

Relazione del progetto di Multimodal Interaction

Anno accademico 2022/2023

Jessica Frabotta – 1758527

Alessio Ferrone – 1751859

Contatti:

[frabotta.1758527@studenti.uniroma1.it](mailto:frabotta.1758527@studenti.uniroma1.it)

[ferrone.1751859@studenti.uniroma1.it](mailto:ferrone.1751859@studenti.uniroma1.it)

# Indice

Introduzione.....	3
Idee dietro il progetto.....	7
Architettura del sistema.....	9
Scelta dell'hardware.....	10
Raccolta dei dati ed addestramento del modello.....	14
Progettazione del sistema.....	24
Implementazione del sistema.....	35
Real-time performance e test di usabilità.....	64
Conclusioni e sviluppi futuri.....	65
Bibliografia.....	66

## Introduzione

L'olfatto è un senso spesso sottovalutato ma potente, che influenza fortemente anche gli altri sensi e le nostre emozioni. Infatti, esso è responsabile di circa l'80% di ciò che gustiamo[1]. Senza l'olfatto, il nostro senso del gusto è limitato solo a cinque sensazioni distinte, tutti gli altri sapori che sperimentiamo provengono dall'olfatto. Questo è il motivo per cui, quando il nostro naso è ostruito, ad esempio da un raffreddore, la maggior parte degli alimenti sembra insipida o priva di sapore.



Figura 1: Snoopy, 10 settembre 1958

Il senso dell'olfatto è strettamente legato alla memoria[2], probabilmente più di quanto lo siano gli altri sensi. Coloro che hanno una piena funzione olfattiva potrebbero essere in grado di pensare a odori che evocano particolari ricordi; ad esempio, il profumo di un frutteto in fiore può suscitare ricordi di un picnic dell'infanzia. Questo può accadere spesso in modo spontaneo, con un odore che agisce come un trigger nel richiamare un evento o un'esperienza da tempo dimenticata.

L'olfatto è anche un senso altamente "emotivo". L'industria dei profumi è costruita su questa connessione, con i profumieri che sviluppano fragranze che cercano di comunicare una vasta gamma di emozioni e sentimenti, dall'amore al potere, dalla vitalità al relax.

Su un piano più personale, l'olfatto è estremamente importante quando si tratta di attrazione tra due persone. Delle ricerche dimostrano che il nostro odore corporeo, prodotto dai geni che compongono il nostro sistema immunitario, può aiutarci a scegliere il nostro partner in modo subconscio.

Ma come possiamo "insegnare" ad una macchina a percepire degli odori?

L'electronic nose, o "naso elettronico," è un dispositivo progettato per rilevare odori e composti chimici volatili nell'aria. Si basa su sensori chimici e software di riconoscimento dei pattern per emulare il senso dell'olfatto umano.

Nel corso degli anni, i nasi elettronici hanno visto significativi progressi grazie a innovazioni scientifiche e tecnologiche.

Uno dei principali avanzamenti riguarda la progettazione di sensori sempre più sofisticati.

Questi sensori sono in grado di rilevare una vasta gamma di composti chimici con maggiore precisione e sensibilità.

Un altro importante sviluppo è stato l'adozione di algoritmi di machine learning e intelligenza artificiale per l'elaborazione dei dati raccolti dai sensori. Questi algoritmi consentono una migliore identificazione e classificazione degli odori, rendendo i nasi elettronici più affidabili nell'analisi dei dati complessi.



Figura 2: il naso elettronico realizzato per questo progetto

Questi dispositivi hanno molteplici applicazioni, come il controllo di qualità alimentare, l'analisi farmaceutica, il monitoraggio ambientale, la diagnosi medica e la sicurezza.

Tuttavia, i nasi elettronici hanno limiti, come la limitata specificità, la variabilità nella sensibilità, i costi elevati e le interferenze ambientali.

L'obiettivo di questo progetto è la creazione di un naso elettronico "smart", integrato con una Alexa Skill e un'applicazione mobile.

Nel corso degli ultimi anni, l'intelligenza artificiale ha fatto notevoli progressi. Ora è possibile eseguire molti algoritmi di apprendimento automatico (compresi alcuni modelli di deep learning piuttosto complessi) su microcontrollori ed è possibile combinare letture provenienti da una varietà di sensori per creare dei dataset utili per addestrare un dispositivo in grado di classificare odori.

Come mostrato nella figura 2, il naso elettronico che abbiamo realizzato è basato su un microcontrollore che esegue un modello di classificazione, fornisce una predizione dell'odore e la mostra sul proprio schermo. Quando vengono rilevati odori "pericolosi" come il gas dell'accendino, il dispositivo inizia ad emettere un suono. Questa funzionalità può essere abilitata o disabilitata premendo il pulsante con l'icona  sul pannello frontale del microcontrollore.

Le applicazioni per Alexa e per lo smartphone consentono di monitorare in tempo reale l'odore percepito dal naso elettronico e la composizione chimica dell'aria in quel momento anche se ci si trova in un'altra stanza o fuori casa.

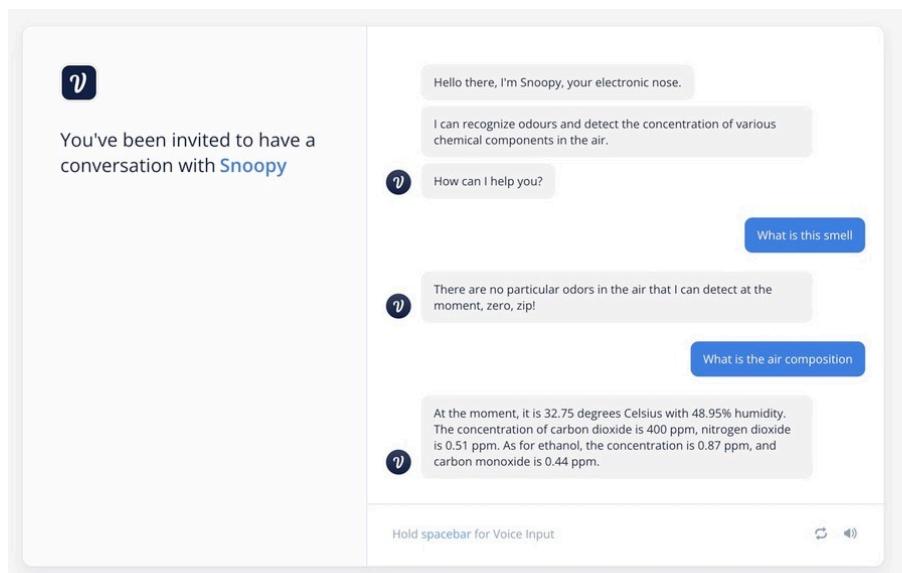


Figura 3: Alexa Skill associata al naso elettronico

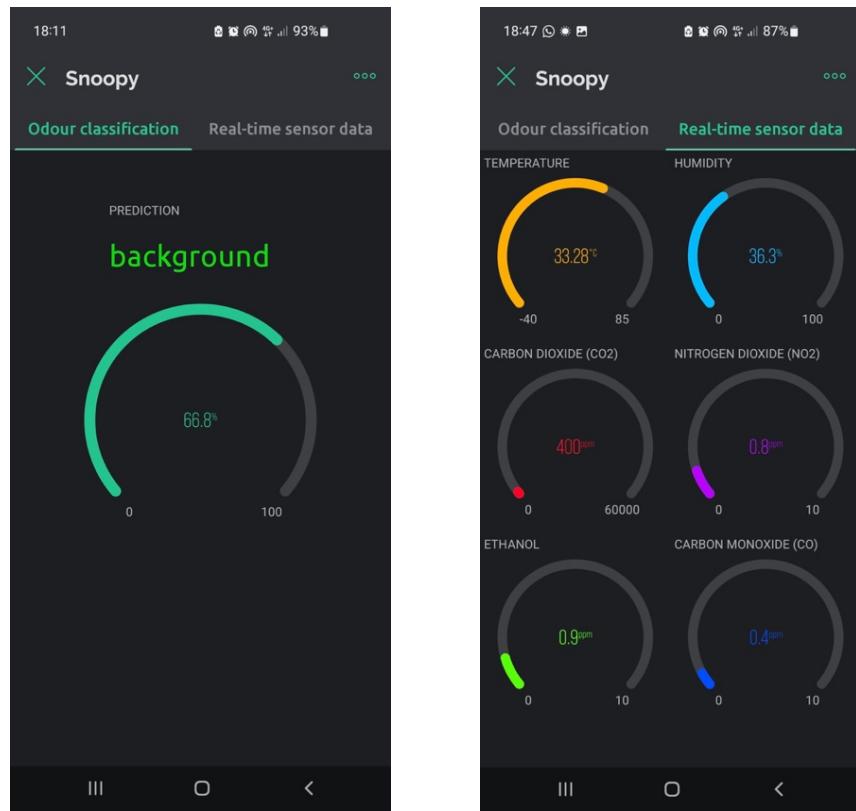


Figura 4: applicazione per smartphone  
associata al naso elettronico

Nei capitoli successivi, verranno forniti dettagli sulle scelte implementative, gli approfondimenti condotti e tutti gli aspetti che hanno portato alla realizzazione di questo progetto.

## Idee dietro il progetto

Il naso umano e il naso elettronico rappresentano due approcci diversi per la rilevazione e il riconoscimento degli odori.

Il naso umano è un sistema biologico incredibilmente sofisticato e versatile. Utilizza recettori degli odori specializzati nella mucosa olfattiva per percepire una vasta gamma di odori, distinguendo tra sottili sfumature e associandoli a esperienze personali. Questo processo coinvolge il cervello, che elabora le informazioni sensoriali e le confronta con le memorie olfattive memorizzate.

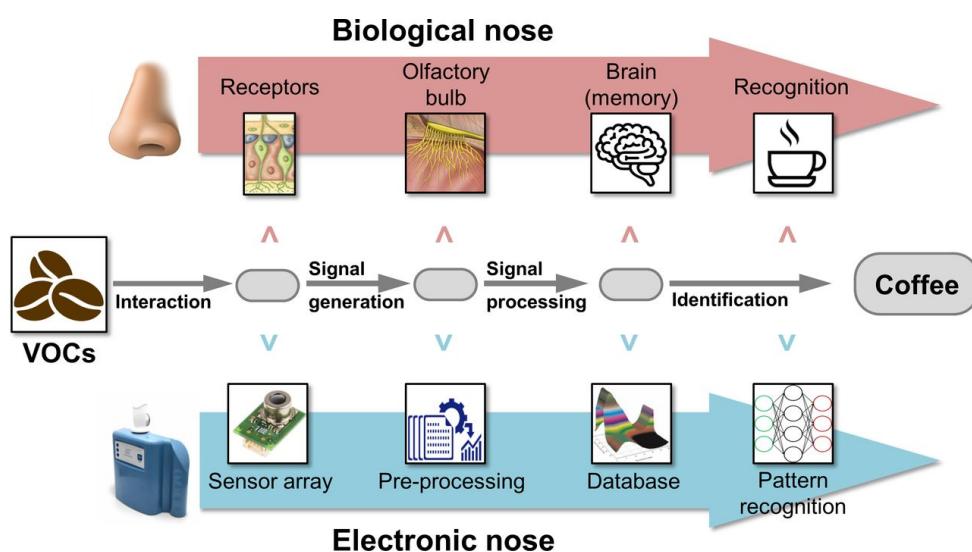


Figura 5: naso biologico vs naso elettronico

D'altra parte, il naso elettronico è una creazione tecnologica che sfrutta sensori chimici per rilevare le sostanze chimiche nell'aria. I dati provenienti dai sensori vengono successivamente elaborati da un sistema di gestione dati. Questa fase di preprocessing è essenziale poiché i segnali possono variare a causa di diversi fattori ambientali, come la temperatura o l'umidità. La normalizzazione dei dati è una delle operazioni fondamentali in questa fase, in quanto riduce le variazioni e garantisce la comparabilità tra i sensori.

Per quanto riguarda il riconoscimento degli odori, il naso elettronico utilizza algoritmi di machine learning, come le reti neurali. Queste reti sono addestrate su un insieme di dati noti, dove le risposte desiderate sono conosciute. Durante il processo di addestramento, la rete neurale impara a pesare correttamente le connessioni tra i neuroni per minimizzare l'errore tra l'output previsto e l'output desiderato. Una volta addestrate, le reti neurali sono in grado di riconoscere odori sconosciuti analizzando le risposte dei sensori e producendo una probabilità per ciascuna classe odore.

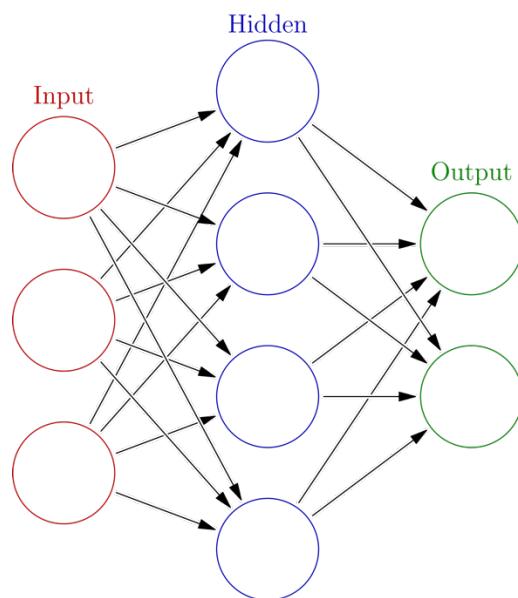


Figura 6: struttura di una rete neurale

## Architettura del sistema

Il nostro sistema è composto da quattro componenti principali:

- Il naso elettronico vero e proprio: comprende un microprocessore e dei sensori che consentono la rilevazione della composizione chimica dell'aria e la classificazione degli odori. Il microprocessore è in grado di connettersi a Internet.
- Il cloud di Blynk: riceve e archivia i dati rilevati dal naso elettronico.
- L'applicazione per smartphone: permette il monitoraggio remoto del naso, mostrando in tempo reale le variazioni dei componenti chimici nell'aria e la classificazione degli odori. Ottiene le informazioni necessarie interrogando il cloud di Blynk.
- La Skill di Alexa: utilizzando delle richieste HTTP GET, acquisisce le informazioni necessarie per rispondere vocalmente all'utente quando vengono poste domande riguardo all'odore corrente e ai componenti presenti nell'aria.

