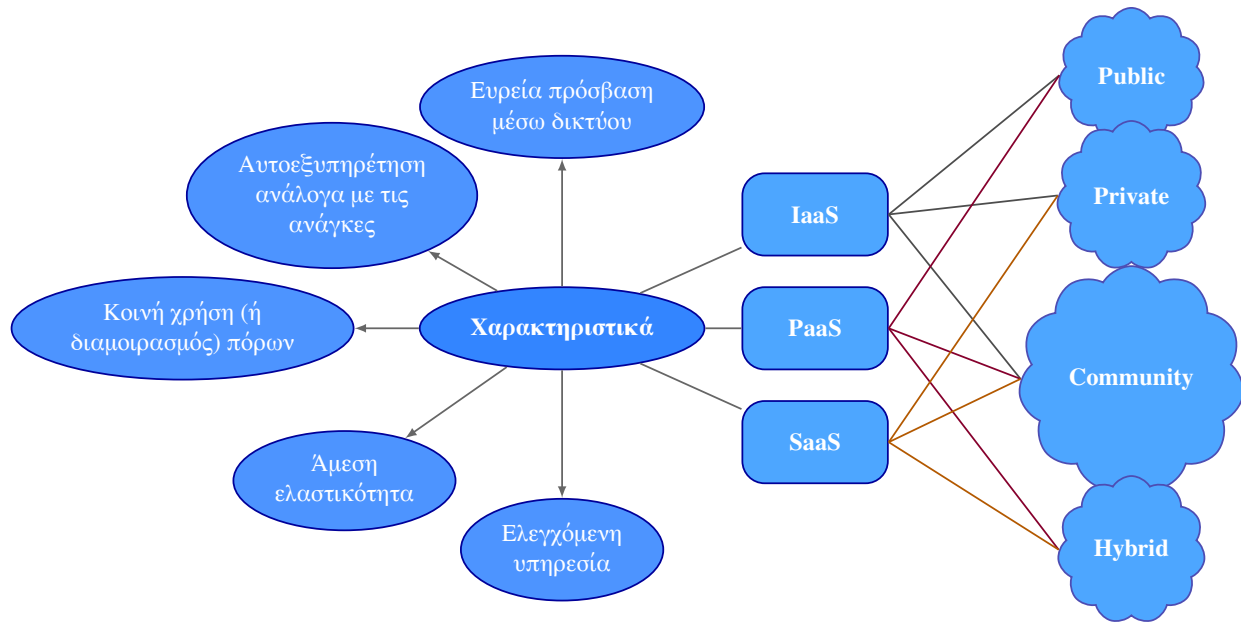
Πίνακας 1.7: Σύγκριση των μοντέλων διάθεσης cloud [9].

1.4.6 Σχέσεις μεταξύ των μοντέλων υλοποίησης και των μοντέλων υπηρεσιών cloud

Τα μοντέλα υπηρεσιών(IaaS,PaaS,SaaS)και τα μοντέλα υλοποίησης δεν λειτουργούν ανεξάρτητα, αλλά συνδέονται στενά μεταξύ τους. Στην πράξη, δεν μπορούν να συνδυαστούν τυφλά όλοι οι τύποι υπηρεσιών με όλα τα περιβάλλοντα Cloud, καθώς υπάρχουν συγκεκριμένοι τεχνικοί περιορισμοί.Πιο αναλυτικά:

- Το IaaS μπορεί να υποστηρίξει ένα Private,Community ή Public Cloud.
- Το PaaS μπορεί να εφαρμοστεί σε ένα Public, Community ή Hybrid Cloud
- Το SaaS μπορεί να λειτουργήσει σε ένα Private, Public ή Hybrid περιβάλλον.

Παρά,τους περιορισμούς αυτούς, οι διαθέσιμες επιλογές για μια επιχείριση παραμένουν αρκετές. Όπως αναφέρθηκε και στο κεφάλαιο 1.1 ο ορισμός του NIST αποτελείται από πέντε βασικά χαρακτηριστικά, τρία μοντέλα υπηρεσιών και τέσσερα μοντέλα υλοποίησης. Όλα αυτά τα στοιχεία δεν λειτουργούν αυτόνομα,αλλά συνδέονται στενά και αλληλεπιδρούν το ένα με το άλλο.



Σχήμα 1.7: Σχέσεις μεταξύ των μοντέλων υλοποίησης και των μοντέλων υπηρεσιών cloud.

Το παραπάνω σχήμα αποτυπώνει αυτή ακριβώς την σύνδεση. Από τον τρόπο που έχει δομηθεί, είναι εύκολα κατανοητό ότι μια στρατηγική Cloud μπορεί να πάρει πολλές διαφορετικές μορφές, ανάλογα με τις ανάγκες της κάθε εταιρείας. Παραδείγματος χάριν, μια επιχείριση μπορεί να δοκιμάσει πιλοτικά έναν συγκεκριμένο συνδυασμό και, αν το αποτέλεσμα είναι ικανοποιητικό, μπορεί είτε να επεκτείνει την χρήση του είτε ακόμα και να δοκιμάσει μια νέα προσέγγιση. Επιπρόσθετα, δίνεται η δυνατότητα να επιστρατευτούν πολλαπλά μοντέλα υλοποίησης για την υποστήριξη μιας ή περισσότερων υπηρεσιών Cloud.

1.5 Θετικά και Αρνητικά του Cloud Computing

Από την εμφάνιση της τεχνολογίας αυτής μέχρι και σήμερα ,έχουν γίνει πολλές έρευνες που καταγράφουν τα θετικά αλλά και τα αρνητικά της στοιχειά. Τα πλεονεκτήματα του Cloud Computing αποτελούν το βασικό κίνητρο για να μεταφέρει μια εταιρεία τις δραστηριότητες της στο Cloud. Σε αντίθετη περίπτωση, η σωστή επισήμανση των μειονεκτημάτων επιτρέπει στις επιχειρήσεις να προετοιμάζονται κατάλληλα και να αναπτύσσουν στρατηγικές, με στόχο την πρόληψη τυχόν προβλημάτων ή ακόμα και απωλειών. Ο παρακάτω πίνακας συνοψίζει ορισμένα από τα κυριότερα πλεονεκτήματα και μειονεκτήματα, τα οποία αναλύονται λεπτομερέστερα στις επόμενες υποενότητες.

Πλεονεκτήματα	Μειονεκτήματα
Μείωση Κόστους	Νομικά Ζητήματα
Αυξημένη Συνεργασία	Ασφάλεια
Ανεξαρτησία Χρόνου και Τοποθεσίας	Μεταφορά Δεδομένων
Αποκατάσταση από Καταστροφή (Disaster Recovery)	

Πίνακας 1.8: Πλεονεκτήματα και μειονεκτήματα του υπολογιστικού νέφους.

1.5.1 Πλεονεκτήματα του Cloud Computing

- **Μειωμένο Κόστος:** Ένα από τα πιο βασικά αλλά και ισχυρά κίνητρα για την μετάβαση μιας επιχείρησης στο περιβάλλον του Cloud Computing είναι η συνολική μείωση των δαπανών που απαιτούνται για την δημιουργία και την ομαλή λειτουργία μιας υποδομής πληροφορικής. Ο λόγος για τον οποίο συμβαίνει αυτό, είναι διότι παύει να υφίσταται η ανάγκη πανάκριβου υλικού εξοπλισμού, όπως servers και συστήματα αποθήκευσης, καθώς και του απαραίτητου λογισμικού που χρειάζονται για την λειτουργία τους. Παράλληλα, με την μεταφορά των δεδομένων στο Cloud, υπάρχει μείωση των λειτουργικών εξόδων που αφορούν τις διαδικασίες συντήρησης των συστημάτων, την τεχνική υποστήριξη, αλλά και την κατανάλωση ηλεκτρικού ρεύματος για την ψύξη και την τροφοδοσία των τοπικών εγκαταστάσεων.
- **Ενισχυμένη συνεργασία:** Όπως αναφέρθηκε αναλυτικά σε προηγούμενο κεφάλαιο, υπάρχουν συγκεκριμένα μοντέλα υλοποίησης του Cloud τα οποία προσφέρουν τη δυνατότητα πρόσβασης και χρήσης των υποδομών τους από περισσότερες από μια επιχειρήσεις ταυτόχρονα. Με αυτόν τον τρόπο, οι εταιρίες αποκτούν ένα εξαιρετικά ισχυρό εργαλείο συνεργασίας. Το πλεονέκτημα αυτό προκύπτει, από τα συμβαλλόμενα μέρη τα οποία μπορούν πλέον να μοιράζονται, να ανταλλάσσουν και να αξιοποιούν δεδομένα, ψηφιακά αρχεία και εξειδικευμένες υπηρεσίες με απίστευτη ταχύτητα και ευκολία, καλύπτοντας έτσι τις αποστάσεις μεταξύ πολλαπλών και διαφορετικών τμημάτων.
- **Ανεξαρτησία Χώρου και Τοποθεσίας:** Ένα από τα πιο σημαντικά οφέλη που προσφέρει η τεχνολογία του cloud computing είναι η πλήρης αποδέσμευση των εταιρικών λειτουργιών από γεωγραφικούς και χρονικούς περιορισμούς. Όλα τα κρίσιμα δεδομένα, οι εφαρμογές και οι πληροφορίες που χρειάζεται μια επιχείρηση για να λειτουργήσει, είναι μόνιμα προσβάσιμα κάθε στιγμή της ημέρας. Πράγμα που σημαίνει, ότι μπορούν να αξιοποιηθούν από οποιοδήποτε μέρος, αρκεί να υπάρχει μια σύνδεση στο ίντερνετ. Η αρχιτεκτονική αυτή, δίνει τη δυνατότητα στο προσωπικό να εργάζεται εξ αποστάσεως με την ίδια ευκολία σαν να βρισκόταν στην εταιρεία. Η συγκεκριμένη ελευθερία κινήσεων βελτιώνει την ισορροπία μεταξύ επαγγελματικής και προσωπικής ζωής στοχεύοντας σε σημαντική ενίσχυση της παραγωγικότητας.
- **Ανάκτηση δεδομένων μετά από Καταστροφή:** Η τεχνολογία του Cloud Computing παρέχει μηχανισμούς για αυτοματοποιημένα και προγραμματισμένα αντίγραφα ασφαλείας σε όλο το δίκτυο, δίνοντας την δυνατότητα να αποθηκεύονται τα δεδομένα σε απομακρυσμένα κέντρα δεδομένων[10].

1.5.2 Αδυναμίες του Cloud Computing

- **Νομικά Ζητήματα:** Η βασική ιδέα του Cloud Computing στηρίζεται στην αποθήκευση δεδομένων και πληροφοριών σε εικονικά κέντρα δεδομένων, τα οποία βρίσκονται διάσπαρτα σε διάφορες τοποθεσίες ανά τον κόσμο. Παρόλα αυτά, υπάρχουν διαφορετικές προσεγγίσεις όσον αφορά τον έλεγχο πρόσβασης, ενώ το γεγονός ότι δεν έχει συμφωνηθεί ακόμη ένα κοινό, παγκόσμιο ρυθμιστικό πλαίσιο αποτελεί καθοριστικό παράγοντα για τον οποίο πολλές εταιρείες αποφεύγουν αυτή την μετάβαση στο Cloud.
- **Ασφάλεια:** Ένας ακόμα πολύ σοβαρός παράγοντας αποθάρρυνσης για την μετάβαση στο Cloud είναι οι ανησυχίες σχετικά με το θέμα της ασφάλειας. Η υποκλοπή ευαίσθητων δεδομένων, τα οποία είναι αποθηκευμένα σε έναν εικονικό διακομιστή αποτελεί έναν από τους μεγαλύτερους κινδύνους αυτής της τεχνολογίας.
- **Μεταφορά δεδομένων:** Στο Cloud Computing, χρησιμοποιείται ένας απομακρυσμένος διακομιστής για την αποθήκευση δεδομένων. Εξαιτίας αυτού, ο χρόνος μεταφοράς αρχείων για μεγάλα πακέτα δεδομένων μπορεί να προκαλέσει δυσκολίες και καθυστερήσεις στους χρήστες.

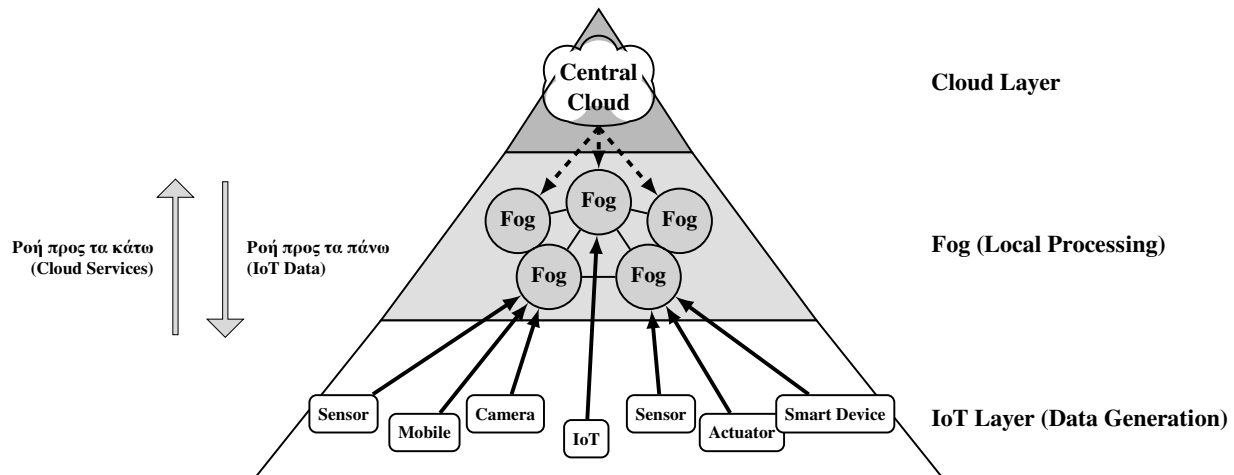
1.6 Edge Computing

Στην παρούσα ενότητα πραγματοποιείται μια αναλυτική παρουσίαση της τεχνολογίας, της Υπολογιστικής Άκρου (Edge Computing) και του ορισμού της. Επιπλέον, επιχειρείται μια ιστορική αναδρομή στην εξέλιξή της, ενώ παράλληλα εξετάζονται διεξοδικά οι επιμέρους τεχνολογίες που τη συνθέτουν. Στη συνέχεια, παρατίθενται οι βασικές διαφορές μεταξύ του Cloud και του Edge Computing και, τέλος, επισημαίνονται τα κυριότερα πλεονεκτήματα καθώς και τα μειονεκτήματα που παρουσιάζει η συγκεκριμένη αρχιτεκτονική.

1.6.1 Ορισμός

Η ραγδαία εξέλιξη και η ευρεία διάδοση του Διαδικτύου των πραγμάτων (Internet of Things-IoT), έχουν επιτρέψει την σύνδεση ενός μεγάλου αριθμού κινητών και σταθερών συσκευών, οι οποίες εξυπηρετούν εφαρμογές που απαιτούν λειτουργία σε πραγματικό χρόνο. Ωστόσο, η αποκλειστική χρήση του Cloud Computing έφερε στο φως σημαντικές προκλήσεις, όπως η υψηλή καθυστέρηση απόκρισης, το αυξημένο κόστος μεταφοράς δεδομένων, την έλλειψη επίγνωσης της γεωγραφικής τοποθεσίας και την περιορισμένη υποστήριξη της κινητικότητας των συσκευών. Προκειμένου να ξεπεραστούν αυτοί οι σημαντικοί περιορισμοί σε μια εποχή όπου οι υπολογιστικές υποδομές αποτελούν αναπόσπαστο κομμάτι της καθημερινότητας μας, αναπτύχθηκαν εναλλακτικές αρχιτεκτονικές, με κυριότερη το Edge Computing.

Με τον όρο Edge Computing, αναφερόμαστε στις τεχνολογίες εκείνες που επιτρέπουν την εκτέλεση υπολογιστικών διεργασιών στο άκρο του δικτύου [11]. Όπως μπορούμε να διακρίνουμε και από το Σχήμα 1.8 οι διεργασίες αυτές αφορούν τόσο τα δεδομένα που κινούνται προς τα κάτω εκ μέρους των υπηρεσιών Cloud, όσο και την ροή δεδομένων προς τα πάνω για λογαριασμό των συσκευών IoT. Πιο συγκεκριμένα, ως <<άκρο>> ορίζουμε οποιονδήποτε υπολογιστικό ή διαδικτυακό πόρο που βρίσκεται στη διαδρομή ανάμεσα στις πηγές των δεδομένων και στα κεντρικά Data Centers του Cloud. Η κεντρική ιδέα του Edge Computing είναι, ότι η επεξεργασία πρέπει να γίνεται όσο το δυνατόν πιο κοντά στην πηγή παραγωγής των δεδομένων.



Σχήμα 1.8: Υπολογιστικό μοντέλο Edge Computing.

Είναι σημαντικό να αναφερθεί ότι, στην επιστημονική βιβλιογραφία ο όρος αυτός χρησιμοποιείται συχνά, εναλλακτικά με τον όρο Fog Computing [12]. Εντούτοις, αξίζει να σημειωθεί ότι η βασική διάφορα είναι ότι το Edge Computing εστιάζει περισσότερο στην πλευρά των ίδιων των συσκευών, ενώ το Fog Computing δίνει μεγαλύτερη έμφαση στην υποδομή του δικτύου.

1.6.2 Ιστορική Αναδρομή

Η εδραίωση του Edge Computing αποτελεί τη φυσική εξέλιξη των κατανεμημένων συστημάτων και των δικτύων παράδοσης περιεχομένου, όπως αναλύεται εκτενώς στην βιβλιογραφία. Οι ρίζες της τεχνολογίας εντοπίζονται στα τέλη της δεκαετίας του 1990 με την δημιουργία των Δικτύων Παράδοσης Περιεχομένου (Content Delivery Networks-CDNs), όπου εισήγαγαν την προσωρινή υποστήριξη της κινητικότητας των συσκευών και της εξοικονόμησης ενέργειας που οδήγησε στην ανάπτυξη των Cloudlets και του Fog Computing, μεταφέροντας σταδιακά τους υπολογιστικούς πόρους από το κεντρικό Cloud προς τις τοπικές συσκευές. Ωστόσο, με την μαζική εμφάνιση του Διαδικτύου των Πραγμάτων, όπου οι τεράστιοι όγκοι δεδομένων κατέστησαν σαφές, ότι οι κεντροποιημένα αρχιτεκτονικά αδυνατούσε να ανταποκριθεί σε εφαρμογές πραγματικού χρόνου. Στη σημερινή εποχή που ζούμε, το Edge Computing αναγνωρίζεται ως ένα αυτόνομο τεχνολογικό πρότυπο, το οποίο, μέσα από τις βασικές τεχνολογίες που το υποστηρίζουν, γεφυρώνει το χάσμα μεταξύ της παραγωγής δεδομένων και της άμεσης, τοπικής επεξεργασίας τους [13] [14].

Χρονολογία	Σταθμός / Ορόσημο	Περιγραφή & Συμβολή στην Αρχιτεκτονική
Τέλη 1990s	Δίκτυα CDNs	Πρώτη απόπειρα μείωσης της γεωγραφικής απόστασης με την προσωρινή αποθήκευση στατικών δεδομένων του Web κοντά στους χρήστες.
2009	Εισαγωγή των Cloudlets	Εμφάνιση «μικρόν-κέντρων δεδομένων» στο άκρο. Χρησιμοποιήθηκαν για την αποφόρτωση υπολογιστικού έργου (computation offloading) από κινητές συσκευές.
2012	Fog Computing (Cisco)	Επέκταση της δομής του Cloud προς το τοπικό δίκτυο. Εισάγει μια ιεραρχική αρχιτεκτονική που περιλαμβάνει και τη διαχείριση της δικτυακής υποδομής.
2014	Multi-access Edge (MEC)	Τυποποίηση από τον ETSI. Έφερε την υπολογιστική ισχύ και τις υπηρεσίες cloud απευθείας μέσα στο δίκτυο κινητής τηλεφωνίας.
2015 - 2016	Εδραίωση Edge Computing	Ο όρος αποκτά αυτόνομη οντότητα στη βιομηχανία. Εστιάζει στην εκτέλεση διεργασιών στην πλευρά των συσκευών για την υποστήριξη του IoT.
Σήμερα	Υποστηρικτικές Τεχνολογίες	Ενσωμάτωση προηγμένων τεχνολογιών (5G/6G, Virtualization, Containers, Τεχνητή Νοημοσύνη) για αυτόνομη λήψη αποφάσεων σε πραγματικό χρόνο.

Πίνακας 1.9: Ιστορική αναδρομή του Edge Computing

1.7 Η Αναγκαιότητα του Edge Computing

Καθώς οι απομακρυσμένοι servers του Cloud είναι τοποθετημένοι σε συγκεκριμένα σημεία, είναι πρακτικά αδύνατο να απορροφήσουν και να επεξεργαστούν, έγκαιρα τον τεράστιο όγκο κίνησης από χιλιάδες έξυπνες συσκευές IoT που βρίσκονται διάσπαρτες ανά τον κόσμο. Για τον λόγο αυτό, η μετακίνηση της υπολογιστικής ισχύος πιο κοντά στις ίδιες τις συσκευές αποτελεί πλέον μια αναπόφευκτη εξέλιξη και όχι μια προαιρετική επιλογή.

Το Edge Computing προέκυψε από ανάγκες, οι οποίες παρουσιάζουν ιδιαίτερο ενδιαφέρον αν μελετηθούν λεπτομερώς: από την μια πλευρά με βάση τις απαιτήσεις των χρηστών για ταχύτητα, και από την άλλη με βάση τις ανάγκες των πάροχων για βέλτιστη διαχείριση του δικτύου. Παρακάτω, γίνεται μια ανάλυση των αιτιών που οδήγησαν στην εμφάνιση του Edge Computing[14]:

- **Real-Time QoS:** Οι σύγχρονες συσκευές μπορούν πλέον να είναι πιο ισχυρές από ποτέ, αλλά οι περιορισμένοι πόροι τους κάνουν πιο δύσκολη την εκτέλεση διεργασιών πραγματικού χρόνου με εγγυημένη Ποιότητα Υπηρεσιών (QoS). Παρόλο που το Cloud Computing αποτελεί μια πολύτιμη λύση προσφέροντας άφθονη επεξεργαστική ισχύ, οι εφαρμογές IoT αντιμετωπίζουν σημαντικό πρόβλημα όσο αναφορά το θέμα της καθυστέρησης. Λόγω, λοιπόν, της ανάγκης για διαδραστικότητα και απόκριση σε πραγματικό χρόνο, οι παραδοσιακές υποδομές Cloud δεν επαρκούν ως μοναδική λύση, εξαιτίας των καθυστερήσεων που προκαλεί η μεταφορά των δεδομένων μέσω του διαδικτύου. Αντίθετα, η τοποθέτηση τοπικών servers κοντά στις συσκευές μειώνει την καθυστέρηση χάρη στο υψηλό εύρος ζώνης του τοπικού δικτύου (LAN).
- **Διάρκεια Ζωής Μπαταρίας:** Η κατανάλωση ενέργειας είναι μία από τις σημαντικότερες παραμέτρους όταν μελετάμε φορητές συσκευές. Έχει παρατηρηθεί ότι, ενώ η επεξεργαστική ισχύς των έξυπνων τηλεφώνων βελτιώνεται σταθερά, η διάρκεια ζωής της μπαταρίας τους δεν ακολουθεί τον ίδιο ρυθμό. Για να αντιμετωπιστεί αυτό το πρόβλημα, μια αποτελεσματική λύση είναι η μεταφορά διεργασιών, το λεγόμενο **Task offloading**, όπου χρησιμοποιείται ως βασική μέθοδος μείωσης της ενεργειακής κατανάλωσης. Το Task offloading πραγματοποιείται είτε προς κεντρικούς διακομιστές Cloud είτε προς τοπικούς edge servers. Παρόλο που η μεταφορά των διεργασιών εκτός συσκευής μειώνει την απαιτούμενη ενέργεια, η χρήση edge servers προσφέρει ακόμη μεγαλύτερη εξοικονόμηση. Συμπερασματικά, η αρχιτεκτονική του Edge Computing είναι απαραίτητη για την προστασία της μπαταρίας των συσκευών, ειδικά όταν εκτελούν εφαρμογές πραγματικού χρόνου.
- **Ρύθμιση της Κίνησης του Κεντρικού Δικτύου:** Ο αυξανόμενος όγκος δεδομένων από δισεκατομμύρια συσκευές IoT, σε συνδυασμό με το περιορισμένο εύρος ζώνης του κεντρικού δικτύου, έχουν ως αποτέλεσμα να προκαλείται συμφόρηση στο δίκτυο. Στο παραδοσιακό μοντέλο, όλη η κίνηση δρομολογείται υποχρεωτικά στο Cloud, με αποτέλεσμα την επιβάρυνση της δικτυακής υποδομής. Αντίθετα, η επεξεργασία των δεδομένων τοπικά στο άκρο απελευθερώνει το κεντρικό δίκτυο και βελτιστοποιεί τη χρήση του bandwidth. Συνεπώς, το Edge Computing επιλύει το πρόβλημα της συμφόρησης σε ολόκληρη τη διαδρομή των δεδομένων με το ελάχιστο δυνατό κόστος.

- **Επεκτασιμότητα:** Λόγω της μεγάλης αύξησης των έξυπνων συσκευών, δημιουργείται πρόβλημα όσον αφορά την επεκτασιμότητα, καθώς η αποστολή τεράστιων όγκων δεδομένων προκαλεί συμφόρηση στις κεντρικές υποδομές δεδομένων του Cloud. Το παραδοσιακό, κεντρικό μοντέλο του Cloud αδυνατεί πλέον να διαχειριστεί αυτόν τον φόρτο εργασίας. Αντίθετα, η κατανομή και η αναπαραγωγή των υπηρεσιών σε εικονικές μηχανές που βρίσκονται σε διάσπαρτους Edge Servers επιτρέπει τη δυναμική δρομολόγηση των αιτημάτων σε γειτονικούς κόμβους όταν ένας Server υπερφορτωθεί. Παράλληλα, η τοπική επεξεργασία των δεδομένων στο edge περιορίζει τον όγκο κίνησης που καταλήγει στο Cloud, απαλλάσσοντας το από τη συνεχή ανάγκη για επέκταση.

1.8 Σχετικές Υπολογιστικές Τεχνολογίες και Αρχιτεκτονικές

Η έλευση του Edge Computing δεν έγινε ξαφνικά, αλλά είναι το αποτέλεσμα μιας σταδιακής μετακίνησης των υπολογιστικών πόρων από το κεντρικό Cloud προς το άκρο του δικτύου. Στην πορεία αυτή αναπτύχθηκαν διάφορες ενδιάμεσες αρχιτεκτονικές, όπου η καθεμία προσπάθησε να λύσει με τον δικό της τρόπο τα προβλήματα της καθυστέρησης, της κατανάλωσης ενέργειας και του bandwidth. Η κατανόηση αυτών των τεχνολογιών, που είτε προηγήθηκαν είτε αναπτύχθηκαν παράλληλα με το Edge Computing, είναι απαραίτητη για να γίνει ξεκάθαρο πώς λειτουργεί και τι προσφέρει το σύγχρονο υπολογιστικό άκρο, όπως θα δούμε στη συνέχεια.

1.8.1 Κινητό Υπολογιστικό Νέφος

Το Κινητό Υπολογιστικό Νέφος (Mobile Cloud Computing - MCC) αποτελεί ουσιαστικά την ενσωμάτωση της τεχνολογίας του υπολογιστικού νέφους στις φορητές συσκευές, με στόχο να τις καταστήσει πλούσιες σε πόρους όσον αφορά την υπολογιστική ισχύ, τη μνήμη, την αποθήκευση, την ενέργεια και την επίγνωση περιβάλλοντος [15].

Στο πλαίσιο αυτό, το MCC συνδυάζει τις τεχνολογίες του Mobile Computing, Mobile Internet και του Cloud Computing σε ένα ενιαίο σύστημα. Στόχος του είναι η μεταφορά των εργασιών από τις περιορισμένες φορητές συσκευές προς ισχυρά, απομακρυσμένα κέντρα δεδομένων, προσφέροντας κατ' απαίτηση υπολογιστικούς πόρους ανεξάρτητα από την τοποθεσία.

Η αρχιτεκτονική οργάνωση του MCC εμφανίζεται με δύο βασικές μορφές:

- **Μοντέλο βασισμένο σε υποδομές (Infrastructure-based MCC):** Αφορά τη χρήση των κλασικών, σταθερών υλικών υποδομών και διακομιστών των υπολογιστικών κέντρων νέφους για την εξυπηρέτηση των χρηστών.
- **Ad-hoc υπολογιστικό νέφος (Ad-hoc mobile cloud):** Είναι ένα συνεργατικό μοντέλο, όπου μια ομάδα γειτονικών φορητών συσκευών λειτουργεί ως ένα τοπικό, προσωρινό «cloud» που μοιράζεται πόρους και πρόσβαση στο διαδίκτυο.

Η σύγχρονη τάση του MCC επιβάλλει τη μεταφορά των υπολογιστικών πόρων οι οποίοι παραδοσιακά βρίσκονταν αποκλειστικά σε απομακρυσμένους διακομιστές πιο κοντά στον τελικό χρήστη. Για να επιτευχθεί αυτό, επιστρατεύεται η χρήση των Cloudlets, τα οποία μεταφέρουν την υπολογιστική ισχύ του Cloud στο τοπικό δίκτυο (LAN). Με τον τρόπο αυτό, προσφέρουν εξαιρετικά χαμηλή καθυστέρηση (latency) και απαλλάσσουν τις συσκευές από την ανάγκη συνεχούς ad-hoc επικοινωνίας. Οι βασικοί στόχοι που οδήγησαν στην ανάπτυξη αυτών των τεχνολογιών, καθώς και τα εγγενή πλεονεκτήματα που προσφέρει το MCC, συνοψίζονται στον Πίνακα 1.10.

Κατηγορία	Βασικοί Στόχοι Ανάπτυξης	Πλεονεκτήματα Παραδοσιακού Cloud
Συσκευή	Επέκταση διάρκειας ζωής μπαταρίας Αντιμετώπιση έλλειψης πόρων (μνήμης & υπολογιστικής ισχύος)	Απελευθέρωση από τη συνεχή σύνδεση σε πηγές τροφοδοσίας
Υποδομή	Εικονικοποίηση πόρων (Virtualization) Υποστήριξη απαιτητικών εφαρμογών	Δυναμική διάθεση πόρων Υψηλή επεξεργαστική επεκτασιμότητα
Διαχείριση	Τοπική επεξεργασία (μέσω Cloudlets)	Διαμοιρασμός πόρων σε πολλούς χρήστες (Multi-tenancy)

Πίνακας 1.10: Στόχοι Ανάπτυξης και Λειτουργικά Οφέλη του MCC

1.8.2 Cloudlet

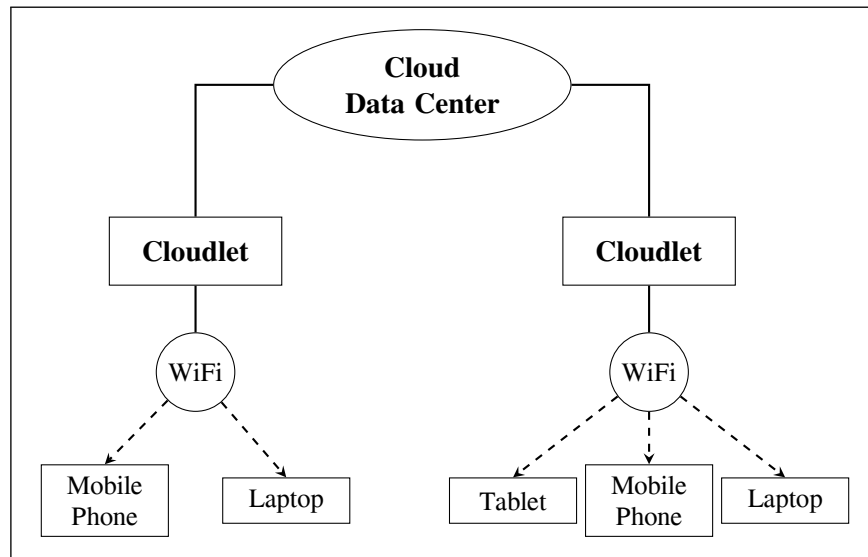
Για την αντιμετώπιση των περιορισμών που επιβάλλει η γεωγραφική απόσταση των κεντρικών υποδομών νέφους, αναπτύχθηκε και εισήχθη η αρχιτεκτονική των Cloudlets. Το Cloudlet ορίζεται ουσιαστικά σαν ένα υπολογιστικό νέφος σε μικρογραφία το οποίο βρίσκεται εγκατεστημένο στο άκρο του δικτύου [16]. Ενώ το παραδοσιακό Cloud προσφέρει τεράστια υπολογιστική ισχύ, η απόσταση του από τον χρήστη προκαλεί μεγάλες καθυστερήσεις λόγω των πολλαπλών διαδικτυακών αλμάτων (multi-hop). Παράλληλα, οι φορητές συσκευές χαρακτηρίζονται από περιορισμένη μνήμη, επεξεργαστική ισχύ και διάρκεια μπαταρίας, με αποτέλεσμα να αδυνατούν να υποστηρίξουν τις απαιτητικές εφαρμογές πραγματικού χρόνου. Για την αντιμετώπιση αυτών των περιορισμών, τα Cloudlets μεταφέρουν τις δυνατότητες του Cloud σε απόσταση ενός μόλις άλματος από τον χρήστη.