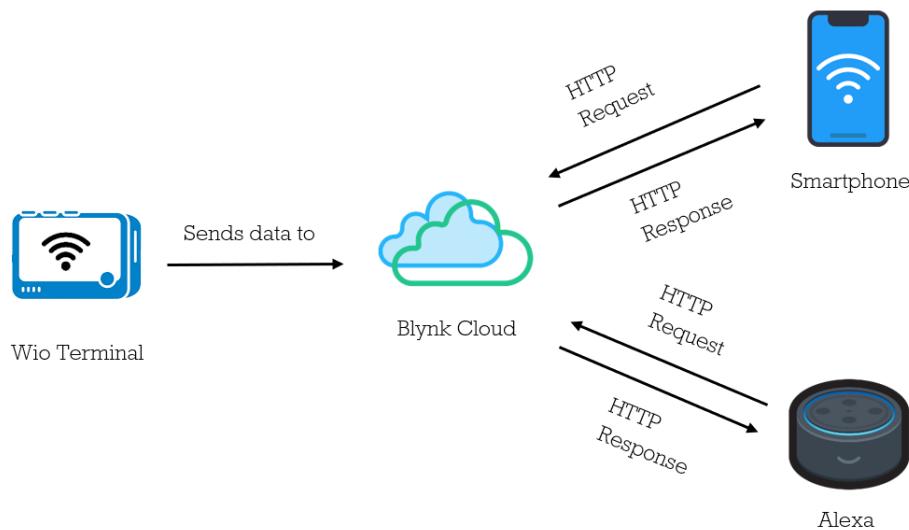


Figura 7: architettura del sistema

## Scelta dell'hardware

Per la realizzazione del naso elettronico, abbiamo dovuto acquistare diversi componenti. Era necessario trovare un microcontrollore sufficientemente potente da eseguire un classificatore e, allo stesso tempo, offrire la possibilità di fornire un feedback visivo delle previsioni tramite uno schermo. Un'altra priorità era la capacità di connettersi a Internet per poterlo integrare con altri dispositivi. Alla fine, abbiamo scelto il Wio Terminal, che è diventato il cervello del nostro naso elettronico.



Figura 8: Wio Terminal

Wio Terminal[3] è un dispositivo hardware creato da Seeed Studio, un'azienda specializzata in prodotti elettronici open source e sviluppo di hardware. Wio Terminal è essenzialmente un microcontroller basato su un SAMD51, un potente chip che combina un processore dual-core a 32 bit, connettività Wi-Fi e Bluetooth, e una serie di periferiche di input/output.

Le caratteristiche principali di Wio Terminal includono:

- Display LCD a colori da 2,4 pollici: Il dispositivo è dotato di un display a colori che permette di visualizzare informazioni e interagire con il dispositivo.
- Porte di espansione Grove: Wio Terminal è compatibile con il sistema di connettività Grove di Seeed Studio, che consente di collegare facilmente vari sensori e moduli senza la necessità di saldature.

- Interfaccia USB-C: Wio Terminal dispone di una porta USB-C per l'alimentazione elettrica e la connessione a un computer per la programmazione.

Oltre al modulo Realtek RTL8720DN, fondamentale per la connettività Wi-Fi, sono state sfruttate altre interessanti caratteristiche di Wio Terminal in questo progetto:

- 5-Way Switch: si tratta del pulsante situato frontalmente sul pannello LCD del dispositivo.
- Buzzer: è un dispositivo elettronico che produce un suono o un segnale acustico quando viene alimentato con tensione elettrica. È costituito da un piccolo altoparlante e da un oscillatore incorporato, che genera vibrazioni sonore quando riceve una corrente elettrica. Queste vibrazioni producono un suono udibile dall'orecchio umano.
- Slot per MicroSD Card: esso è particolarmente utile se si desidera visualizzare immagini sullo schermo del Wio Terminal, poiché è necessario salvarle precedentemente su una scheda MicroSD.

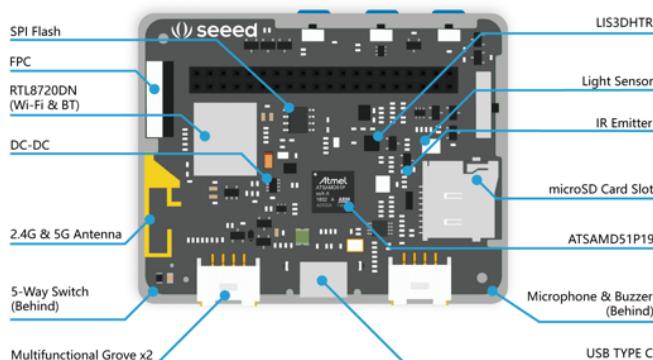


Figura 9: Hardware Overview di Wio Terminal

È possibile programmare Wio Terminal utilizzando Arduino, MicroPython o ArduPy. Per questo progetto abbiamo scelto l'IDE Arduino in quanto offre una vasta gamma di librerie predefinite e dispone di un'ampia documentazione.

Passiamo ora alla descrizione dei sensori. Tutti questi sensori sono stati progettati da Seeed Studio e possono essere facilmente collegati a Wio Terminal tramite una delle due porte Grove situate nella parte inferiore del dispositivo.

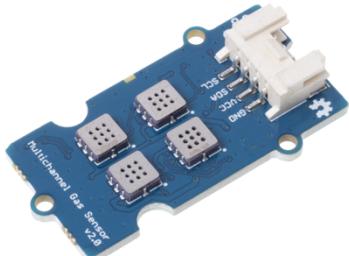


Figura 10: Grove - Gas Sensor V2 (Multichannel)

Grove - Gas Sensor V2 (Multichannel)[4] è composto da quattro elementi sensoriali completamente indipendenti in un unico package.

Esso è in grado di rilevare i seguenti componenti chimici:

- monossido di carbonio (CO);
- diossido di azoto (NO<sub>2</sub>);
- etanolo (C<sub>2</sub>H<sub>5</sub>CH);
- composti organici volatili (VOC).

Grove - VOC and eCO<sub>2</sub> Gas Sensor (SGP30)[5] è un sensore di gas che è in grado di misurare la concentrazione di due importanti componenti dell'aria:

- TVOC (Total Volatile Organic Compounds): questa misura rappresenta la concentrazione totale di composti organici volatili nell'aria. I composti organici volatili sono una classe di sostanze chimiche che possono evaporare facilmente in aria e includono vari composti come solventi, idrocarburi, prodotti chimici industriali e altri.
- eCO<sub>2</sub> (equivalent Carbon Dioxide): Questa misura rappresenta l'equivalente di anidride carbonica (CO<sub>2</sub>) nell'aria, anche se il sensore non misura direttamente la CO<sub>2</sub>. L'eCO<sub>2</sub> è una stima della concentrazione di CO<sub>2</sub> basata sui livelli di TVOC rilevati, poiché la presenza di TVOC è spesso correlata alla presenza di CO<sub>2</sub>.

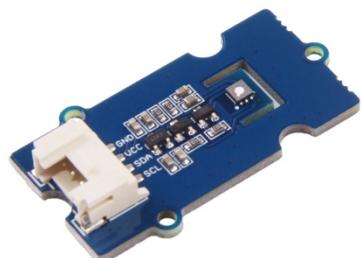


Figura 11:  
Grove-VOC and eCO<sub>2</sub> Gas Sensor (SGP30)

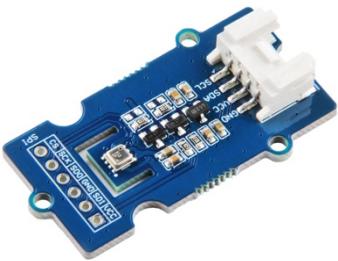


Figura 12: Grove -Temperature Humidity Pressure Gas (BME680)

Il sensore Grove -Temperature Humidity Pressure Gas (BME680)[6] è un sensore multifunzione in grado di misurare contemporaneamente la temperatura, la pressione e l'umidità.

Wio Terminal offre solo due porte Grove, ma desideravamo comunque utilizzare tutti e tre i sensori, quindi abbiamo dovuto acquistare un altro componente: il Grove - I2C Hub[7], un dispositivo progettato per semplificare la connessione e l'espansione di dispositivi I2C, consentendoci di collegare più sensori e periferiche al nostro progetto senza complicazioni aggiuntive.

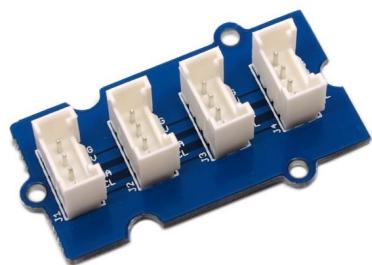


Figura 13: Grove - I2C Hub

Nel progetto è stato necessario visualizzare delle immagini sullo schermo di Wio Terminal, pertanto, come già spiegato in precedenza, è stato indispensabile l'utilizzo di una scheda MicroSD.

Nell'idea iniziale, il naso elettronico non doveva solo fornire un feedback visivo sulla classificazione avvenuta, ma anche un feedback uditorio in cui la stringa dell'odore classificato veniva letta da Wio Terminal. Nonostante la presenza di numerosi componenti integrati, in Wio Terminal non c'è un componente dedicato all'audio, ad eccezione del buzzer, che tuttavia può produrre solo suoni molto semplici. Abbiamo quindi acquistato una scheda di estensione audio chiamata ReSpeaker 2-Mic Hat, ma questa offriva una qualità del suono distorta e ovattata. Inoltre, la nostra priorità era sempre stata quella di avere un naso elettronico portatile e di dimensioni ridotte. Tuttavia, con l'aggiunta di questo componente, ciò non era più possibile. Di conseguenza, abbiamo deciso di riprogettare l'interazione e nella versione finale del progetto, questo componente non è stato più utilizzato.

## Raccolta dei dati ed addestramento del modello

In questa sezione, verrà descritto il processo di raccolta dei dati utilizzati per addestrare il modello alla base del naso elettronico.

In questo contesto, le classi di odori che abbiamo deciso di predire sono:

- Background (odore neutrale nell'ambiente)
- Caffè
- Aceto
- Martini ( bevanda alcolica)
- Camomilla
- Tè
- Gas dell'accendino
- Cacao



Figura 14: rilevazione dell'aceto

Il naso elettronico può essere anche un dispositivo di sicurezza. Da qui nasce l'idea di segnalare all'utente eventuali fughe di gas in casa.

Chiaramente, vista l'impossibilità di poter usare altre sorgenti di gas più pericolose, abbiamo deciso di usare il gas dell'accendino per la raccolta dati.

I dati sono stati rilevati con l'hardware specificato nel capitolo precedente: il dispositivo Wio Terminal equipaggiato con i tre sensori Grove - Gas Sensor V2 (Multichannel), Grove - VOC and eCO<sub>2</sub> Gas Sensor (SGP30) e Grove - Temperature Humidity Pressure Gas (BME680).

Per semplicità, nel resto della relazione, indicheremo i tre sensori rispettivamente come:

- Multichannel Gas
- SGP30
- BME680

Per integrare correttamente i sensori nel codice Arduino (contenuto nel file Main.ino), abbiamo utilizzato tre librerie specifiche:

1. Multichannel\_Gas\_GMXXX.h per il sensore Multichannel Gas.
2. seeed\_bme680.h per il sensore ambientale BME680.
3. sgp30.h per il sensore SGP30.

```
#include "Multichannel_Gas_GMXXX.h"
#include "seeed_bme680.h"
#include "sgp30.h"
```

Figura 15: come vengono importate le librerie Multichannel\_Gas\_GMXXX.h, seeed\_bme680.h e sgp30.h nel file Main.ino

I sensori sono stati inizializzati nel metodo setup(). Come mostrato in figura 16, il sensore Multichannel Gas è inizializzato con gas.begin(Wire, 0x08), il sensore BME680 con bme680.init(), e il sensore SGP30 con sgp\_probe(). Dei messaggi di errore vengono visualizzati sulla porta seriale nel caso in cui l'inizializzazione dei sensori fallisca.

```
// Initialize gas sensors
gas.begin(Wire, 0x08);

// Initialize environmental sensor
while (!bme680.init()) {
    Serial.println("Trying to initialize BME680...");
    delay(1000);
}

// Initialize VOC and eCO2 sensor
while (sgp_probe() != STATUS_OK) {
    Serial.println("Trying to initialize SGP30...");
    delay(1000);
}
```

Figura 16: inizializzazione dei sensori

Per quanto riguarda invece il campionamento dei dati, questo è stato eseguito a una frequenza specifica di 4 Hz, ovvero 4 campioni al secondo. Per ciascun campione sono state raccolte 8 letture da ciascun sensore per andare a formare dei files con nomenclatura “{label dell'odore rilevato}.{timestamp in millisecondi}.csv”.

Nel ciclo in figura 17, vengono eseguite le letture dai sensori, e i dati vengono memorizzati nelle variabili appropriate. Nel caso in cui la lettura da uno dei sensori fallisca, viene visualizzato un messaggio di errore sulla porta seriale.

In particolare, le letture dal sensore Multichannel Gas sono state salvate nelle variabili gm\_no2\_v, gm\_eth\_v, gm\_voc\_v, gm\_co\_v. Le letture dal sensore SGP30 sono state memorizzate in sgp\_tvoc e sgp\_co2 mentre quelle dal sensore BME680 sono state salvate nelle variabili bme\_temp, bme\_hum e bme\_press.

Tutte le letture dei sensori sono poi state inviate alla porta seriale.

```

for (int i = 0; i < NUM_SAMPLES; i++) {
    // Take timestamp so we can hit our target frequency
    timestamp = millis();

    // Read from Multichannel sensor
    gm_no2_v = gas.calcVol(gas.getGM102B());
    gm_eth_v = gas.calcVol(gas.getGM302B());
    gm_voc_v = gas.calcVol(gas.getGM502B());
    gm_co_v = gas.calcVol(gas.getGM702B());

    // Read BME680 environmental sensor
    if (bme680.read_sensor_data()) {
        Serial.println("Error: Could not read from BME680");
        return;
    }

    bme_temp = bme680.sensor_result_value.temperature;
    bme_hum = bme680.sensor_result_value.humidity;
    bme_press = bme680.sensor_result_value.pressure;

    // Read SGP30 sensor
    sgp_err = sgp_measure_iaq_blocking_read(&sgp_tvoc, &sgp_co2);
    if (sgp_err != STATUS_OK) {
        Serial.println("Error: Could not read from SGP30");
        return;
    }
}

```

Figura 17: lettura dati dei sensori

```

Serial.print(timestamp);
Serial.print(",");
Serial.print(bme_temp);
Serial.print(",");
Serial.print(bme_hum);
Serial.print(",");
Serial.print(bme_press / PA_IN_KPA);
Serial.print(",");
Serial.print(sgp_co2);
Serial.print(",");
Serial.print(sgp_tvoc);
Serial.print(",");
Serial.print(gm_voc_v);
Serial.print(",");
Serial.print(gm_no2_v);
Serial.print(",");
Serial.print(gm_eth_v);
Serial.print(",");
Serial.print(gm_co_v);
Serial.println();

```

Figura 18: Dati stampati sulla porta seriale

Questo passo serve a facilitare l'estrapolazione di file in formato CSV. In questo contesto viene anche inviato un header che specifica i nomi delle colonne.

```
Serial.println("timestamp,temp,humd,pres,co2,voc1,voc2,no2,eth,co");
```

Figura 19: header

Stampando i dati in questo formato viene consentito ad uno script Python denominato "collect\_csv\_from\_serial\_port.py" di catturare i dati in tempo reale dalla porta seriale e utilizzarli per costruire il dataset. Per eseguire con successo lo script, è essenziale fornire alcuni argomenti quando si avvia il file da linea di comando. La struttura del comando deve essere la seguente:

'python collect\_csv\_from\_serial\_port.py -p {identificativo della porta seriale su cui ascoltare} -b 115200 -d {percorso di destinazione} -l {label dell'odore rilevato}'.

Per ciascuno degli 8 odori menzionati in precedenza, sono state effettuate 9 misurazioni in diverse condizioni di temperatura e umidità. Queste misurazioni sono state condotte durante diverse fasce orarie, tra cui il mattino, il pomeriggio e la sera, in vari giorni, sia all'interno che all'esterno, presso due località distinte: Pofi (FR) e Cori (LT), le due località in cui viviamo.

Questa diversità di contesti e ambienti di rilevazione ha contribuito a rendere il nostro dataset più rappresentativo della realtà.

In ogni sessione di rilevazione, abbiamo registrato 150 campioni, ognuno composto da 8 letture (come mostrato nella figura 20, 8 letture costituiscono un file csv) dello stesso odore.

	<b>timestamp</b>	<b>temp</b>	<b>humd</b>	<b>pres</b>	<b>co2</b>	<b>voc1</b>	<b>voc2</b>	<b>no2</b>	<b>eth</b>	<b>co</b>
1	60488286	31.06	56.41	99.85	16522	60000	2.50	1.55	2.07	2.75
2	60488536	31.07	56.31	99.85	15378	60000	2.51	1.55	2.07	2.74
3	60488786	31.07	56.18	99.85	14206	60000	2.51	1.55	2.07	2.73
4	60489036	31.07	56.05	99.85	13158	60000	2.51	1.55	2.07	2.70
5	60489286	31.08	55.89	99.85	12014	60000	2.51	1.55	2.07	2.69
6	60489536	31.08	55.74	99.85	11504	60000	2.50	1.55	2.06	2.68
7	60489786	31.09	55.61	99.85	11192	60000	2.51	1.55	2.06	2.68
8	60490036	31.09	55.49	99.85	10943	60000	2.51	1.55	2.06	2.67

Figura 20: uno dei file csv che compongono il nostro dataset

Il risultato di questa raccolta dati è un dataset contenente 86.912 istanze.

Tuttavia, prima di procedere con la creazione del nostro classificatore di odori, abbiamo eseguito una fase di preprocessing dei dati. Questa fase è stata essenziale per assicurare qualità e coerenza al nostro dataset.

Per prima cosa, abbiamo effettuato un'analisi attenta dei dati. Abbiamo verificato che il numero di campioni per ciascuna classe di odori fosse bilanciato, il che ha reso superflue tecniche di data augmentation.

Questo equilibrio tra classi è stato confermato attraverso la creazione e lo studio di grafici dimostrativi. Successivamente, abbiamo esaminato le correlazioni tra le diverse classi di odore mediante la costruzione di matrici di correlazione. Questo ci ha permesso di identificare eventuali relazioni tra le diverse caratteristiche dei dati. Inoltre, abbiamo creato grafici violin per visualizzare la distribuzione dei dati in relazione a ciascuna classe di odore.

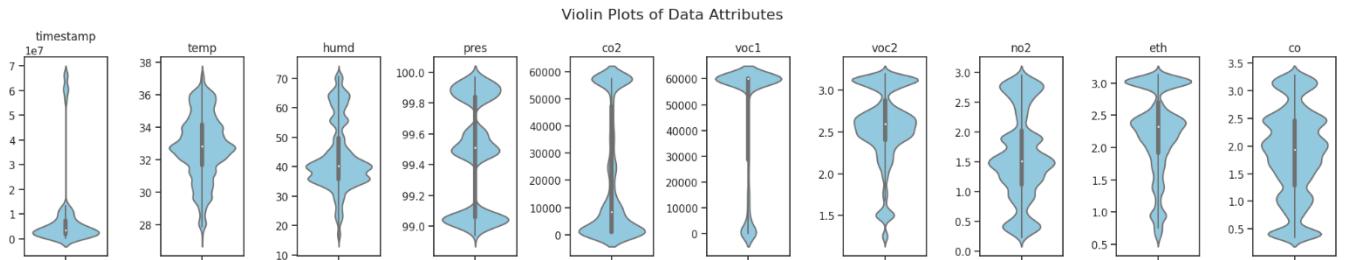


Figura 21: violin plots

Durante questa fase di analisi, abbiamo rilevato che la feature relativa alla pressione atmosferica non era discriminante per il nostro task di riconoscimento degli odori. La pressione atmosferica tendeva a variare in modo trascurabile nel contesto dell'ambiente in cui operavamo.

Pertanto, abbiamo preso la decisione di rimuovere questa feature dai nostri dati al fine di ridurne la complessità.

Per quanto riguarda le altre features, tranne "timestamp," abbiamo applicato una normalizzazione. Questo passaggio è stato fondamentale per garantire che tutte le features avessero valori compresi nello stesso intervallo (tra 0 e 1) per tutte le istanze del dataset. La normalizzazione è stata un fattore chiave per migliorare l'efficacia e la robustezza del nostro modello di machine learning durante la successiva fase di

addestramento. Per ulteriori dettagli sul data preprocessing vedere il file “Data\_preprocessing.ipynb”.

A questo punto il nostro dataset era pronto per l'addestramento.

Tuttavia, ci siamo trovati di fronte a diverse sfide significative per garantire che il nostro classificatore di odori potesse operare in tempo reale su un dispositivo con risorse limitate come il Wio Terminal. Il problema più grande era l'addestramento di un modello sufficientemente potente ma allo stesso tempo costituito da pochi parametri, considerando le restrizioni legate al nostro hardware.

Per superare questi ostacoli, abbiamo sfruttato Edge Impulse, una potente piattaforma online specializzata in intelligenza artificiale e edge computing. Questa piattaforma gioca un ruolo fondamentale nell'agevolare lo sviluppo, l'addestramento e l'implementazione di modelli di machine learning e intelligenza artificiale direttamente su dispositivi edge come microcontrollori, sistemi embedded e sensori. Supporta il caricamento, l'addestramento di modelli di classificazione e regressione, e offre strumenti per ottimizzarne le prestazioni.

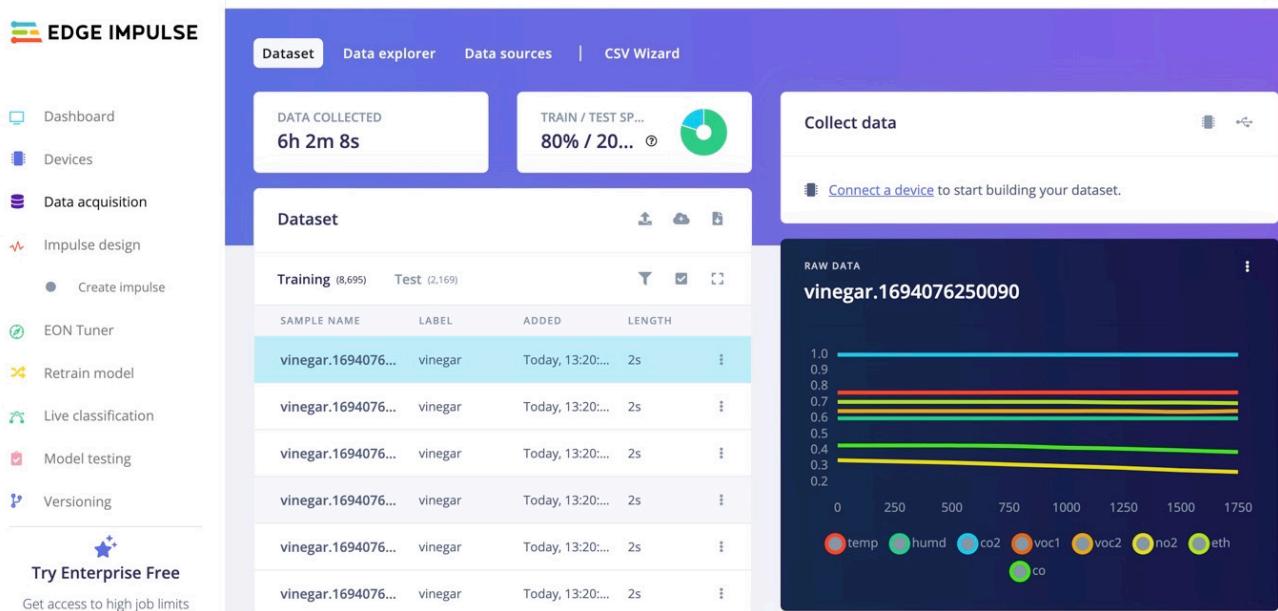


Figura 22: Edge Impulse

Questa piattaforma è stata fondamentale perché ci ha permesso di addestrare il nostro modello utilizzando risorse GPU potenti, ottenendo un modello compatibile con l'hardware del Wio Terminal, che richiede una bassa potenza di calcolo e di memoria.

Il processo è iniziato con il caricamento del nostro vasto dataset, composto da 86.912 istanze di dati, sulla piattaforma Edge Impulse. Al fine di ottimizzare l'efficienza dell'allenamento e dei test, abbiamo diviso il dataset in due parti: l'80% (69.530 istanze) è stato dedicato all'addestramento, mentre il restante 20% (17.382 istanze) è stato riservato per i test. Nella figura 22 è mostrato come appare il dataset una volta caricato su Edge Impulse dopo la suddivisione.

Prima di creare il classificatore per i nostri 8 odori, abbiamo eseguito un tuning degli iperparametri per individuare la configurazione ottimale per il nostro modello. Questo processo è stato reso possibile grazie alla funzionalità di EON Tuner fornita da Edge Impulse. L'EON Tuner ci ha permesso di esplorare una vasta gamma di parametri al fine di generare il miglior modello possibile in base ai nostri dati.

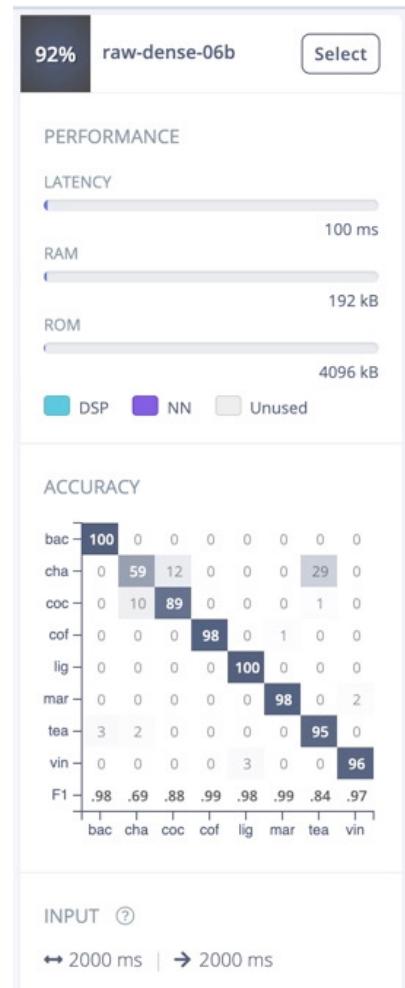


Figura 23: EON Tuner

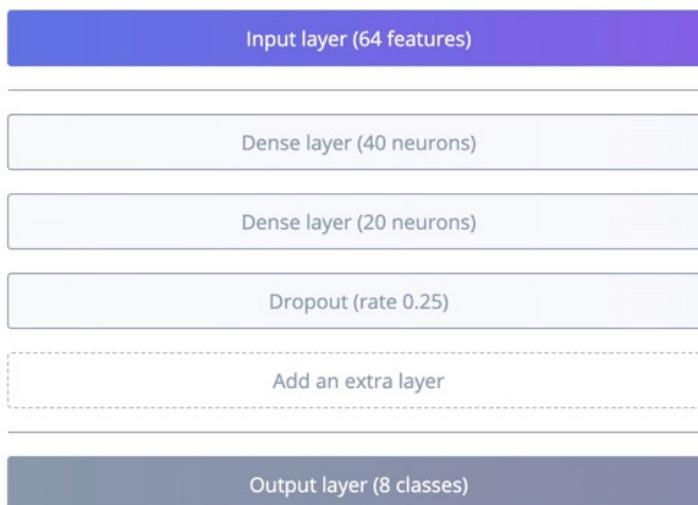


Figura 24: rete neurale

In figura 24 è mostrata la rete neurale ottenuta dopo il tuning degli iperparametri. La rete è sviluppata utilizzando la libreria Keras che è integrata in Edge Impulse.

I dati sono stati elaborati dal primo layer, noto come "raw data", e poi passati attraverso i vari layer della rete, tra cui layer densi con 40 neuroni e 20 neuroni, e un layer di dropout con una percentuale di dropout del 25% per mitigare il rischio di overfitting. L'output del modello include le probabilità associate alle diverse classi di odori.

Durante l'addestramento, il modello è stato sottoposto a 300 epocha con un tasso di apprendimento (learning rate) di 0.0005. Questo processo è stato completato in circa 20 minuti grazie alle potenti GPU messe a disposizione da Edge Impulse.

Durante l'addestramento, il modello ha raggiunto un'accuratezza del 97%.

Per perfezionare e rendere più affidabile il modello abbiamo addestrato anche un anomaly detector.

L'obiettivo di un anomaly detector è individuare le istanze di dati considerate "rumorose" o anomale. Queste istanze rappresentano valori outlier rispetto alle medie dei valori delle features per ciascun odore specifico durante la fase di rilevazione dei dati. Allenare un anomaly detector è stato essenziale per migliorare la robustezza del nostro classificatore nel generalizzare su dati mai visti in precedenza. L'individuazione di queste istanze anomale ha permesso al modello di rilevare gli odori in modo più accurato e coerente, evitando potenziali classificazioni errate dovute a dati rumorosi.



Figura 25: risultato anomaly detection

Allenato l'anomaly detector, il modello è stato testato sul 20% del dataset riservato per i test, ottenendo un'accuratezza del 98%. È importante notare che abbiamo valutato le prestazioni del modello utilizzando anche l'indicatore F1-score, che tiene conto sia della precision che della recall del modello. In generale, l'F1-score mostra valori molto elevati, tendendo a 1 per la maggior parte delle classi di odori, ad eccezione delle classi "camomilla" (90%) e "cacao" (94%), dove l'F1-score è leggermente inferiore. Infatti, osservando il grafico sottostante, emerge che in alcune occasioni questi due odori vengono classificati in modo errato, con una tendenza a confonderli tra loro o con il tè, poiché hanno composizioni particolarmente simili.

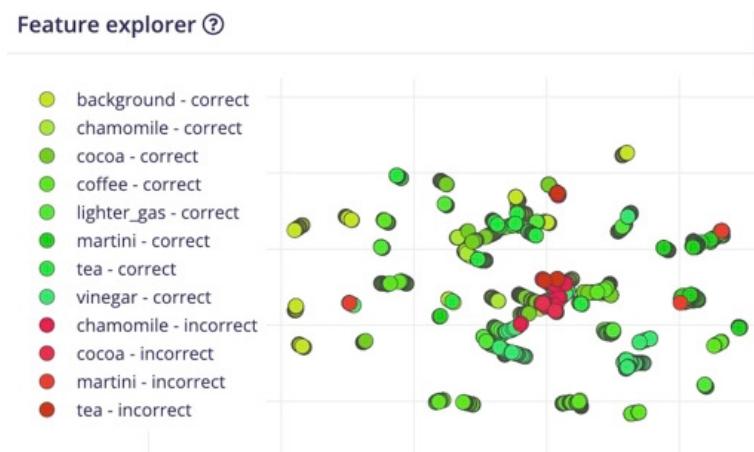


Figura 26: visualizzazione della classificazione dei dati come punti su un piano

Ciononostante, come si può notare nella matrice in figura 27, questi risultati sono particolarmente promettenti per la classe "lighter\_gas", dove avere accuracy ma soprattutto F1-score alti era fondamentale per rilevare eventuali fughe di gas in modo affidabile.