Για να μπορέσουμε να κατανοήσουμε καλύτερα το πως επιτυγχάνεται η άμεση εξυπηρέτηση, είναι σημαντικό να συζητηθούν και να αναλυθούν τα επίπεδα της αρχιτεκτονικής του συστήματος. Η δομή αυτή γεφυρώνει το χάσμα ανάμεσα στις συσκευές του χρήστη και το κεντρικό Cloud, όπως αποτυπώνεται σχηματικά στο σχήμα 1.9.

Η αρχιτεκτονική είναι σχεδιασμένη σε τρία διακριτά επίπεδα:

1. **Επίπεδο Φορητών Συσκευών:** Το επίπεδο αυτό περιλαμβάνει τα έξυπνα κινητά τηλέφωνα, τα τάμπλετ και γενικά όλες τις IoT συσκευές που χρειάζονται επιπλέον υπολογιστικούς πόρους για να εκτελούν τις εφαρμογές τους χωρίς να εξαντλήσουν την μπαταρία τους.
2. **Επίπεδο Cloudlet:** Αποτελεί τον ενδιάμεσο σταθμό στο άκρο του δικτύου. Οι συσκευές συνδέονται απευθείας με το Cloudlet μέσω τοπικού ασύρματου δικτύου. Επειδή η απόσταση είναι ελάχιστη, οι εργασίες υπολογίζονται και επιστρέφουν στον χρήστη σε κλάσματα του δευτερολέπτου.

3. **Επίπεδο Κεντρικού Cloud:** Οι servers του Cloud παραμένουν στο παρασκήνιο. Το Cloudlet επικοινωνεί μαζί τους για να συγχρονίσει δεδομένα ή για να τους προωθήσει πολύ βαριές εργασίες που δεν απαιτούν άμεση απόκριση.



Σχήμα 1.9: Αρχιτεκτονική συστήματος Cloudlet .

1.8.3 Fog Computing

Η CISCO εισήγαγε για πρώτη φορά τον όρο Fog Computing ή υπολογιστικό νέφος ομίχλης το 2012, το οποίο αναδύθηκε ως ένα νέο παράδειγμα με σκοπό την επέκταση των δυνατοτήτων του Cloud Computing στο άκρο του δικτύου [17]. Έχει σχεδιαστεί πρωτογενώς για να εξυπηρετήσει τις ανάγκες του περιβάλλοντος IoT, με κύριο στόχο του να είναι η μεταφορά υπολογιστικής ισχύος και πόρων αποθήκευσης εγγύτερα στα σημεία όπου παράγονται τα δεδομένα.

Σε αντίθεση με το παραδοσιακό, κεντρικό μοντέλο του Cloud Computing, η αρχιτεκτονική του Fog δίνει προτεραιότητα στην επεξεργασία δεδομένων απευθείας στις συσκευές του άκρου ή σε ενδιάμεσους κόμβους. Με αυτόν τον τρόπο επιτυγχάνεται το φιλτράρισμα και η τοπική επεξεργασία των πληροφοριών, διασφαλίζοντας ότι μόνο ο ελάχιστος απαραίτητος όγκος δεδομένων μεταδίδεται τελικά στο κεντρικό Cloud.

Είναι σημαντικό να αναφερθεί ότι η τεχνολογία του Fog Computing προσφέρει σημαντικά πλεονεκτήματα, όπως εξαιρετικά χαμηλούς χρόνους απόκρισης, υποστήριξη κινητικότητας, επίγνωση τοποθεσίας και ευρεία ασύρματη πρόσβαση. Επιπρόσθετα, θεωρείται μια πιο αποδοτική προσέγγιση ως προς τη διαχείριση πόρων, καθώς αξιοποιεί τις υπάρχουσες δυνατότητες των συσκευών του άκρου, μειώνοντας την εξάρτηση από βαριές κεντρικές υποδομές υλικού εξοπλισμού και λογισμικού. Αξίζει να σημειωθεί ότι το Fog Computing παρουσιάζει ισχυρές ομοιότητες με την έννοια του Multi-access Edge Computing (MEC), καθώς και τα δύο μοντέλα επιδιώκουν τη βελτιστοποίηση της απόδοσης μέσω της αποκέντρωσης των υπολογιστικών διαδικασιών.

1.8.4 Υπολογιστική Άκρου Πολλαπλής Πρόσβασης

Η ανάγκη για υποστήριξη εφαρμογών με κρίσιμους χρόνους απόκρισης στα δίκτυα πέμπτης γενιάς ανέδειξε τα εγγενή προβλήματα της παραδοσιακής αρχιτεκτονικής MCC. Οι μεγάλες γεωγραφικές αποστάσεις των τελικών χρηστών από τα κέντρα δεδομένων και η μετάδοση όγκου πληροφορίας είχαν σαν αποτέλεσμα μεγάλες καθυστερήσεις και υπερφόρτωση των δικτύων μεταφοράς δεδομένων [18]. Για την αντιμετώπιση του προβλήματος αυτού, η Υπολογιστική Άκρου Πολλαπλής Πρόσβασης - MEC εισήγαγε μια κατανεμημένη αρχιτεκτονική, η οποία μεταφέρει τις υπολογιστικές λειτουργίες, την αποθήκευση και την κυκλοφορία δεδομένων στο άκρο του δικτύου.

Ο όρος αυτός προτάθηκε αρχικά το 2014 από τον Ευρωπαϊκό Οργανισμό Τηλεπικοινωνιακών Προτύπων (ETSI), ο οποίος όρισε την τεχνολογία ως [19]:

«...μια νέα πλατφόρμα που παρέχει δυνατότητες πληροφορικής και υπολογιστικού νέφους εντός του Δικτύου Ραδιοπρόσβασης (RAN), σε άμεση εγγύτητα με τους κινητούς συνδρομητές».

Στην πράξη, το MEC επιτρέπει σε σταθμούς βάσης (Base Stations - BSs) και σε σημεία ασύρματης πρόσβασης (Access Points - APs) να λειτουργούν ως τοπικοί servers, αναλαμβάνοντας την εκτέλεση απαιτητικών εργασιών και απαλλάσσοντας τις τερματικές συσκευές από τον υπολογιστικό φόρτο.

Η υλοποίηση του MEC βασίζεται σε ένα εικονικοποιημένο περιβάλλον το οποίο αξιοποιεί τρεις θεμελιώδεις τεχνολογίες δικτύωσης:

- **Εικονικοποίηση Δικτυακών Λειτουργιών (Network Functions Virtualization - NFV):** Επιτρέπει σε μια συσκευή του άκρου να εξυπηρετεί ταυτόχρονα πολλαπλούς χρήστες μέσω της δημιουργίας Εικονικών Μηχανών (Virtual Machines - VMs).
- **Δίκτυα Καθοριζόμενα από Λογισμικό (Software-Defined Networks - SDN):** Διευκολύνουν τη δυναμική και κλιμακώσιμη διαχείριση των υπηρεσιών μέσω της αφαίρεσης των δικτυακών λειτουργιών.
- **Δίκτυα Κεντροποιημένα στην Πληροφορία (Information-Centric Networks - ICN):** Μετατοπίζουν το παράδειγμα από την αναγνώριση βάσει κόμβου (host-centric) στην αναγνώριση βάσει περιεχομένου, υποστηρίζοντας υπολογισμούς με επίγνωση του περιβάλλοντος.

Είναι σημαντικό να αναφερθεί ότι η έννοια του Fog Computing εισήχθη ως μια γενικευμένη μορφή του Edge Computing, περιλαμβάνοντας ένα ευρύτερο σύνολο συσκευών στο ακρο. Παρά τις αρχικές θεωρητικές διαφοροποιήσεις, οι περιοχές του Fog Computing και του MEC ταυτίζονται σε μεγάλο βαθμό στο πλαίσιο της σύγχρονης βιβλιογραφίας, με αποτέλεσμα οι δύο όροι να χρησιμοποιούνται συχνά ως συνώνυμοι. Ο στόχος των δύο αυτών μοντέλων είναι κοινός. Πιο συγκεκριμένα, επιδιώκουν τη βελτιστοποίηση της απόδοσης, τη δραστική μείωση της καθυστέρησης σε επίπεδο millisecond και την εξοικονόμηση ενέργειας των τελικών συσκευών, καθιστώντας εφικτή την υποστήριξη απαιτητικών εφαρμογών όπως η Επαυξημένη Πραγματικότητα και το Διαδίκτυο των Πραγμάτων.

1.8.5 Πλεονεκτήματα & Μειονεκτήματα

Η χρήση του Edge Computing εξασφαλίζει πολλαπλά οφέλη, ωστόσο συνοδεύεται από προκλήσεις που πρέπει να συνεκτιμηθούν κατά τον σχεδιασμό σύγχρονων δικτυακών υποδομών. Η μεταφορά της υπολογιστικής ισχύος στο άκρο του δικτύου προσφέρει άμεση βελτιστοποίηση στην ταχύτητα επεξεργασίας και ενισχύει την προστασία των δεδομένων, ωστόσο εισάγει περιορισμούς που σχετίζονται με το κόστος και τη διαχείριση των διάσπαρτων πόρων. Στον Πίνακα 1.11 αποτυπώνεται μια συνοπτική συγκριτική ανάλυση των κυριότερων πλεονεκτημάτων και μειονεκτημάτων του Edge Computing, όπως αυτά προκύπτουν μέσα από τη βιβλιογραφία [20].

Πλεονεκτήματα	Μειονεκτήματα
Τοπική Επεξεργασία: Η επεξεργασία των δεδομένων εκτελείται πιο κοντά στην πηγή παραγωγής τους.	Υψηλότερο Κόστος: Απαιτείται σημαντική αρχική επένδυση σε hardware, άδειες λογισμικού και εξειδικευμένο προσωπικό πληροφορικής.
Ενισχυμένη Ιδιωτικότητα: Οι κόμβοι λειτουργούν ως ενδιάμεσο επίπεδο για το φιλτράρισμα και την ανωνυμοποίηση των δεδομένων.	Περιορισμένοι Πόροι: Οι συσκευές του άκρου διαθέτουν σαφώς μικρότερη υπολογιστική ισχύ συγκριτικά με τα κεντρικά Data Centers.
Χαμηλός Φόρτος Δικτύου: Μειώνεται ο όγκος των δεδομένων που μεταδίδονται, εξοικονομώντας εύρος ζώνης.	Περιορισμοί στη Μηχανική Μάθηση: Ανεπαρκής ισχύς για απαιτητικές εργασίες, όπως η εκπαίδευση μεγάλων μοντέλων ML.
Μείωση Καθυστερήσης: Δραστική μείωση στην καθυστέρηση δικτύου, επιτρέποντας τη λήψη αποφάσεων σε πραγματικό χρόνο.	Δυσκολία Κλιμάκωσης: Η επέκταση της υποδομής είναι σύνθετη, καθώς απαιτεί προσθήκη νέου φυσικού εξοπλισμού στο πεδίο.

Πίνακας 1.11: Πλεονεκτήματα και Μειονεκτήματα του Edge Computing .

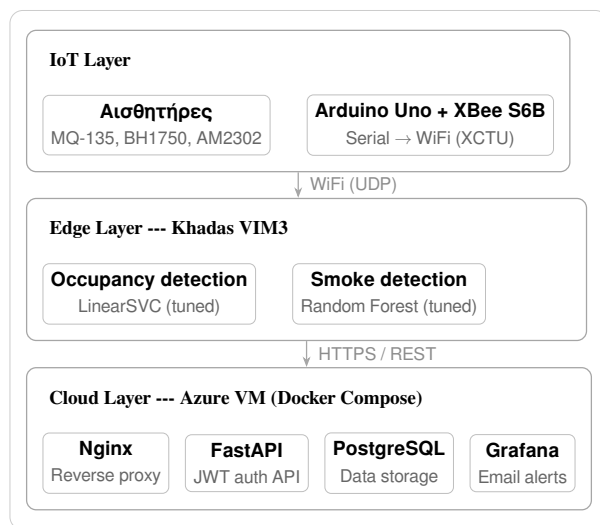
Κεφάλαιο 2

Αρχιτεκτονική Και Υλοποίηση

2.1 Επισκόπηση Αρχιτεκτονικής του Συστήματος

Η υλοποίηση του συστήματος ακολουθεί το μοντέλο του Cloud-Edge-IoT Continuum, οργανωμένο σε τρία διακριτά επίπεδα, καθένα με ξεχωριστό ρόλο στην αλυσίδα από τη συλλογή δεδομένων μέχρι την τελική οπτικοποίηση και ειδοποίηση:

- **IoT Layer:** Συλλογή αναλογικών/ψηφιακών μετρήσεων από το φυσικό περιβάλλον και μετάδοσή τους ασύρματα.
- **Edge Layer:** Τοπική επεξεργασία και παραγωγή προβλέψεων κοντά στην πηγή των δεδομένων, μέσω μοντέλων μηχανικής μάθησης.
- **Cloud Layer:** Κεντρική αποθήκευση, έκθεση API, οπτικοποίηση και ειδοποίηση, αναπλεγμένα σε δημόσια υποδομή cloud.



Σχήμα 2.1: Αρχιτεκτονική Συστήματος σε Τρία Επίπεδα (IoT, Edge, Cloud).

Ο διαχωρισμός αυτός ακολουθεί τη λογική του continuum: Κάθε επίπεδο εκτελεί τις λειτουργίες για τις οποίες είναι υπεύθυνο. Το edge βρίσκεται κοντά στην πηγή των δεδομένων, οπότε προσφέρει μικρότερη καθυστέρηση και στέλνει παρακάτω μόνο ό,τι χρειάζεται. Το cloud, από την άλλη, δίνει μόνιμη αποθήκευση, κεντρική διαχείριση και εύκολη πρόσβαση από οποιοδήποτε σημείο και να είναι. Η συνολική αρχιτεκτονική, παρουσιάζεται στην Εικόνα 2.1.

2.2 IoT Layer

2.2.1 Αισθητήρες και Συνδεσμολογία

Η συσκευή IoT της υλοποίησης χρησιμοποιεί τέσσερις αισθητήρες, όπως φαίνεται στον πίνακα 2.1.

Αισθητήρας	Μέγεθος που μετράει
MQ-135	Ποιότητα αέρα / αέρια (VOC, CO2 κ.ά.)
BH1750	Ένταση φωτισμού (lux)
AM2302	Θερμοκρασία / Υγρασία
SGP40	Δείκτης πτητικών οργανικών ενώσεων (VOC Index)

Πίνακας 2.1: Αισθητήρες της τερματικής συσκευής IoT.

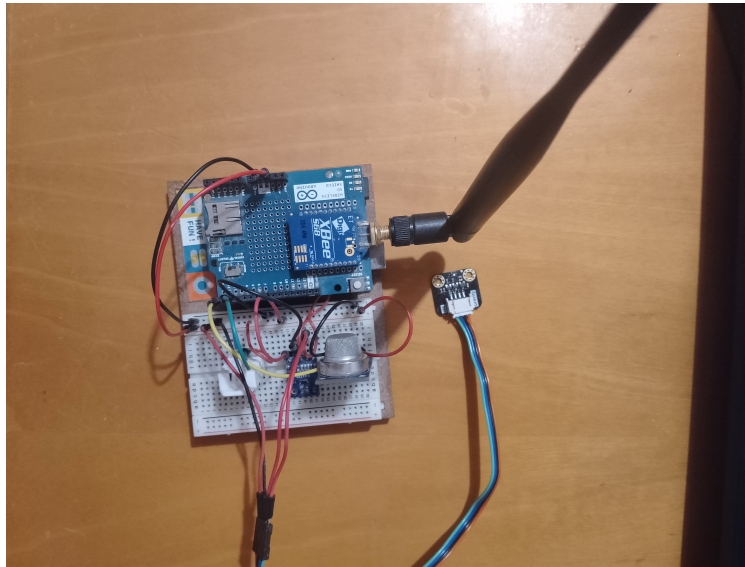
Η συνδεσμολογία κάθε αισθητήρα στο Arduino Uno συνοψίζεται στον Πίνακα 2.2. Το AM2302 συνδέεται σε ψηφιακό pin, το MQ-135 σε αναλογικό pin (καθώς η έξοδός του είναι αναλογική τάση), ενώ το BH1750 επικοινωνεί μέσω του διαύλου I2C, χρησιμοποιώντας τα αντίστοιχα pins SDA/SCL του Arduino Uno.

Αισθητήρας	Σύνδεση στο Arduino Uno
AM2302 (DHT22)	Ψηφιακό pin D2
MQ-135	Αναλογικό pin A0
BH1750	I2C (SDA/SCL = A4/A5)
SGP40	I2C (SDA/SCL = A4/A5, διεύθυνση 0x59)

Πίνακας 2.2: Συνδεσμολογία αισθητήρων στο Arduino Uno.

Αξίζει να σημειωθεί ότι οι παραπάνω αισθητήρες δεν αποτελούν από μόνοι τους "συσκευές IoT". Μια συσκευή IoT ορίζεται τυπικά από τον συνδυασμό τριών ιδιοτήτων: (α) αίσθηση/ενεργοποίηση, (β) στοιχειώδη επεξεργασία, και (γ) ικανότητα δικτυακής επικοινωνίας. Οι αισθητήρες MQ-135, BH1750 και AM2302 είναι **transducers** δηλαδή μετατρέπουν ένα φυσικό μέγεθος σε ηλεκτρικό σήμα, χωρίς δική τους διεύθυνση δικτύου ή επεξεργαστική ικανότητα. Η πλήρης συσκευή IoT σχηματίζεται από το σύνολο Arduino Uno και από το XBee S6B, οι αισθητήρες αυτοί αποτελούν τα "όργανα αντίληψης" αυτής της συσκευής.

Η τοποθέτηση των αισθητήρων πάνω στο Arduino shield φαίνεται στην Εικόνα 2.2.



Σχήμα 2.2: Διάταξη αισθητήρων πάνω στο Arduino Uno με το Wireless SD Shield και το XBee S6B.

MQ-135 (Ποιότητα αέρα). Αισθητήρας ημιαγωγού μετάλλου-οξειδίου, ευαίσθητος κυρίως σε CO₂ και άλλα αέρια/VOC. Η έξοδός του είναι αναλογική τάση που αντιστοιχεί στην αντίσταση του αισθητήριου στοιχείου, η οποία μεταβάλλεται ανάλογα με τη συγκέντρωση αερίων στο περιβάλλον.



Σχήμα 2.3: Ο αισθητήρας MQ-135 [21].

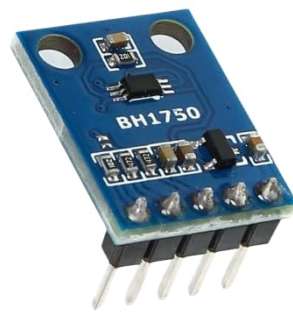
Βαθμονόμηση. Ο αισθητήρας MQ-135 απαιτεί βαθμονόμηση πριν χρησιμοποιηθεί για αξιόπιστες μετρήσεις, καθώς η βασική του αντίσταση (R_0) διαφέρει ανά μονάδα και ανά συνθήκες περιβάλλοντος. Η διαδικασία που ακολουθήθηκε ήταν η εξής:

1. Ο αισθητήρας αφέθηκε εκτεθειμένος σε **καθαρό αέρα** (εξωτερικός χώρος, μπαλκόνι) για περίπου μία ώρα, ώστε να σταθεροποιηθεί.
2. Κατά τη διάρκεια αυτή, εκτελέστηκε ξεχωριστό sketch βαθμονόμησης (Κώδικας **A'.1.1.1**), το οποίο καλεί επαναλαμβανόμενα τη συνάρτηση `getRZero()` της βιβλιοθήκης MQ135, τυπώνοντας την τιμή αναφοράς `RZero` κάθε δευτερόλεπτο.

3. Η τιμή RZero που προέκυψε (≈ 30.0) καταγράφηκε και χρησιμοποιήθηκε ως σταθερά αναφοράς (myRZero) στον τελικό κώδικα του Arduino.

Στον τελικό κώδικα, η βαθμονομημένη τιμή χρησιμοποιείται για τον υπολογισμό της συγκέντρωσης CO₂ σε ppm (Κώδικας A'.1.1.2), ενώ παράλληλα εφαρμόζεται εξομάλυνση σε 10 διαδοχικές μετρήσεις αντίστασης πριν τον τελικό υπολογισμό, ώστε να μειωθεί ο θόρυβος μέτρησης.

BH1750 (Ένταση φωτισμού). Ψηφιακός αισθητήρας φωτός με διεπαφή I2C, ο οποίος επιστρέφει απευθείας τιμή έντασης φωτισμού σε lux, χωρίς να απαιτείται πρόσθετη βαθμονόμηση από τον χρήστη, η μετατροπή γίνεται εσωτερικά από το ίδιο το chip.



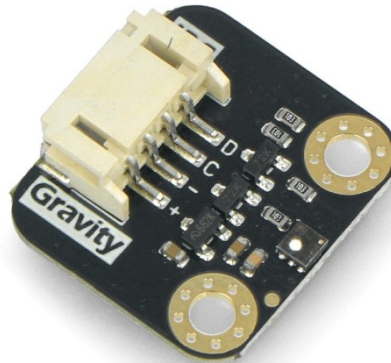
Σχήμα 2.4: Ο αισθητήρας BH1750 [22].

AM2302 (Θερμοκρασία / Υγρασία). Ψηφιακός αισθητήρας που συνδυάζει έναν αισθητήρα υγρασίας τύπου χωρητικότητας και έναν αισθητήρα θερμοκρασίας, με ενσωματωμένο μετατροπέα analog-to-digital. Όπως και το BH1750, επιστρέφει βαθμονομημένες τιμές απευθείας μέσω του πρωτοκόλλου επικοινωνίας του, χωρίς να χρειάζεται επιπλέον ρύθμιση.



Σχήμα 2.5: Ο αισθητήρας AM2302 [23].

SGP40 (Δείκτης πτητικών οργανικών ενώσεων). Ψηφιακός αισθητήρας ποιότητας αέρα με διεπαφή I2C, ο οποίος παρέχει απευθείας τιμή VOC Index σε κλίμακα 0–500, χωρίς να απαιτείται χειροκίνητη βαθμονόμηση από τον χρήστη. Ο αισθητήρας διαθέτει ενσωματωμένο αλγόριθμο προσαρμογής γραμμής βάσης (*baseline algorithm*), ο οποίος προσαρμόζεται σταδιακά στις συνθήκες του περιβάλλοντος κατά τα πρώτα λεπτά λειτουργίας. Η ανάγνωση μπορεί προαιρετικά να αντισταθμιστεί με τις τιμές θερμοκρασίας και υγρασίας από τον αισθητήρα AM2302, βελτιώνοντας την ακρίβεια του υπολογιζόμενου δείκτη.



Σχήμα 2.6: Ο αισθητήρας SGP40 [24].

2.2.2 Arduino Uno

Οι αισθητήρες είναι τοποθετημένοι πάνω σε προσαρμοσμένο shield πάνω στο Arduino Uno, το οποίο αναλαμβάνει τη συλλογή των μετρήσεων (ανάγνωση αναλογικών/ψηφιακών σημάτων) και τη μετάδοσή τους σειριακά προς τη μονάδα XBee.



Σχήμα 2.7: Arduino Uno Rev3 [25].

Το πλήρες sketch που εκτελείται στο Arduino (Κώδικας Α'.3, Παράρτημα Α) συνδυάζει τους τέσσερις αισθητήρες και ακολουθεί την εξής λογική:

Μεταβλητές και Αρχικοποίηση Αντικειμένων:

Γίνεται δήλωση των αντικειμένων dht (τύπου DHT22, στον ψηφιακό ακροδέκτη 2) και gasSensor (τύπου MQ135, στον αναλογικό ακροδέκτη A0), καθώς και τα αντικείμενα lightMeter της βιβλιοθήκης BH1750 και sgp40 της βιβλιοθήκης DFRobot_SGP40, τα οποία επικοινωνούν μέσω του κοινού διαύλου I2C. Επίσης γίνεται ο ορισμός των σταθερών μεταβλητών SEA_LEVEL_PRESSURE_HPA στα 1023.25 hPa, η πρότυπη πίεση στάθμης θάλασσας, καθώς και της ALTITUDE_M στα 620 μέτρα, οι οποίες χρησιμοποιούνται για τον υπολογισμό της ατμοσφαιρικής πίεσης.

Συνάρτηση estimateTVOC(int mq135Raw):

Λέχεται ως όρισμα την ακατέργαστη τιμή ADC του MQ-135 (εύρος 0 έως 1023) και επιστρέφει μια εκτίμηση TVOC σε ppb, μέσω μιας τμηματικής γραμμικής συνάρτησης τριών κλάδων: (1) για τιμές κάτω από 400, η έξοδος ταυτίζεται με την είσοδο· (2) για τιμές μεταξύ 400 και 650, εφαρμόζεται κλίση $\times 3.2$ πάνω από τη βάση των 400· (3) για τιμές άνω των 650, εφαρμόζεται πολύ μεγαλύτερη κλίση ($58800/373 \approx 157.6$) πάνω από τη βάση των 1200, ώστε οι υψηλές τιμές ADC να αντιστοιχίζονται σε αναλογικά πολύ υψηλότερες τιμές TVOC. Πρόκειται για εμπειρική προσέγγιση και όχι άμεση μέτρηση, καθώς ο MQ-135 δεν διαθέτει ειδικό αισθητήρα για το στοιχείο TVOC. Ο λόγος επιλογής αυτών των ορίων ήταν ώστε το ανώτατο άκρο του εύρους ADC (1023) να αντιστοιχίζεται σε τιμή TVOC κοντά στο ανώτατο όριο τιμών του Smoke Detection IoT Dataset (βλ. Ενότητα 3.2.2), ώστε η κλίμακα των εκτιμώμενων τιμών να είναι συγκρίσιμη με αυτή στην οποία εκπαιδεύτηκε το μοντέλο. Η τιμή αυτή αποτελεί το χαρακτηριστικό TVOC που τροφοδοτεί το μοντέλο Smoke Detection (βλ. Ενότητα 3.2), και παραμένει ανεξάρτητη από την πραγματική μέτρηση VOC Index του αισθητήρα SGP40.

Συνάρτηση setup():

Γίνεται η αρχικοποίηση της σειριακής επικοινωνίας στα 9600 baud, του διαύλου I2C (Wire.begin()) και του αισθητήρα DHT22. Στη συνέχεια γίνεται έλεγχος επιτυχούς αρχικοποίησης του BH1750 (lightMeter.begin()), με εκτύπωση μηνύματος σφάλματος αν αποτύχει, καθώς και του SGP40 (sgp40.begin(10000)), όπου εφαρμόζεται επανάληψη της προσπάθειας αρχικοποίησης μέχρι την επιτυχή ανίχνευση της μονάδας, με χρόνο προθέρμανσης 10 δευτερολέπτων, όπως ορίζεται από τον κατασκευαστή. Έπειτα, εφαρμόζεται επιπλέον καθυστέρηση 10 δευτερολέπτων (delay(10000)), απαραίτητη για την προθέρμανση του MQ-135 πριν την πρώτη αξιόπιστη μέτρηση. Τέλος, εκτυπώνεται η γραμμή επικεφαλίδων του CSV (Θερμοκρασία, Υγρασία, CO2, Φως, TVOC, VOC Index SGP40, Πίεση).

Συνάρτηση loop() & Ανάγνωση DHT22:

Σε κάθε επανάληψη, διαβάζονται η υγρασία και η θερμοκρασία από τον DHT22. Αν κάποια από τις τιμές δεν είναι έγκυρη (isnan()), κάτι που συμβαίνει συχνά σε αυτούς τους αισθητήρες λόγω του χρονισμού της ψηφιακής γραμμής επικοινωνίας, ο βρόχος διακόπτεται πρόωρα με return, χωρίς αποστολή δεδομένων, και επαναλαμβάνεται μετά από 2 δευτερόλεπτα.

Ανάγνωση και Εξομάλυνση MQ-135:

Γίνεται λήψη 10 διαδοχικών μετρήσεων μέσω της συνάρτησης gasSensor.getResistance(), με ενδιάμεση καθυστέρηση 50 ms η καθε-

μία, και υπολογίζεται ο μέσος όρος τους (`avg_resistance`). Η εξομάλυνση αυτή μειώνει τον θόρυβο της μεμονωμένης ανάγνωσης, χαρακτηριστικά έντονο σε αναλογικούς αισθητήρες αερίων τύπου MOS. Από τον μέσο όρο υπολογίζεται η συγκέντρωση CO₂ σε ppm, μέσω εκθετικής συνάρτησης βαθμονόμησης ($116.6020682 \cdot (\text{avg_resistance}/R_0)^{-2.769034857}$), με χρήση της βαθμονομημένης τιμής αναφοράς `myRZero` (βλ. Ενότητα 2.2.1).

Εκτίμηση TVOC και Φωτισμού:

Λαμβάνεται μια νέα ανάγνωση ADC του MQ-135 (`analogRead(CO2PIN)`), ανεξάρτητη από την προηγούμενη εξομαλυνμένη μέτρηση, η οποία τροφοδοτεί τη συνάρτηση `estimateTVOC()` για τον υπολογισμό της τιμής TVOC. Παράλληλα, διαβάζεται η ένταση φωτισμού από τον BH1750 (`lightMeter.readLightLevel()`) και υπολογίζεται η ατμοσφαιρική πίεση μέσω της `calculatePressure()`.

Ανάγνωση SGP40:

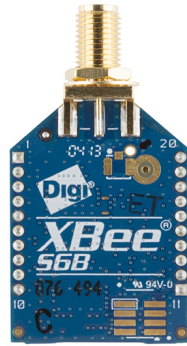
Πριν την ανάγνωση, καλείται η συνάρτηση `sgp40.setRhT()`, στην οποία περνιούνται οι τιμές υγρασίας και θερμοκρασίας από τον DHT22, ώστε να επιτευχθεί αντιστάθμιση (*compensation*) της μέτρησης. Στη συνέχεια, καλείται η `sgp40.getVocIndex()`, η οποία επιστρέφει τον δείκτη VOC Index σε κλίμακα 0–500. Σε αντίθεση με την τιμή TVOC που προκύπτει από τον MQ-135, πρόκειται για άμεση μέτρηση του αισθητήρα και όχι για εμπειρική εκτίμηση. Η τιμή αυτή καταγράφεται παράλληλα με το TVOC ως συμπληρωματική, ανεξάρτητη πληροφορία ποιότητας αέρα, χωρίς ωστόσο να τροφοδοτεί κάποιο από τα μοντέλα μηχανικής μάθησης (βλ. Ενότητα 3.2).