Essendo la rilevazione di una potenziale fuga di gas una delle funzionalità critiche del nostro sistema, minimizzare il numero di falsi negativi e falsi positivi era cruciale. Nel primo caso, anche un solo falso negativo, ovvero una situazione in cui il sistema non rileva una vera fuga di gas, avrebbe potuto comportare gravi conseguenze. Nel secondo caso, avere falsi positivi e ricevere continue segnalazioni di perdite di gas anche quando non sono effettivamente presenti avrebbe reso il sistema inaffidabile per l'utente.

### Model testing results

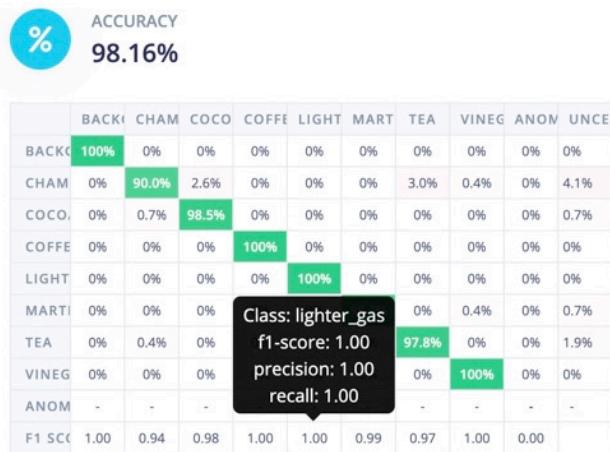


Figura 27: performance della classe “lighter\_gas”

Una volta completato il processo di addestramento del nostro modello e dell'anomaly detector, Edge Impulse consente di scaricare un file compresso (formato zip) contenente una libreria Arduino con il nostro modello allenato e dei metodi utili per eseguire il classificatore con Wio Terminal.



Figura 28: schermata riepilogativa mostrata su Edge Impulse dopo aver scaricato la libreria

Nei capitoli successivi verrà mostrato come il sistema è stato progettato e implementato utilizzando questo modello di machine learning.

## Progettazione del sistema

Nella progettazione di un sistema, il processo inizia con la definizione delle funzionalità chiave che il sistema deve avere. Le funzionalità rappresentano le capacità generali del sistema. Successivamente, si utilizzano queste funzionalità come blocchi fondamentali per creare i casi d'uso, che rappresentano scenari specifici in cui gli utenti o i sistemi interagiscono con il sistema per raggiungere obiettivi specifici.

Un diagramma dei casi d'uso è una rappresentazione visuale di questi scenari, mostrando gli attori (utenti o sistemi esterni) coinvolti e le azioni che compiono per raggiungere i loro scopi.

Queste sono le funzionalità che il sistema deve eseguire:

1. Attivazione dei sensori
2. Connessione ad Internet
3. Rilevazione della composizione chimica dell'aria
  - 3.1. Classificazione degli odori
    - 3.1.1. Visualizzazione della classificazione sul display
    - 3.1.2. Emissione di un suono quando la classificazione è "lighter\_gas"
    - 3.1.3. Invio dati a Blynk Cloud
      - 3.1.3.1. Monitoraggio attraverso smartphone
        - 3.1.3.1.1. Visualizzazione della classificazione degli odori
        - 3.1.3.1.2. Visualizzazione della composizione chimica nell'aria
        - 3.1.3.1.3. Ricezione di notifiche
      - 3.1.3.2. Monitoraggio attraverso Alexa
        - 3.1.3.2.1. Fornire informazioni riguardo la classificazione corrente
        - 3.1.3.2.2. Fornire informazioni riguardo la composizione dell'aria
        - 3.1.3.2.3. Fornire assistenza all'utente

#### 4. Visualizzazione di icone sul display

#### 5. Disabilitazione dell'emissione di suoni

- █ Queste funzionalità sono eseguite da Wio Terminal
- █ Queste funzionalità sono eseguite dall'applicazione per smartphone
- █ Queste funzionalità sono eseguite dalla Skill di Alexa

Da queste funzionalità abbiamo ricavato il seguente diagramma dei casi d'uso:

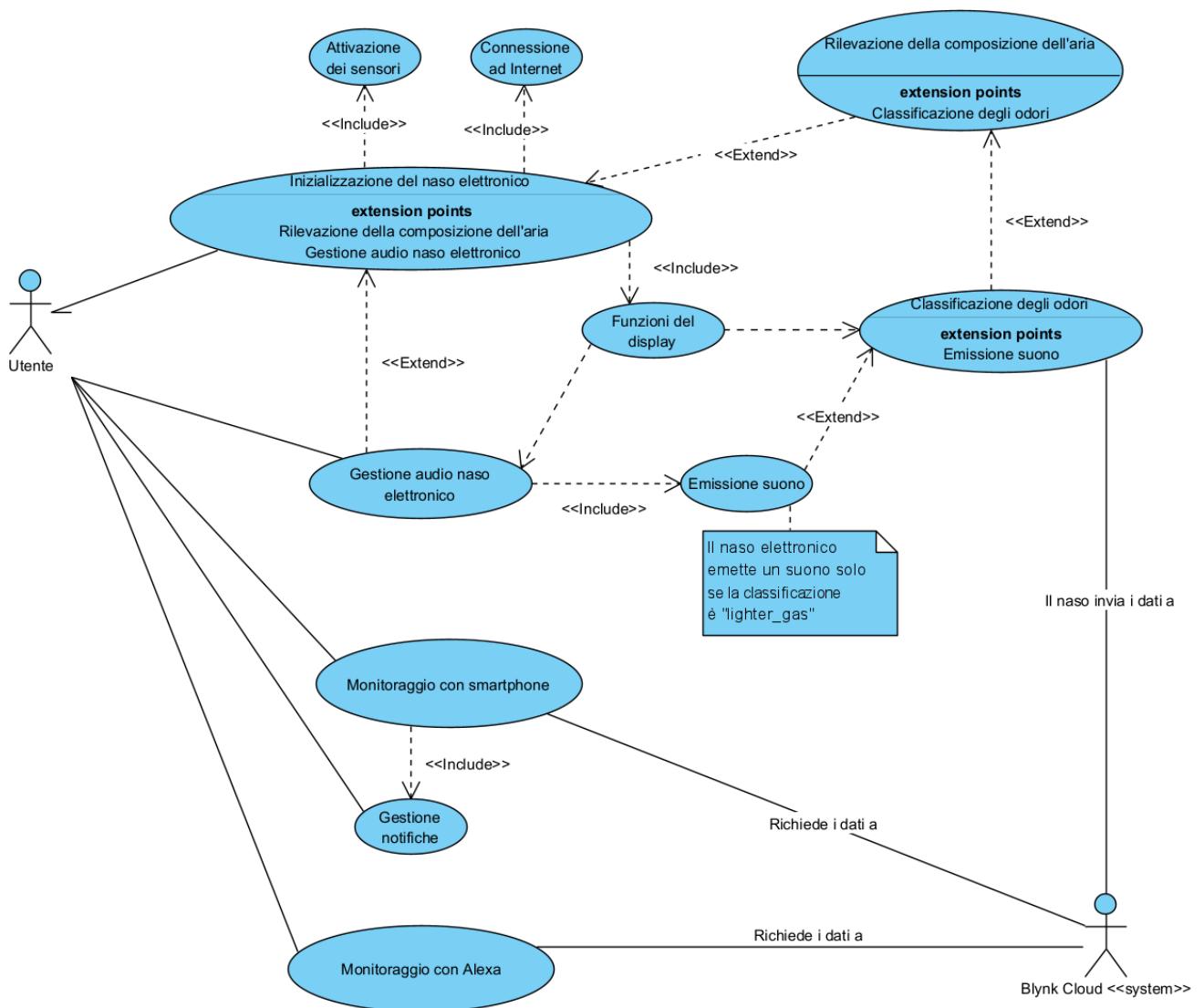


Figura 29: diagramma dei casi d'uso

I diagrammi di attività servono a rappresentare visivamente il flusso di lavoro o le sequenze di azioni all'interno di un sistema o di un processo. Sono utilizzati per modellare le attività, le decisioni, le condizioni e le interazioni tra gli oggetti o gli attori in un sistema.

I diagrammi di attività che verranno presentati sono:

- Inizializzazione naso elettronico
- Classificazione degli odori
- Gestione audio
- Monitoraggio con smartphone
- Gestione notifiche
- Monitoraggio con Alexa

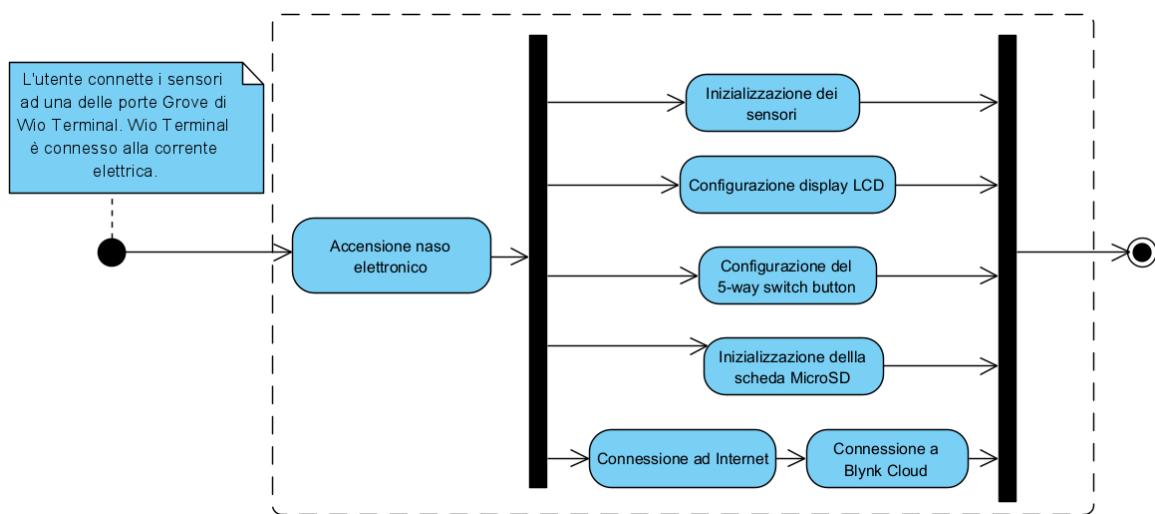


Figura 30: diagramma attività “Inizializzazione naso elettronico”

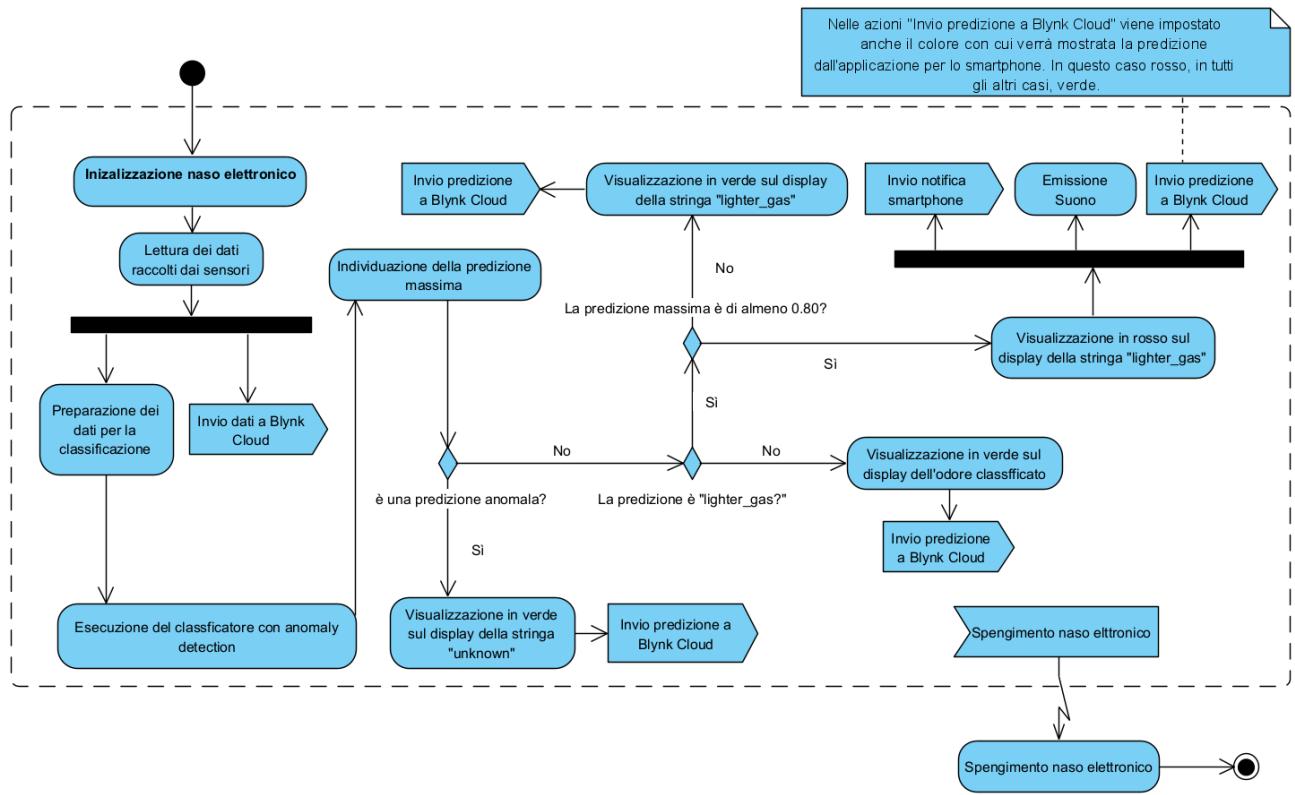


Figura 31: diagramma attività “Classificazione degli odori”

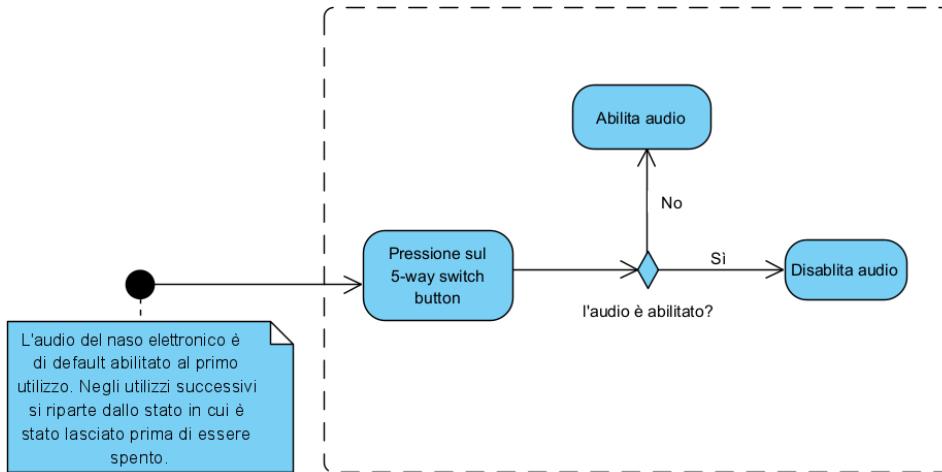


Figura 32: diagramma attività “Gestione audio”

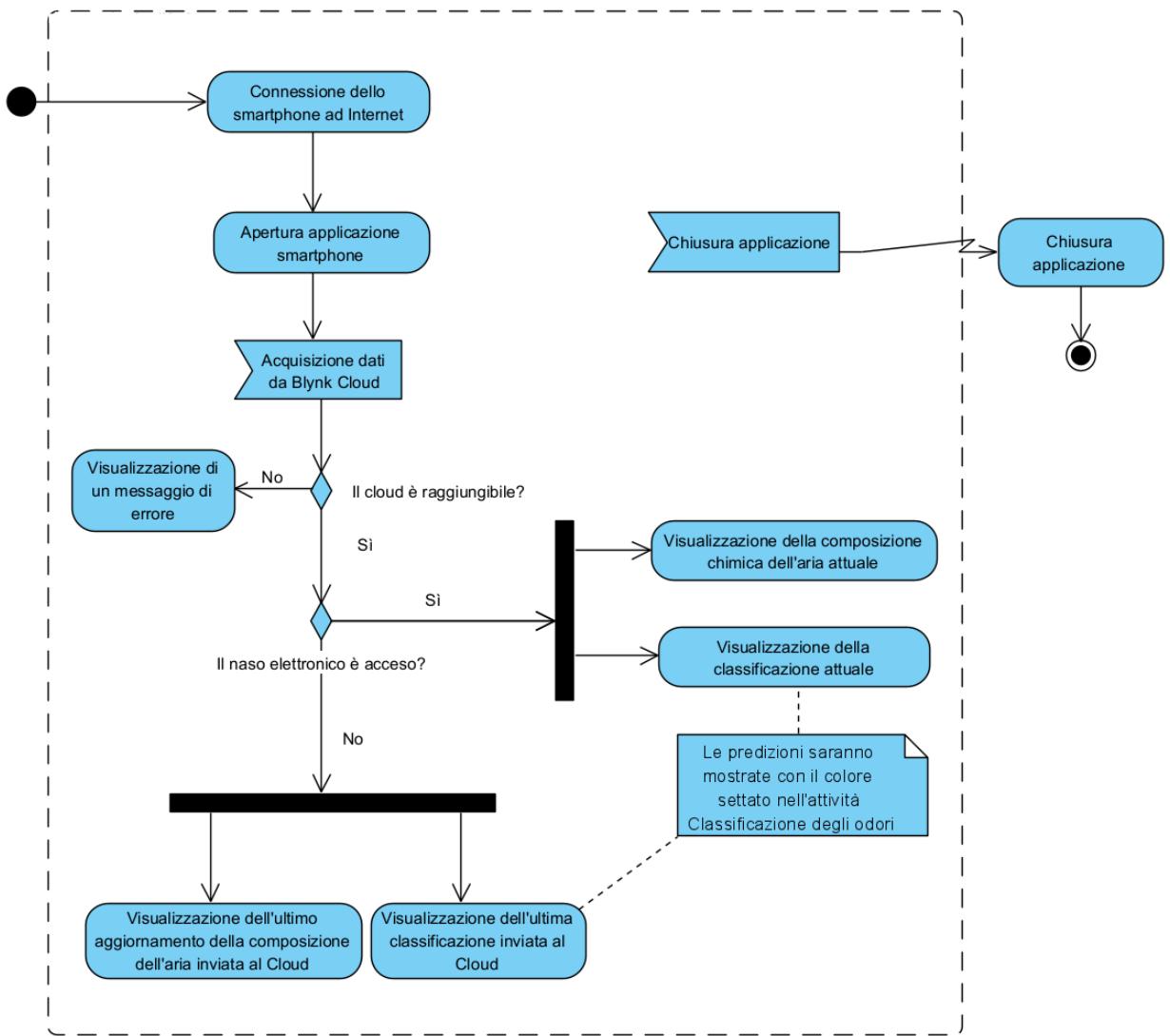


Figura 33: diagramma attività “Monitoraggio con smartphone”

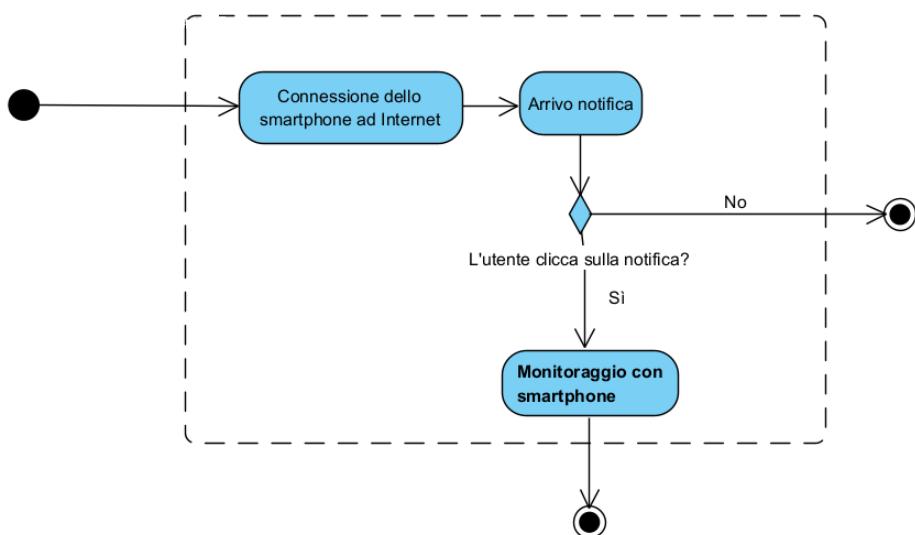


Figura 34: diagramma attività “Gestione notifiche”

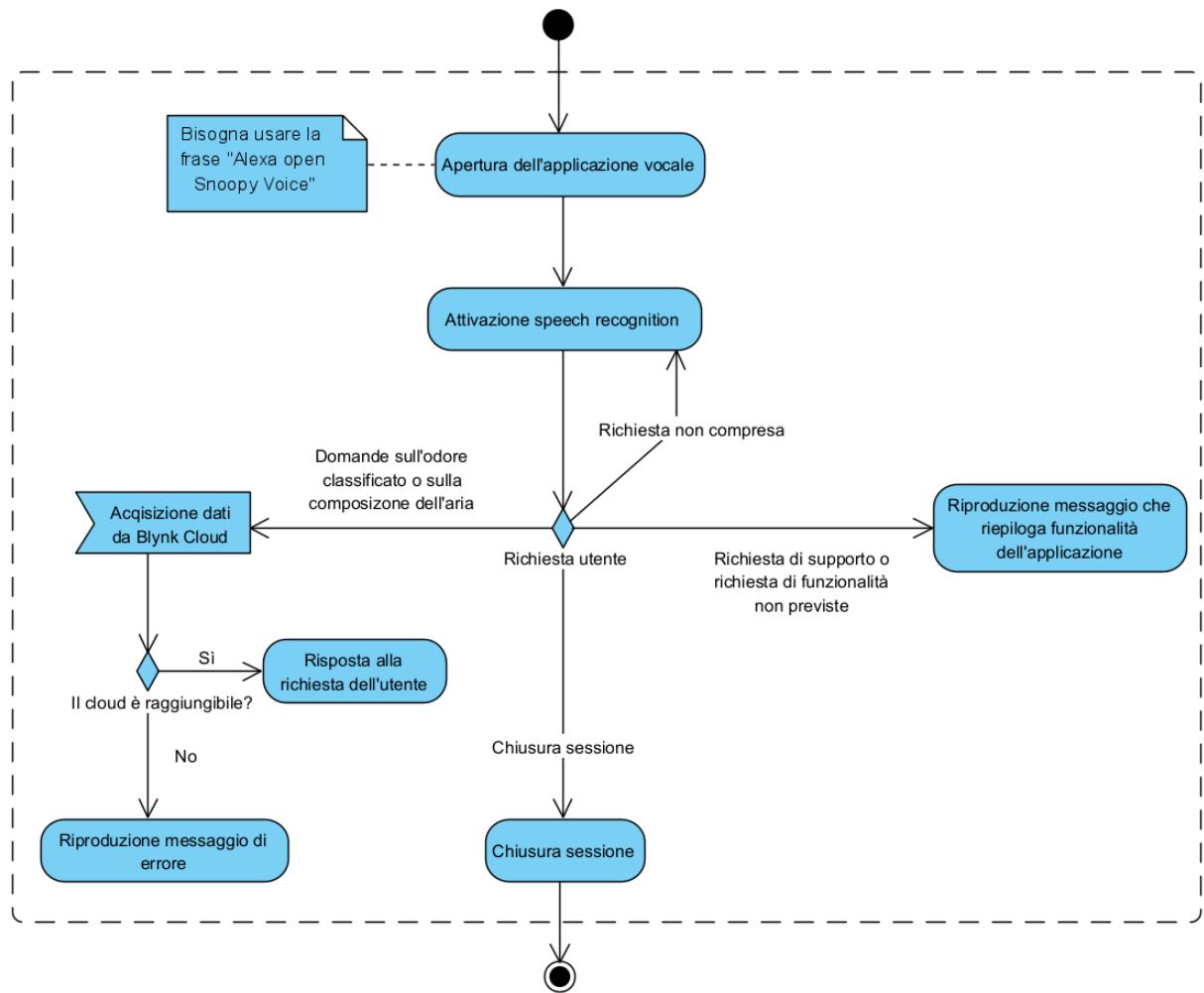


Figura 35: diagramma attività “Monitoraggio con Alexa”

I diagrammi di sequenza servono a rappresentare visivamente l'interazione tra gli oggetti o le componenti di un sistema software nel tempo. Essi mostrano come gli oggetti comunicano tra loro attraverso messaggi, evidenziando l'ordine. Questi diagrammi sono utilizzati per analizzare, progettare e documentare il comportamento dinamico dei sistemi software, facilitando la comprensione delle sequenze di azioni e delle collaborazioni tra gli elementi del sistema.

I diagrammi di attività che verranno presentati sono:

- Gestione audio
- Inizializzazione naso elettronico e classificazione degli odori
- Monitoraggio con smartphone
- Gestione notifiche
- Monitoraggio con Alexa

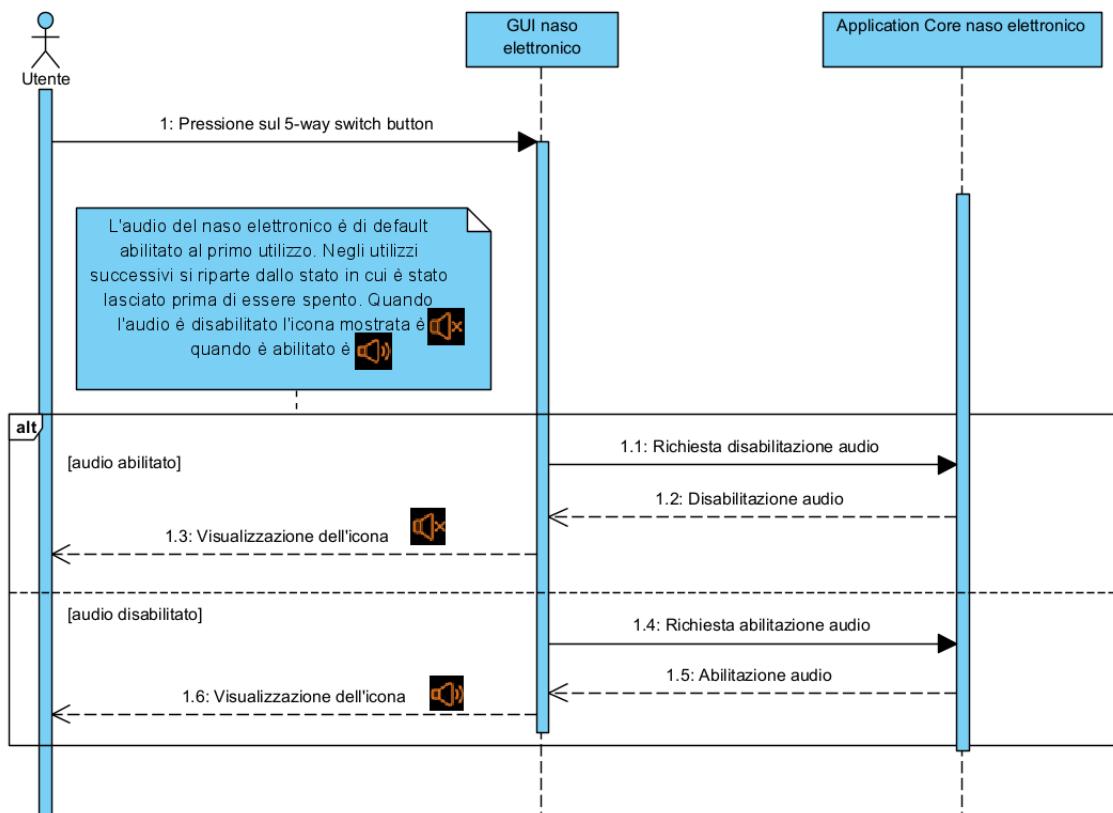


Figura 36: diagramma di sequenza "Gestione audio"