Αποστολή Δεδομένων και Χρονισμός:

Οι επτά τιμές (θερμοκρασία, υγρασία, CO₂, φωτισμός, TVOC, VOC Index SGP40, πίεση) στέλνονται σειριακά σε ενιαία γραμμή, διαχωρισμένες με κόμμα, όπου και γίνεται η μετάδοσή τους μέσω του XBee. Ο βρόχος ολοκληρώνεται με καθυστέρηση 2 δευτερολέπτων (`delay(2000)`), η οποία καθορίζει τη συχνότητα αποστολής νέων μετρήσεων προς το Edge Layer.

2.2.3 Δικτύωση: XBee S6B (ρύθμιση μέσω XCTU)

Η μετάδοση των δεδομένων προς το Edge Layer γίνεται μέσω της μονάδας XBee S6B, η οποία παρέχει ασύρματη σύνδεση WiFi. Η ρύθμιση του XBee πραγματοποιήθηκε μέσω της εφαρμογής XCTU, ώστε η μονάδα να είναι ανιχνεύσιμη και προσβάσιμη από το Khadas VIM3 στο τοπικό δίκτυο.



Σχήμα 2.8: XBee S6B WiFi Module [26].

Το Wireless SD Shield που χρησιμοποιήθηκε μοιράζεται τα ίδια pins TX/RX ανάμεσα στο XBee και στο USB-to-serial του Arduino. Αυτό σημαίνει ότι, για να επικοινωνήσει η εφαρμογή XCTU απευθείας με το XBee μέσω USB, ο μικροελεγκτής του Arduino πρέπει προσωρινά να απενεργοποιηθεί, ώστε να μην παρεμβαίνει στη σειριακή γραμμή. Αυτό επιτυγχάνεται βραχυκυκλώνοντας με ένα καλώδιο τα pins RESET και GND πάνω στο shield, κρατώντας τον μικροελεγκτή σε κατάσταση reset όσο διαρκεί η ρύθμιση μέσω XCTU.

Οι βασικές παράμετροι δικτύου και λειτουργίας που παραμετροποιήθηκαν στο XBee S6B μέσω του περιβάλλοντος XCTU συγκεντρώνονται στον Πίνακα 2.3.

Κατηγορία	Παράμετρος (AT Command)	Τιμή Ρύθμισης
Network	AH (Network Type)	Infrastructure [2]
Network	CE (Infrastructure Mode)	STA mode [2]
Network	ID (SSID)	XBeeKhadass
Network	EE (Encryption Enable)	WPA2 (AES) security [2]
Network	IP (IP Protocol)	UDP [0]
Network	MA (IP Addressing Mode)	DHCP [0]
Serial Interfacing	AP (API Enable)	Transparent Mode [0]

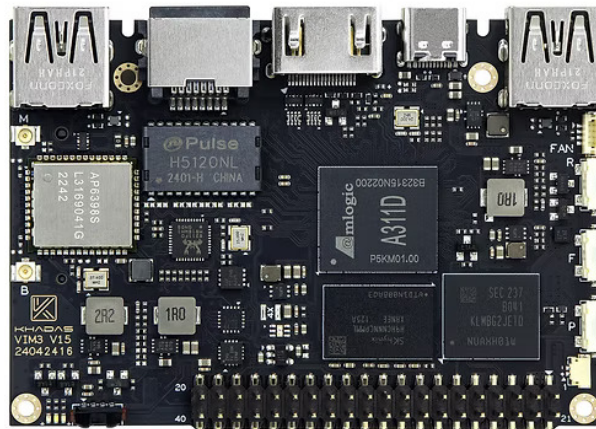
Πίνακας 2.3: Βασικές παράμετροι ρυθμίσεων του XBee S6B στο XCTU.

Όπως φαίνεται στον Πίνακα 2.3, η μονάδα ρυθμίστηκε σε **Transparent Mode** (AP = 0), λειτουργία στην οποία τα δεδομένα που στέλνονται σειριακά από το Arduino μεταδίδονται αυτούσια μέσω WiFi, χωρίς πρόσθετη κωδικοποίηση πλαισίου (framing), απλοποιώντας την υλοποίηση στην πλευρά λήψης.

2.3 Edge Layer

2.3.1 Khadas VIM3

Το edge κομμάτι του συστήματος υλοποιείται σε ένα Khadas VIM3, το οποίο αναλαμβάνει τη λήψη των δεδομένων από το IoT Layer και την εκτέλεση δύο ανεξάρτητων μοντέλων μηχανικής μάθησης σε πραγματικό χρόνο, τοπικά, χωρίς εξάρτηση από το cloud για την διαδικασία εξαγωγής προβλέψεων. Η συγκεκριμένη συσκευή επιλέχθηκε καθώς διαθέτει τον επεξεργαστή Amlogic A311D (τετραπύρηνο ARM Cortex-A73 και διπύρηνο Cortex-A53), καθώς και ενσωματωμένη μονάδα επεξεργασίας νευρωνικών δικτύων (NPU) ισχύος έως 5.0 TOPS.



Σχήμα 2.9: Το Khadas VIM3 [27].

2.3.1.1 Διαμόρφωση Ασύρματου Δικτύου (Hotspot) και Δρομολόγησης

Για την ασύρματη επικοινωνία με το IoT Layer, το Khadas VIM3 ρυθμίστηκε ώστε να λειτουργεί ως τοπικό σημείο πρόσβασης (Wi-Fi Hotspot / Access Point). Η ρύθμιση πραγματοποιήθηκε στη γραμμή εντολών (Ubuntu Linux) αξιοποιώντας τα εργαλεία NetworkManager (nmcli), Netplan, καθώς και εντολές διαχείρισης δικτύου του πυρήνα Linux. Η συσκευή διαθέτει δύο ασύρματες διεπαφές, όπου και η αρχική δοκιμή έγινε στην ενσωματωμένη διεπαφή wlan0, ωστόσο η τελική, λειτουργική διαμόρφωση πραγματοποιήθηκε στη διεπαφή wlan1, η οποία και χρησιμοποιείται στην πράξη.

Τα βήματα και οι εντολές που χρησιμοποιήθηκαν αναλύονται παρακάτω:

1. **Δημιουργία και Παραμετροποίηση του Wi-Fi Hotspot:** Δημιουργήθηκε το ασύρματο σημείο πρόσβασης με όνομα δικτύου (SSID) XBeeKhadas (βλ. Πίνακα 2.3) και ρυθμίστηκαν οι παράμετροι εκπομπής (Channel 6, ζώνη 802.11b/g, κρυπτογράφηση WPA-PSK):

```
sudo nmcli device wifi hotspot ifname wlan1 ssid XBeeKhadas
password *****
```

```
sudo nmcli connection modify "XBeeKhadas" 802-11-wireless.band bg
sudo nmcli connection modify "XBeeKhadas" 802-11-wireless.channel
6
sudo nmcli connection modify "XBeeKhadas" 802-11-wireless-security
.key-mgmt wpa-psk
sudo nmcli connection modify "XBeeKhadas" 802-11-wireless-security
.pmf 1
sudo nmcli connection up "XBeeKhadas"
```

2. **Απόδοση Στατικής Διεύθυνσης IP και Ρύθμιση Netplan:** Για να διασφαλιστεί ότι το Khadas διατηρεί τη σταθερή διεύθυνση 10.42.0.1/24 ως Gateway του τοπικού δικτύου, ορίστηκε στατική IP στη διεπαφή wlan1:

```
sudo ip addr add 10.42.0.1/24 dev wlan1
```

Παράλληλα, δημιουργήθηκε το αρχείο ρυθμίσεων /etc/netplan/02-wlan1-hotspot.yaml για τη μονιμοποίηση των δικτυακών ρυθμίσεων:

```
sudo chmod 600 /etc/netplan/*.yaml
sudo netplan generate
sudo netplan apply
```

3. **Ενεργοποίηση IP Forwarding & NAT :** Προκειμένου το Khadas VIM3 να δρομολογεί την κίνηση μεταξύ του τοπικού ασύρματου δικτύου (wlan1) και της ενσύρματης σύνδεσης (eth0), ενεργοποιήθηκε η προώθηση πακέτων στον πυρήνα και ορίστηκε ο κανόνας NAT μέσω iptables:

```
sudo sysctl -w net.ipv4.ip_forward=1
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

4. **Έλεγχος Συνδεσιμότητας και Διαγνωστικά (Debugging):** Κατά τη φάση δοκιμών, η επιτυχής σύνδεση της μονάδας XBee S6B (η οποία έλαβε την IP 10.42.0.35) και η ροή των πακέτων UDP επιβεβαιώθηκαν με τις ακόλουθες διαγνωστικές εντολές:

```
# Έλεγχος συνδεδεμένων συσκευών στο ασύρματο δίκτυο
ip neigh show dev wlan1
sudo iw dev wlan1 station dump
```

```
# Παρακολούθηση εισερχόμενων πακέτων UDP από το XBee (10.42.0.35)
sudo tcpdump -i wlan1 host 10.42.0.35
```

Με την παραπάνω αρχιτεκτονική δικτύου, το Khadas VIM3 λειτουργεί ως αυτόνομος κόμβος (Access Point & Gateway), λαμβάνοντας με αξιοπιστία τα πακέτα UDP από το XBee S6B για την τοπική εκτέλεση των μοντέλων Μηχανικής Μάθησης.

2.3.2 Λήψη Δεδομένων (UDP)

Στο Khadas VIM3 εκτελείται σε Python server που λαμβάνει τα εισερχόμενα πακέτα UDP από τη μονάδα XBee του IoT Layer, τα οποία περιέχουν τις τιμές των αισθητήρων.

Η ορθή λειτουργία ολόκληρης της αλυσίδας από τη λήψη δεδομένων μέσω UDP, την εκτέλεση των μοντέλων, μέχρι την αποστολή του αποτελέσματος στο Cloud Layer επιβεβαιώθηκε στην πράξη, όπως φαίνεται στην πραγματική έξοδο του συστήματος (Κώδικας 2.1).

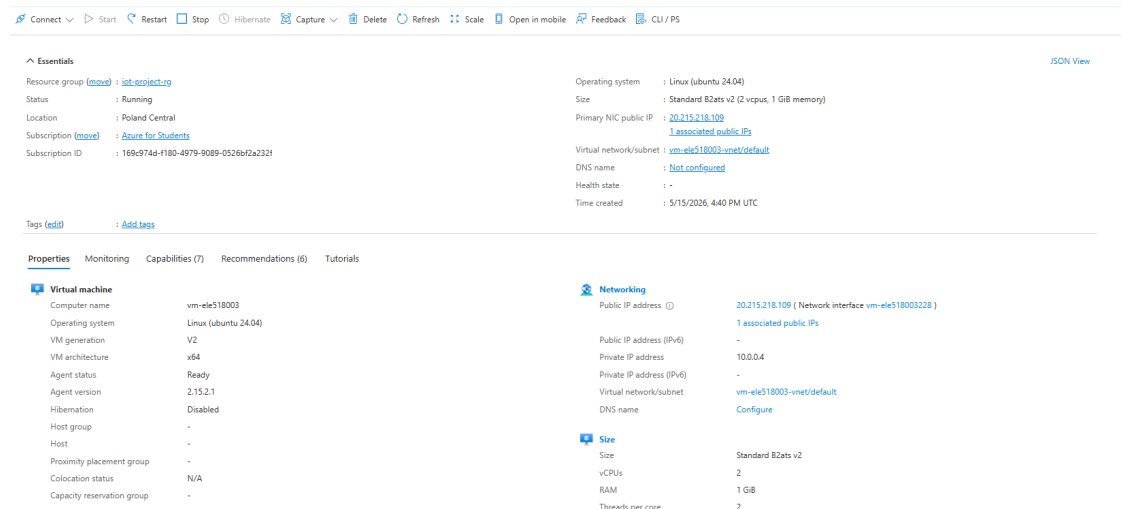
```
[API] Δεδομένα εστάλησαν επιτυχώς στο /smoke!
[2026-08-23 11:47:52] Από IP: 10.42.0.35
  └ Θερμοκρασία : 32.6 °C
  └ Υγρασία      : 59.4 \%
  └ CO2          : 103.21 ppm
  └ Φωτεινότητα : 10.83 Lux
  └ TVOC         : 246.0 ppb
  └ Πίεση       : 940.95 hPa
>>> [Occupancy] Πρόβλεψη: EMPTY
[API] Δεδομένα εστάλησαν επιτυχώς στο /occupancy!
>>> [Smoke] Πρόβλεψη: OK (20.0\%)
Στατιστικά -- Occupancy: \{'EMPTY':2,'OCCUPANCY': 0\} |
Smoke: \{'OK': 2, 'FIRE': 0\}
```

Κώδικας 2.1: Πραγματική έξοδος του Khadas VIM3, Επιβεβαίωση λήψης δεδομένων από την εκτέλεση των μοντέλων Μηχανικής Μάθησης και την επιτυχή αποστολή στο Cloud API.

2.4 Cloud Layer

2.4.1 Azure VM — Υποδομή και Εικονικοποίηση

Το cloud backend είναι αναπτυγμένο σε εικονική μηχανή (VM) στο Microsoft Azure, με λειτουργικό σύστημα Ubuntu 24.04 LTS (περιοχή Poland Central, μέγεθος Standard B2ats v2 2 vCPUs, 1 GiB RAM), μέσω του προγράμματος Azure for Students. Η επιλογή εικονικοποιημένης υποδομής αντί για φυσικό εξοπλισμό επιτρέπει ευελιξία στην κλιμάκωση των πόρων και στην κεντρική διαχείριση, χωρίς την ανάγκη φυσικής συντήρησης server.



Σχήμα 2.10: Επισκόπηση της εικονικής μηχανής (VM) στο Microsoft Azure Portal.

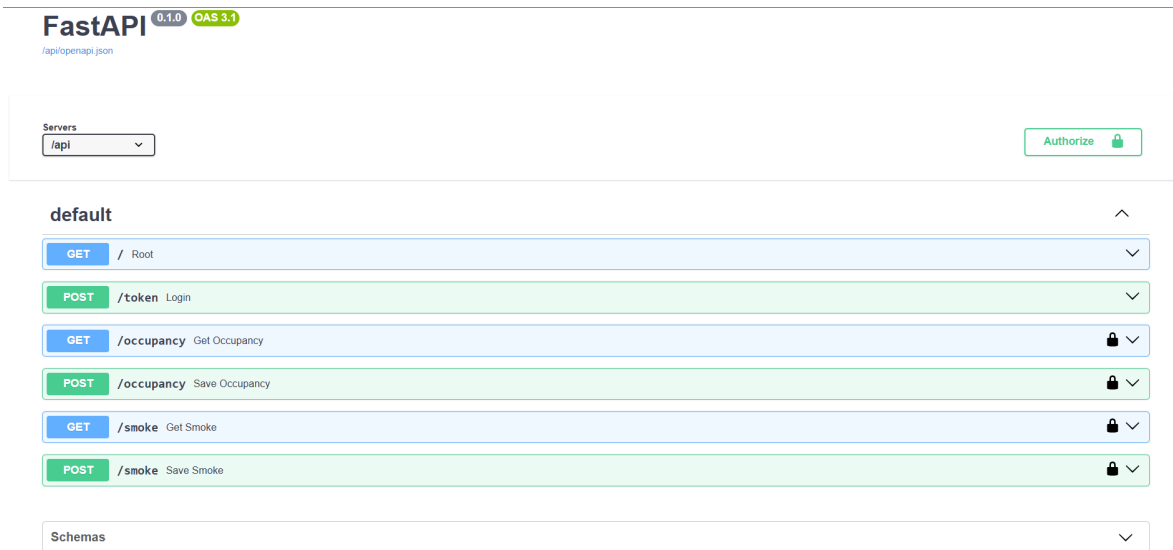
2.4.2 Docker Containers (FastAPI, PostgreSQL, Grafana, Nginx)

Η ενορχήστρωση του backend γίνεται μέσω Docker Compose, με τέσσερα containers:

- **FastAPI:** REST API για την επικοινωνία με τα δεδομένα.
- **PostgreSQL:** Κεντρική αποθήκευση των δεδομένων από τους αισθητήρες και τα αποτελέσματα από τα μοντέλα μηχανικής μάθησης.
- **Grafana:** Οπτικοποίηση δεδομένων σε πραγματικό χρόνο, με ειδοποιήσεις μέσω email όταν ενεργοποιούνται κατάλληλα κατώφλια (π.χ. ανίχνευση καπνού).
- **Nginx:** Reverse proxy, ώστε η κίνηση προς τα υπόλοιπα containers να διέρχεται από ένα ενιαίο σημείο εισόδου.

Ο ορισμός των containers και της μεταξύ τους σχέσης (ports, volumes, dependencies) γίνεται μέσω ενός αρχείου `docker-compose.yml` (Κώδικας Α'4). Τα ευαίσθητα δεδομένα (κωδικοί βάσης, διαπιστευτήρια SMTP) **δεν** είναι καταχωρημένα μέσα στο αρχείο, αλλά περνιούνται μέσω μεταβλητών περιβάλλοντος από ξεχωριστό αρχείο `.env`, το οποίο δεν περιλαμβάνεται στο σύστημα ελέγχου εκδόσεων (`.gitignore`).

Η δομή του API, όπως παρουσιάζεται μέσω του αυτόματα παραγόμενου Swagger UI (`/api/docs`), φαίνεται στην Εικόνα 2.11 περιλαμβάνει endpoints για αυθεντικοποίηση (`/token`) και για αποθήκευση/ανάκτηση δεδομένων ανά μοντέλο (`/occupancy`, `/smoke`), με τα endpoints ελεγχόμενης πρόσβασης να επισημαίνονται με το εικονίδιο κλειδώματος.



Σχήμα 2.11: Επισκόπηση των endpoints του API μέσω Swagger UI.

Ένα πραγματικό αίτημα προς το endpoint `/occupancy`, εκτελεσμένο απευθείας μέσα από το Swagger UI, επιστρέφει επιτυχώς κωδικό απάντησης **200** μαζί με τα αποθηκευμένα δεδομένα (Εικόνα 2.12), επιβεβαιώνοντας την ορθή λειτουργία του API σε πραγματικές συνθήκες.



Σχήμα 2.12: Επιτυχής απάντηση (200 OK) του endpoint `/occupancy`, με πραγματικά δεδομένα από τη βάση.

2.4.3 Ασφάλεια & Πρόσβαση (JWT, DuckDNS, SSL)

Η πρόσβαση στο API προστατεύεται με JWT authentication. Η ροή αυθεντικοποίησης ακολουθεί το τυπικό μοντέλο login → token → authenticated request, όπως φαίνεται στον Κώδικα 2.2.

```
# 1. Login - apoktisi token
POST /login
Content-Type: application/json

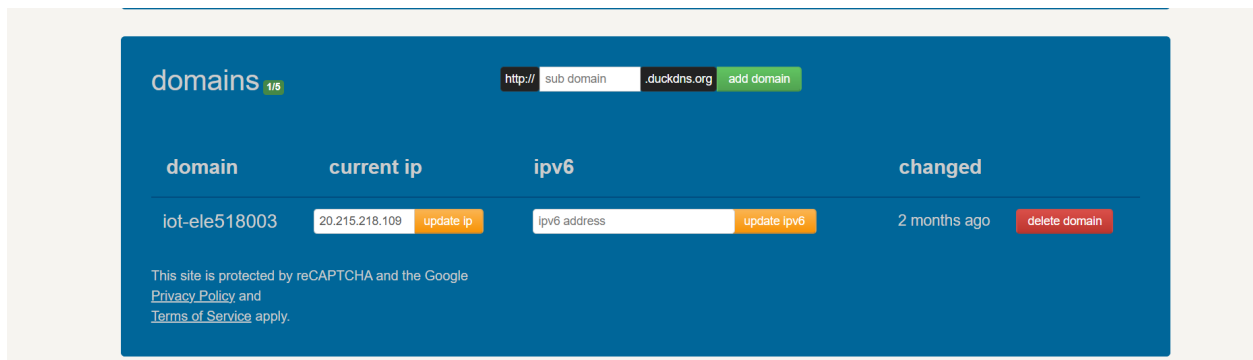
{
  "username": "user",
  "password": "*****"
}

# Apantisi:
{
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "bearer"
}

# 2. Xrisi tou token se prostateumeno request
GET /sensors/latest
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Κώδικας 2.2: Ενδεικτική ροή αυθεντικοποίησης JWT: απόκτηση token και χρήση του σε προστατευμένο αίτημα.

Η πρόσβαση στο VM γίνεται μέσω domain που καταχωρήθηκε στο DuckDNS (Εικόνα 2.13).

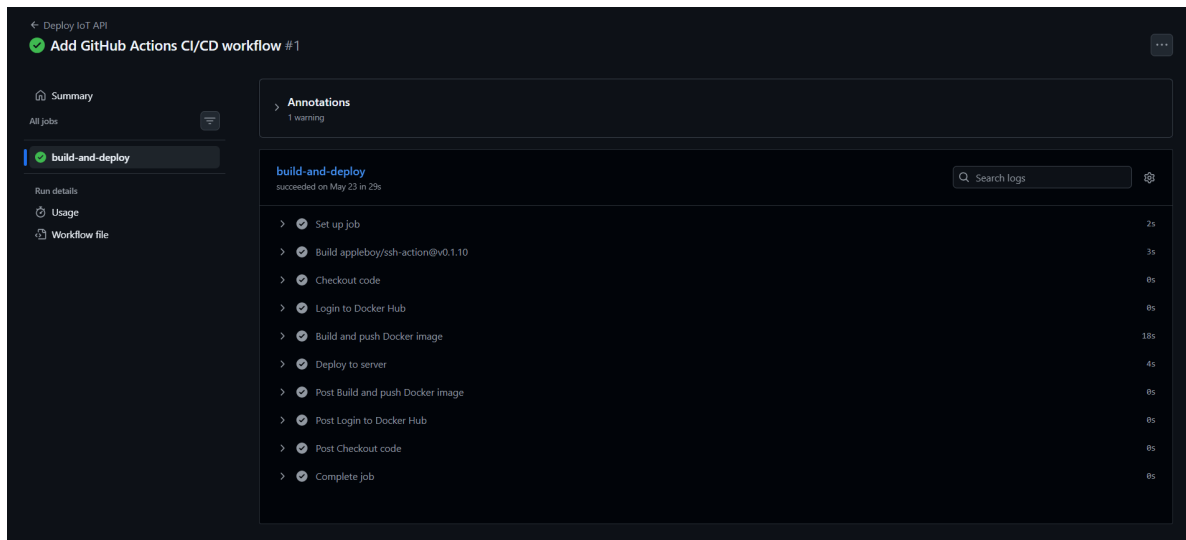


Σχήμα 2.13: Το καταχωρημένο domain στο DuckDNS.

Για την κρυπτογράφηση της επικοινωνίας (HTTPS) επιχειρήθηκε αρχικά η χρήση πιστοποιητικού Let's Encrypt, το οποίο τελικά δεν κατέστη εφικτό λόγω περιορισμών στη διαχείριση DNS του λογαριασμού Azure for Students. Ως εναλλακτική λύση χρησιμοποιήθηκε **self-signed SSL certificate** όπως φαίνεται στην εικόνα 2.13.

2.4.4 CI/CD Pipeline

Για τη συνεχή ενσωμάτωση και ανάπτυξη (CI/CD), διαμορφώθηκε μια ροή εργασιών του GitHub Actions στο repository του project, η οποία κατασκευάζει τις εικόνες Docker και τις προωθεί στο Docker Hub, αυτοματοποιώντας τη διαδικασία της τελικής ανάπτυξης. Το pipeline περιλαμβάνει τα έξι βήματα: έλεγχο του κώδικα, σύνδεση στο Docker Hub, κατασκευή και προώθηση της νέας εικόνας, και τέλος την αυτόματη εγκατάσταση στον server μέσω SSH. Μια επιτυχημένη εκτέλεση του pipeline φαίνεται στην Εικόνα 2.14.



Σχήμα 2.14: Επιτυχής εκτέλεση του CI/CD pipeline μέσω GitHub Actions.

Κεφάλαιο 3

Μηχανική Μάθηση Και Διασύνδεση

Στο παρόν κεφάλαιο παρουσιάζεται η μεθοδολογία ανάπτυξης, εκπαίδευσης, αξιολόγησης και ενσωμάτωσης των μοντέλων Μηχανικής Μάθησης στο σύστημα, καθώς και η τελική διασύνδεση του Edge επιπέδου (Khadass VIM3) με την υποδομή Cloud (Azure VM). Η επίλυση των προβλημάτων ταξινόμησης βασίστηκε στη χρήση δύο δημόσια διαθέσιμων συνόλων δεδομένων (datasets) από το περιβάλλον IoT.

3.1 Δεδομένα

Για την εκπαίδευση και των δύο μοντέλων μηχανικής μάθησης χρησιμοποιήθηκαν δημόσια, ελεύθερα διαθέσιμα datasets, καθένα κατάλληλο για το αντίστοιχο πρόβλημα ταξινόμησης.

Occupancy Detection Dataset [28]: περιέχει μετρήσεις θερμοκρασίας, φωτισμού, CO₂, υγρασίας και ένδειξη κατοχής χώρου (occupied/not occupied), συλλεγμένες σε πραγματικό περιβάλλον γραφείου.

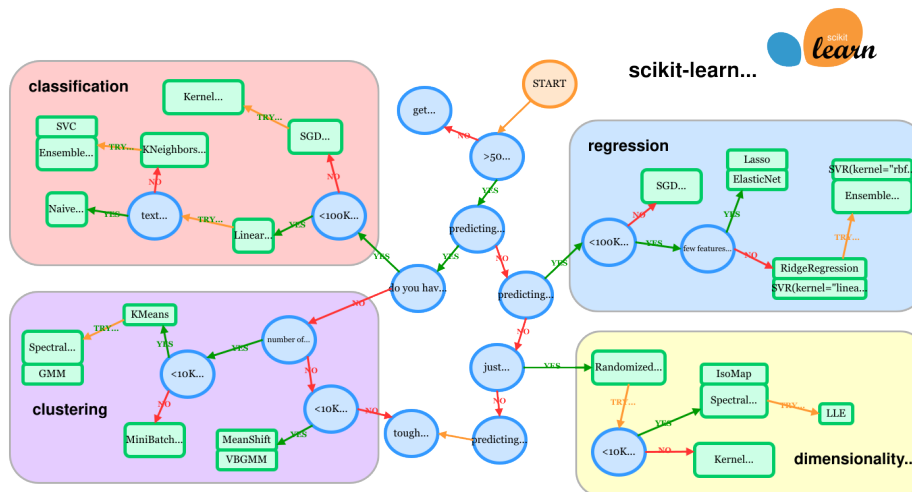
Smoke Detection IoT Dataset [29]: περιέχει μετρήσεις θερμοκρασίας, υγρασίας, TVOC, πίεσης και άλλων μεγεθών, με ετικέτα ανίχνευσης φωτιάς/καπνού (Fire Alarm).

Και τα δύο datasets επιλέχθηκαν καθώς τα διαθέσιμα χαρακτηριστικά τους αντιστοιχούν, σε μεγάλο βαθμό, στα μεγέθη που μπορούν πραγματικά να μετρηθούν από τους αισθητήρες της υλοποίησης (βλ. Ενότητα 2.2), επιτρέποντας τη μεταφορά ενός εκπαιδευμένου μοντέλου από το dataset στο πραγματικό edge περιβάλλον.

Αξίζει να σημειωθεί η διαφορά ανάμεσα στα δύο datasets: το Occupancy Detection Dataset διατίθεται ήδη χωρισμένο σε ξεχωριστά αρχεία εκπαίδευσης και ελέγχου (DataTraining.csv, DataTest.csv), γεγονός που απλοποίησε τη διαδικασία, καθώς δεν απαιτήθηκε χειροκίνητος διαχωρισμός (train/test split). Αντίθετα, το Smoke Detection IoT Dataset διατίθεται ως ενιαίο αρχείο, οπότε ο διαχωρισμός σε σύνολα εκπαίδευσης και ελέγχου πραγματοποιήθηκε χειροκίνητα κατά την προεπεξεργασία.

3.2 Scikit-Learn

Για την υλοποίηση των μοντέλων Μηχανικής Μάθησης αξιοποιήθηκε η βιβλιοθήκη *Scikit-Learn* η οποία χρησιμοποιεί την Python γλώσσα προγραμματισμού, ένα άλλο εργαλείο το οποίο χρησιμοποιήσαμε είναι το Jupyter Notebooks το οποίο μας δίνει την δυνατότητα να τρέχουμε με διαφορετική σειρά τα cells που περιέχουν τον κωδικά που χρειαζόμαστε. Το Jupyter Notebooks είναι ένα χρήσιμο εργαλείο για την ανάπτυξη και τον πειραματισμό με το κώδικα. Η επιλογή των υποψήφιων αλγορίθμων για δοκιμή δεν έγινε τυχαία, αλλά καθοδηγήθηκε εν μέρει από το επίσημο διάγραμμα ροής του scikit-learn για την επιλογή κατάλληλου μοντέλου [30], το οποίο για προβλήματα ταξινόμησης με επισημασμένα (labeled) δεδομένα και λιγότερα από 100.000 δείγματα προτείνει ως πρώτη επιλογή το LinearSVC [31]. Το σχετικό διάγραμμα ροής παρουσιάζεται στην Εικόνα 3.1.



Σχήμα 3.1: Χάρτης επιλογής αλγορίθμων της βιβλιοθήκης Scikit-Learn.

Ανάλυση συσχέτισης: Πριν την εκπαίδευση οποιουδήποτε μοντέλου, εξετάστηκε η στατιστική σχέση ανάμεσα στα χαρακτηριστικά και τη μεταβλητή-στόχο, μέσω τριών διαφορετικών συντελεστών συσχέτισης: του συντελεστή Pearson [32], που μετρά γραμμική συσχέτιση, και των συντελεστών Kendall και Spearman, που μετρούν μονοτονική (όχι απαραίτητα γραμμική) σχέση μέσω κατάταξης (rank) των τιμών. Η χρήση και των τριών μετρικών, αντί μίας, επιτρέπει τον εντοπισμό τόσο γραμμικών όσο και μη-γραμμικών αλλά μονοτονικών σχέσεων ανάμεσα στα χαρακτηριστικά και την κατάσταση κατοχής του χώρου.

Κλιμάκωση χαρακτηριστικών: (Feature Scaling). Τα χαρακτηριστικά κλιμακώθηκαν με το StandardScaler [33], το οποίο μετασχηματίζει κάθε χαρακτηριστικό ώστε να έχει μέσο όρο 0 και τυπική απόκλιση 1 (αφαίρεση μέσου όρου, διαίρεση με την τυπική απόκλιση). Η κλιμάκωση είναι απαραίτητη για αλγορίθμους όπως το LinearSVC, των οποίων η συνάρτηση κόστους επηρεάζεται από την κλίμακα των εισερχόμενων τιμών χωρίς κλιμάκωση, χαρακτηριστικά με μεγαλύτερο εύρος τιμών (π.χ. CO2 σε ppm) θα κυριαρχούσαν τεχνητά έναντι χαρακτηριστικών με μικρότερο εύρος (π.χ. Humidity Ratio).

Επιλογή Χαρακτηριστικών: Για να διαπιστωθεί αν όλα τα διαθέσιμα χαρακτηριστικά ήταν πράγματι απαραίτητα, εφαρμόστηκε Recursive Feature Elimination with Cross-Validation (RFECV) [34]: μια επαναληπτική διαδικασία που αφαιρεί σταδιακά το λιγότερο σημαντικό χαρακτηριστικό και επαναξιολογεί την απόδοση του μοντέλου μέσω διασταυρούμενης επικύρωσης [35], με στόχο τον εντοπισμό του μικρότερου δυνατού υποσυνόλου χαρακτηριστικών που διατηρεί την ακρίβεια πρόβλεψης.

3.2.1 Μεθοδολογία για το Occupancy Detection Dataset

Τα δεδομένα από το Occupancy Dataset είναι χωρισμένα σε train και test datasets, οπότε δεν χρειάζεται να χωρίσουμε τα δεδομένα για την εκπαίδευση των αλγόριθμων. Πριν αρχίσουμε την επεξεργασία των δεδομένων κοιτάζουμε αν υπάρχει κάποια συσχέτιση ανάμεσα στα χαρακτηριστικά τα οποία θέλουμε να προβλέψουμε, στην προκειμένη περίπτωση είναι το Occupancy. Για να δούμε αν υπάρχει κάποια συσχέτιση χρησιμοποιήσαμε τα στατιστικά εργαλεία Pearson και Kendall correlation. Από τον πίνακα 3.1 παρατηρούμε ότι υπάρχει συσχέτιση με το χαρακτηριστικό Light.

Χαρακτηριστικά (Feature)	Pearson Correlation	Kendall Correlation
Temperature	0.538220	0.439836
Humidity	0.132964	0.105824
Light	0.907352	0.734405
CO2	0.712235	0.537307
HumidityRatio	0.300282	0.209033
Occupancy	1.000000	1.000000

Πίνακας 3.1: Συσχέτιση των χαρακτηριστικών ως προς Occupancy χρησιμοποιώντας τους Pearson και Kendall correlation.

Στη συνέχεια, προχωρήσαμε στο στάδιο της προεπεξεργασίας των δεδομένων (Data Preprocessing). Αρχικά, διαχωρίσαμε τα δεδομένα στις ανεξάρτητες μεταβλητές X (features) και στην εξαρτημένη μεταβλητή y (label). Ακολούθως, εφαρμόσαμε την τεχνική της τυποποίησης (Standardization), η οποία αποτελεί βασική προϋπόθεση για την ορθή λειτουργία πολλών αλγορίθμων της βιβλιοθήκης *scikit-learn*, καθώς αλγόριθμοι που βασίζονται στον υπολογισμό αποστάσεων (όπως τα SVM και KNN) είναι ιδιαίτερα ευαίσθητοι στην κλίμακα των δεδομένων.

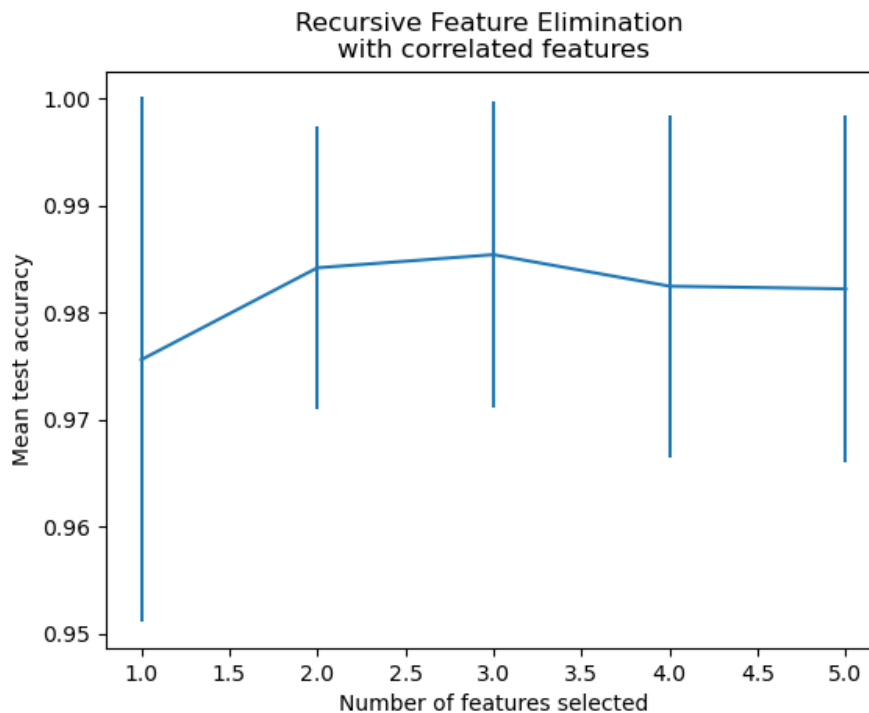
Στόχος της τυποποίησης είναι η μετατροπή της κατανομής των δεδομένων ώστε να έχουν μέση τιμή 0 και τυπική απόκλιση 1. Για την υλοποίηση αυτού του βήματος χρησιμοποιήθηκε η κλάση `StandardScaler` της Python. Στον Πίνακα 3.2 παρουσιάζονται τα στατιστικά στοιχεία των χαρακτηριστικών πριν και μετά την εφαρμογή της κλιμάκωσης, όπου επιβεβαιώνεται η επιτυχής μετατροπή τους.

Χαρακτηριστικά	Before Scale		After Scale	
	Mean	Std Dev	Mean	Std Dev
Temperature	20.6191	1.0169	20.6191	1.0000
Humidity	25.7315	5.5309	25.7315	1.0000
Light	119.5194	194.7438	119.5194	1.0000
CO2	606.5462	314.3016	606.5462	1.0000
HumidityRatio	0.0039	0.0009	0.0039	1.0000

Πίνακας 3.2: Μέσος όρος (Mean) και Τυπική Απόκλιση (Standard Deviation) των χαρακτηριστικών πριν και μετά την εφαρμογή της κλιμάκωσης (scaling).

Επιλογή Χαρακτηριστικών (Feature Elimination):

Έχοντας τα δεδομένα κλιμακωμένα, ελέγξαμε αν όλα τα διαθέσιμα χαρακτηριστικά συνεισφέρουν ουσιαστικά στην εκπαίδευση του μοντέλου. Η διατήρηση περιττών πληροφοριών μπορεί να προκαλέσει υπερεκπαίδευση (overfitting) και να αυξήσει άσκοπα τον υπολογιστικό χρόνο. Χρησιμοποιήσαμε την τεχνική Recursive Feature Elimination with Cross-Validation (RFECV), ορίζοντας ως εκτιμητή τη Λογιστική Παλινδρόμηση (Logistic Regression) και πραγματοποιώντας διασταυρούμενη επικύρωση 5 τμημάτων (5-fold CV). Όπως απεικονίζεται στο Διάγραμμα 3.2, ο αλγόριθμος υπέδειξε ότι η μέγιστη ακρίβεια επιτυγχάνεται με τη χρήση μόνο 3 χαρακτηριστικών, και συγκεκριμένα των: Temperature, Light και CO2.



Σχήμα 3.2: Αποτελέσματα του RFECV.

Επιλογή και Σύγκριση Αλγορίθμων βάσει Scikit-Learn Map:

Λαμβάνοντας υπόψη τη φύση του προβλήματός μας (πρόβλεψη κατηγορίας - classification), το γεγονός ότι έχουμε επισημασμένα δεδομένα (labeled data) και ότι το μέγεθος του dataset είναι μικρότερο από 100.000 δείγματα, το επίσημο διάγραμμα ροής της scikit-learn (Εικόνα 3.1) προτείνει ως βέλτιστη αρχική επιλογή τον αλγόριθμο **Linear Support Vector Classification (LinearSVC)**. Ως εναλλακτική προσέγγιση προτείνεται ο **K-Nearest Neighbors (KNeighborsClassifier)**.

Πραγματοποιήσαμε μια δοκιμαστική εκπαίδευση χρησιμοποιώντας τα 3 βέλτιστα χαρακτηριστικά και τις προεπιλεγμένες παραμέτρους των δύο αλγορίθμων, με στόχο την αρχική σύγκριση της απόδοσής τους μέσω Διασταυρούμενης Επικύρωσης 5 τμημάτων (5-fold Cross-Validation). Η αναλυτική αξιολόγηση και σύγκριση των δύο αλγορίθμων παρουσιάζεται στο Κεφάλαιο 4.

Έχοντας καταλήξει στον *LinearSVC*, το επόμενο βήμα είναι η βελτιστοποίηση των υπερπαραμέτρων του (Hyperparameter Tuning). Προκειμένου όμως να γίνει αυτό με επιστημονικά ορθό τρόπο και να αποφευχθεί το φαινόμενο της διαρροής δεδομένων (Data Leakage), δημιουργήσαμε ένα *Pipeline* (κλάση `make_pipeline` της *scikit-learn*)¹. Στο Pipeline που δημιουργήσαμε χρησιμοποιήσαμε το *StandardScaler* και το *LinearSVC*. Για να μεγιστοποιήσουμε την απόδοση του αλγορίθμου τις τιμές για τις παραμέτρους που παρουσιάζονται στον πίνακα 3.3.

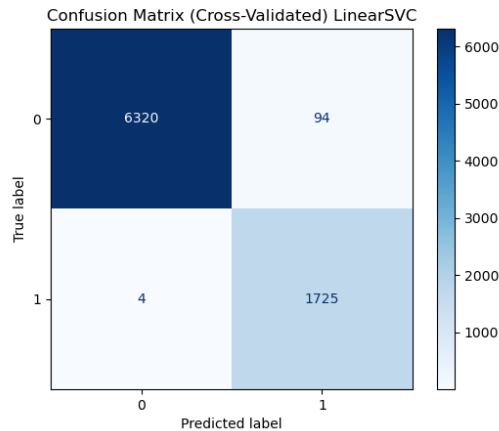
Υπερπαραμέτρος (Parameter)	Τιμές Αναζήτησης (Grid Values)
C	0.01, 0.1, 1, 10, 100, 200, 400, 500, 600
loss	'hinge', 'squared_hinge'
dual	'auto', True
multi_class	'ovr', 'crammer_singer'
max_iter	500, 800, 1000, 2500, 5000, 10000, 75000

Πίνακας 3.3: Ο χώρος αναζήτησης (Parameter Grid) για τη βελτιστοποίηση των υπερπαραμέτρων του αλγορίθμου *LinearSVC*.

Έχοντας διασφαλίσει τη διαδικασία μέσω του Pipeline, εφαρμόσαμε τη μέθοδο *Grid Search (GridSearchCV)* για την αναζήτηση των ιδανικών υπερπαραμέτρων. Το πλέγμα αναζήτησης περιέλαβε την παράμετρο κόστους C, τη συνάρτηση απώλειας (loss), την πολυκατηγορική στρατηγική (multi_class) και το μέγιστο όριο επαναλήψεων max_iter. Εξετάστηκαν συνολικά 504 συνδυασμοί. Ο βέλτιστος συνδυασμός που αναδείχθηκε ήταν: C=400, dual=True, loss='squared_hinge', max_iter=800 και multi_class='ovr'.

Έπειτα έγινε χρήση των παραμέτρων από τον αλγόριθμο βελτιστοποίησης (solver) για να εκπαιδευσουμε το τελικό μοντέλο όπου και στη συνέχεια, εφαρμόσαμε διασταυρούμενη επικύρωση (cross-validation), παρατηρήσαμε καλύτερα αποτελέσματα στο πίνακα σύγχυσης με ποιο αποδοτικά αποτελέσματα στα False Positive από 19 σε 4 και σε False Negative από 220 σε 94. Τα αποτελέσματα αυτά παρουσιάζονται στο σχήμα 4.1α' και οι βελτιστοποιήσεις στο σχήμα 4.2.

¹<https://scikit-learn.org/stable/modules/compose.html#pipeline-chaining-estimators>



Σχήμα 3.3: Αποτελέσματα του Confusion Matrix με τις βέλτιστες παραμέτρους

3.2.2 Μεθοδολογία για το Smoke Detection Dataset

Η μεθοδολογία που χρησιμοποιήθηκε είναι παρόμοια με το προηγούμενο dataset, όπου στη συγκεκριμένη περίπτωση πρόκειται για ένα δυαδικό πρόβλημα κατηγοριοποίησης, το οποίο ανιχνεύει την ύπαρξη φωτιάς μέσα σε ένα δωμάτιο. Το συγκεκριμένο dataset περιέχει 14 διαφορετικά χαρακτηριστικά:

- Θερμοκρασία, Υγρασία
- VOC, eCO₂, Raw H₂, Raw Ethanol
- Πίεση
- PM_{1.0}, PM_{2.5}
- NC_{0.5}, NC_{1.0}, NC_{2.5}
- CNT

Αρχικά, μέσα από τη διαδικασία του Pre-Processing των δεδομένων, έγινε αφαίρεση χαρακτηριστικών των οποίων οι αισθητήρες που χρησιμοποιήθηκαν δεν μπορούν να μετρήσουν. Ειδικότερα, τα χαρακτηριστικά που παρέμειναν είναι η Θερμοκρασία, η Υγρασία, το TVOC και η πίεση.

Αξίζει να σημειωθεί ότι το χαρακτηριστικό TVOC αντιστοιχεί στην εμπειρική εκτίμηση που προκύπτει από τον αισθητήρα MQ-135 (βλ. Ενότητα 2.2.2), καθώς αυτή ήταν η κλίμακα τιμών με βάση την οποία εκπαιδεύτηκε το μοντέλο. Ο αισθητήρας SGP40, ο οποίος προστέθηκε στη συσκευή IoT σε μεταγενέστερο στάδιο της υλοποίησης, παρέχει ανεξάρτητη, πραγματική μέτρηση δείκτη ποιότητας αέρα (VOC Index) σε διαφορετική κλίμακα τιμών (0–500) και δεν χρησιμοποιείται ως χαρακτηριστικό εισόδου στο παρόν μοντέλο.

Σε αντίθεση με το προηγούμενο dataset, στο Smoke Detection dataset χρειάστηκε να γίνει διαχωρισμός σε σύνολα εκπαίδευσης 80% και ελέγχου 20% μέσω της συνάρτησης `train_test_split`. Μια λεπτομέρεια που πρέπει να επισημανθεί είναι ότι ο αριθμός των διαθέσιμων δεδομένων στην κατηγορία που θέλουμε να προβλέψουμε έχει διαφορετική αναλογία. Για να αποτρέψουμε οποιοδήποτε πρόβλημα στο μοντέλο που εκπαιδεύσαμε, κρατήσαμε τις ίδιες αναλογίες όπως στο αρχικό dataset, αναδιατάσσοντας ωστόσο τις εγγραφές.

Στο επόμενο βήμα έχουμε την εκπαίδευση των αλγορίθμων. Όπως και στην προηγούμενη ενότητα, χρησιμοποιούμε pipelines για αποφυγή data-leaks. Οι αλγόριθμοι που χρησιμοποιήθηκαν είναι ο LinearSVC, ο KNeighborsClassifier και ο RandomForestClassifier. Κάθε αλγόριθμος έχει το δικό του pipeline, όπου έγινε χρήση του RobustScaler, μια συνάρτηση που αφαιρεί τη διάμεσο τιμή και κλιμακώνει τα δεδομένα σύμφωνα με το εύρος των ποσοστημορίων.

Κάθε αλγόριθμος αξιολογήθηκε με τη χρήση cross-validation. Η δοκιμαστική σύγκριση των τριών αλγορίθμων (LinearSVC, KNeighborsClassifier, RandomForestClassifier) μέσω Confusion Matrix παρουσιάζεται αναλυτικά στο Κεφάλαιο 4.

Το επόμενο βήμα στη μεθοδολογία μας είναι να βρούμε τις κατάλληλες παραμέτρους για τους αλγόριθμους KNeighborsClassifier και RandomForestClassifier με την χρήση της συνάρτησης GridSearchCV. Στους πίνακες 3.4, 3.5 παρουσιάζεται ο χώρος της αναζήτησης για τους αλγόριθμους.

Υπερπαραμέτρος (Parameter)	Τιμές Αναζήτησης (Grid Values)
n_neighbors	3, 5, 7, 11, 21
weights	'uniform', 'distance'
leaf_size	15, 20, 30, 40, 50
p	1, 2

Πίνακας 3.4: Ο χώρος αναζήτησης (Parameter Grid) για τη βελτιστοποίηση των υπερπαραμέτρων του αλγορίθμου KNeighborsClassifier.

Υπερπαραμέτρος (Parameter)	Τιμές Αναζήτησης (Grid Values)
n_estimators	50, 100, 200
criterion	'gini', 'entropy'
max_depth	10, 20, 30
min_samples_split	2, 5, 10
min_samples_leaf	1, 2, 4

Πίνακας 3.5: Ο χώρος αναζήτησης (Parameter Grid) για τη βελτιστοποίηση των υπερπαραμέτρων του αλγορίθμου RandomForestClassifier.

Τα αποτελέσματα της αναζήτησης παρουσιάζονται για κάθε αλγόριθμο στον Πίνακα 3.6, τα οποία χρησιμοποιήθηκαν για την εκπαίδευση και την αξιολόγηση των αλγορίθμων. Στην Εικόνα 4.5 παρουσιάζονται τα αποτελέσματα των αλγορίθμων με τη χρήση των βέλτιστων παραμέτρων και η απόδοσή τους στο test set.

KNeighborsClassifier

Υπερπαράμετρος (Parameter)	Βέλτιστη Τιμή (Best Value)
leaf_size	15
n_neighbors	5
p	1
weights	'distance'

RandomForestClassifier

Υπερπαράμετρος (Parameter)	Βέλτιστη Τιμή (Best Value)
criterion	'entropy'
max_depth	20
min_samples_leaf	1
min_samples_split	2
n_estimators	200

Πίνακας 3.6: Οι βέλτιστες τιμές των υπερπαραμέτρων που προέκυψαν για τους αλγόριθμους KNeighborsClassifier και RandomForestClassifier.

3.2.3 Εξαγωγή τελικών μοντέλων

Μετά την αξιολόγηση, την επιτυχή εκπαίδευση αλλά και την βελτιστοποίηση των μοντέλων μηχανικής μάθησης, το επόμενο σημαντικό βήμα είναι η αποθήκευση τους το λεγόμενο Model Persistence. Με αυτή την διαδικασία διατηρείται η κατάσταση του εκπαιδευμένου αλγορίθμου σε ένα εξωτερικό αρχείο, με στόχο να μπορεί να φορτωθεί και να χρησιμοποιηθεί μελλοντικά για την παραγωγή προβλέψεων σε νέα δεδομένα, χωρίς να απαιτείται να επαναληφθεί από την αρχή η χρονοβόρα διαδικασία της εκπαίδευσης.

Μια συνηθισμένη πρακτική όσο αναφορά την εξαγωγή και αποθήκευση μοντέλων στην Python, είναι η χρήση της ενσωματωμένης βιβλιοθήκης `pickle`. Ωστόσο, για τα μοντέλα που αναπτύχθηκαν μέσω της βιβλιοθήκης Scikit-Learn, επιλέχθηκε η ειδική βιβλιοθήκη `joblib`, ακολουθώντας την επίσημη σύσταση της τεκμηρίωσης του Scikit-Learn [36].

Οι λόγοι για τους οποίους επιλέχθηκε το `joblib` έναντι του απλού `pickle` περιγράφονται παρακάτω:

- **Καλύτερη απόδοση σε μεγάλους όγκους δεδομένων:** Το `joblib` είναι φτιαγμένο ώστε να μπορεί να διαχειρίζεται πολύ πιο αποδοτικά αντικείμενα, τα οποία περιέχουν μεγάλους πίνακες της βιβλιοθήκης NumPy. Επειδή οι εκτιμητές του Scikit-Learn συνήθως αποθηκεύουν μεγάλους πίνακες εσωτερικά για να μπορέσουν να κρατήσουν τις παραμέτρους της εκπαίδευσης τους, το `joblib` προσφέρει ταχύτερους χρόνους αποθήκευσης αλλά και φόρτωσης.
- **Καλύτερη διαχείριση μνήμης:** Προσφέρει εξελιγμένους μηχανισμούς διαχείρισης και αντικατάστασης μνήμης. Πράγμα, που τον καθιστά ιδιαίτερα σημαντικό όταν τα μοντέλα πρόκειται να μεταφερθούν και να εκτελεστούν στο Edge Layer του συστήματος, όπου η διαθέσιμη μνήμη και οι υπολογιστικοί πόροι πρέπει να αξιοποιούνται με τον καλύτερο δυνατό τρόπο.

Εφαρμόζοντας αυτή την μεθοδολογία, τα δυο μοντέλα εξήχθησαν σε μορφή αρχείων .pkl. Το pipeline του LinearSVC για το Occupancy Detection αποθηκεύτηκε ως `occupancy_mode_pipeline.pkl`, ενώ το αντίστοιχο μοντέλο του Random_Forest_Classifier για το Smoke Detection αποθηκεύτηκε ως `smoke_detection_model_RandomForest.pkl` μέσω της συνάρτησης `joblib.dump()`. Τα αρχεία αξιοποιήθηκαν στο Edge επίπεδο για την τοπική επεξεργασία των δεδομένων σε πραγματικό χρόνο.

3.3 Εκτέλεση Μοντέλων σε Πραγματικό Χρόνο & Ανάλυση Κώδικα Εκτέλεσης Μοντέλων

Μετά την εκπαίδευση αλλά και εξαγωγή των μοντέλων Μηχανικής Μάθησης, το επόμενο βήμα είναι η ανάπτυξη και εκτέλεση τους στο Edge Layer, για την παράγωγή προβλέψεων σε πραγματικό χρόνο. Για τον λόγο αυτό, αναπτύχθηκαν δυο ξεχωριστά Jupyter Notebooks τα οποία λειτουργούν ως UDP servers, αναλύοντας τις μετρήσεις των αισθητήρων και αποστέλλοντας τα αποτελέσματα στο Cloud. Οι πλήρεις κώδικες των υλοποιήσεων παρατίθενται στο Παράρτημα Α.3.

Κατά την αρχική υλοποίηση, τα δύο μοντέλα εκτελούνταν ως ξεχωριστά Jupyter Notebooks, καθένα με το δικό του UDP server στην ίδια θύρα λήψης (`LISTEN_PORT = 39361`). Στην πράξη, αυτό οδήγησε σε σύγκρουση θύρας: όταν και τα δύο *scripts* προσπαθούσαν να δεσμεύσουν ταυτόχρονα την ίδια θύρα, το ένα από τα δύο έπαυε να λαμβάνει δεδομένα, ενώ σε ορισμένες περιπτώσεις, μετά από μικρό χρονικό διάστημα, σταματούσε να λειτουργεί και το δεύτερο. Για την οριστική επίλυση του προβλήματος, τα δύο *scripts* ενοποιήθηκαν σε ένα ενιαίο Jupyter Notebook, το οποίο φορτώνει και τα δύο μοντέλα στην ίδια διεργασία και χρησιμοποιεί ένα και μοναδικό UDP socket για τη λήψη των δεδομένων από το Arduino. Η λογική εκτέλεσης αναλύεται παρακάτω.

3.3.1 Ενοποιημένος Κώδικας Εκτέλεσης Μοντέλων

- **Εισαγωγή Βιβλιοθηκών και Φόρτωση και των Δύο Μοντέλων:** Εισάγονται οι απαραίτητες βιβλιοθήκες (`socket`, `joblib`, `pandas`, `requests` κ.λπ.) και απενεργοποιούνται οι προειδοποιήσεις SSL του `urllib3`. Σε αντίθεση με την προηγούμενη υλοποίηση, εδώ φορτώνονται **και τα δύο** μοντέλα (`occ_model`, `smoke_model`) στην ίδια διεργασία μέσω `joblib.load()`, πριν την έναρξη του κύριου βρόχου.
- **Ρύθμιση Δικτύου και Ενιαίος UDP Server:** Δημιουργείται **ένα και μοναδικό** `socket` στη διεύθυνση `10.42.0.1:39361`, το οποίο πλέον αναλαμβάνει τη λήψη δεδομένων και για τα δύο μοντέλα ταυτόχρονα, εξαλείφοντας τη σύγκρουση θύρας που περιγράφηκε παραπάνω.
- **Επικοινωνία με το Cloud API:** Η λογική αυθεντικοποίησης (`get_token()`) παραμένει ίδια με πριν. Ωστόσο, η συνάρτηση `send_to_api()` τροποποιήθηκε ώστε να δέχεται πλέον ως παράμετρο το `endpoint` (`send_to_api(endpoint, data)`), επιτρέποντας στο ίδιο κομμάτι κώδικα να στέλνει σωστά τα δεδομένα του κάθε μοντέλου στο αντίστοιχο `endpoint` του API (`/occupancy` ή `/smoke`), αντί να χρησιμοποιείται ένα σταθερό, κοινό `endpoint` και για τα δύο.

- **Προετοιμασία Καταγραφής:** Διατηρούνται δύο ξεχωριστά αρχεία CSV (`occupancy_log.csv`, `smoke_log.csv`), καθένα με τις δικές του επικεφαλίδες, ώστε τα δεδομένα κάθε μοντέλου να καταγράφονται τοπικά, εξασφαλίζοντας τον πλήρη διαχωρισμό τους.
- **Μετρητές:** Αρχικοποιούνται ξεχωριστές δομές δεδομένων για την καταγραφή στατιστικών για κάθε μοντέλο (`occ_counts`, `smoke_counts`), καθώς και μεταβλητές παρακολούθησης της προηγούμενης κατάστασης (`previous_occ`, `previous_fire`) για τον εντοπισμό αλλαγών κατάστασης σε πραγματικό χρόνο.
- **Κύριος Βρόχος - `recvfrom()`:** Σε κάθε επανάληψη, το *script* λαμβάνει ένα πακέτο δεδομένων (`temp`, `humidity`, `co2`, `lux`, `tvoc`, `pressure`) και εκτελεί διαδοχικά και τις δύο προβλέψεις πάνω στο ίδιο πακέτο:
 - Πρώτα υπολογίζεται η πρόβλεψη `occupancy` (χρησιμοποιώντας τα χαρακτηριστικά `Temperature`, `Light`, `CO2`) και αποστέλλεται στο `/occupancy`.
 - Στη συνέχεια υπολογίζεται η πρόβλεψη `smoke` (χρησιμοποιώντας τα χαρακτηριστικά `Temperature`, `Humidity`, `TVOC`, `Pressure`), όπου εφαρμόζεται ο κανόνας υπερίσχυσης της θερμοκρασίας (`temp > 50.0 → FIRE`) και αποστέλλεται στο `/smoke`.

Και τα δύο *blocks* περιβάλλονται από τον ίδιο μηχανισμό διαχείρισης σφαλμάτων (`ValueError`, `IndexError`) και τον μηχανισμό ομαλού τερματισμού (`except KeyboardInterrupt / finally: sock.close()`) που υπήρχε στις αρχικές, ξεχωριστές υλοποιήσεις.

Ο πλήρης κώδικας της υλοποίησης παρατίθεται στο Παράρτημα A.3.

3.4 Cloud: Docker Ορχήστρωση και Διασύνδεση

Η ενορχήστρωση του backend στο Cloud Layer γίνεται μέσω Docker Compose, με τέσσερα containers: FastAPI, PostgreSQL, Grafana και Nginx. Το πλήρες αρχείο ορισμού (`docker-compose.yml`) παρατίθεται στο Παράρτημα A'.4, ενώ παρακάτω αναλύεται λεπτομερώς κάθε service ξεχωριστά.

3.4.1 Ανάλυση των Services

Postgres: Χρησιμοποιεί την επίσημη εικόνα `postgres:15`. Το όνομα βάσης, ο χρήστης και ο κωδικός πρόσβασης δεν είναι ενσωματωμένα μέσα στο αρχείο, αλλά περνιούνται ως μεταβλητές περιβάλλοντος (`${POSTGRES_DB}` κ.λπ.) από ξεχωριστό αρχείο `.env`, το οποίο δεν συμπεριλαμβάνεται στο σύστημα ελέγχου (`.gitignore`) για λόγους ασφάλειας. Ο κατάλογος `postgres_data` ορίζεται ως `named volume`, διασφαλίζοντας ότι τα δεδομένα της βάσης υπάρχουν ακόμα κι αν το container επανεκκινηθεί ή αντικατασταθεί. Το port 5432 εκτίθεται στον host, επιτρέποντας απευθείας σύνδεση για debugging αν χρειαστεί. Η πολιτική `restart: always` εξασφαλίζει ότι το container επανεκκινείται αυτόματα σε περίπτωση `crash` ή `reboot` του VM.

API: Σε αντίθεση με τα υπόλοιπα services που χρησιμοποιούν έτοιμες εικόνες, το API container κατασκευάζεται τοπικά (`build: ./api`) από τον κωδικά που φτιάχτηκε στο FastAPI όπως μπορούμε να δούμε στο Παράρτημα Α'.4, μέσω του Dockerfile που βρίσκεται στον αντίστοιχο υποφάκελο. Η οδηγία `depends_on: postgres` διασφαλίζει ότι το container του PostgreSQL ξεκινάει πρώτο, είναι σημαντικό να αναφερθεί ότι αυτό εγγυάται μόνο τη σειρά εκκίνησης, όχι ότι η βάση είναι ήδη έτοιμη να δεχτεί συνδέσεις τη στιγμή που ξεκινάει το api.

Grafana: Χρησιμοποιεί την επίσημη εικόνα `grafana/grafana:latest`, αντιστοιχίζοντας τη με την θύρα:3000 χρησιμοποιώντας το `named volume (grafana_data)` για τη μόνιμη αποθήκευση των dashboards και των ρυθμίσεων. Οι μεταβλητές `GF_SMTP_*` ενεργοποιούν και ρυθμίζουν την αποστολή email ειδοποιήσεων μέσω Gmail SMTP, με τα διαπιστευτήρια επίσης να προέρχονται από το αρχείο `.env`.

Nginx: Λειτουργεί ως reverse proxy, αντιστοιχίζοντας τα ports 80 (HTTP, ανακατεύθυνση σε HTTPS) και 443 (HTTPS). Μέσω volumes, φορτώνεται το αρχείο ρυθμίσεων (`nginx.conf`) και το ζεύγος self-signed πιστοποιητικό/κλειδιού (βλ. Ενότητα 2.4.3). Η οδηγία `depends_on` εξασφαλίζει ότι τα api και grafana έχουν ήδη ξεκινήσει πριν το Nginx, αφού είναι αυτά που προωθεί. Τα volumes `certbot_data/certbot_www` και οι προσαρμοσμένοι DNS resolvers (8.8.8.8, 8.8.4.4) αποτελούν τον κωδικά που γράφτηκε αρχικά, όπου περιέχει την ανεπιτυχή προσπάθεια απόκτησης πιστοποιητικού Let's Encrypt (βλ. Ενότητα 2.4.3), όπου και παρέμειναν στο αρχείο χωρίς πρακτική χρήση πλέον, καθώς τελικά χρησιμοποιήθηκε το self-signed πιστοποιητικό.

3.4.2 Cloud: Αποθήκευση Δεδομένων στη Βάση Δεδομένων

Η μόνιμη αποθήκευση των δεδομένων του συστήματος πραγματοποιείται στη βάση δεδομένων PostgreSQL (`container: iot-postgres`), η οποία τρέχει ως ένα από τα τέσσερα services του `docker-compose.yml` (βλ. Ενότητα 3.4). Η βάση έχει το όνομα `iotdb` και περιέχει δύο πίνακες `occupancy` και `smoke`, έναν για κάθε μοντέλο μηχανικής μάθησης του συστήματος. Κάθε φορά που το API λαμβάνει μια νέα πρόβλεψη από το Edge Layer μέσω του αντίστοιχου endpoint, η εγγραφή αποθηκεύεται σε πραγματικό χρόνο στον αντίστοιχο πίνακα, ώστε να είναι διαθέσιμη τόσο για ιστορική ανάλυση όσο και για την οπτικοποίηση μέσω του Grafana (βλ. Ενότητα 3.4.2). Η δομή των δύο πινάκων παρουσιάζεται στους Πίνακες 3.7 και 3.8.

Στήλη (Column)	Τύπος (Type)	Περιγραφή
<code>id</code>	<code>integer</code>	Πρωτεύον κλειδί , αυτόματη αρίθμηση
<code>timestamp</code>	<code>timestamp</code>	Χρονοσήμανση της μέτρησης
<code>temperature</code>	<code>double precision</code>	Θερμοκρασία (°C)
<code>humidity</code>	<code>double precision</code>	Υγρασία (%)
<code>co2</code>	<code>double precision</code>	Συγκέντρωση CO2 (ppm)
<code>lux</code>	<code>double precision</code>	Ένταση φωτισμού (lux)
<code>voc</code>	<code>double precision</code>	Εκτιμώμενη τιμή TVOC »
<code>prediction</code>	<code>varchar(20)</code>	Πρόβλεψη μοντέλου (occupied/not occupied)

Πίνακας 3.7: Δομή του πίνακα `occupancy` στη βάση δεδομένων `iotdb`.