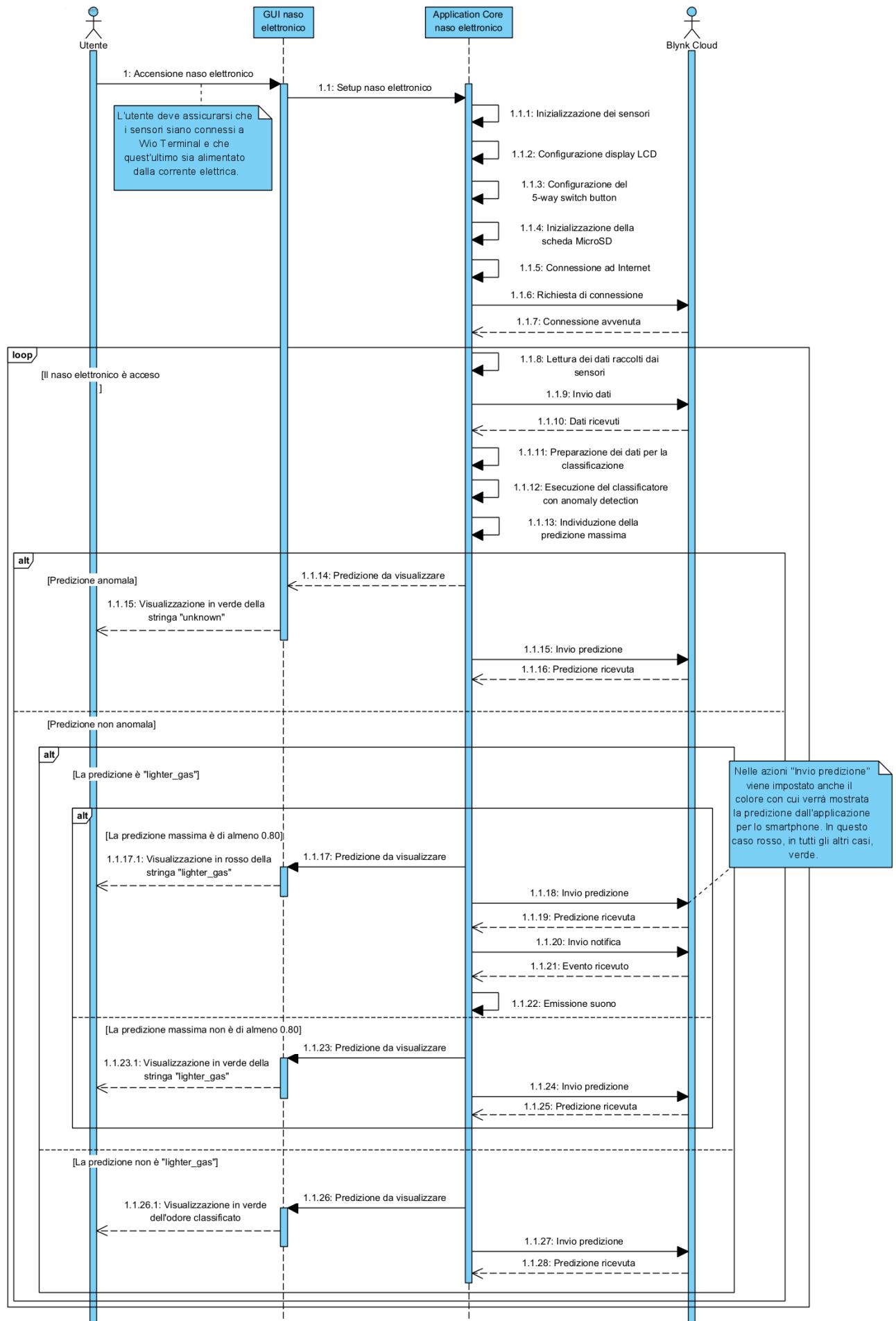


Figura 37: diagramma di sequenza "Inizializzazione naso elettronico e classificazione degli odori"

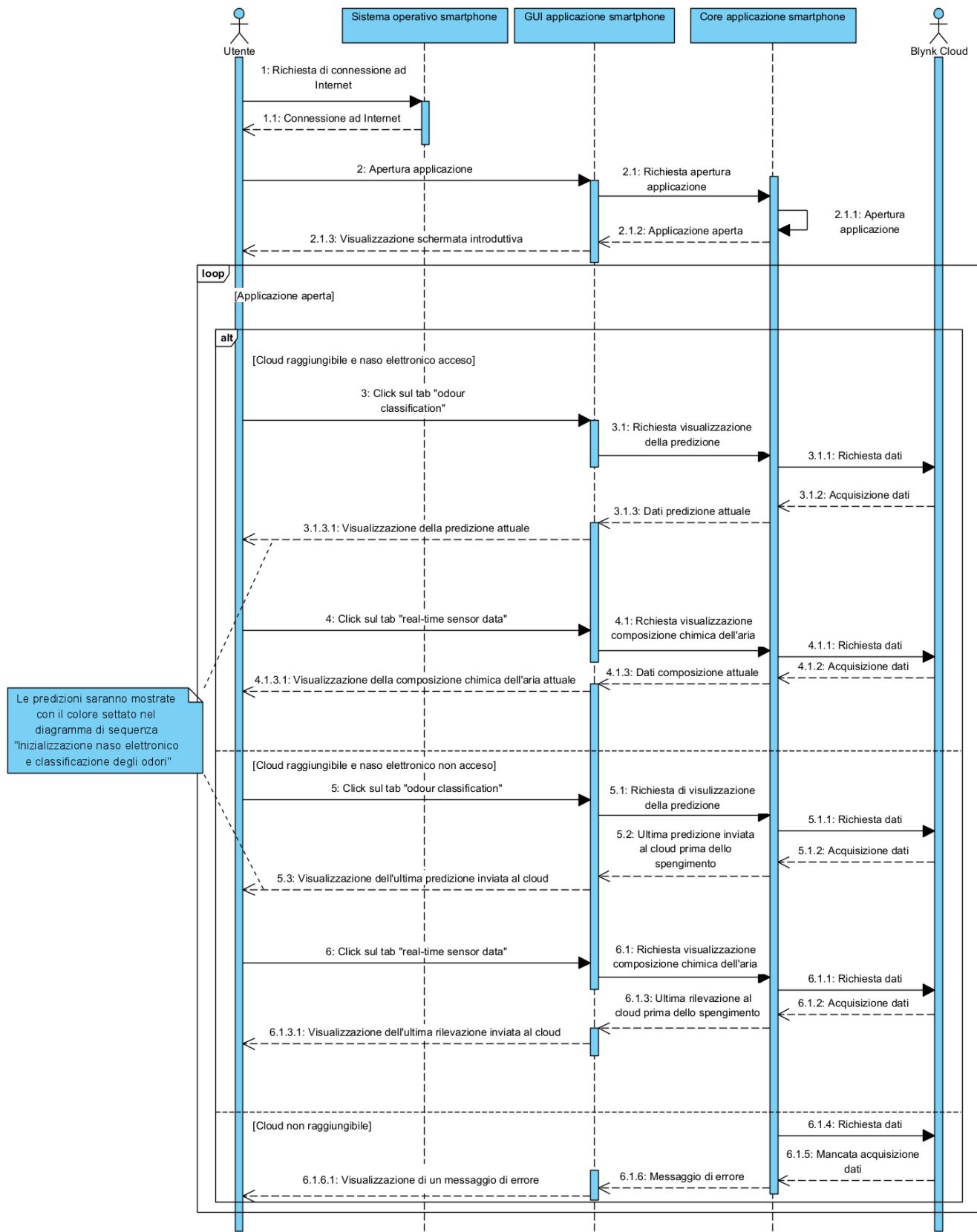


Figura 38: diagramma di sequenza "Monitoraggio con smartphone"

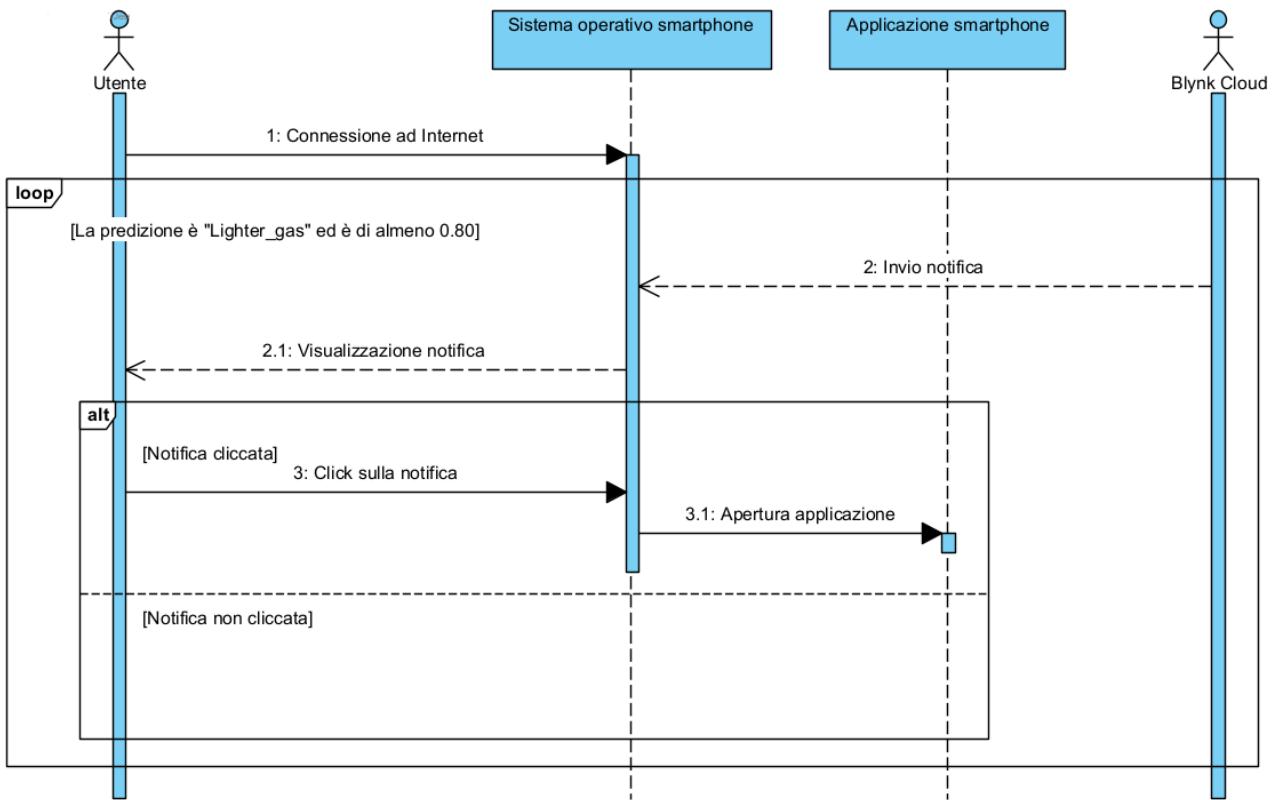


Figura 39: diagramma di sequenza “Gestione notifiche”

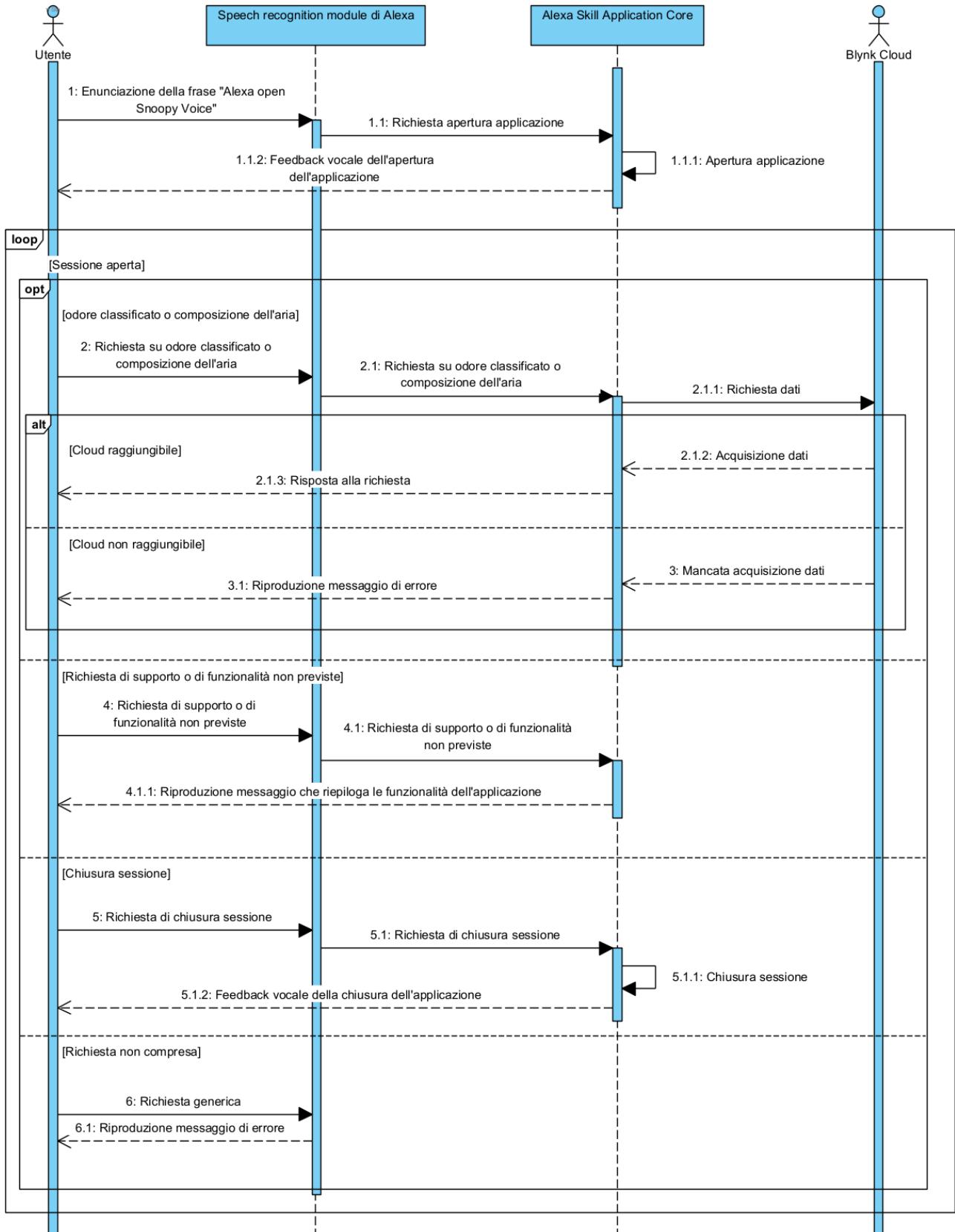


Figura 40: diagramma di sequenza “Monitoraggio con Alexa”

## Implementazione del sistema

In questo capitolo, ci concentreremo sui punti salienti dell'implementazione del codice Arduino per il nostro naso elettronico, dell'applicazione mobile e dell'applicazione per Alexa.

Iniziamo dal codice Arduino, il quale è interamente raccolto nel file Main.ino.

Una delle attività cruciali nell'implementazione del naso elettronico è l'inizializzazione dei sensori del dispositivo. Tuttavia, è importante notare che questa parte è stata già dettagliatamente descritta nel capitolo "Raccolta dei dati ed addestramento del modello". Pertanto, in questa sezione, eviteremo di ripetere tali dettagli e ci concentreremo invece su altre componenti dell'implementazione.

In particolare, è stato essenziale stabilire una connessione tra il nostro dispositivo Wio Terminal e il servizio Blynk, che ci consente di monitorare e controllare il naso elettronico tramite un app mobile disponibile sia per iOS che per Android.

Abbiamo utilizzato la libreria BlynkEdgent per semplificare questa connessione. Per iniziare, abbiamo chiamato la funzione BlynkEdgent.begin() nel metodo setup().

```
BlynkEdgent.begin(); // Initialize the connection with Blynk.
```

Figura 41: inizializzazione della connessione con Blynk

Dopodiché nel metodo loop() abbiamo usato la funzione BlynkEdgent.run() per mantenere aperta la connessione, inviare e ricevere dati.

```
BlynkEdgent.run(); // BlynkEdgent.run() is a main Blynk routine responsible  
// for keeping connection alive, sending data, receiving data
```

Figura 42: mantenimento della connessione con Blynk

Un'altra configurazione importante è quella della scheda microSD. Essa deve essere correttamente riconosciuta dal dispositivo Wio Terminal poiché memorizza le icone utilizzate per fornire un feedback visivo all'utente riguardo allo stato del suono, cioè se il suono è abilitato o disabilitato.

```
//Initialize SD card
if(!SD.begin(SDCARD_SS_PIN, SDCARD_SPI)) {
| while (1);
}
```

Figura 43: inizializzazione della MicroSD

In questa sezione di codice, stiamo usando la libreria Seeed\_FS.h per accedere alla scheda microSD. La funzione SD.begin() è utilizzata per inizializzare la comunicazione con la scheda microSD.