Στήλη (Column)	Τύπος (Type)	Περιγραφή
id	integer	Πρωτεύον κλειδί , αυτόματη αρίθμηση
timestamp	timestamp	Χρονοσήμανση της μέτρησης
temperature	double precision	Θερμοκρασία (°C)
humidity	double precision	Υγρασία (%)
co2	double precision	Συγκέντρωση CO2 (ppm)
lux	double precision	Ένταση φωτισμού (lux)
voc	double precision	Εκτιμώμενη τιμή TVOC»
tvoc	double precision	Συνολικές πτητικές οργανικές ενώσεις (TVOC, ppb)
voc_index_sgp40	integer	Πραγματικός δείκτης ποιότητας αέρα (VOC Index, 0–500)
pressure	double precision	Ατμοσφαιρική πίεση (hPa)
prediction	varchar(20)	Πρόβλεψη μοντέλου (FIRE/OK)
confidence	double precision	Βεβαιότητα (confidence) της πρόβλεψης (0--1)

Πίνακας 3.8: Δομή του πίνακα smoke στη βάση δεδομένων iotdb.

3.4.3 Παρακολούθηση σε Πραγματικό Χρόνο (Grafana)

Το κομμάτι της οπτικοποίησης επιβεβαιώθηκε μέσω του Grafana, με πραγματικά δεδομένα λειτουργίας του συστήματος. Κάθε *panel* του *dashboard* συνδέεται απευθείας με τη βάση δεδομένων PostgreSQL (iotdb) μέσω του ενσωματωμένου data source του Grafana για PostgreSQL, χωρίς να μεσολαβεί κάποιο επιπλέον API.

Κάθε *panel* εκτελεί ένα ξεχωριστό SQL ερώτημα, το οποίο αντλεί την τιμή μιας συγκεκριμένης στήλης από τον πίνακα occupancy ή smoke, ταξινομημένη ως προς τη χρονοσήμανση (timestamp). Σε όλα τα ερωτήματα χρησιμοποιήθηκε η εντολή `$__timeFilter(timestamp)`, η οποία επιτρέπει στο Grafana να περιορίζει αυτόματα τα αποτελέσματα στο χρονικό διάστημα που έχει επιλέξει ο χρήστης από το *dashboard*, χωρίς να χρειάζεται να ξαναγραφτεί το ερώτημα. Με αυτόν τον τρόπο, όλα τα *panels* ανταποκρίνονται με συνεπή τρόπο στην επιλογή χρονικού διαστήματος.

Το τελικό *dashboard*, όπως διαμορφώθηκε, αποτελείται από οκτώ *panels* (Εικόνα 3.4):

- θερμοκρασία δωματίου
- επίπεδα διοξειδίου του άνθρακα (CO2)
- κατάσταση του χώρου (παρουσία ή απουσία ατόμου)
- ανίχνευση φωτιάς μέσα στον χώρο
- υγρασία
- επίπεδα φωτεινότητας
- συγκέντρωση πτητικών οργανικών ενώσεων (TVOC)
- πραγματικός δείκτης ποιότητας αέρα (VOC Index) από τον αισθητήρα SGP40

Παρακάτω παρατίθεται το SQL ερώτημα για κάθε *panel* ξεχωριστά.

Έπειτα από την ενοποίηση του κώδικα εκτέλεσης (βλ. Ενότητα 3.3.1), και οι δύο πίνακες (occupancy, smoke) λαμβάνουν τις πλήρεις μετρήσεις αισθητήρων ανά πακέτο. Για λόγους συνέπειας, όλα τα *panels* περιβάλλοντος (TEMP ROOM, Level CO2, Humidity, TVOC, VOC Index (SGP40), Lux) αντλούν δεδομένα ενιαία από τον πίνακα smoke.



Σχήμα 3.4: Το τελικό Grafana *dashboard* με τα οκτώ *panels* παρακολούθησης σε πραγματικό χρόνο.

TEMP ROOM: Απεικονίζει τη χρονοσειρά της θερμοκρασίας δωματίου, όπως προκύπτει από τον πίνακα *smoke*.

```
SELECT
  timestamp::timestamp AS time,
  temperature
FROM smoke
WHERE $__timeFilter(timestamp::timestamp)
ORDER BY timestamp ASC;
```

Κώδικας 3.1: Ερώτημα SQL για το *panel* "TEMP ROOM".

Level CO2: Απεικονίζει τη χρονοσειρά της συγκέντρωσης CO2, επίσης από τον πίνακα *smoke*.

```
SELECT
  timestamp::timestamp AS time,
  co2
FROM smoke
WHERE $__timeFilter(timestamp::timestamp )
ORDER BY timestamp ASC;
```

Κώδικας 3.2: Ερώτημα SQL για το *panel* "Level Co2".

Κατάσταση Χώρου: Απεικονίζει την τελική πρόβλεψη (prediction) του μοντέλου LinearSVC για την κατάσταση κατάληψης του χώρου, σε μορφή *state timeline*.

```
SELECT
    timestamp AS "time",
    prediction AS Κατάσταση"
FROM occupancy
WHERE $__timeFilter(timestamp)
    AND prediction IN ('EMPTY', 'OCCUPANCY')
ORDER BY timestamp ASC;
```

Κώδικας 3.3: Ερώτημα SQL για το *panel* “Κατάσταση Χώρου”.

Smoke Detection: Απεικονίζει, επίσης ως *state timeline*, την πρόβλεψη του μοντέλου RandomForestClassifier για την ανίχνευση καπνού/φωτιάς. Η στήλη prediction μετατρέπεται σε δυαδική τιμή (1 για FIRE, 0 για OK) μέσω της έκφρασης CASE WHEN, ώστε να είναι δυνατή η οπτική διαφοροποίηση των δύο καταστάσεων με χρώμα στο *panel*.

```
SELECT
    timestamp::timestamp AS time,
    CASE WHEN prediction = 'FIRE' THEN 1 ELSE 0 END AS Κατάσταση"
FROM smoke
WHERE $__timeFilter(timestamp::timestamp)
ORDER BY timestamp ASC;
```

Κώδικας 3.4: Ερώτημα SQL για το *panel* “Smoke Detection”.

Humidity: Απεικονίζει τη χρονοσειρά της σχετικής υγρασίας, από τον πίνακα smoke.

```
SELECT
    timestamp::timestamp AS time,
    humidity
FROM smoke
WHERE $__timeFilter(timestamp::timestamp)
ORDER BY timestamp ASC;
```

Κώδικας 3.5: Ερώτημα SQL για το *panel* “Humidity”.

TVOC : Απεικονίζει τη χρονοσειρά της εκτιμώμενης τιμής TVOC (από τον αισθητήρα MQ-135), από τον πίνακα smoke.

```
SELECT
    timestamp::timestamp AS time,
    tvoc
FROM smoke
WHERE $__timeFilter(timestamp::timestamp )
ORDER BY timestamp ASC;
```

Κώδικας 3.6: Ερώτημα SQL για το *panel* “TVOC”.

VOC Index (SGP40) : Απεικονίζει τη χρονοσειρά του πραγματικού δείκτη ποιότητας αέρα (VOC Index), όπως μετράται απευθείας από τον αισθητήρα SGP40, σε αντιδιαστολή με το πεδίο TVOC που αποτελεί εμπειρική εκτίμηση από τον MQ-135 (βλ. Ενότητα 2.2.2).

```
SELECT
    timestamp::timestamp AS time,
    voc_index_sgp40
FROM smoke
WHERE $__timeFilter(timestamp::timestamp)
ORDER BY timestamp ASC;
```

Κώδικας 3.7: Ερώτημα SQL για το *panel* "VOC Index (SGP40)".

Lux: Απεικονίζει τη χρονοσειρά της έντασης φωτισμού, από τον πίνακα *smoke*.

```
SELECT
    timestamp::timestamp AS time,
    lux
FROM smoke
WHERE $__timeFilter(timestamp::timestamp)
ORDER BY timestamp ASC;
```

Κώδικας 3.8: Ερώτημα SQL για το *panel* "Lux".

Επιπλέον, στο Grafana έχει ρυθμιστεί ένας κανόνας ειδοποίησης (*alert rule*) με όνομα <<Smoke Detection Alert>>, ο οποίος παρακολουθεί σε πραγματικό χρόνο την εμφάνιση κατάστασης FIRE στον πίνακα *smoke*. Το query του κανόνα μετρά τον αριθμό των εγγραφών με πρόβλεψη FIRE στο τελευταίο λεπτό:

```
SELECT COUNT(*) FROM smoke
WHERE prediction = 'FIRE'
AND timestamp > NOW() - INTERVAL '1 minute'
```

Κώδικας 3.9: Ερώτημα SQL του *alert rule* "Smoke Detection Alert".

Η συνθήκη ενεργοποίησης ορίζεται ως *Is above 0*, δηλαδή ο κανόνας πυροδοτείται όταν υπάρχει έστω και μία εγγραφή FIRE μέσα στο παραπάνω χρονικό περιθώριο. Η αξιολόγηση του κανόνα πραγματοποιείται κάθε 1 λεπτό, ενώ έχει οριστεί περίοδος αναμονής 1 λεπτού πριν η κατάσταση θεωρηθεί οριστικά ως <<Firing>>, όπως και μηδενική διάρκεια παραμονής μετά τη λήξη του συμβάντος (*keep firing for: 0s*), ώστε η ειδοποίηση επίλυσης να αποστέλλεται άμεσα μόλις το query επιστρέψει ξανά μηδενικό αποτέλεσμα. Η αποστολή των ειδοποιήσεων γίνεται μέσω ηλεκτρονικού ταχυδρομείου, χρησιμοποιώντας το προκαθορισμένο σημείο επικοινωνίας *email-alert*. Η ρύθμιση του SMTP διακομιστή που καθιστά δυνατή αυτή την αποστολή περιγράφεται στην Ενότητα 3.4.1, στην ανάλυση του service *grafana*.

Κεφάλαιο 4

Αποτελέσματα

4.1 Αξιολόγηση Μοντέλων Μηχανικής Μάθησης

Στην παρούσα ενότητα παρουσιάζονται τα αριθμητικά αποτελέσματα της εκπαίδευσης, της σύγκρισης και της τελικής αξιολόγησης των δύο μοντέλων μηχανικής μάθησης, όπως προέκυψαν από τη μεθοδολογία που περιγράφηκε στο Κεφάλαιο 3.

4.1.1 Occupancy Detection

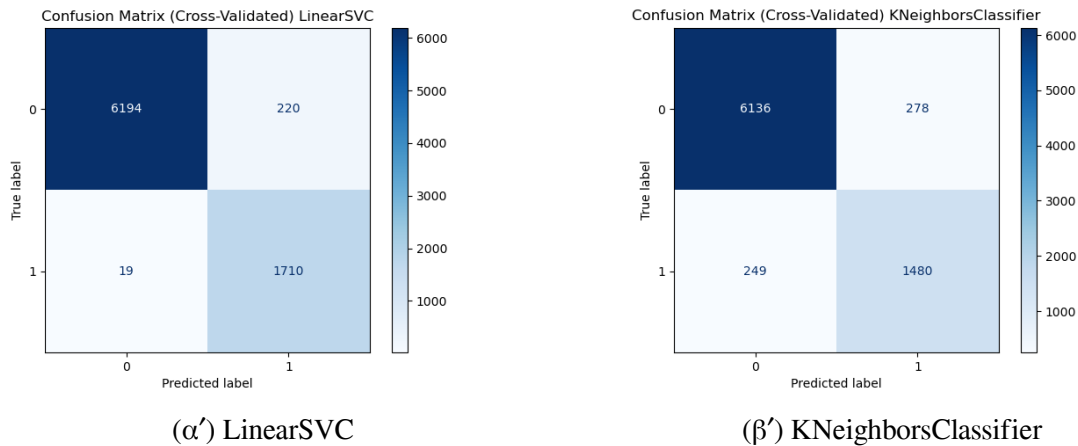
Πραγματοποιήσαμε μια δοκιμαστική εκπαίδευση χρησιμοποιώντας τα 3 βέλτιστα χαρακτηριστικά και τις προεπιλεγμένες παραμέτρους των δύο αλγορίθμων. Η αξιολόγηση έγινε με τη μέθοδο Διασταυρούμενης Επικύρωσης 5 τμημάτων (5-fold Cross-Validation), με μετρική το *F1-Macro Score*. Τα αποτελέσματα παρατίθενται στον Πίνακα 4.1. Παρατηρούμε ότι ο *LinearSVC* παρουσιάζει σταθερά υψηλές αποδόσεις σε όλα τα folds (από 0.896 έως 0.995). Αντίθετα, ο *KNeighborsClassifier* (με $k=5$) εμφανίζει σημαντική αστάθεια, καθώς στο 3ο fold η απόδοσή του πέφτει στο 0.661.

Algorithm	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
LinearSVC	0.9291184	0.98375872	0.99542746	0.99272077	0.89624136
KNeighborsClassifier	0.93006032	0.95725483	0.66142675	0.98124908	0.99088202

Πίνακας 4.1: Αποτελέσματα των αλγορίθμων ανά Fold.

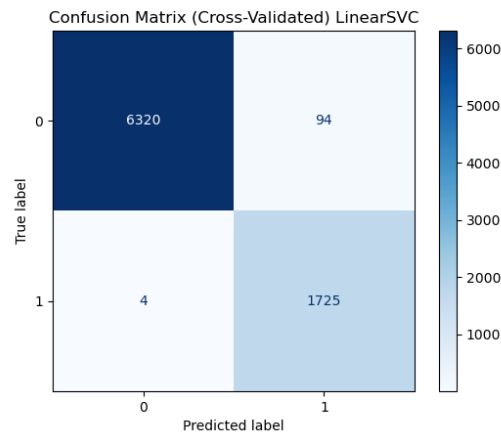
Για να επιβεβαιώσουμε περαιτέρω την υπεροχή του *LinearSVC*, αναλύσαμε τους Πίνακες Σύγχυσης (Confusion Matrices) των δύο αλγορίθμων (Εικόνα 4.1). Ο πίνακας αυτός απεικονίζει τα True Positives και True Negatives (στην κύρια διαγώνιο) έναντι των False Positives (κάτω αριστερά) και False Negatives (πάνω δεξιά). Ο *LinearSVC* εμφάνισε αισθητά λιγότερα σφάλματα ταξινόμησης συγκριτικά με τον *KNeighborsClassifier*, επιβεβαιώνοντας πως είναι ο καταλληλότερος αλγόριθμος για τη λύση του προβλήματός μας.

Έπειτα έγινε χρήση των παραμέτρων από τον αλγόριθμο βελτιστοποίησης (solver) για να εκπαιδευσουμε το τελικό μοντέλο όπου και στη συνέχεια, εφαρμόσαμε διασταυρούμενη επικύρωση (cross-validation), παρατηρήσαμε καλύτερα αποτελέσματα στο πίνακα σύγχυσης με πιο αποδοτικά αποτελέσματα στα False Positive από 19 σε 4 και σε False Negative από 220 σε 94.



Σχήμα 4.1: Σύγκριση των αλγορίθμων χρησιμοποιώντας το Confusion matrix

Τα αποτελέσματα αυτά παρουσιάζονται στο σχήμα 4.1α' και οι βελτιστοποιήσεις στο σχήμα 4.2.



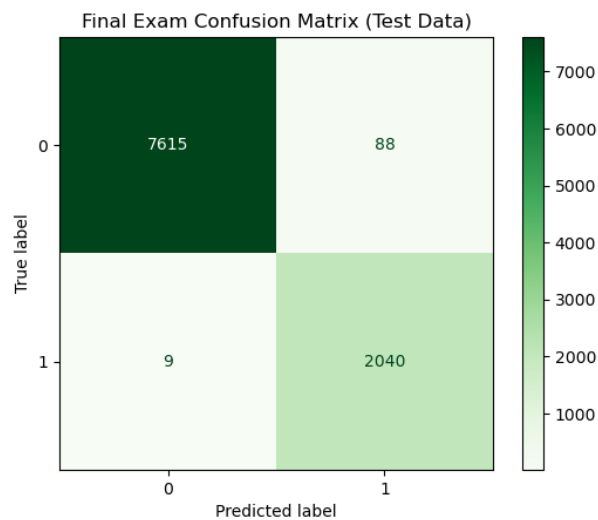
Σχήμα 4.2: Αποτελέσματα του Confusion Matrix με τις βέλτιστες παραμέτρους

Το μοντέλο μας μέχρι στιγμής έχει εκπαιδευτεί σε δεδομένα εκπαίδευσης (training data), οπότε για να δούμε την πραγματική απόδοση του μοντέλου θα πρέπει να γίνει έλεγχος με καινούρια δεδομένα. Τα δεδομένα αυτά βρίσκονται στο test set μας. Με τη βοήθεια της συνάρτησης `classification_report`, η οποία δημιουργεί μια αναφορά κειμένου, εμφανίζονται οι κύριες μετρήσεις ταξινόμησης στον Πίνακα 4.2. Το μοντέλο μας έχει ικανοποιητικά αποτελέσματα και στις δυο κατηγορίες που το εκπαιδεύσαμε, καθώς και υψηλό F1-Score.

Κλάση (Class)	Precision	Recall	F1-score	Support
0	1.00	0.99	0.99	7703
1	0.96	1.00	0.98	2049
Accuracy			0.99	9752
Macro Avg	0.98	0.99	0.99	9752
Weighted Avg	0.99	0.99	0.99	9752

Πίνακας 4.2: Αναφορά ταξινόμησης (Classification Report) για την αξιολόγηση του μοντέλου.

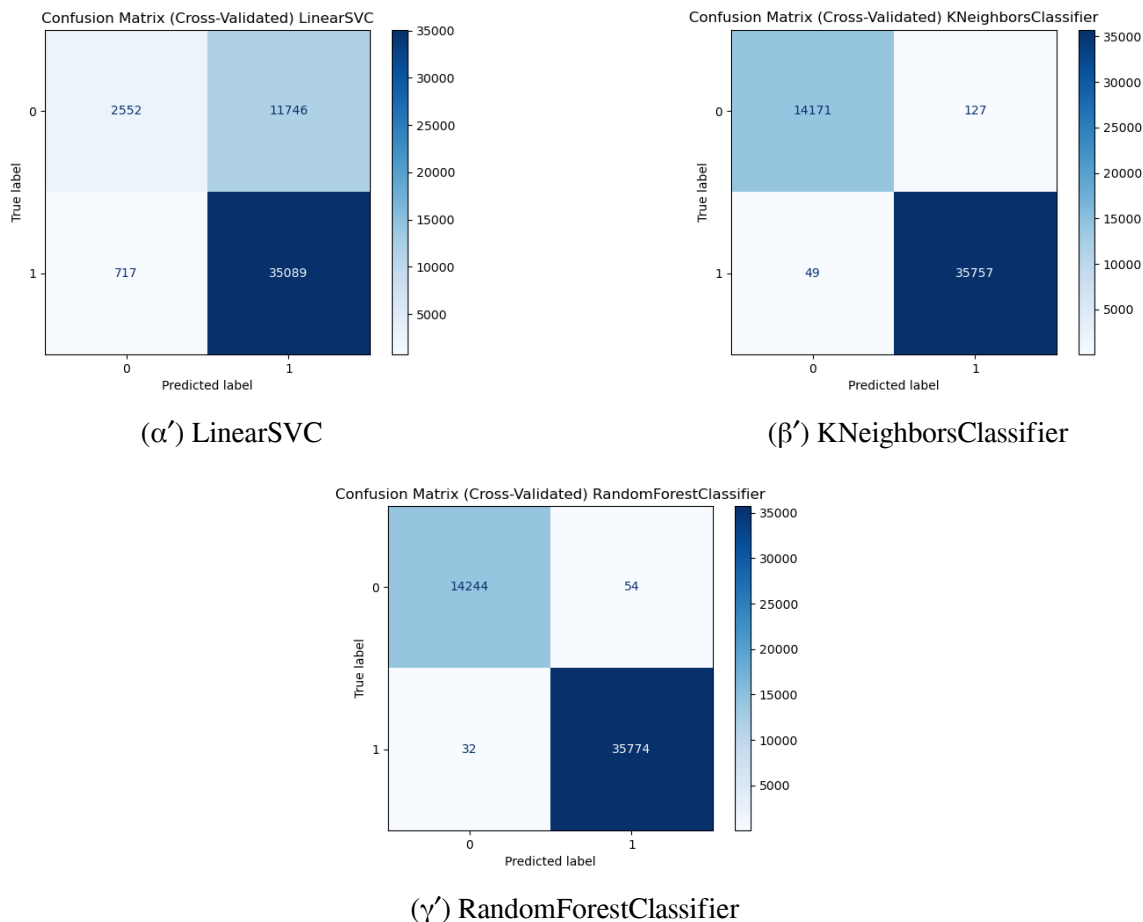
Ιδιαίτερο ενδιαφέρον παρουσιάζουν τα αποτελέσματα του πίνακα σύγχυσης (Confusion Matrix) στα δεδομένα του test set, τα οποία απεικονίζονται στην Εικόνα 4.3. Παρατηρούμε μόλις 88 False Positives και 9 False Negatives, αποτελέσματα που είναι παρόμοια με τις αντίστοιχες επιδόσεις στο training set.



Σχήμα 4.3: Αποτελέσματα από τον Confusion Matrix.

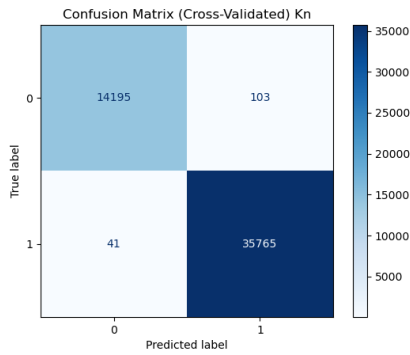
4.1.2 Smoke Detection

Κάθε αλγόριθμος αξιολογήθηκε με την χρήση του cross-validation και υπολογίστηκε ο Confusion Matrix όπως παρουσιάζεται στην εικόνα 4.4.

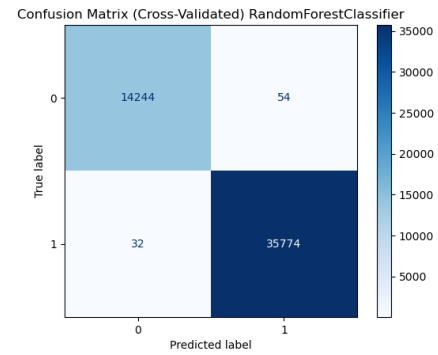


Σχήμα 4.4: Αποτελέσματα των αλγορίθμων χρησιμοποιώντας το Confusion Matrix

Με βάση τα αποτελέσματα στην εικόνα 4.4 παρατηρούμε ότι ο LinearSVC έχει 11746 False Negative ενώ μόνο 2552 True Positive. Αυτό σημαίνει ότι ο αλγόριθμος πιστεύει ότι υπάρχει φωτιά ενώ στην πραγματικότητα δεν υπάρχει. Πολύ καλύτερα αποτελέσματα έχουν ο KNeighborsClassifier και ο RandomForestClassifier, όπου τα αποτελέσματά τους είναι αρκετά κοντά. Στην Εικόνα 4.5 παρουσιάζονται τα αποτελέσματα των αλγορίθμων KNeighborsClassifier και RandomForestClassifier με τη χρήση των βέλτιστων παραμέτρων (βλ. Πίνακα 3.6, Κεφάλαιο 3) και η απόδοσή τους στο training set.

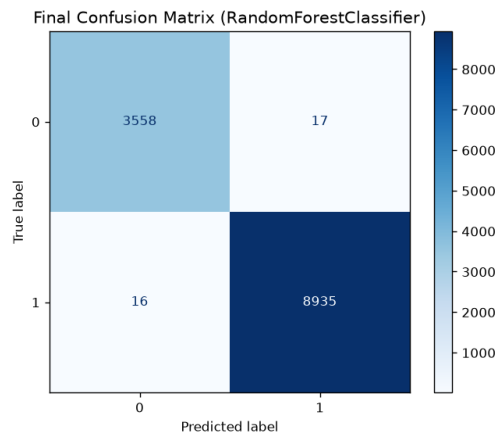


(α') KNeighborsClassifier



(β') RandomForestClassifier

Σχήμα 4.5: Σύγκριση των αλγορίθμων χρησιμοποιώντας το Confusion matrix στο trainigset



Σχήμα 4.6: Αποτελέσματα του RandomForest στο testset

Καλύτερα αποτελέσματα παρουσιάζει ο RandomForest όπως παρατηρούμε στην εικόνα 4.6. Στο πίνακα 4.3 παρουσιάζονται τα αποτελέσματα του RandomForest από το testset.

Κλάση (Class)	Precision	Recall	F1-score	Support
0	1.00	1.00	1.00	3575
1	1.00	1.00	1.00	8951
Accuracy			1.00	12526
Macro Avg	1.00	1.00	1.00	12526
Weighted Avg	1.00	1.00	1.00	12526

Πίνακας 4.3: Αναφορά ταξινόμησης (Classification Report) για την αξιολόγηση του μοντέλου.

4.2 Μοντέλο Ανίχνευσης Κατάληψης Χώρου (Occupancy Detection)

Στο κεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα της λειτουργίας του συστήματος σε πραγματικές συνθήκες, όπως καταγράφηκαν μέσω του Grafana (βλ. Ενότητα 3.4.3), για τα δυο μοντέλα μηχανικής μάθησης: το μοντέλο ανίχνευσης κατάληψης χώρου (occupancy detection) και το μοντέλο ανίχνευσης καπνού (smoke detection). Για κάθε ένα μοντέλο εξετάζονται δυο διαφορετικές καταστάσεις λειτουργίας: μια κατάσταση ηρεμίας ή αλλιώς (*baseline*), όπου δεν υπάρχει κάποιο ερέθισμα προς ανίχνευση, και μια κατάσταση ενεργοποίησης, όπου δέχεται το αντίστοιχο ερέθισμα. Είναι σημαντικό να αναφερθεί ότι τα αποτελέσματα παρουσιάζονται όπως ακριβώς προέκυψαν από το πραγματικό σύστημα, ακόμα και στις περιπτώσεις που δεν ανταποκρίθηκαν πλήρως στην αναμενόμενη συμπεριφορά, ώστε η αξιολόγηση να είναι ειλικρινής και τεκμηριωμένη.

4.2.1 Κατάσταση Ηρεμίας

Στην πρώτη δοκιμή, το δωμάτιο παρέμεινε κενό και τα φώτα κλειστά, προσομοιώνοντας συνθήκες απουσίας. Στην Εικόνα 4.7 παρουσιάζονται οι μετρήσεις θερμοκρασίας δωματίου, του CO₂, της φωτεινότητας σε Lux, της υγρασίας καθώς και του δείκτη ποιότητας του αέρα TVOC, μαζί με την αντίστοιχη πρόβλεψη του μοντέλου και το αποτέλεσμα του μοντέλου ανίχνευσης καπνού, όπως καταγράφηκαν στο Grafana για το συγκεκριμένο χρονικό διάστημα



Σχήμα 4.7: Αποτελέσματα του μοντέλου occupancy detection σε κατάσταση ηρεμίας (κενός χώρος, φώτα κλειστά).

Όπως φαίνεται στην Εικόνα 4.7, το *panel* <<Κατάσταση Χώρου>> παρέμεινε σταθερά στην τιμή EMPTY καθ' όλη τη διάρκεια της δοκιμής, δηλαδή το μοντέλο προέβλεψε σωστά ότι δεν υπήρχε άτομο μέσα στον χώρο. Η τιμή φωτισμού (Lux) παρέμεινε σε πολύ χαμηλά επίπεδα (8 έως 13), λόγω του ότι τα φώτα στο δωμάτιο ήταν κλειστά, η θερμοκρασία παρέμεινε σχετικά σταθερή (32.6 έως 32.8°C) και το CO₂ κυμάνθηκε σε φυσιολογικά επίπεδα εσωτερικού χώρου (106 έως 110 ppm). Η υγρασία κυμάνθηκε μεταξύ 57.2% και 58.2%, ενώ ο δείκτης TVOC κυμάνθηκε μεταξύ 248 και 252, τιμές αναμενόμενες για κενό χώρο χωρίς την ύπαρξη ατόμου. Παράλληλα, το *panel* <<Smoke Detection>> εμφάνισε την τιμή OK, υποδεικνύοντας ότι δεν ανιχνεύθηκε καπνός κατά τη διάρκεια της δοκιμής.

4.2.2 Κατάσταση Ενεργοποίησης

Στη δεύτερη δοκιμή, εισήχθη ερέθισμα αντίστοιχο με το χαρακτηριστικό Light που χρησιμοποιεί το μοντέλο (βλ. Ενότητα 3.1), μέσω της έκθεσης του αισθητήρα φωτισμού σε ισχυρή πηγή φωτός (φακός κινητού τηλεφώνου) πάνω από τον αισθητήρα, προσομοιώνοντας απότομη αύξηση της φωτεινότητας του χώρου. Στην Εικόνα 4.8 παρουσιάζονται τα αντίστοιχα αποτελέσματα.



Σχήμα 4.8: Αποτελέσματα του μοντέλου occupancy detection σε κατάσταση ενεργοποίησης (έκθεση του αισθητήρα φωτός σε φακό κινητού τηλεφώνου).

Στην Εικόνα 4.8 παρατηρούμε ότι, όπως μπορούμε να διακρίνουμε, το *panel* <<Κατάσταση Χώρου>> άλλαξε σε OCCUPANCY, επιβεβαιώνοντας τη σωστή ανίχνευση. Η πιο χαρακτηριστική μεταβολή παρατηρείται στην τιμή φωτισμού, η οποία αυξήθηκε απότομα από τα επίπεδα ηρεμίας (8 έως 13 Lux) σε τιμές μεταξύ 900 και 1050 Lux, γεγονός που αποτέλεσε και το κυρίαρχο σήμα για την αλλαγή πρόβλεψης του μοντέλου. Το CO₂ παρέμεινε σε παρόμοια, μάλλον ελαφρώς χαμηλότερα επίπεδα σε σχέση με την κατάσταση ηρεμίας (101 έως 103 ppm), ενώ η θερμοκρασία εμφάνισε μια μικρή αύξηση (33.0 έως 33.1°C). Η υγρασία κυμάνθηκε μεταξύ 57.9% και 58.4%, ενώ ο δείκτης TVOC παρέμεινε επίσης σε αντίστοιχα επίπεδα (244 έως

248). Όπως και στην προηγούμενη δοκιμή, το *panel* <<Smoke Detection>> εμφάνισε την τιμή OK, καθώς η δοκιμή αυτή αφορούσε αποκλειστικά το μοντέλο ανίχνευσης κατάληψης χώρου. Το γεγονός ότι το CO₂ δεν παρουσίασε αντίστοιχη μεταβολή με τον φωτισμό παραμένοντας μάλιστα στα ίδια ή και χαμηλότερα επίπεδα επιβεβαιώνει ότι, στη συγκεκριμένη δοκιμή, η πρόβλεψη του μοντέλου βασίστηκε σχεδόν αποκλειστικά στο χαρακτηριστικό *Light*, κάτι αναμενόμενο δεδομένου ότι το ερέθισμα αφορούσε στοχευμένα τον αισθητήρα φωτισμού και όχι πραγματική μεταβολή των συνθηκών του χώρου.

4.3 Μοντέλο Ανίχνευσης Καπνού (Smoke Detection)

4.3.1 Κατάσταση Ηρεμίας

Αντίστοιχα, για το μοντέλο ανίχνευσης καπνού, η πρώτη δοκιμή πραγματοποιήθηκε σε συνθήκες απουσίας καπνού/φωτιάς, με το δωμάτιο σε φυσιολογική κατάσταση, χρησιμοποιώντας τον ενοποιημένο κώδικα εκτέλεσης (βλ. Ενότητα 3.3.1). Στην Εικόνα 4.9 παρουσιάζονται οι μετρήσεις TVOC, θερμοκρασίας, CO₂, υγρασίας και φωτεινότητας, καθώς και η αντίστοιχη πρόβλεψη του μοντέλου, όπως καταγράφηκαν στο Grafana για το συγκεκριμένο χρονικό διάστημα.



Σχήμα 4.9: Αποτελέσματα του μοντέλου smoke detection σε κατάσταση ηρεμίας (απουσία καπνού/φωτιάς), με τον ενοποιημένο κώδικα εκτέλεσης.