La condizione if (!SD.begin(...)) verifica se l'inizializzazione della scheda microSD è avvenuta correttamente. Se l'inizializzazione non ha successo, il programma entra in un loop infinito (while (1);) che impedisce al dispositivo di proseguire nell'esecuzione del codice.

Un altro passaggio essenziale è la configurazione dello schermo LCD di Wio Terminal poiché il display viene utilizzato per mostrare varie informazioni all'utente.

```
// Configure LCD
tft.begin();
tft.setRotation(3);
tft.setFreeFont(&FreeMonoBoldOblique24pt7b);
tft.setTextColor(TFT_GREEN);
tft.fillScreen(TFT_BLACK);
```

Figura 44: inizializzazione del display LCD

L'istruzione tft.begin() inizializza il display LCD e lo prepara per l'output grafico. Dopo questa istruzione, il display è pronto per ricevere dati e comandi per la visualizzazione. Le restanti istruzioni servono a settare la rotazione del display, il font e il colore di default (verde) delle stringhe che vengono visualizzate. Infine, tft.fillScreen(TFT\_BLACK) riempie l'intero schermo con il colore nero (TFT\_BLACK).

Altra inizializzazione necessaria è stata quella del 5-way switch button, ovvero il bottone tondo e blu situato nella parte frontale in basso a destra del Wio Terminal.

```
//Configure the 5-way switch button as input  
pinMode(WIO_5S_PRESS, INPUT_PULLUP);
```

Figura 45: inizializzazione del 5-way switch button

In particolare, in questo codice stiamo settando il bottone alla modalità "INPUT\_PULLUP". Questa configurazione è fondamentale per il nostro sistema in quanto il bottone sarà utilizzato per attivare o disattivare il suono in base allo stato in cui si trovava prima di essere premuto.

L'utilizzo di "INPUT\_PULLUP" implica che, di default, il pin associato al pulsante sarà in uno stato "HIGH", ma quando il pulsante viene premuto, il pin passerà a uno stato "LOW". Questo ci permette di rilevare facilmente il cambiamento di stato del pulsante e di agire di conseguenza. Quindi, se il suono era attivo prima di premere il pulsante, passerà allo stato disattivato quando verrà premuto, e viceversa.

Dopo aver completato le inizializzazioni necessarie, ci siamo concentrati sulla preparazione dei dati. Prima di avviare il classificatore, è essenziale acquisire i dati grezzi provenienti dai sensori, immagazzinarli in un buffer e normalizzarli per portarli allo stesso ordine di grandezza dei dati usati per l'addestramento del modello. Nel codice in figura 46 viene mostrato come abbiamo normalizzato i dati arrivati in tempo reale dal dispositivo Wio Terminal.

```
float mins[] = {  
    27.08, 15.23, 400.0, 0.0, 1.22, 0.23, 0.6, 0.35  
};  
float ranges[] = {  
    10.3, 56.59, 56930.0, 60000.0, 1.98, 2.73, 2.53, 2.93  
};  
for (int j = 0; j < READINGS_PER_SAMPLE; j++) {  
    temp = raw_buf[(i * READINGS_PER_SAMPLE) + j] - mins[j];  
    raw_buf[(i * READINGS_PER_SAMPLE) + j] = temp / ranges[j];  
}
```

Figura 46: normalizzazione dei dati

La normalizzazione è stata eseguita utilizzando due array: "mins" e "ranges". Questi array sono stati calcolati in precedenza tramite l'esecuzione del notebook "Data\_preprocessing.ipynb". L'array "mins" contiene i valori minimi trovati per ciascuna delle 8 diverse features dei dati. Questi valori rappresentano il punto più basso registrato per ciascuna feature durante la raccolta dei dati. L'array "ranges" contiene 8 valori, ognuno dei quali rappresenta la differenza tra il valore massimo e il valore minimo registrato per ciascuna delle 8 features. Questi valori indicano l'intervallo o la variazione nei dati per ciascuna feature.

Una volta ottenuti questi dati, i valori grezzi raccolti dai sensori vengono memorizzati in un buffer chiamato "raw\_buf". Ogni lettura del sensore viene posizionata nell'array in base alla loro sequenza. Ad esempio, la temperatura viene memorizzata in "raw\_buf" nella posizione  $(i * \text{READINGS\_PER\_SAMPLE}) + 0$ , l'umidità in  $(i * \text{READINGS\_PER\_SAMPLE}) + 1$  e così via. Dopo aver memorizzato i dati grezzi, viene eseguita la normalizzazione. Per ogni lettura del sensore, viene sottratto il valore minimo corrispondente da "mins" e il risultato viene diviso per l'intervallo corrispondente da "ranges". In questo modo, tutte le letture vengono portate a uno standard comune che faciliterà la classificazione.

Su questi dati possiamo eseguire l'inferenza, ossia il processo mediante il quale il modello di machine learning precedentemente addestrato effettua previsioni sui dati in ingresso.

Prima di tutto è necessario includere il file della libreria di Edge Impulse contenente il modello di machine learning addestrato.

```
#include "electronic_nose6_inferencing.h"
```

Figura 47: include della libreria con il modello addestrato

Vediamo ora come avviene l'inferenza.

```

ei_impulse_result_t result = {0};
err = run_classifier(&signal, &result, DEBUG_NN);
if (err != EI_IMPULSE_OK) {
    ei_printf("ERROR: Failed to run classifier (%d)\r\n", err);
    return;
}

// Find maximum prediction
for (int i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {
    if (result.classification[i].value > max_val) {
        max_val = result.classification[i].value;
        max_idx = i;
    }
}

```

Figura 48: codice che esegue l'inferenza

Qui i punti salienti:

- `ei_impulse_result_t result = {0};`: qui stiamo dichiarando una variabile chiamata `result` di tipo `ei_impulse_result_t`, che verrà utilizzata per memorizzare i risultati dell'inferenza. La variabile viene inizializzata a zero.
- `err = run_classifier(&signal, &result, DEBUG_NN);`: in questa istruzione, stiamo eseguendo il classificatore sul segnale dei dati preelaborati. La funzione `run_classifier` prende tre argomenti:
  1. `&signal`: questo è un puntatore al segnale dei dati preelaborati che verrà utilizzato per l'inferenza.
  2. `&result`: questo è un puntatore alla variabile `result` che abbiamo dichiarato in precedenza e in cui verranno memorizzati i risultati dell'inferenza.
  3. `DEBUG_NN`: questo parametro indica se vogliamo attivare la modalità di debug. Se `DEBUG_NN` è vero, verranno stampate informazioni di debug.
- `if (err != EI_IMPULSE_OK) { ... }`: qui stiamo gestendo eventuali errori nell'esecuzione del classificatore. Se la predizione non è riuscita e `err` non è uguale a `EI_IMPULSE_OK`, verrà stampato un messaggio di errore e la funzione verrà interrotta.
- `for (int i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {...}`: in questo ciclo `for`, stiamo iterando attraverso tutte le etichette di classificazione possibili. Il modello di machine learning è stato

addestrato per riconoscere diverse classi, ognuna delle quali rappresenta un odore specifico. Stiamo cercando la classe con il valore di previsione massimo tra tutte le classi. Questo ci darà la previsione finale del modello per l'istanza di dati in ingresso.

Fino a questo punto, abbiamo dedicato la nostra attenzione principalmente agli aspetti logici del sistema. Tuttavia, ora è giunto il momento di esplorare la dimensione interattiva del progetto ossia gli elementi grafici da visualizzare sullo schermo del dispositivo Wio Terminal, gli effetti sonori e l'integrazione con l'applicazione per smartphone e con quella per Alexa.

L'applicazione per smartphone è stata sviluppata sfruttando la piattaforma Blynk[8].

Blynk è una Low-Code IoT Software Platform che consente di creare applicazioni mobili o web personalizzate per controllare e monitorare dispositivi connessi. Fornisce API semplici per la condivisione di dati tra hardware e app, con il supporto di numerosi modelli di hardware, come ESP32, Arduino, Raspberry Pi e molti altri tra cui Wio Terminal. Inoltre, come già rimarcato più volte, offre l'accesso ad un cloud gratuito e molto performante.

A tal proposito Blynk offre app mobili native per iOS e Android che consentono di creare queste applicazioni personalizzate. L'app Blynk funziona in due modalità:

- modalità sviluppatore: la funzione principale della modalità sviluppatore nell'app mobile è quella di creare e modificare l'interfaccia utente per il template del dispositivo specifico.

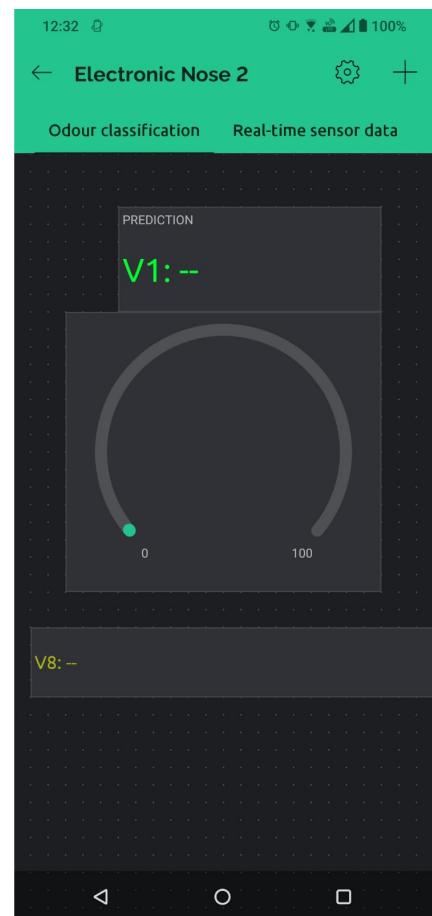


Figura 49: modalità sviluppatore app Blynk

Questo processo è possibile attraverso l'utilizzo di widget, elementi modulari dell'interfaccia utente che possono essere posizionati sulla schermata. Ogni widget svolge una funzione specifica ed ha le proprie impostazioni in base alla sua funzionalità.

- modalità utente finale: questa modalità è utilizzata dagli utenti finali che possono visualizzare e gestire le informazioni e le notifiche provenienti dai dispositivi legati ad un determinato template ma non possono modificare l'interfaccia.

Il primo passo per utilizzare i servizi di Blynk è registrarsi sulla piattaforma. Dopodiché deve essere creato un template e deve essere configurata la prima connessione con il dispositivo con cui si vuole comunicare (in questo caso Wio Terminal). In questo senso è fondamentale il ruolo svolto dalle variabili `BLYNK_TEMPLATE_ID` e `BLYNK_DEVICE_NAME` che sono utilizzate per l'autenticazione del dispositivo e devono essere settate all'interno del codice Arduino per consentire la comunicazione.

```
#define BLYNK_TEMPLATE_ID "TMPL4pdRtiE9E"  
#define BLYNK_TEMPLATE_NAME "Electronic Nose 2"
```

Figura 50: autenticazione del dispositivo

A questo punto, è possibile accedere finalmente alla modalità sviluppatore e creare un'interfaccia utente personalizzata per le nostre specifiche esigenze, utilizzando widget. I widget più comuni includono pulsanti, indicatori, grafici e display di testo.

Possiamo anche personalizzare l'aspetto e le funzionalità di ciascun widget, programmandoli per adattarli perfettamente ai nostri scopi.

Nella nostra applicazione, abbiamo utilizzato dei widget chiamati 'gauge' che ci consentono di visualizzare in tempo reale i valori rilevati dal Wio Terminal, rappresentando anche i valori limite massimi e minimi associati alla capacità dei sensori o all'unità di misura.

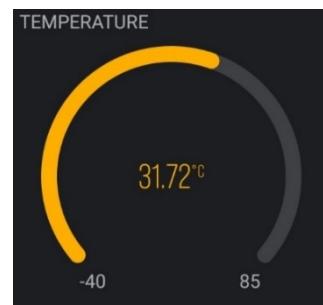


Figura 51: gauge widget

Abbiamo anche aggiunto un widget dedicato alla visualizzazione delle previsioni ottenute dal nostro modello di riconoscimento degli odori. Parlando della struttura dell'app, questa è divisa in due tab (ossia 2 diverse sezioni):

- “Odour classification”: mostra l’odore rilevato dal naso elettronico, insieme ad un widget ‘Gauge’ che indica l’accuratezza della previsione.
- “Real-time sensor data”: è dedicato alla visualizzazione dei dati in tempo reale provenienti dai sei principali componenti provenienti dai 3 sensori del Wio Terminal: temperatura, umidità, anidride carbonica, diossido di azoto, etanolo e monossido di carbonio. Abbiamo scelto di far visualizzare questi 6 componenti dell’aria e non tutti perché Blynk permette l’utilizzo gratuito di solo 10 widget per app. Tuttavia, i componenti mostrati risultano essere quelli più intuitivi e significativi per un utente medio che non ha molta dimestichezza con la chimica.



Figura 52: i due tab

I dati possono essere visualizzati attraverso i widget grazie alla configurazione dei virtual pin. I virtual pin consentono lo scambio di qualsiasi dato tra l’hardware e l’app mobile di Blynk fungendo in un certo senso da ponte.

Detto ciò, l’implementazione del codice per Wio Terminal e dell’app non poteva essere trattata separatamente. Abbiamo dovuto lavorare simultaneamente su entrambe le piattaforme per garantire coerenza e uniformità in tutte le fasi del progetto. Difatti le due piattaforme dovevano essere coordinate sia logicamente che stilisticamente.

Tant’è che un aspetto che abbiamo voluto tenere in forte considerazione, è stato quello di voler garantire un’uniformità dei colori tra le interfacce di Wio Terminal e l’app. Questo concetto non riguarda solo l’aspetto estetico, ma ha un impatto significativo anche sulla user experience complessiva.

Per cui, abbiamo scelto di utilizzare il colore verde per rappresentare una condizione normale. Questo è un principio ampiamente accettato nella progettazione dell'esperienza utente. Quando l'utente vede il verde, capisce istantaneamente che tutto sta funzionando correttamente e che non ci sono problemi. Abbiamo applicato questo principio sia su Wio Terminal che sull'app Blynk. Quando una predizione indica una situazione normale e l'anomalia è sotto una soglia prestabilita, presentiamo le informazioni in verde su entrambe le piattaforme. Questo aiuta l'utente a riconoscere rapidamente che tutto è sotto controllo.

D'altra parte, abbiamo utilizzato il colore rosso per evidenziare eventuali situazioni critiche. Quando il sistema rileva una condizione critica, come una possibile perdita di gas (ad esempio, un valore di "lighter\_gas" superiore a 0.80), il rosso viene utilizzato per attirare l'attenzione dell'utente e comunicare l'urgenza della situazione. In questo modo, l'utente comprende immediatamente che deve prestare attenzione e intraprendere le azioni necessarie.

L'uniformità dei colori tra le due interfacce, Wio Terminal e Blynk, migliora notevolmente l'usabilità complessiva del nostro sistema. Gli utenti possono passare da una piattaforma all'altra senza confusione, poiché sanno che gli stessi colori rappresentano gli stessi stati o risultati. Questa coerenza aiuta a evitare malintesi ed è essenziale per rendere l'uso del sistema intuitivo.