Όπως φαίνεται στην Εικόνα 4.9, το *panel* <<Smoke Detection>> παρέμεινε σταθερά στην τιμή OK καθ' όλη τη διάρκεια της δοκιμής, δηλαδή το μοντέλο προέβλεψε σωστά την απουσία φωτιάς. Η θερμοκρασία κυμάνθηκε μεταξύ 32.3°C και 32.7°C, ενώ η συγκέντρωση CO₂ κυμάνθηκε μεταξύ 86 και 94 ppm. Η υγρασία κυμάνθηκε μεταξύ 58.5% και 60.5%, ενώ ο δείκτης TVOC παρουσίασε τιμές μεταξύ 234 και 242 ppb, χαρακτηριστικό της θορυβώδους φύσης της εκτίμησης TVOC μέσω του αισθητήρα MQ-135. Η φωτεινότητα κυμάνθηκε μεταξύ 20 και 28 Lux. Στο ίδιο χρονικό διάστημα, το *panel* <<Κατάσταση Χώρου>> παρέμεινε σταθερά στην

τιμή EMPTY, καθώς πλέον, μετά την ενοποίηση του κώδικα εκτέλεσης, και τα δύο μοντέλα λαμβάνουν και επεξεργάζονται το ίδιο πακέτο δεδομένων ταυτόχρονα (βλ. Ενότητα 3.3.1).

4.3.2 Κατάσταση Ενεργοποίησης

Στη δεύτερη δοκιμή, εισήχθη ερέθισμα φωτιάς με στοχευμένη θέρμανση του αισθητήρα θερμοκρασίας/υγρασίας (AM2302) μέσα από ένα πιστολάκι το οποίο έβγαζε θερμό αέρα, ώστε η μετρούμενη θερμοκρασία να ξεπεράσει το όριο των 50°C και να ενεργοποιηθεί ο κανόνας υπερίσχυσης του συστήματος (βλ. Ενότητα 3.3.2). Στην Εικόνα 4.10 παρουσιάζονται τα αντίστοιχα αποτελέσματα.



Σχήμα 4.10: Αποτελέσματα του μοντέλου smoke detection σε κατάσταση ενεργοποίησης (ανίχνευση φωτιάς μέσω υπερθέρμανσης αισθητήρα).

Όπως φαίνεται στην Εικόνα 4.10, η θερμοκρασία παρέμεινε αρχικά σταθερή γύρω στους ~30°C, αντανακλώντας τη φυσιολογική κατάσταση του χώρου πριν την έναρξη της δοκιμής. Με την εφαρμογή της πηγής θερμότητας, η θερμοκρασία ανέβηκε απότομα σε ένα μέγιστο περίπου ~72°C, και στη συνέχεια μειώθηκε σταδιακά, επιστρέφοντας στα αρχικά επίπεδα (~30°C) μέχρι το τέλος του απεικονιζόμενου διαστήματος.

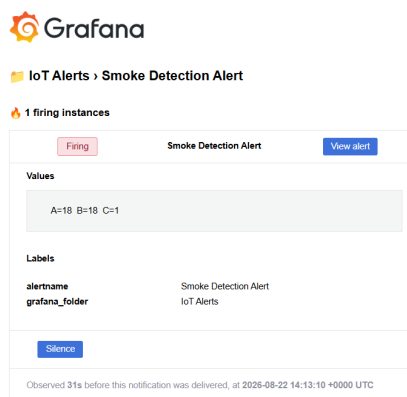
Ανίχνευση: Αντίστοιχα, το *panel <<Smoke Detection>>* παρέμεινε στην τιμή OK πριν και μετά τη θέρμανση, ενώ κατά τη διάρκεια της αιχμής της θερμοκρασίας μεταπήδησε ορθά στην τιμή FIRE, επιστρέφοντας στο OK μόλις η θερμοκρασία υποχώρησε κάτω από το όριο ενεργοποίησης του κανόνα υπερίσχυσης, επιβεβαιώνοντας τη σωστή, αμφίδρομη λειτουργία του συστήματος τόσο κατά την ανίχνευση όσο και κατά την επίλυση του συμβάντος.

TVOC (MQ-135): Είναι σημαντικό να σημειωθεί ότι ο δείκτης δεν ακολούθησε αντίστοιχη πορεία με τη θερμοκρασία· αντιθέτως, από μια αρχική τιμή γύρω στα 250 ppb, έπεσε απότομα σε τιμές κοντά στο μηδέν ακριβώς κατά την έναρξη της θέρμανσης, παρέμεινε χαμηλός και ασταθής καθ' όλη τη διάρκεια του συμβάντος φωτιάς, και επανήλθε σταδιακά προς τα αρχικά επίπεδα μόνο μετά το πέρας της δοκιμής. Αυτή η αντίστροφη, αντί για ανάλογη συμπεριφορά

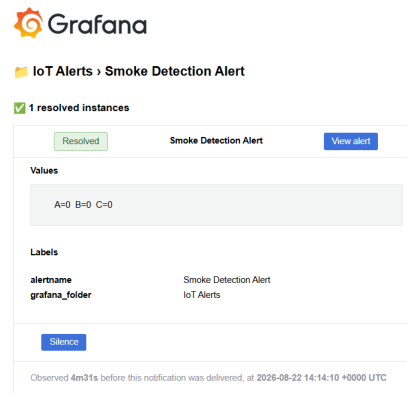
επιβεβαιώνει για ακόμη μία φορά ότι η ανίχνευση φωτιάς στο σύστημα στηρίζεται ουσιαστικά στον κανόνα θερμοκρασίας και όχι στην πρόβλεψη του μοντέλου RandomForestClassifier μέσω του TVOC· πρόκειται για έναν ουσιαστικό περιορισμό του υλοποιημένου μοντέλου ανίχνευσης καπνού.

VOC Index (SGP40): Αντίθετα, ο πραγματικός δείκτης του αισθητήρα παρουσίασε μια σαφώς πιο συνεπή και ερμηνεύσιμη απόκριση: Ξεκινώντας από σταθερή τιμή γύρω στο ~ 100 , αυξήθηκε σταδιακά σε ένα μέγιστο περίπου ~ 175 λίγο μετά την αιχμή της θερμοκρασίας, και στη συνέχεια μειώθηκε ομαλά, επιστρέφοντας στο αρχικό επίπεδο των ~ 100 μέχρι το τέλος του διαστήματος μια καμπύλη ανόδου-πτώσης σαφώς συσχετισμένη με την εξέλιξη του θερμοκικού συμβάντος. Αν και ο δείκτης αυτός δεν τροφοδοτεί το μοντέλο ανίχνευσης καπνού (βλ. Ενότητα 2.2.2), η πιο σταθερή και ερμηνεύσιμη απόκρισή του σε σχέση με την ασταθή ένδειξη του MQ-135 υποδεικνύει ότι θα μπορούσε να αποτελέσει πολλά υποσχόμενο χαρακτηριστικό εισόδου για μελλοντική επανεκπαίδευση του μοντέλου (βλ. Κεφάλαιο 5).

Λοιπές μεταβολές: Η υγρασία παρουσίασε πτωτική τάση από περίπου 40% σε ένα ελάχιστο γύρω στο 20% κατά τη διάρκεια της θέρμανσης, αναμενόμενη συνέπεια της τοπικής αύξησης της θερμοκρασίας του αισθητήρα, με σταδιακή ανάκαμψη προς το 30% μετά το πέρας της δοκιμής. Η φωτεινότητα εμφάνισε δύο παροδικές πτώσεις (στην αρχή και προς το τέλος του διαστήματος), οι οποίες δεν φαίνεται να σχετίζονται άμεσα με το θερμικό συμβάν, πιθανότατα λόγω τυχαίας παρεμπόδισης του αισθητήρα κατά τη διάρκεια της δοκιμής. Η τιμή CO₂ παρουσίασε απότομη πτώση από 100 σε σχεδόν 0 ppm κατά την έναρξη της θέρμανσης, με σταδιακή επαναφορά προς τα αρχικά επίπεδα μέχρι το τέλος του διαστήματος, συμπεριφορά που οφείλεται πιθανότατα στην επίδραση της υψηλής θερμοκρασίας στην αντίσταση του αισθητήρα MQ-135 (βλ. Ενότητα 2.2.1) και όχι σε πραγματική μεταβολή της συγκέντρωσης CO₂ του χώρου. Στο ίδιο διάστημα, το *panel* <<Κατάσταση Χώρου>> παρέμεινε σταθερά στην τιμή EMPTY, χωρίς να επηρεαστεί από τη δοκιμή φωτιάς.



Σχήμα 4.11: Ειδοποίηση (alert) ενεργοποίησης (*firing*) που στάλθηκε μέσω *email* κατά την ανίχνευση φωτιάς.



Σχήμα 4.12: Ειδοποίηση (alert) επίλυσης (*resolved*) που στάλθηκε μέσω *email* μετά την επιστροφή σε κανονική κατάσταση.

Κεφάλαιο 5

Συμπεράσματα

Στην παρούσα διπλωματική εργασία σχεδιάστηκε, υλοποιήθηκε και τεκμηριώθηκε ένα πλήρες σύστημα Cloud-Edge-IoT continuum, τριών επιπέδων: ένα Arduino Uno με τέσσερις αισθητήρες στο επίπεδο IoT, ένα Khadas VIM3 που αναλαμβάνει την εκτέλεση των μοντέλων μηχανικής μάθησης τοπικά στο επίπεδο Edge, και μια εικονική μηχανή Azure με Containerized backend (FastAPI, PostgreSQL, Grafana, Nginx) στο επίπεδο Cloud. Η ορθή λειτουργία ολόκληρης της αλυσίδας, από τη μέτρηση μέχρι την απεικόνιση σε πραγματικό χρόνο, επιβεβαιώθηκε στην πράξη, ικανοποιώντας τον αρχικό στόχο ο οποίος ήταν, η απόδειξη των αρχών του Cloud-Edge-IoT continuum μέσω ενός λειτουργικού συστήματος.

Συμπεράσματα από την υλοποίηση:

- Η ενσωμάτωση ετερογενών τεχνολογιών όπως η ασύρματη δικτύωση, η τοπική εκτέλεση των μοντέλων μηχανικής μάθησης και το περιβάλλον Cloud μέσω containers σε ένα ενιαίο σύστημα, ανέδειξε πρακτικά ζητήματα όπως περιορισμοί πόρων, ανθεκτικότητα αλλά και την ασφαλή διαχείριση ευαίσθητων δεδομένων πρόσβασης, που συχνά παραλείπονται σε θεωρητικές μελέτες του continuum.
- Το μοντέλο occupancy detection εκπαιδεύτηκε με δεδομένα από γραφείο, όπου ο φωτισμός γίνεται με λαμπτήρες φθορισμού. Επειδή ένα σπίτι έχει διαφορετικό φωτισμό, το μοντέλο δεν είναι σίγουρο ότι θα δουλέψει το ίδιο καλά, για τον λόγο αυτό χρειάζεται να εκπαιδεύσουμε ξανά το μοντέλο με νέα δεδομένα τα οποία θα συλλεχθούν από τον χώρο από τον οποίο θέλουμε να πάρουμε μετρήσεις ώστε το τελικό αποτέλεσμα να είναι πιο ακριβές.
- Η τιμή TVOC που τροφοδοτεί το μοντέλο Smoke Detection εξακολουθεί να προκύπτει από εμπειρική εκτίμηση πάνω στην ανάγνωση του MQ-135, και όχι από άμεση μέτρηση, καθώς διατηρήθηκε συμβατή με την κλίμακα τιμών του dataset εκπαίδευσης (βλ. Ενότητα 3.2.2). Παρότι στην τελική υλοποίηση προστέθηκε ο αισθητήρας SGP40, ο οποίος παρέχει άμεση μέτρηση δείκτη ποιότητας αέρα (VOC Index), η ενσωμάτωσή του ως χαρακτηριστικό εισόδου στο μοντέλο θα απαιτούσε νέα συλλογή δεδομένων εκπαίδευσης και επανεκπαίδευση, κάτι που αποτελεί προτεινόμενη κατεύθυνση για μελλοντική εργασία. Η τιμή της ατμοσφαιρικής πίεσης εξακολουθεί να προκύπτει από υπολογισμό βάσει σταθερού υψομέτρου, λόγω απουσίας ειδικού αισθητήρα, και πρέπει να ερμηνεύεται ως προσεγγιστική.

- Η στατική δρομολόγηση (NAT/iptables) ήταν επαρκής για την κλίμακα ενός edge κόμβου, αλλά δεν θα κλιμακωνόταν αυτόματα σε σενάριο πολλαπλών κόμβων, όπου θα ήταν προτιμότερη μια προσέγγιση software-defined networking.
- Η εικονική μηχανή Azure λειτουργεί στο όριο των δυνατοτήτων της για τη φιλοξενία τεσσάρων containers μέσω Docker Compose, γεγονός που προκάλεσε πρακτικά προβλήματα ανθεκτικότητας κατά την ανάπτυξη.

Στο μέλλον, η υλοποίηση θα μπορούσε να διαφοροποιηθεί ως εξής:

- Αντικατάσταση των έμμεσων μετρήσεων με τους κατάλληλους αισθητήρες και re-training των μοντέλων σε δεδομένα από το πραγματικό περιβάλλον ανάπτυξης, ώστε το μοντέλο να προσαρμοστεί καλύτερα στο πραγματικό περιβάλλον λειτουργίας.
- Επέκταση σε πολλαπλούς edge κόμβους, με χρήση Kubernetes αντί για Docker Compose, το οποίο θα προσέφερε αυτόματη κλιμάκωση, δυναμικό εντοπισμό υπηρεσιών και κυρίως αυτόματο εντοπισμό και επανεκκίνηση των containers που έχουν σταματήσει να λειτουργούν σωστά, κάτι που θα απέτρεπε τη χειροκίνητη παρέμβαση που απαιτήθηκε όταν ο πυρήνας του Linux τερμάτισε αναγκαστικά το Grafana λόγω έλλειψης μνήμης.
- Ωστόσο, αυτό έχει κόστος: το ίδιο το Kubernetes να χρειάζεται αρκετούς υπολογιστικούς πόρους για να λειτουργήσει, οι οποίοι δύσκολα θα χωρούσαν σε μια εικονική μηχανή με μόλις 1 GiB RAM. Γι' αυτό παραμένει κατάλληλο μόνο για μια μεγαλύτερη, μελλοντική υλοποίηση με πολλαπλά containers και κόμβους, και όχι για το τρέχον σύστημα.
- Ενσωμάτωση του πραγματικού δείκτη VOC Index του αισθητήρα SGP40 ως χαρακτηριστικό εισόδου στο μοντέλο Smoke Detection, σε αντικατάσταση της εμπειρικής εκτίμησης TVOC από τον MQ-135. Όπως παρατηρήθηκε στην Ενότητα 4.3.2 (βλ. Κεφάλαιο 4), ο δείκτης αυτός παρουσίασε σαφώς πιο σταθερή και ερμηνεύσιμη απόκριση κατά τη διάρκεια δοκιμής φωτιάς σε σχέση με την ασταθή ένδειξη του MQ-135, υποδεικνύοντας ότι θα μπορούσε να βελτιώσει σημαντικά την ακρίβεια του μοντέλου, με την προϋπόθεση συλλογής νέων δεδομένων εκπαίδευσης στη νέα κλίμακα τιμών (0–500).
- Τα μοντέλα τα οποία δημιουργήθηκαν θα μπορούσαν να χρησιμοποιηθούν με smart assistant εφαρμογές για την αυτοματοποίηση ρουτινών. Για παράδειγμα το occupancy μοντέλο θα μπορούσε με βάση τις προβλέψεις να ανοίγει ή να κλείνει τον κλιματισμό σε έναν χώρο, με την προϋπόθεση να διαθέτει αισθητήρες ο χώρος. Αντίστοιχα, το μοντέλο smoke detection θα μπορούσε να ενεργοποιεί αυτόματα μηχανισμούς ασφαλείας, όπως το κλείσιμο της παροχής φυσικού αερίου ή την ενεργοποίηση συστήματος εξαερισμού, μειώνοντας τον χρόνο απόκρισης σε ένα πραγματικό συμβάν φωτιάς.

Βιβλιογραφία

- [1] P. Mell και T. Grance, "The NIST Definition of Cloud Computing", National Institute of Standards και Technology (NIST), Gaithersburg, MD, USA, Special Publication 800-145, Σεπτ. 2011.
- [2] ClipGround, *Cloud Computing Clipart*, <https://clipground.com/images/cloud-computing-clipart-10.jpg>, Πρόσβαση: Ιούνιος 20 2026.
- [3] T. Erl, R. Puttini και M. Zaigham, *Cloud Computing: Concepts, Technology & Architecture*. Upper Saddle River, NJ, USA: Prentice Hall, 2013.
- [4] D. Gurung, S. M. Z. U. Rashid, Z. u. Abdeen και S. Rath, "Cloud Revolution: Tracing the Origins and Rise of Cloud Computing", στο *2026 IEEE 16th Annual Computing and Communication Workshop and Conference (CCWC)*, 2026, σσ. 1100–1106. doi: [10.1109/CCWC67433.2026.11393790](https://doi.org/10.1109/CCWC67433.2026.11393790).
- [5] W.-T. Tsai, X. Bai και Y. Huang, "Software-as-a-service (SaaS): perspectives and challenges", *Science China Information Sciences*, τόμ. 57, αρθμ. 5, σσ. 1–15, 2014.
- [6] H. Nouhas, A. Belangour και M. Nassar, "Cloud and Edge Computing Architectures: A Survey", στο *2023 IEEE 11th Conference on Systems, Process & Control (ICSPC)*, 2023, σσ. 210–215. doi: [10.1109/ICSPC59664.2023.10420123](https://doi.org/10.1109/ICSPC59664.2023.10420123).
- [7] A. Choudhary, P. K. Verma και P. Rai, "Comparative Study of Various Cloud Service Providers: A Review", στο *2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)*, 2022, σσ. 1–8. doi: [10.1109/ICPECTS56089.2022.10047594](https://doi.org/10.1109/ICPECTS56089.2022.10047594).
- [8] S. Gupta, A. Gupta και G. Shankar, "Cloud Computing: Services, Deployment Models and Security Challenges", στο *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*, 2021, σσ. 414–418. doi: [10.1109/ICOSEC51865.2021.9591794](https://doi.org/10.1109/ICOSEC51865.2021.9591794).
- [9] SaM Solutions. "Advantages and Comparison of Cloud Computing Models". Available at: [samsolutions.us](https://samsolutions.us/advantages-and-disadvantages-of-cloud-deployment-models/), επίσκεψη 29 Ιούν. 2026. διεύθν.: <https://samsolutions.us/advantages-and-disadvantages-of-cloud-deployment-models/>.
- [10] S. F. Mousavi Shoshtari, "Cloud Computing Adoption in Iran as a Developing Country: A Tentative Framework Based on Experiences from Iran", Doctoral Dissertation / Research Thesis, Luleå University of Technology, Department of Computer Science, Electrical και Space Engineering, Sweden, 2013.

- [11] W. Shi, J. Cao, Q. Zhang, Y. Li και L. Xu, "Edge Computing: Vision and Challenges", *IEEE Internet of Things Journal*, τόμ. 3, αρθμ. 5, σσ. 637–646, 2016. doi: [10.1109/JIOT.2016.2579198](https://doi.org/10.1109/JIOT.2016.2579198).
- [12] OpenFog Consortium Architecture Working Group, "OpenFog Architecture Overview", OpenFog Consortium, White Paper, 2016, Accessed on December 7, 2016. διεύθν.: https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf.
- [13] S. Douch, M. R. Abid, K. Zine-Dine, D. Bouzidi και D. Benhaddou, "Edge Computing Technology Enablers: A Systematic Lecture Study", *IEEE Access*, τόμ. 10, σσ. 69 264–69 302, 2022. doi: [10.1109/ACCESS.2022.3183634](https://doi.org/10.1109/ACCESS.2022.3183634).
- [14] Deepak, M. K. Upadhyay και M. Alam, "Edge Computing: Architecture, Application, Opportunities, and Challenges", στο *2023 3rd International Conference on Technological Advancements in Computational Sciences (ICTACS)*, 2023, σσ. 695–702. doi: [10.1109/ICTACS59847.2023.10390171](https://doi.org/10.1109/ICTACS59847.2023.10390171).
- [15] A. u. R. Khan, M. Othman, S. A. Madani και S. U. Khan, "A Survey of Mobile Cloud Computing Application Models", *IEEE Communications Surveys & Tutorials*, τόμ. 16, αρθμ. 1, σσ. 393–413, 2014. doi: [10.1109/SURV.2013.062613.00160](https://doi.org/10.1109/SURV.2013.062613.00160).
- [16] M. Babar, M. S. Khan, F. Ali, M. Imran και M. Shoaib, "Cloudlet Computing: Recent Advances, Taxonomy, and Challenges", *IEEE Access*, τόμ. 9, σσ. 29 609–29 622, 2021. doi: [10.1109/ACCESS.2021.3059072](https://doi.org/10.1109/ACCESS.2021.3059072).
- [17] S. Dong κ.ά., "Task offloading strategies for mobile edge computing: A survey", *Computer Networks*, τόμ. 254, σ. 110 791, 2024, issn: 1389-1286. doi: [10.1016/j.comnet.2024.110791](https://doi.org/10.1016/j.comnet.2024.110791).
- [18] Y. Mao, C. You, J. Zhang, K. Huang και K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective", *IEEE Communications Surveys & Tutorials*, τόμ. 19, αρθμ. 4, σσ. 2322–2358, 2017. doi: [10.1109/COMST.2017.2745201](https://doi.org/10.1109/COMST.2017.2745201).
- [19] M. Patel κ.ά., "Mobile Edge Computing -- Introductory Technical White Paper", European Telecommunications Standards Institute (ETSI), αδημοσίευτη ερευνητική εργασία White Paper No. 11, Σεπτ. 2014. διεύθν.: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge-computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf.
- [20] F. C. Andriulo, M. Fiore, M. Mongiello, E. Traversa και V. Zizzo, "Edge Computing and Cloud Computing for Internet of Things: A Review", *Informatics*, τόμ. 11, αρθμ. 4, σ. 71, 2024. doi: [10.3390/informatics11040071](https://doi.org/10.3390/informatics11040071).
- [21] Waveshare Electronics. "MQ-135 Gas Sensor Board", επίσκεψη 20 Μάι. 2024. διεύθν.: https://www.waveshare.com/wiki/MQ-135_Gas_Sensor.
- [22] ShillehTek. "GY-302 BH1750 Pre-Soldered Light Intensity Module", επίσκεψη 20 Μάι. 2024. διεύθν.: <https://shillehtek.com/products/shillehtek-gy-302-bh1750-pre-soldered-light-intensity-module>.

-
- [23] Grobotronics. "ASAIR AM2302 (DHT22) Humidity and Temperature Sensor", επίσκεψη 20 Μάι. 2024. διεύθν.: <https://grobotronics.com/asair-am2302-dht22-humidity-and-temperature-sensor.html>.
- [24] DFRobot, *Gravity: SGP40 Air Quality Sensor SKU: SEN0394*, https://wiki.dfrobot.com/Gravity_SGP40_Air_Quality_Sensor_SKU_SEN0394, Accessed: 2026, DFRobot.
- [25] Devobox. "Arduino Uno Rev3", επίσκεψη 20 Μάι. 2024. διεύθν.: <https://www.devobox.com/el/official-boards/1-arduino-uno-rev3-8058333490090.html>.
- [26] TinyOS Shop. "XBee S6B WiFi Module - RPSMA", επίσκεψη 25 Ιούλ. 2026. διεύθν.: https://www.tinyosshop.com/index.php?route=product/product&product_id=810.
- [27] Khadas. "VIM3 Single Board Computer", επίσκεψη 25 Ιούλ. 2026. διεύθν.: <https://www.khadas.com/product-page/vim3>.
- [28] Saumit GP. "Occupancy Detection Dataset", επίσκεψη 26 Ιούλ. 2026. διεύθν.: <https://www.kaggle.com/datasets/saumitgp/occupancy-detection-dataset>.
- [29] Deepcontractor. "Smoke Detection Dataset", επίσκεψη 26 Ιούλ. 2026. διεύθν.: <https://www.kaggle.com/datasets/deepcontractor/smoke-detection-dataset>.
- [30] scikit-learn developers. "Choosing the Right Estimator", επίσκεψη 4 Αύγ. 2026. διεύθν.: https://scikit-learn.org/stable/machine_learning_map.html.
- [31] scikit-learn developers. "sklearn.svm.LinearSVC", επίσκεψη 4 Αύγ. 2026. διεύθν.: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html>.
- [32] Wikipedia contributors. "Pearson Correlation Coefficient", επίσκεψη 4 Αύγ. 2026. διεύθν.: https://en.wikipedia.org/wiki/Pearson_correlation_coefficient.
- [33] scikit-learn developers. "sklearn.preprocessing.StandardScaler", επίσκεψη 4 Αύγ. 2026. διεύθν.: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>.
- [34] scikit-learn developers. "sklearn.feature_selection.RFECV", επίσκεψη 4 Αύγ. 2026. διεύθν.: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html.
- [35] scikit-learn developers. "Cross-validation: Evaluating Estimator Performance", επίσκεψη 4 Αύγ. 2026. διεύθν.: https://scikit-learn.org/stable/modules/cross_validation.html.
- [36] scikit-learn developers, *Model persistence --- scikit-learn documentation*, 2026. επίσκεψη 8 Αύγ. 2026. διεύθν.: https://scikit-learn.org/stable/model_persistence.html.

Παράρτημα Α

Παρατήματα Κώδικα

Α'.1 Κώδικας για τους Αισθητήρες

Α'.1.1 MQ-135

Α'.1.1.1 Βαθμονόμηση

```
#include "MQ135.h"

MQ135 gasSensor = MQ135(A0);

void setup() {
  Serial.begin(9600);
  Serial.println("Υπολογισμός RZero...");
}

void loop() {
  float rzero = gasSensor.getRZero();
  Serial.print("Η τιμή του RZero είναι: ");
  Serial.println(rzero);
  delay(1000);
}
```

Κώδικας Α.1: Sketch βαθμονόμησης για εύρεση της τιμής αναφοράς RZero του MQ-135.

Α'.1.1.2 Υπολογισμός συγκέντρωσης CO2

```
float myRZero = 30.0;

float resistance_sum = 0;
for (int i = 0; i < 10; i++) {
  resistance_sum += gasSensor.getResistance();
  delay(50);
}
```

```
float avg_resistance = resistance_sum / 10.0;
```

```
float co2_ppm = 116.6020682 * pow((avg_resistance / myRZero),  
-2.769034857);
```

Κώδικας Α.2: Υπολογισμός συγκέντρωσης CO2 με χρήση της βαθμονομημένης τιμής RZero.

Α'.1.2 Πλήρες Sketch Καταγραφής Μετρήσεων

Το πλήρες sketch που εκτελείται στο Arduino Uno, συνδυάζοντας και τους τέσσερις αισθητήρες (MQ-135, BH1750, AM2302,SGP40), παρουσιάζεται στον Κώδικα Α'.3. Η επεξήγηση της λειτουργίας του παρατίθεται αναλυτικά στην Ενότητα 2.2 του Κεφαλαίου 2.

```
#include <Wire.h>
#include "DHT.h"
#include "MQ135.h"
#include <BH1750.h>
#include <DFRobot_SGP40.h>

#define DHTPIN 2
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

#define CO2PIN A0
MQ135 gasSensor = MQ135(CO2PIN);

float myRZero = 30.0;

const float SEA_LEVEL_PRESSURE_HPA = 1013.25;
const float ALTITUDE_M = 620.0;

BH1750 lightMeter;
DFRobot_SGP40 sgp40;

float estimateTVOC(int mq135Raw) {
    float tvoc;
    if (mq135Raw < 400) {
        tvoc = mq135Raw;
    } else if (mq135Raw < 650) {
        tvoc = 400 + (mq135Raw - 400) * 3.2;
    } else {
        tvoc = 1200 + (mq135Raw - 650) * (58800.0 / 373.0);
    }
    return tvoc;
}

float calculatePressure() {
```

```

    return SEA_LEVEL_PRESSURE_HPA * pow(1.0 - (2.25577e-5 * ALTITUDE_M),
        5.25588);
}

void setup() {
    Serial.begin(9600);
    Wire.begin();
    dht.begin();

    if (!lightMeter.begin()) {
        Serial.println("SFALMA: O BH1750 den vrethike!");
    }

    while (!sgp40.begin(10000)) {
        Serial.println("SFALMA: O SGP40 den vrethike, prospathw ksana...");
        delay(1000);
    }

    delay(10000);
    Serial.println("Thermokrasia (C), Ygrasia (%), CO2 (ppm), Fos (Lux), TVOC (ppb)
        ), VOC_Index_SGP40, Piesi (hPa)");
}

void loop() {
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    if (isnan(h) || isnan(t)) {
        delay(2000);
        return;
    }

    float resistance_sum = 0;
    for (int i = 0; i < 10; i++) {
        resistance_sum += gasSensor.getResistance();
        delay(50);
    }
    float avg_resistance = resistance_sum / 10.0;
    float co2_ppm = 116.6020682 * pow((avg_resistance / myRZero),
        -2.769034857);

    int mq135_raw = analogRead(CO2PIN);
    float tvoc_ppb = estimateTVOC(mq135_raw);

    sgp40.setRhT(h, t);
    uint16_t voc_index = sgp40.getVocIndex();

```

```

float lux = lightMeter.readLightLevel();
float pressure_hpa = calculatePressure();

Serial.print(t);
Serial.print(",");
Serial.print(h);
Serial.print(",");
Serial.print(co2_ppm);
Serial.print(",");
Serial.print(lux);
Serial.print(",");
Serial.print(tvoc_ppb);
Serial.print(",");
Serial.print(voc_index);
Serial.print(",");
Serial.println(pressure_hpa);

delay(2000);
}

```

Κώδικας A.3: Πλήρες sketch καταγραφής μετρήσεων από τους τέσσερις αισθητήρες, με εκτίμηση TVOC, μέτρηση VOC Index και υπολογισμό πίεσης.

A'.2 Machine Learnig

A'.2.1 Ocupancy ML

```

[1]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import GenericUnivariateSelect, _
    chi2
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.neighbors import NearestNeighbors
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn import metrics
from sklearn.model_selection import cross_val_predict

```

```
from sklearn.metrics import confusion_matrix, \
    ConfusionMatrixDisplay
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

import matplotlib.pyplot as plt
%matplotlib inline
%matplotlib widget
```

```
[2]: # Load the training dataset
train_data = pd.read_csv('OccupancyData/DataTraining.csv')
# Convert date columne from str to datetime
train_data['date'] = pd.to_datetime(train_data['date'])
# Drop the No colum
train_data = train_data.drop(["Unnamed: 0"], axis=1)
train_data
```

```
[2]:
```

		date	Temperature	Humidity	Light	
	CO2	\				
0	2015-04-02	17:51:00	23.18	27.2720	426.0	721.250000
1	2015-04-02	17:51:00	23.15	27.2675	429.5	714.000000
2	2015-04-02	17:53:00	23.15	27.2450	426.0	713.500000
3	2015-04-02	17:54:00	23.15	27.2000	426.0	708.250000
4	2015-04-02	17:55:00	23.10	27.2000	426.0	704.500000
...		...	...	...	...	...
8138	2015-10-02	09:29:00	21.05	36.0975	433.0	787.250000
8139	2015-10-02	09:29:00	21.05	35.9950	433.0	789.500000
8140	2015-10-02	09:30:00	21.10	36.0950	433.0	798.500000
8141	2015-10-02	09:32:00	21.10	36.2600	433.0	820.333333
8142	2015-10-02	09:33:00	21.10	36.2000	447.0	821.000000

	HumidityRatio	Occupancy
0	0.004793	1
1	0.004783	1
2	0.004779	1
3	0.004772	1
4	0.004757	1
...	...	...
8138	0.005579	1
8139	0.005563	1
8140	0.005596	1
8141	0.005621	1
8142	0.005612	1

[8143 rows x 7 columns]

```
[3]: # Do the same for the test data.
test_data = pd.read_csv('OccupancyData/DataTest.csv')
test_data['date'] = pd.to_datetime(test_data['date'],
    ↪format='mixed', dayfirst=True)
test_data = test_data.drop(["Unnamed: 0"], axis=1)
```

A'.2.1.1 See if the test and traing data are the same

```
[4]: test_data.describe()
```

```
[4]:
```

	date	Temperature	Humidity
count	9752	9752.000000	9752.000000
mean	2015-02-15 00:03:11.333059840	21.001768	29.891910
min	2015-02-11 14:48:00	19.500000	21.865000
25%	2015-02-13 07:25:00	20.290000	26.642083
50%	2015-02-15 00:03:30	20.790000	30.200000
75%	2015-02-16 16:40:15	21.533333	32.700000
max	2015-02-18 09:19:00	24.390000	39.500000
std	NaN	1.020693	3.952844

	CO2	HumidityRatio	Occupancy
count	9752.000000	9752.000000	9752.000000
mean	753.224832	0.004589	0.210111
min	484.666667	0.003275	0.000000
25%	542.312500	0.004196	0.000000
50%	639.000000	0.004593	0.000000
75%	831.125000	0.004998	0.000000
max	2076.500000	0.005769	1.000000
std	297.096114	0.000531	0.407408

```
[5]: train_data.describe()
```

```
[5]:
```

	date	Temperature	Humidity
count	8143	8143.000000	8143.000000
mean	2015-07-05 05:02:39.980351232	20.619084	25.731507
min	2015-04-02 17:51:00	19.000000	16.745000
25%	2015-06-02 03:46:30	19.700000	20.200000
50%	2015-07-02 13:41:00	20.390000	26.222500
75%	2015-08-02 23:37:30	21.390000	30.533333
max	2015-10-02 09:33:00	23.180000	39.117500
std	NaN	1.016916	5.531211

	CO2	HumidityRatio	Occupancy
count	8143.000000	8143.000000	8143.000000
mean	606.546243	0.003863	0.212330
min	412.750000	0.002674	0.000000
25%	439.000000	0.003078	0.000000
50%	453.500000	0.003801	0.000000
75%	638.833333	0.004352	0.000000
max	2028.500000	0.006476	1.000000
std	314.320877	0.000852	0.408982

A'.2.1.2 Coeficientes

Take the statistical coefficients to see if they can reveal something from the data. Specifically, we will use the [Pearson Coefficient](#) via the [pandas Coefficient](#) function.

```
[6]: train_data[["Temperature", "Humidity", "Light",
                  "CO2", "HumidityRatio", "Occupancy"]].corr('pearson')
```

```
[6]:
```

	Temperature	Humidity	Light	CO2
HumidityRatio	-0.141759	0.649942	0.559894	0.151762
Temperature	1.000000	-0.141759	0.037828	0.439023
Humidity	-0.141759	1.000000	0.037828	0.439023
Light	0.649942	0.037828	1.000000	0.151762
CO2	0.559894	0.439023	0.151762	1.000000

Light	0.649942	0.037828	1.000000	0.664022	0.
↪230420					
CO2	0.559894	0.439023	0.664022	1.000000	0.
↪626556					
HumidityRatio	0.151762	0.955198	0.230420	0.626556	1.
↪000000					
Occupancy	0.538220	0.132964	0.907352	0.712235	0.
↪300282					

	Occupancy
Temperature	0.538220
Humidity	0.132964
Light	0.907352
CO2	0.712235
HumidityRatio	0.300282
Occupancy	1.000000

An explanation of **kaendall** that measure the rank

```
[7]: train_data[["Temperature", "Humidity", "Light",
                 "CO2", "HumidityRatio", "Occupancy"]].corr('kendall')
```

```
[7]:
```

	Temperature	Humidity	Light	CO2	↪
↪HumidityRatio \					
Temperature	1.000000	-0.110486	0.415524	0.454057	0.
↪085116					
Humidity	-0.110486	1.000000	0.000452	0.169497	0.
↪808516					
Light	0.415524	0.000452	1.000000	0.337398	0.
↪130087					
CO2	0.454057	0.169497	0.337398	1.000000	0.
↪271553					
HumidityRatio	0.085116	0.808516	0.130087	0.271553	1.
↪000000					
Occupancy	0.439836	0.105824	0.734405	0.537307	0.
↪209033					

	Occupancy
Temperature	0.439836
Humidity	0.105824
Light	0.734405
CO2	0.537307
HumidityRatio	0.209033
Occupancy	1.000000

```
[8]: train_data[["Temperature", "Humidity", "Light",
                "CO2", "HumidityRatio", "Occupancy"]].corr('spearman')
```

```
[8]:
```

	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
Temperature	1.000000	-0.193389	0.565237	0.636906	0.104765	0.532830
Humidity	-0.193389	1.000000	0.007065	0.223519	0.940327	0.129235
Light	0.565237	0.007065	1.000000	0.473103	0.169057	0.804645
CO2	0.636906	0.223519	0.473103	1.000000	0.378357	0.656651
HumidityRatio	0.104765	0.940327	0.169057	0.378357	1.000000	0.255836
Occupancy	0.532830	0.129235	0.804645	0.656651	0.255836	1.000000

A'.2.1.3 Split the dataset to the features and the labes

```
[9]: X = train_data[["Temperature", "Humidity", "Light",
                    "CO2", "HumidityRatio"]]
     y = train_data[["Occupancy"]]
```

```
[10]: X.shape
```

```
[10]: (8143, 5)
```

A'.2.1.4 Preprosering the data

The goal is the values to be **scaled**. This can be done on the features by removing the mean and scaling to unit variance

```
[11]: scaler = StandardScaler().fit(X)

     feature_means = dict(zip(X.columns, scaler.mean_))
     feature_standarDev = dict(zip(X.columns, scaler.scale_))
```

```

print("--- Feature Means ---")
for feature, mean in feature_means.items():
    print(f"{feature:15} : {mean:>10.4f}")
print("--- Feature Standar Deviation ---")
for feature, std in feature_standarDev.items():
    print(f"{feature:15} : {std:>10.4f}")

```

```

--- Feature Means ---
Temperature      :    20.6191
Humidity         :    25.7315
Light           :   119.5194
CO2              :   606.5462
HumidityRatio    :     0.0039
--- Feature Standar Deviation ---
Temperature      :     1.0169
Humidity         :     5.5309
Light           :   194.7438
CO2              :   314.3016
HumidityRatio    :     0.0009

```

```

[12]: # transform the data
X_scaled = scaler.transform(X)
# Create a dictionary using zip
feature_means = dict(zip(X.columns, scaler.mean_))
feature_standarDev = dict(zip(X.columns, X_scaled.std(axis=0)))
print("--- Feature Means ---")
for feature, mean in feature_means.items():
    print(f"{feature:15} : {mean:>10.4f}")
print("--- Feature Standar Deviation ---")
for feature, std in feature_standarDev.items():
    print(f"{feature:15} : {std:>10.4f}")

```

```

--- Feature Means ---
Temperature      :    20.6191
Humidity         :    25.7315
Light           :   119.5194
CO2              :   606.5462
HumidityRatio    :     0.0039
--- Feature Standar Deviation ---
Temperature      :     1.0000
Humidity         :     1.0000
Light           :     1.0000
CO2              :     1.0000
HumidityRatio    :     1.0000

```

Α'.2.1.5 Feature elimination

Try to find if we need all features. We can use Recursive Feature Elimination with cross-validation to select features ([RFECV](#)).

```
[13]: min_features_to_select = 1  # Minimum number of features to_
      ↪consider
      clf = LogisticRegression() # The algorithm for features_
      ↪eliminations
      cv = StratifiedKFold(5) # Take a cross validation for

      rfecv = RFECV(
          estimator=clf,
          step=1,
          cv=cv,
          scoring="accuracy",
          min_features_to_select=min_features_to_select,
          n_jobs=2,
      )

      # Use the scaled features
      # https://numpy.org/doc/stable/reference/generated/numpy.ravel.
      ↪html
      rfecv.fit(X_scaled, y.values.ravel())

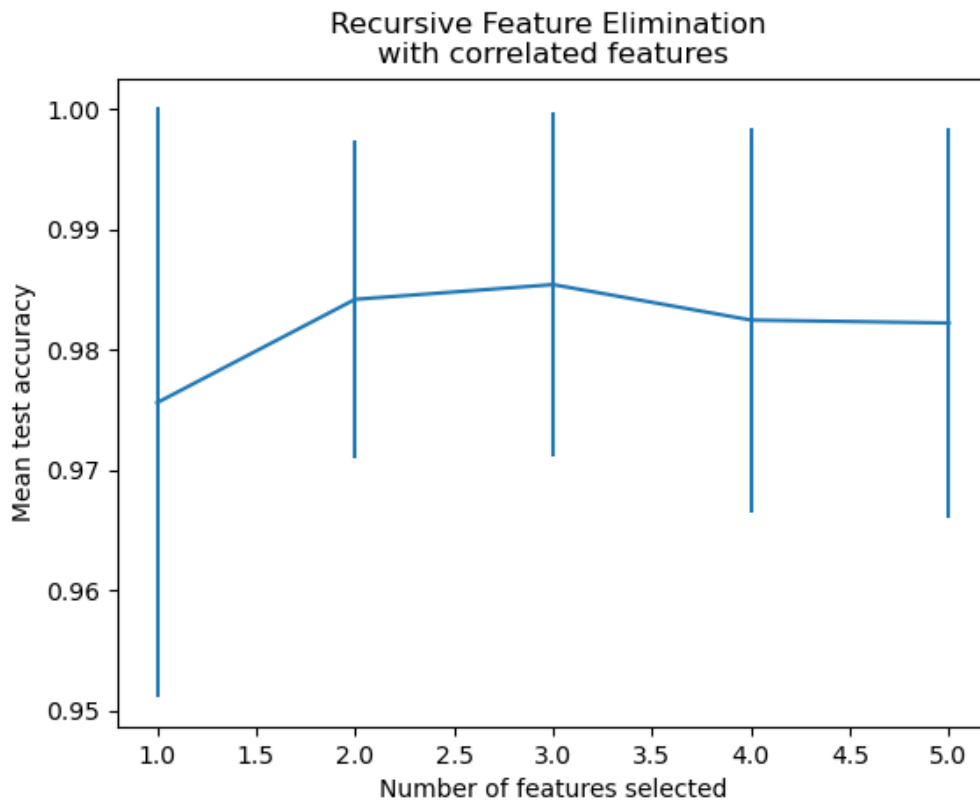
      print(f"Optimal number of features: {rfecv.n_features_}")
```

Optimal number of features: 3

See visually in plot who the number feature perform ([link](#))

```
[14]: data = {
      key: value
      for key, value in rfecv.cv_results_.items()
      if key in ["n_features", "mean_test_score", "std_test_score"]
      }
      cv_results = pd.DataFrame(data)
      plt.figure()
      plt.xlabel("Number of features selected")
      plt.ylabel("Mean test accuracy")
      plt.errorbar(
          x=cv_results["n_features"],
          y=cv_results["mean_test_score"],
          yerr=cv_results["std_test_score"],
      )
```

```
plt.title("Recursive Feature Elimination \nwith correlated_
↪features")
plt.show()
```



Take the optimal feature and use it to train a model

```
[15]: # The 'support_' attribute is a boolean array (True for_
↪selected, False for discarded)
selected_features = X.columns[rfevc.support_]

print(f"Optimal number of features: {rfevc.n_features_}")
print(f"Selected features: {list(selected_features)}")

# Take the best featues that have been scaled and store in a_
↪new vatiabile
X_best = X_scaled[:, rfevc.support_]
X_best
```

Optimal number of features: 3
 Selected features: ['Temperature', 'Light', 'CO2']

```
[15]: array([[2.51847007, 1.57376283, 0.36494808],
            [2.48896731, 1.59173515, 0.34188106],
            [2.48896731, 1.57376283, 0.34029023],
            ...,
            [0.47294534, 1.60970748, 0.61073113],
            [0.47294534, 1.60970748, 0.68019732],
            [0.47294534, 1.68159679, 0.68231843]])
```

A'.2.1.6 Model Training

What model is suitable for me? There is no single correct answer. Trial and experimentation can give us these answers. We can use this [map](#) as a blueprint. According to our data, we are in the classification box. So we test a Linear SVC and a KNeighbors Classifier and see how they perform using the F1-score. To be more precise, we will use cross-validation (CV).

Run a [LinearSVC](#) model for our training data. Use the F1-score to evaluate the performance of the model.

```
[16]: linearSVC = LinearSVC(random_state=0, tol=1e-5)
      scores = cross_val_score(linearSVC, X_best, y.values.ravel(),
      ↪cv=5, scoring='f1_macro')
      scores
```

```
[16]: array([0.9291184 , 0.98375872, 0.99542746, 0.99272077, 0.
      ↪89624136])
```

Do the same for [KNeighborsClassifier](#)

```
[17]: neigh = KNeighborsClassifier(n_neighbors=5)
      scores2 = cross_val_score(neigh, X_best, y.values.ravel(),
      ↪cv=5, scoring='f1_macro')
      scores2
```

```
[17]: array([0.93006032, 0.95725483, 0.66142675, 0.98124908, 0.
      ↪99088202])
```

A'.2.1.7 Compare the results for the two algorithms

We can see from the initial results that LinearSVC performs better in terms of the F1-score. We can use the [sklearn. Confusion matrix](#) from the cross-validation to see clearer results.

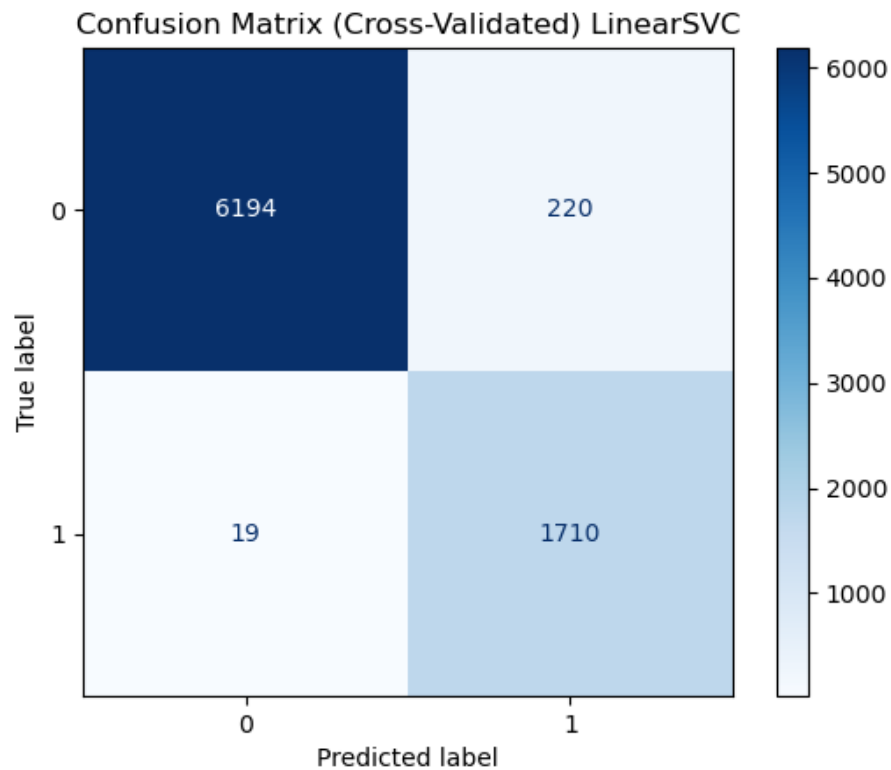
```
[18]: y_pred = cross_val_predict(linearSVC, X_best, y.values.ravel(),
      ↪cv=5)

      cm = confusion_matrix(y.values.ravel(), y_pred)
```

```

disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Cross-Validated) LinearSVC")
plt.show()

```



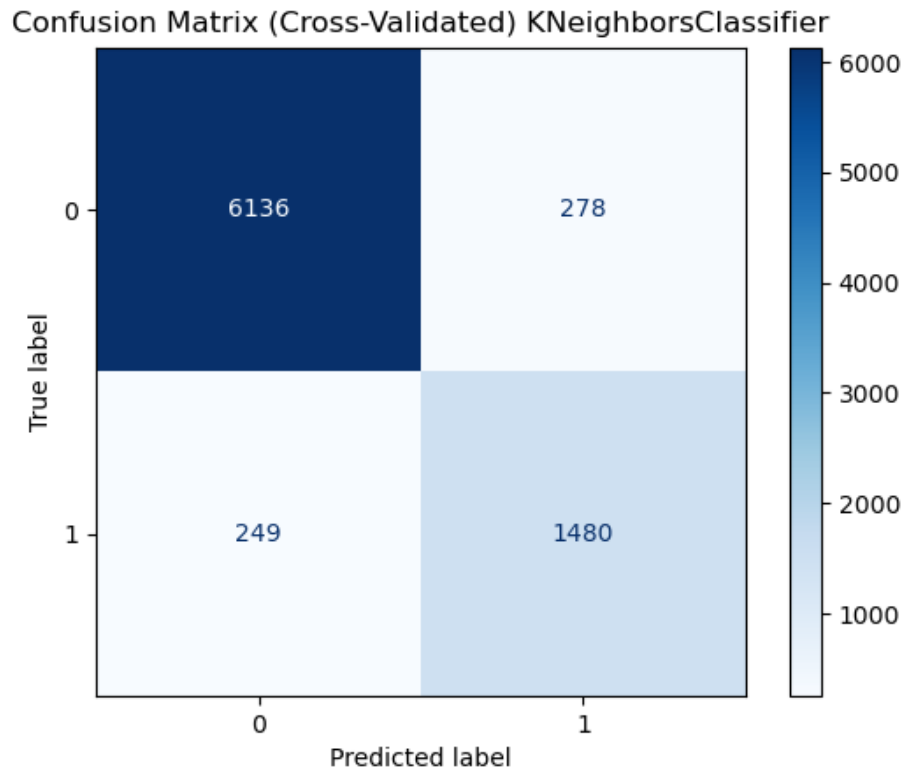
```

[19]: y_pred = cross_val_predict(neigh, X_best, y.values.ravel(),
    ↪cv=5)

cm = confusion_matrix(y.values.ravel(), y_pred)

# 3. Visualize it
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Cross-Validated) ↪
    ↪KNeighborsClassifier")
plt.show()

```



We can see clearly that LinearSVC performs better. So now we start the process of fine-tuning the model.

A'.2.1.8 Tunig params

We can use grid search to find the best parameters for our model. We provide the solver with parameters and it tries to find the best combination. Each parameter we add increases the computation exponentially (C: 9, loss: 2, multi_class: 2, dual: 2, max_iter: 7). This results in a total of 504 combinations with 5 cross-validations, scoring for the F1-score. To avoid data leakage to the model, we use a Pipeline.

```
[20]: # Take the raw data from the best features
X_selected = X[['Temperature', 'Light', 'CO2']]

# Create a pipe line
pipeline = make_pipeline(
    StandardScaler(),
    LinearSVC(random_state=0, tol=1e-5, dual="auto")
)

[21]: param_grid = {'linearsvc__C': [0.01, 0.1, 1, 10, 100, 200, 400, 500, 600],
                    'linearsvc__loss': ['hinge', 'squared_hinge'],
```

```

        'linearsvc__dual': ['auto', True],
        'linearsvc__multi_class' : ['ovr', 'crammer_singer'],
        'linearsvc__max_iter' : [500, 800 ,1000 , 2500,
↪, 5000, 10000, 75000]}

clf = GridSearchCV(pipeline, param_grid, cv=5,
↪scoring='f1_macro', n_jobs = -1)
clf.fit(X_selected, y.values.ravel())
clf.best_params_

```

```

[21]: {'linearsvc__C': 400,
       'linearsvc__dual': True,
       'linearsvc__loss': 'squared_hinge',
       'linearsvc__max_iter': 800,
       'linearsvc__multi_class': 'ovr'}

```

We use the parameters from the solver to train the final model. Next, we perform cross-validation, plot the confusion matrix, and compare it with the previous confusion matrix.

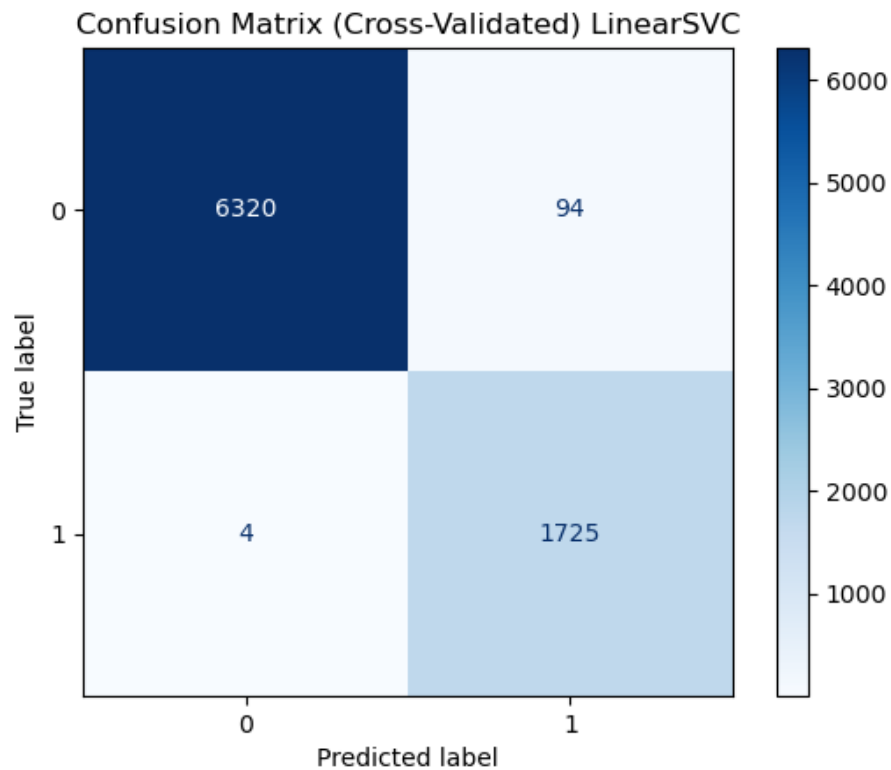
```

[22]: y_pred = cross_val_predict(clf.best_estimator_, X_selected, y.
↪values.ravel(), cv=5)

cm = confusion_matrix(y.values.ravel(), y_pred)

# 3. Visualize it
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Cross-Validated) LinearSVC")
plt.show()

```



View the cross-validation for F1-score from the new train of the linearSVC model with the tuned parameters.

```
[23]: scores = cross_val_score(clf.best_estimator_, X_best, y.values.  
    ↪ ravel(), cv=5, scoring='f1_macro')  
scores
```

```
[23]: array([0.94396922, 0.98375872, 0.99542746, 0.99634519, 0.  
    ↪ 99450627])
```

A'.2.1.9 Use the best params and train the final model

Use a [classification report](#) to see how the model performs on the new data.

```
[25]: # --- Scale and fit the ---  
# Separate all 5 features  
X_test_all = test_data[["Temperature", "Humidity", "Light", "  
    ↪ "CO2", "HumidityRatio"]]  
  
X_test_final = X_test_all[['Temperature', 'Light', 'CO2']]  
y_test_final = test_data['Occupancy']
```

```

final_predictions = clf.predict(X_test_final)
# Make predictions for thee new data!

# Comprere the results with actual values from the new data
print("=== FINAL EXAM RESULTS ===")
print(classification_report(y_test_final, final_predictions))

```

```

=== FINAL EXAM RESULTS ===

```

	precision	recall	f1-score	support
0	1.00	0.99	0.99	7703
1	0.96	1.00	0.98	2049
accuracy			0.99	9752
macro avg	0.98	0.99	0.99	9752
weighted avg	0.99	0.99	0.99	9752

See the confusion matrix for the new data

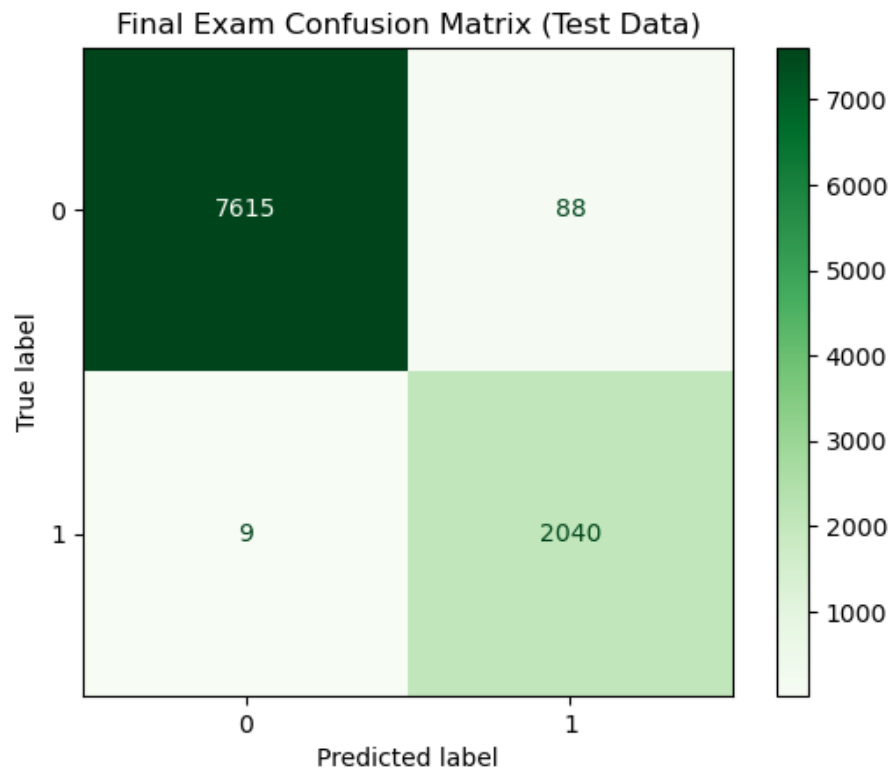
```

[26]: # 1. Calculate the raw numbers for the matrix
cm_final = confusion_matrix(y_test_final, final_predictions)

# 2. Create the visual display
disp_final = ConfusionMatrixDisplay(confusion_matrix=cm_final)

disp_final.plot(cmap=plt.cm.Greens)
plt.title("Final Exam Confusion Matrix (Test Data)")
plt.show()

```



```
[27]: # Extract the coefficients (weights) from the model
best_pipe = clf.best_estimator_

model_step = best_pipe.named_steps['linearsvc']

# Get the coefficients
weights = model_step.coef_[0]
features = ["Temperature", "Light", "CO2"]

# Sort them to see the most important one at the top!
importance_df = pd.DataFrame({'Feature': features, 'Weight': weights})
print(importance_df.sort_values(by='Weight', ascending=False))
```

	Feature	Weight
1	Light	1.100437
2	CO2	0.573252
0	Temperature	-0.049573

```
[28]: # Save model
import joblib
```

```
joblib.dump(clf.best_estimator_, 'occupancy_mode_pipeline.pkl')
```

```
[28]: ['occupancy_mode_pipeline.pkl']
```

```
[32]: # 1. Access the trained LinearSVC model inside the pipeline
```

```
my_model = best_pipe.named_steps['linearsvc']

# 2. Extract the weights and the intercept
weights = my_model.coef_[0]
intercept = my_model.intercept_[0]
features = X.columns

# 3. Print the baseline "flip" point
print(f"Baseline Intercept: {intercept:.4f}\n")

print(features)
```

```
Baseline Intercept: -1.3117
```

```
Index(['Temperature', 'Humidity', 'Light', 'CO2',
      ↪ 'HumidityRatio'],
      dtype='object')
```

A'.2.2 Smoke ML

```
[1]: import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import GenericUnivariateSelect,
    ↪ chi2
from sklearn.feature_selection import SelectPercentile
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.svm import LinearSVC
from sklearn.neighbors import NearestNeighbors
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn import metrics
from sklearn.model_selection import cross_val_predict
```

```
from sklearn.metrics import confusion_matrix, \
    ConfusionMatrixDisplay
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier

import matplotlib.pyplot as plt
%matplotlib inline
%matplotlib widget
```

```
[2]: # Load the dataset
data = pd.read_csv('SmokeDetection/smoke_detection_iot.csv')
data
```

```
[2]:      Unnamed: 0      UTC  Temperature[C]  Humidity[%] \
      ↳TVOC [ppb] \
0      0  1654733331      20.000      57.36      0
1      1  1654733332      20.015      56.67      0
2      2  1654733333      20.029      55.96      0
3      3  1654733334      20.044      55.28      0
4      4  1654733335      20.059      54.69      0
...
62625  62625  1655130047      18.438      15.79      ↳
      ↳625
62626  62626  1655130048      18.653      15.87      ↳
      ↳612
62627  62627  1655130049      18.867      15.84      ↳
      ↳627
62628  62628  1655130050      19.083      16.04      ↳
      ↳638
62629  62629  1655130051      19.299      16.52      ↳
      ↳643

      eCO2[ppm]  Raw H2  Raw Ethanol  Pressure[hPa]  PM1.0  PM2.
      ↳5  NC0.5 \
0      400  12306      18520      939.735  0.00  0.
      ↳00  0.00
1      400  12345      18651      939.744  0.00  0.
      ↳00  0.00
2      400  12374      18764      939.738  0.00  0.
      ↳00  0.00
3      400  12390      18849      939.736  0.00  0.
      ↳00  0.00
```

```

4          400    12403          18921          939.744    0.00    0.
↳00    0.00
...
62625          400    13723          20569          936.670    0.63    0.
↳65    4.32
62626          400    13731          20588          936.678    0.61    0.
↳63    4.18
62627          400    13725          20582          936.687    0.57    0.
↳60    3.95
62628          400    13712          20566          936.680    0.57    0.
↳59    3.92
62629          400    13696          20543          936.676    0.57    0.
↳59    3.90

```

```

          NC1.0  NC2.5    CNT  Fire Alarm
0          0.000  0.000     0           0
1          0.000  0.000     1           0
2          0.000  0.000     2           0
3          0.000  0.000     3           0
4          0.000  0.000     4           0
...
62625    0.673    0.015  5739           0
62626    0.652    0.015  5740           0
62627    0.617    0.014  5741           0
62628    0.611    0.014  5742           0
62629    0.607    0.014  5743           0

```

[62630 rows x 16 columns]

A'.2.2.1 Drop features

Drop all the features that I cannot measure with my sensors

```

[3]: data = data.drop(["Unnamed: 0", "UTC", "eCO2 [ppm]", "Raw H2", "Raw_
↳Ethanol", "PM1.0", "PM2.5", "NC0.5", "NC1.0", "NC2.5", "CNT"], _
↳axis=1)
data.describe()

```

```

[3]:      Temperature[C]  Humidity[%]  TVOC[ppb]  Pressure[hPa]_
↳  Fire Alarm
count      62630.000000  62630.000000  62630.000000  62630.
↳000000  62630.000000
mean         15.970424      48.539499  1942.057528  938.627649_
↳      0.714626

```

```
std      14.359576      8.865367      7811.589055      1.331344
↪      0.451596
min      -22.010000      10.740000      0.000000      930.852000
↪      0.000000
25%      10.994250      47.530000      130.000000      938.700000
↪      0.000000
50%      20.130000      50.150000      981.000000      938.816000
↪      1.000000
75%      25.409500      53.240000      1189.000000      939.418000
↪      1.000000
max      59.930000      75.200000      60000.000000      939.861000
↪      1.000000
```

```
[4]: data.corr('pearson')
```

```
[4]:      Temperature[C]  Humidity[%]  TVOC[ppb]  ↪
↪Pressure[hPa]  \
Temperature[C]      1.000000      -0.243986      0.082442      -0.
↪251203
Humidity[%]      -0.243986      1.000000      -0.488878      0.
↪694614
TVOC[ppb]      0.082442      -0.488878      1.000000      -0.
↪220004
Pressure[hPa]      -0.251203      0.694614      -0.220004      1.
↪000000
Fire Alarm      -0.163902      0.399846      -0.214743      0.
↪249797

      Fire Alarm
Temperature[C]      -0.163902
Humidity[%]      0.399846
TVOC[ppb]      -0.214743
Pressure[hPa]      0.249797
Fire Alarm      1.000000
```

A'.2.2.2 Split the dataset

Separate the dataset into features (X) and labels (y). The initial data is then shuffled and split into training (80%) and test (20%) sets. We use a stratified split to ensure that the original proportion of classes is maintained across both sets.

```
[5]: X = data[["Temperature[C]", "Humidity[%]", "TVOC[ppb]",
           "Pressure[hPa]"]]
y = data[["Fire Alarm"]]
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, shuffle=True, stratify=y,
    random_state=10)
```

A'.2.2.3 Run algorithms

Run a dry run with LinearSVC to see how it performs. We make a pipeline with a scaler and the algorithm. The idea is that in cross-validation, we scale each time the data that we use for training. The [RobustScaler](#) removes the median and scales the data according to the quantile range, and is perfect if our data has outliers. More info [here](#). Start with LinearSVC.

```
[6]: # 2. Create the leak-proof pipeline with the RobustScaler
pipeline = make_pipeline(
    RobustScaler(),
    LinearSVC(random_state=42, dual="auto", max_iter=5000)
)

# 3. Train on the training set
pipeline.fit(X_train, y_train.values.ravel())

# 4. Score it on the test set
print("Test Accuracy:", pipeline.score(X_test, y_test))

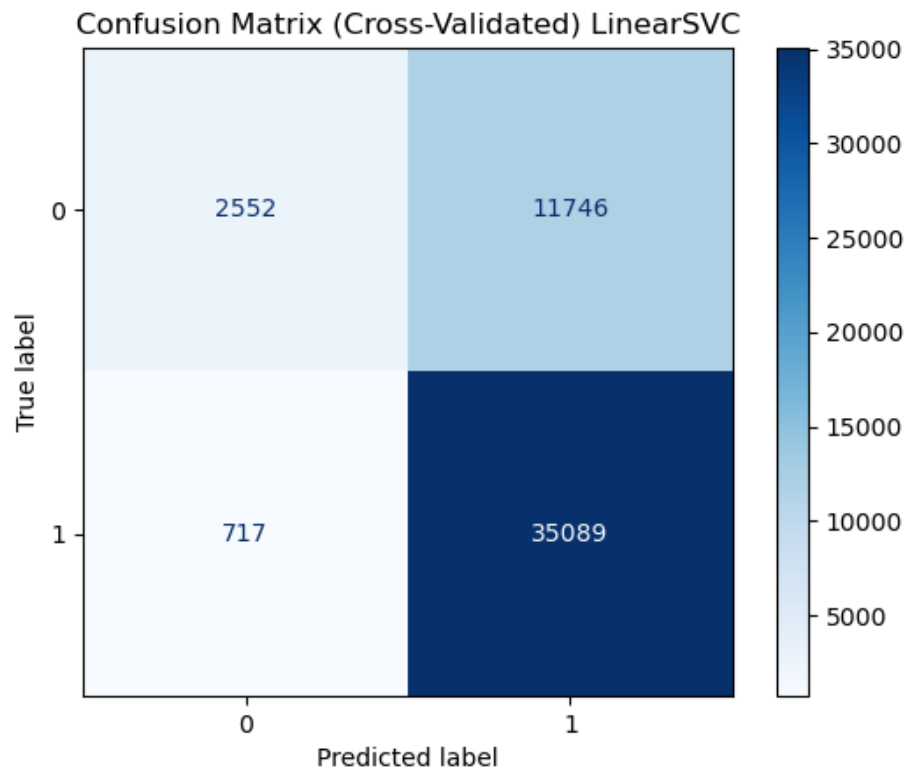
y_pred = cross_val_predict(pipeline, X_train, y_train.values.
    ravel(), cv=5)

# 2. Confusion matrix
cm = confusion_matrix(y_train.values.ravel(), y_pred)

# 3. Draw the matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Cross-Validated) LinearSVC")
plt.show()

scores = cross_val_score(pipeline, X_train, y_train.values.
    ravel(), cv=5, scoring='f1_macro')
print("LinearSVC \nMean score:", scores.mean(), " Scores:",
    scores)
```

Test Accuracy: 0.7547501197509181



LinearSVC

Mean score: 0.5698496391930515 Scores: [0.57325108 0.57533858 0.
 ↪55957441
 0.57079762 0.5702865]

The results from the cross-validation are not ideal due to too many false positives and false negatives. Next, we try a KNeighborsClassifier, which takes the 5 closest neighbors into account.

```
[7]: pipeline2 = make_pipeline(
    RobustScaler(),
    KNeighborsClassifier(n_neighbors=5)
)

# 3. Train on the training set
pipeline2.fit(X_train, y_train.values.ravel())

# 4. Score it on the test set
print("Test Accuracy:", pipeline2.score(X_test, y_test))

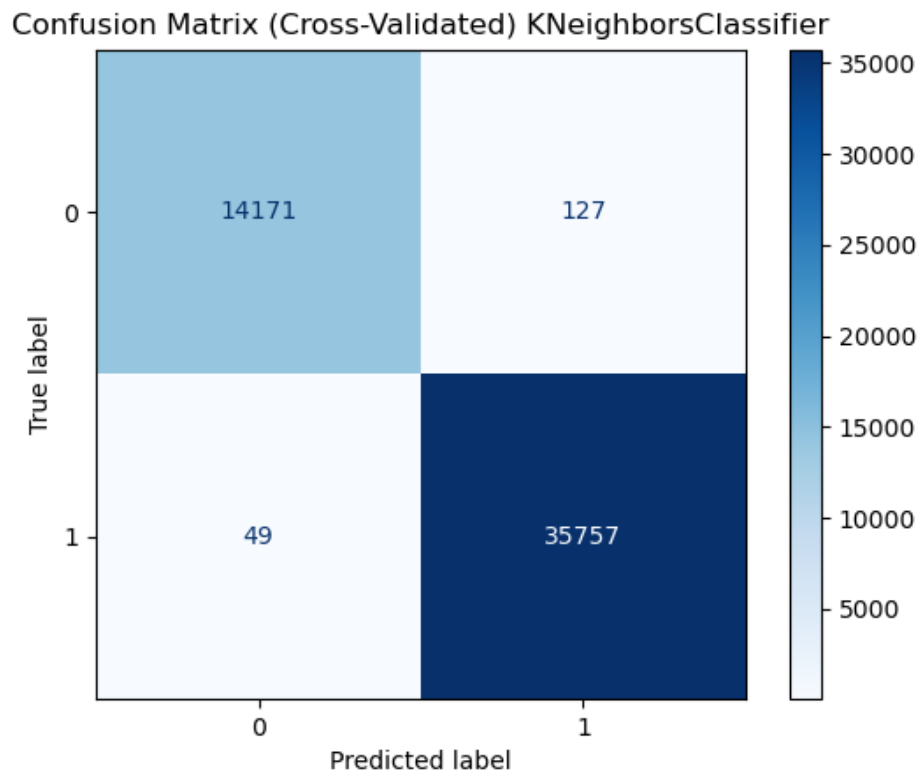
y_pred = cross_val_predict(pipeline2, X_train, y_train.values.
    ↪ravel(), cv=5)
```

```
# 2. Confusion matrix
cm = confusion_matrix(y_train.values.ravel(), y_pred)

# 3. Draw the matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Cross-Validated) \n
↳KNeighborsClassifier")
plt.show()

scores = cross_val_score(pipeline2, X_train, y_train.values.
↳ravel(), cv=5, scoring='f1_macro')
print("KNeighborsClassifier \nMean score:", scores.mean(), "\n
↳Scores:", scores)
```

Test Accuracy: 0.9956091330033531



KNeighborsClassifier
Mean score: 0.995686765664815 Scores: [0.99509672 0.99632176 0.99632487
0.99534484 0.99534563]

KNeighborsClassifier has better results from LinearSVC. Try the [RandomForestClassifier](#).

```
[8]: pipeline3 = make_pipeline(
    RobustScaler(),
    RandomForestClassifier()
)

# 3. Train it strictly on the training set
pipeline3.fit(X_train, y_train.values.ravel())

# 4. Score it on the test set
print("Test Accuracy:", pipeline3.score(X_test, y_test))

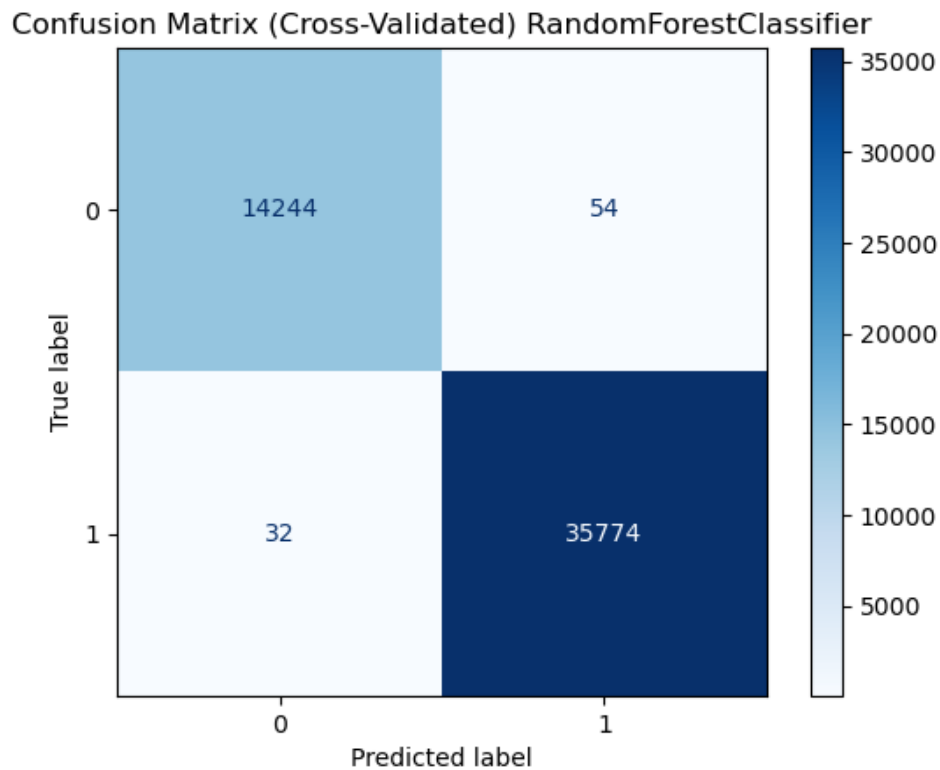
y_pred = cross_val_predict(pipeline3, X_train, y_train.values.
    ↪ravel(), cv=5)

# 2. Now the confusion matrix has the exact 0s and 1s it needs_
    ↪to do the math
cm = confusion_matrix(y_train.values.ravel(), y_pred)

# 3. Draw the matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Cross-Validated)_
    ↪RandomForestClassifier")
plt.show()

scores = cross_val_score(pipeline3, X_train, y_train.values.
    ↪ravel(), cv=5, scoring='f1_macro')
print("RandomForestClassifier \nMean score:", scores.mean(), "_
    ↪Scores:", scores)
```

Test Accuracy: 0.9976848155835861



RandomForestClassifier

Mean score: 0.997821440772455 Scores: [0.99779724 0.99816418 0.99828675
0.99779678 0.99706226]

A'.2.2.4 Tuning Params

Try to find better params for KNeighborsClassifier using grid search.

```
[9]: # Create a pipe line
pipelineGridKn = make_pipeline(
    RobustScaler(),
    KNeighborsClassifier()
)

param_grid = {'kneighborsclassifier__n_neighbors': [3, 5, 7, 11, 21],
              'kneighborsclassifier__weights': ['uniform', 'distance'],
              'kneighborsclassifier__leaf_size' : [15, 20, 30, 40, 50],
```

```
        'kneighborsclassifier__p' : [1, 2]
    }
```

```
clf_Kn = GridSearchCV(pipelineGridKn, param_grid, cv=5,
    ↪scoring='f1_macro', n_jobs= -1)
clf_Kn.fit(X_train, y_train.values.ravel())
clf_Kn.best_params_
```

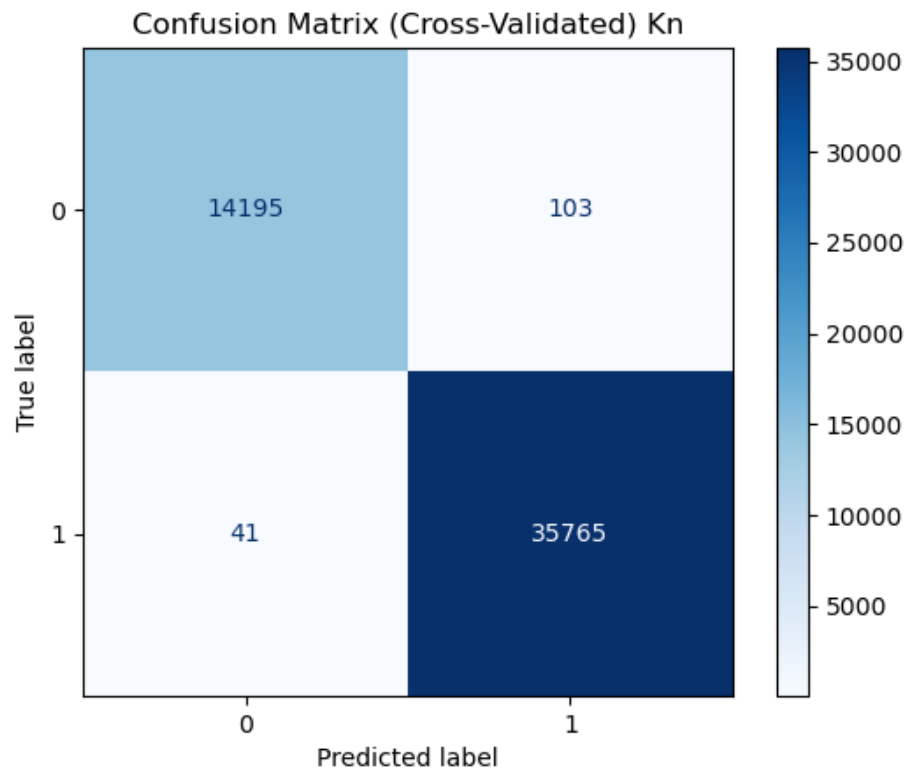
```
[9]: {'kneighborsclassifier__leaf_size': 15,
      'kneighborsclassifier__n_neighbors': 5,
      'kneighborsclassifier__p': 1,
      'kneighborsclassifier__weights': 'distance'}
```

```
[11]: y_pred = cross_val_predict(clf_Kn.best_estimator_, X_train,
    ↪y_train.values.ravel(), cv=5)

cm = confusion_matrix(y_train.values.ravel(), y_pred)

# 3. Visualize it
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Cross-Validated) Kn")
plt.show()

scores = cross_val_score(clf_Kn.best_estimator_, X_train,
    ↪y_train.values.ravel(), cv=5, scoring='f1_macro')
print("KNeighborsClassifier Grid Params \nMean score:", scores.
    ↪mean(), " Scores:", scores)
```



KNeighborsClassifier Grid Params

Mean score: 0.9964721135246796 Scores: [0.9957119 0.99693707 0.99706052
0.99595778 0.9966933]

```
[12]: final_predictions = clf_Kn.predict(X_test)
      # Make predictions for thee new data!

      # Comprere the results with actual values from the new data
      print("=== FINAL RESULTS for Test data ===")
      print(classification_report(y_test, final_predictions))
```

=== FINAL RESULTS for Test data ===

	precision	recall	f1-score	support
0	1.00	0.99	0.99	3575
1	1.00	1.00	1.00	8951
accuracy			1.00	12526
macro avg	1.00	0.99	1.00	12526
weighted avg	1.00	1.00	1.00	12526

Use the grid search for RandomForestClassifier now. Try to find better params for our problem.

```
[13]: pipelineGridRandomForest = make_pipeline(
    RobustScaler(),
    RandomForestClassifier()
)

param_grid = {'randomforestclassifier__n_estimators': [50, 100, ↵
↵200],
              'randomforestclassifier__criterion': ['gini', ↵
↵'entropy'],
              'randomforestclassifier__max_depth': [10, 20, 30],
              'randomforestclassifier__min_samples_split': [2, ↵
↵5, 10],
              'randomforestclassifier__min_samples_leaf': [1, 2, ↵
↵4],
              }

clf_rm = GridSearchCV(pipelineGridRandomForest, param_grid, ↵
↵cv=5, scoring='f1_macro', n_jobs= -1)
clf_rm.fit(X_train, y_train.values.ravel())
clf_rm.best_params_
```

```
[13]: {'randomforestclassifier__criterion': 'entropy',
       'randomforestclassifier__max_depth': 20,
       'randomforestclassifier__min_samples_leaf': 1,
       'randomforestclassifier__min_samples_split': 2,
       'randomforestclassifier__n_estimators': 200}
```

```
[14]: final_predictions = clf_rm.predict(X_test)

# Comprere the results with actual values from the new data
print("=== FINAL RESULTS for Test data ===")
print(classification_report(y_test, final_predictions))
```

```
=== FINAL RESULTS for Test data ===
```

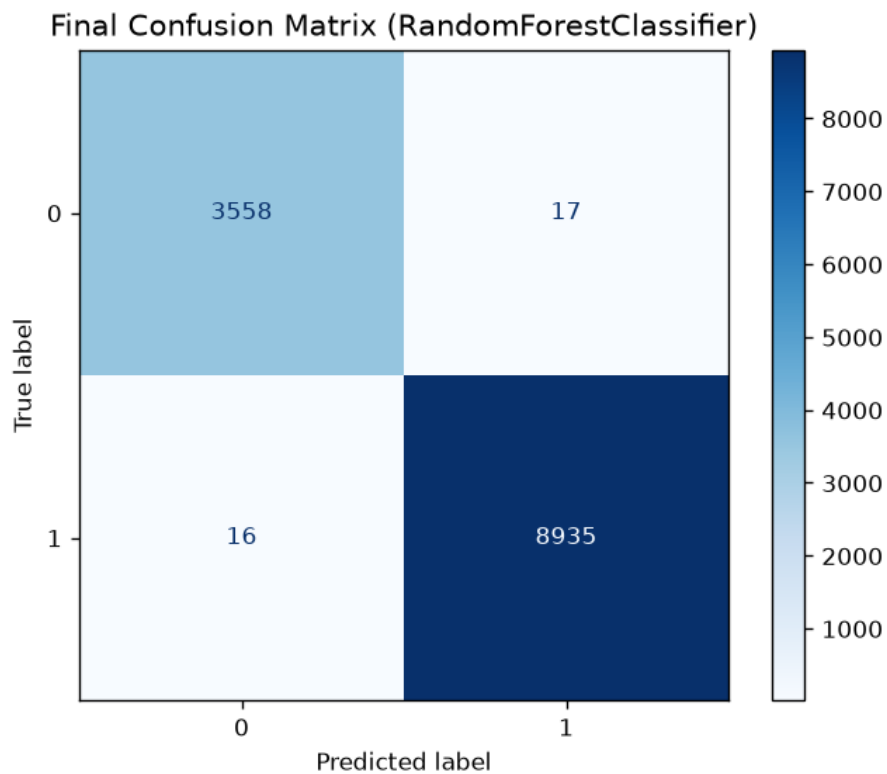
	precision	recall	f1-score	support
0	1.00	1.00	1.00	3575
1	1.00	1.00	1.00	8951
accuracy			1.00	12526
macro avg	1.00	1.00	1.00	12526
weighted avg	1.00	1.00	1.00	12526

```
[15]: # 2. Predict the answers for the unseen test data
final_preds = clf_rm.predict(X_test)

# 3. Grade the Exam: Compare the predictions to the actual
# y_test answers
cm = confusion_matrix(y_test.values.ravel(), final_preds)

# 4. Visualize the Final Confusion Matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.title("Final Confusion Matrix (RandomForestClassifier)")
plt.show()

# 5. Print the final classification report
print("=== FINAL EXAM RESULTS ===")
print(classification_report(y_test, final_preds))
```



```
=== FINAL EXAM RESULTS ===
              precision    recall  f1-score   support

0               1.00      1.00      1.00     3575
```

1	1.00	1.00	1.00	8951
accuracy			1.00	12526
macro avg	1.00	1.00	1.00	12526
weighted avg	1.00	1.00	1.00	12526

```
[17]: # Save model
import joblib

joblib.dump(clf_rm.best_estimator_,
            'smok_detection_model_RandomForest.pkl')
```

```
[17]: ['smok_detection_model_RandomForest.pkl']
```

A'.3 Data Collection

```
[1]: import socket
import joblib
import datetime
import csv
import os
import requests
import pandas as pd
import urllib3
urllib3.disable_warnings(urllib3.exceptions.
                        InsecureRequestWarning)
```

```
[2]: # -----
#
# 1) Φόρτωση ΚΑΙ ΤΩΝ ΔΥΟ μοντέλων στην ίδια διεργασία
# -----
#
occ_model = joblib.load('occupancy_mode_pipeline.pkl')
smoke_model = joblib.load('smok_detection_model_RandomForest.
                       .pkl')
print("[*] Και τα δύο μοντέλα φορτώθηκαν επιτυχώς!")
```

[*] Και τα δύο μοντέλα φορτώθηκαν επιτυχώς!

```
[3]: # -----
#
# 2) ΕΝΑ ΚΑΙ ΜΟΝΑΔΙΚΟ UDP socket (λύνει τη σύγκρουση port)
```

```
# -----
-
LISTEN_IP = "10.42.0.1"
LISTEN_PORT = 39361

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind((LISTEN_IP, LISTEN_PORT))
print(f"[*] UDP Server σε λειτουργία στο {LISTEN_IP}:
↳{LISTEN_PORT}")
```

[*] UDP Server σε λειτουργία στο 10.42.0.1:39361

```
[4]: # -----
-
# 3) Cloud API
# -----
-
API_URL = "https://iot-ele518003.duckdns.org/api"
API_TOKEN = None

def get_token():
    global API_TOKEN
    try:
        response = requests.post(f"{API_URL}/token", data={
            "username": "vim3", "password": "secret"
        }, verify=False, timeout=5)
        if response.status_code == 200:
            API_TOKEN = response.json().get("access_token")
            print("[API] Token ελήφθη επιτυχώς!")
        else:
            print(f"[API Error] get_token status {response.
↳status_code}")
    except Exception as e:
        print(f"[API Error] Αποτυχία σύνδεσης στο /token: {e}")

def send_to_api(endpoint, data):
    global API_TOKEN
    if API_TOKEN is None:
        get_token()
    if API_TOKEN is None:
        return
    headers = {"Authorization": f"Bearer {API_TOKEN}"}
    try:
```

```

        response = requests.post(f"{API_URL}/{endpoint}",
    ↪ json=data,
                                headers=headers, verify=False,
    ↪ timeout=5)
        if response.status_code == 200:
            print(f"[API] Δεδομένα εστάλησαν επιτυχώς στο /
    ↪ {endpoint}!")
        else:
            print(f"[API Error] /{endpoint} status {response.
    ↪ status_code}: {response.text[:100]}")
        except Exception as e:
            print(f"[API Error] Αποτυχία αποστολής στο /{endpoint}:
    ↪ {e}")

```

```

[5]: # -----
-
# 4) Αρχεία καταγραφής (ξεχωριστά, ένα ανά μοντέλο)

occ_log = "occupancy_log.csv"
smoke_log = "smoke_log.csv"

if not (os.path.isfile(occ_log) and os.path.getsize(occ_log) >
    ↪ 0):
    with open(occ_log, mode='a', newline='') as f:
        csv.writer(f).writerow(
            ["Timestamp", "Temperature", "Humidity", "CO2",
    ↪ "Lux", "TVOC", "Prediction"])

if not (os.path.isfile(smoke_log) and os.path.
    ↪ getsize(smoke_log) > 0):
    with open(smoke_log, mode='a', newline='') as f:
        csv.writer(f).writerow(
            ["Timestamp", "Temperature", "Humidity", "CO2",
    ↪ "Lux", "TVOC", "Pressure", "Prediction", "Confidence"])

```

```

[6]: # -----
-
# 5) Μετρητές
# -----
-

occ_counts = {"EMPTY": 0, "OCCUPANCY": 0}
smoke_counts = {"OK": 0, "FIRE": 0}
previous_occ = None
previous_fire = None

```

```
print("[*] Dispatcher σε αναμονή δεδομένων από Arduino...")
print("-" * 60)
```

```
[*] Dispatcher σε αναμονή δεδομένων από Arduino...
```

```
[ ]: # -----
#
# 6) Κύριος βρόχος
# -----
#
try:
    while True:
        data, addr = sock.recvfrom(1024)
        sender_ip = addr[0]
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d_
↪%H:%M:%S")

        try:
            message = data.decode('utf-8').strip()
            values = list(map(float, message.split(',')))

            if len(values) != 6:
                print(f"[!] Σφάλμα: {len(values)} τιμές αντί για
↪6. Μήνυμα: {message}")
                continue

            temp, humidity, co2_ppm, lux, tvoc, pressure = values

            print(f"[{timestamp}] Από IP: {sender_ip}")
            print(f"    | Θερμοκρασία : {temp} °C")
            print(f"    | Υγρασία      : {humidity} %")
            print(f"    | CO2         : {co2_ppm} ppm")
            print(f"    | Φωτεινότητα : {lux} Lux")
            print(f"    | TVOC        : {round(tvoc, 2)} ppb")
            print(f"    | Πίεση         : {pressure} hPa")

            # ----- OCCUPANCY MODEL -----
            occ_input = pd.DataFrame([[temp, lux, co2_ppm]],
                                     columns=['Temperature', '
↪Light', 'CO2'])
            occ_pred = occ_model.predict(occ_input)[0]
            occ_label = "OCCUPANCY" if occ_pred == 1 else "EMPTY"
```

```

occ_counts[occ_label] += 1

if previous_occ is not None and occ_pred !=_
↪previous_occ:
    old = "OCCUPANCY" if previous_occ == 1 else_
↪"EMPTY"
    print(f"    [OCCUPANCY] ΑΛΛΑΓΗ ΚΑΤΑΣΤΑΣΗΣ: {old}_
↪-> {occ_label}")
    previous_occ = occ_pred

print(f"    >>> [Occupancy] Πρόβλεψη: {occ_label}")

with open(occ_log, 'a', newline='') as f:
    csv.writer(f).writerow(
        [timestamp, temp, humidity, co2_ppm, lux,_
↪round(tvoc, 2), occ_label])

send_to_api("occupancy", {
    "temperature": temp, "humidity": humidity, "co2":
↪ co2_ppm,
    "lux": lux, "voc": round(tvoc, 2), "prediction":_
↪occ_label
})

# ----- SMOKE MODEL -----
smoke_input = pd.DataFrame(
    [[temp, humidity, tvoc, pressure]],
    columns=['Temperature[C]', 'Humidity[%]',_
↪'TVOC[ppb]', 'Pressure[hPa]']
)
smoke_pred = smoke_model.predict(smoke_input)[0]
proba = smoke_model.predict_proba(smoke_input)[0]

if smoke_pred == 1 and proba[1] >= 0.90:
    confidence = round(proba[1] * 100, 1)
    smoke_label = "FIRE"
else:
    confidence = round(proba[0] * 100, 1)
    smoke_label = "OK"

if temp > 50.0:
    smoke_label = "FIRE"
    confidence = 100.0

```

```

        smoke_counts[smoke_label if smoke_label in
↪smoke_counts else "FIRE"] += 1

        if previous_fire is not None and smoke_pred !=
↪previous_fire:
            print(f"    [SMOKE] ΑΛΛΑΓΗ ΚΑΤΑΣΤΑΣΗΣ")
            previous_fire = smoke_pred

            print(f"    >>> [Smoke] Πρόβλεψη: {smoke_label}
↪({confidence}%")
            print(f"    Στατιστικά -- Occupancy: {occ_counts} |
↪Smoke: {smoke_counts}")
            print("-" * 60)

            with open(smoke_log, 'a', newline='') as f:
                csv.writer(f).writerow(
                    [timestamp, temp, humidity, co2_ppm, lux,
↪round(tvoc, 2),
                        pressure, smoke_label, confidence])

            send_to_api("smoke", {
                "temperature": temp, "humidity": humidity, "co2":
↪co2_ppm, "lux": lux,
                "voc": round(tvoc, 2),
                "tvoc": round(tvoc, 2),
                "pressure": pressure,
                "prediction": smoke_label, "confidence":
↪confidence
            })

        except ValueError as e:
            print(f"[] Σφάλμα parsing (μη αριθμητικά δεδομένα):
↪{e}")

        except IndexError as e:
            print(f"[] Σφάλμα index: {e}")

    except KeyboardInterrupt:
        print("\n[*] Χειροκίνητος τερματισμός dispatcher.")
    finally:
        sock.close()
        print("[*] Το socket έκλεισε.")

```

[2026-08-23 11:47:50] Από IP: 10.42.0.35

```

├─ Θερμοκρασία : 32.6 °C
├─ Υγρασία      : 59.3 %
├─ CO2          : 102.24 ppm
├─ Φωτεινότητα : 11.67 Lux
├─ TVOC         : 244.0 ppb
└─ Πίεση       : 940.95 hPa
>>> [Occupancy] Πρόβλεψη: EMPTY
[API] Token ελήφθη επιτυχώς!
[API] Δεδομένα εστάλησαν επιτυχώς στο /occupancy!
>>> [Smoke] Πρόβλεψη: OK (20.0%)
Στατιστικά -- Occupancy: {'EMPTY': 1, 'OCCUPANCY': 0} | Smoke:
↪ {'OK': 1,
'FIRE': 0}
-----
[API] Δεδομένα εστάλησαν επιτυχώς στο /smoke!
[2026-08-23 11:47:52] Από IP: 10.42.0.35
├─ Θερμοκρασία : 32.6 °C
├─ Υγρασία      : 59.4 %
├─ CO2          : 103.21 ppm
├─ Φωτεινότητα : 10.83 Lux
├─ TVOC         : 246.0 ppb
└─ Πίεση       : 940.95 hPa
>>> [Occupancy] Πρόβλεψη: EMPTY
[API] Δεδομένα εστάλησαν επιτυχώς στο /occupancy!
>>> [Smoke] Πρόβλεψη: OK (20.0%)
Στατιστικά -- Occupancy: {'EMPTY': 2, 'OCCUPANCY': 0} | Smoke:
↪ {'OK': 2,
'FIRE': 0}

```

A'.4 Data Presentation

```

services:
  postgres:
    image: postgres:15
    container_name: iot-postgres
    environment:
      POSTGRES_DB: ${POSTGRES_DB}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data

```

```
ports:
  - "5432:5432"
restart: always

api:
  build: ./api
  container_name: iot-api
  depends_on:
    - postgres
  restart: always

grafana:
  image: grafana/grafana:latest
  container_name: iot-grafana
  ports:
    - "3000:3000"
  volumes:
    - grafana_data:/var/lib/grafana
  environment:
    - GF_SMTP_ENABLED=true
    - GF_SMTP_HOST=${SMTP_HOST}
    - GF_SMTP_USER=${SMTP_USER}
    - GF_SMTP_PASSWORD=${SMTP_PASSWORD}
    - GF_SMTP_FROM_ADDRESS=${SMTP_USER}
    - GF_SMTP_FROM_NAME=IoT Alerts
    - GF_SERVER_ROOT_URL=${GRAFANA_ROOT_URL}
  restart: always

nginx:
  image: nginx:latest
  container_name: iot-nginx
  dns:
    - 8.8.8.8
    - 8.8.4.4
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./nginx/nginx.conf:/etc/nginx/nginx.conf
    - ./nginx/iot-selfsigned.crt:/etc/nginx/certs/iot-selfsigned.crt
    - ./nginx/iot-selfsigned.key:/etc/nginx/certs/iot-selfsigned.key
    - certbot_data:/etc/letsencrypt
    - certbot_www:/var/www/certbot
  depends_on:
    - api
    - grafana
  restart: always
```

```
volumes:  
  postgres_data:  
  grafana_data:  
  certbot_data:  
  certbot_www:
```

Κώδικας Α.4: Πλήρες αρχείο ορισμού docker-compose.yml (containers PostgreSQL, API, Grafana, Nginx).