Nello stesso contesto, abbiamo poi deciso di utilizzare il suono come canale di comunicazione ridondante. Il suono è un elemento acustico che può richiamare l'attenzione dell'utente in modo più immediato rispetto al semplice cambiamento di colore e può evidenziare l'importanza di un evento. Pertanto, quando vengono rilevate condizioni critiche, come una potenziale perdita di gas, il sistema abilita il suono per fornire una notifica sonora, sia sul dispositivo Wio Terminal che sullo smartphone (se il telefono è in modalità silenziosa chiaramente il feedback diventa di tipo aptico).

Sul dispositivo Wio Terminal l'abilitazione o la disabilitazione del suono è controllata dall'utente mediante il pulsante 5-way switch. Sullo smartphone esiste una sezione di Blynk app in cui è possibile disattivare le notifiche.

Questo aspetto consente agli utenti flessibilità e possibilità di personalizzazione: quando il suono è abilitato, l'utente sa che, in caso di condizioni critiche, verrà avvertito immediatamente. D'altra parte, se il suono è disabilitato, l'utente non sarà disturbato da allarmi acustici, il che può essere particolarmente utile in situazioni in cui si desidera una maggiore discrezione.

La coerenza tra i colori e il suono svolge un ruolo fondamentale nell'integrazione di queste due modalità di comunicazione. Quando il sistema indica una situazione critica utilizzando il colore rosso, l'utente sa che può aspettarsi anche una notifica sonora. In tal modo, la coerenza tra il feedback visivo e quello acustico offre un'esperienza utente più informativa.

Passiamo quindi alla descrizione dell'implementazione degli aspetti appena citati.

```
tft.fillRect(TFT_BLACK);
if (result.anomaly < ANOMALY_THRESHOLD && result.classification[max_idx].label != "lighter_gas") {
    tft.setFreeFont(&FreeMonoBoldOblique24pt7b);
    tft.setTextColor(TFT_GREEN);
    tft.drawString(result.classification[max_idx].label, 12, 60);
    sprintf(str_buf, "%.3f", max_val);
    tft.drawString(str_buf, 60, 120);
    sound_on = sound(sound_on);
    Blynk.setProperty(V1, "color", "#01DF01"); //green HTML color code
    Blynk.virtualWrite(V1, result.classification[max_idx].label);
    Blynk.setProperty(V2, "color", "#23C48E"); //green HTML color code
    Blynk.virtualWrite(V2, result.classification[max_idx].value * 100);
    Blynk.virtualWrite(V8, " ");
}
```

Figura 53: caso 1 - predizione non anomala e diversa da "lighter\_gas"

La condizione mostrata nella figura 53 è progettata per gestire le situazioni in cui il sistema non considera la rilevazione anomala e la predizione non è "lighter\_gas". In questo caso la predizione viene visualizzata con il font "FreeMonoBoldOblique24pt7b" (font grassetto obliquo e di dimensione 24pt) e impostando il colore del testo su verde

(TFT\_GREEN). Oltre alla stringa della classe predetta viene mostrata anche l'accuratezza della predizione (max\_val) che abbiamo formattato come stringa con tre cifre decimali ("%.3f") e viene visualizzata nella posizione (60, 120) del display anch'essa in verde.

Inoltre, vengono passate le informazioni rilevanti a Blynk per consentire il monitoraggio remoto. Abbiamo utilizzato Blynk.virtualWrite per inviare il tipo di odore (result.classification[max\_idx].label) e il valore dell'accuratezza della classificazione

(result.classification[max\_idx].value \* 100) ai rispettivi widget nell'app Blynk (quelli con virtual pin V1 e V2). Il widget con virtual pin V1 è un "value display", il widget con virtual pin V2 è un "gauge".

Per implementare la corrispondenza di colori tra Wio Terminal e l'app Blynk, i colori per i widget con virtual pin V1 e V2 vengono settati a verde attraverso Blynk.setProperty(V1, "color", "#01DF01") e Blynk.setProperty(V2, "color", "#23C48E"). Da notare che per i due widget la tonalità di verde usata è leggermente diversa, per questo il codice HTML rappresentativo dei colori è differente.

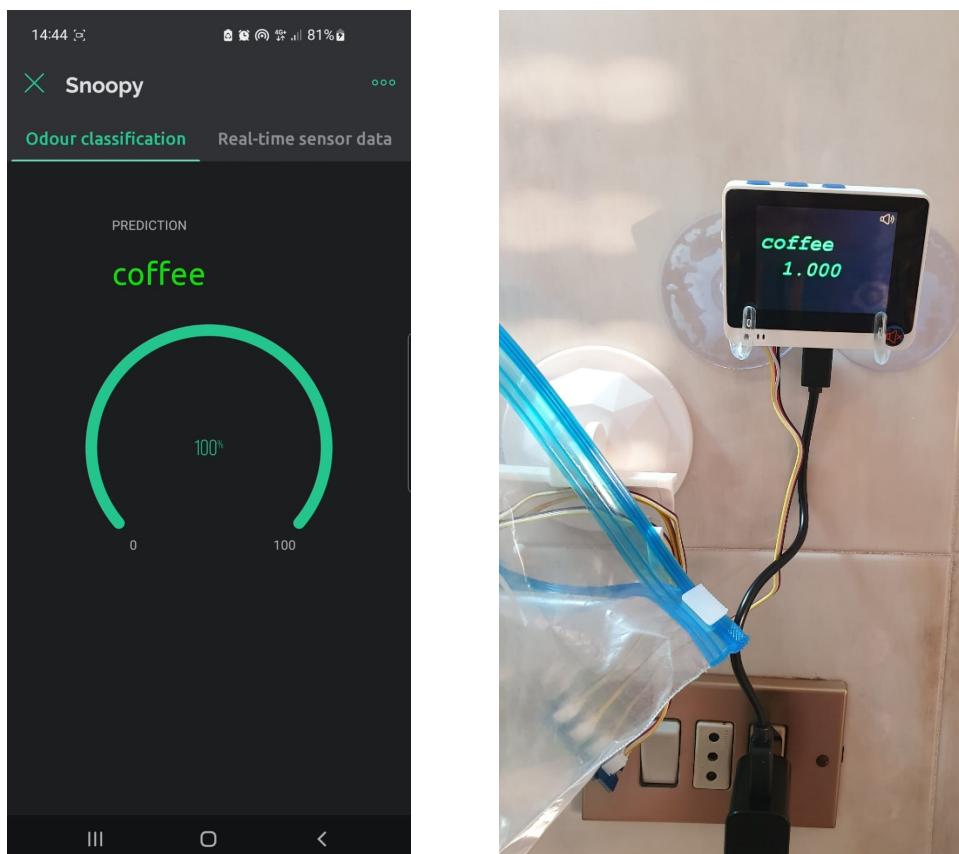


Figura 54: schermate di Wio Terminal ed app quando la predizione non è anomala e è diversa da "lighter\_gas"

Infine, per dare la possibilità all'utente di attivare o disattivare l'audio in qualsiasi momento anche quando come in questo caso non vengono rilevate situazioni critiche, è utilizzata la variabile sound\_on che è collegata alla funzione sound() che gestisce e permette di memorizzare lo stato attuale dell'audio.

```

int sound(int sound_on){
    if (digitalRead(WIO_5S_PRESS) == LOW){
        if (sound_on == 1) {
            sound_on = 0;
            drawImage<uint8_t>("no-sound_neg.bmp", 280, 0);
        }else{
            sound_on = 1;
            drawImage<uint8_t>("volume_neg.bmp", 280, 0);
        }
    }else{
        if (sound_on == 1) {
            drawImage<uint8_t>("volume_neg.bmp", 280, 0);
        }else{
            drawImage<uint8_t>("no-sound_neg.bmp", 280, 0);
        }
    }
    return sound_on;
}

```

Figura 55: funzione sound

La funzione è costituita da una serie di condizioni if...then...else per gestire varie situazioni:

- if (digitalRead(WIO\_5S\_PRESS) == LOW) {...}: questa condizione verifica se il 5-way switch button di Wio Terminal è premuto. Dentro il blocco if, la funzione verifica lo stato corrente di sound\_on. Se sound\_on è 1 (suono abilitato), allora cambia il suo stato a 0 (suono disabilitato) e cambia l'immagine sull'interfaccia grafica di Wio Terminal per mostrare l'icona "no sound\_neg"  nell'angolo in alto a destra. In alternativa, se sound\_on è 0 (suono disabilitato), lo cambia a 1 (suono abilitato) e cambia l'immagine per mostrare l'icona "volume\_neg"  nell'angolo in alto a destra in sostituzione dell'icona dal significato opposto.

- Se il pulsante non è premuto, la funzione verifica ancora una volta lo stato di sound\_on e imposta l'immagine in base allo stato corrente, senza cambiare lo stato.

Per semplificare l'interazione con il pulsante 5-way switch button presente su Wio Terminal, abbiamo adottato un approccio pratico. Abbiamo applicato sopra il pulsante uno sticker con un'immagine iconica , simile all'icona utilizzata sui telecomandi televisivi per il controllo dell'audio. Inoltre, come si può facilmente notare questa icona è la stessa utilizzata sul display di Wio Terminal per rappresentare lo stato di suono disabilitato, proprio per creare una coerenza nella mente dell'utente che riesce così a riconoscere quello che sta facendo, come lo sta facendo e in che stato si trova il sistema.

In figura 56 sono raffigurate due schermate del display di Wio Terminal rappresentanti lo stato in cui il suono è abilitato e quello in cui invece risulta disabilitato.

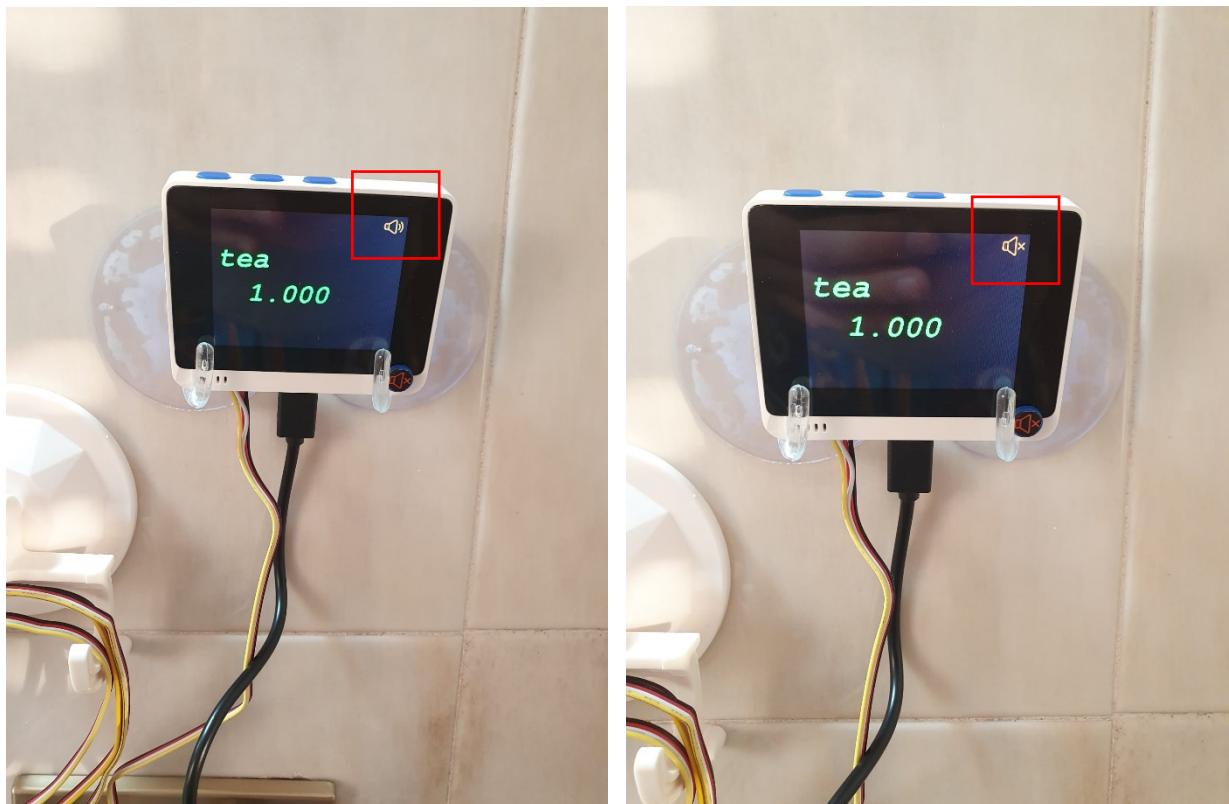


Figura 56: a sinistra l'audio è abilitato, a destra disabilitato

```

else if (result.anomaly < ANOMALY_THRESHOLD
&& result.classification[max_idx].label == "lighter_gas"
&& result.classification[max_idx].value < 0.80){
    tft.setFreeFont(&FreeMonoBoldOblique24pt7b);
    tft.setTextColor(TFT_GREEN);
    tft.drawString(result.classification[max_idx].label, 12, 60);
    sprintf(str_buf, "%.3f", max_val);
    tft.drawString(str_buf, 60, 120);
    sound_on = sound(sound_on);
    Blynk.setProperty(V1, "color", "#01DF01"); // green HTML color code
    Blynk.virtualWrite(V1, result.classification[max_idx].label);
    Blynk.setProperty(V2, "color", "#23C48E"); //green HTML color code
    Blynk.virtualWrite(V2, result.classification[max_idx].value * 100);
    Blynk.virtualWrite(V8, " ");
}

```

Figura 57: caso 2 - predizione non anomala, uguale a "lighter\_gas"  
ma con predizione con valore inferiore a 0.80

La seconda condizione (mostrata in figura 57) è progettata per gestire una situazione in cui il sistema rileva un tipo di odore che non è anomalo e che corrisponde a "lighter\_gas" con un accuracy della predizione inferiore a 0,80. Per indicare che la situazione è ancora sotto controllo perché, anche se l'odore rilevato è "lighter\_gas", non supera ancora una soglia critica di certezza nell'accuratezza, abbiamo deciso di comunque rappresentare la predizione con il colore verde.

Ancora una volta su Wio Terminal, utilizziamo il tipo di carattere "FreeMonoBoldOblique24pt7b" e impostiamo il colore del testo su verde (TFT\_GREEN). La corrispondenza dei colori tra Wio Terminal e l'app Blynk e l'invio dei dati sul cloud, avvengono allo stesso modo della condizione 1. Anche qui la variabile sound\_on è coinvolta nella gestione dell'audio.



Figura 58: "lighter\_gas" in verde su Wio Terminal.

```

    else if (result.anomaly < ANOMALY_THRESHOLD
    && result.classification[max_idx].label == "lighter_gas"
    && result.classification[max_idx].value >= 0.80){
        tft.setFreeFont(&FreeMonoBoldOblique24pt7b);
        tft.setTextColor(TFT_RED);
        sound_on = sound(sound_on);
        tft.drawString(result.classification[max_idx].label, 12, 60);
        sprintf(str_buf, "%.3f", max_val);
        tft.drawString(str_buf, 60, 120);
        tft.setFreeFont(&FreeSansOblique12pt7b);
        tft.setTextColor(TFT_WHITE);
        tft.drawString("Attention, there could be", 25, 180);
        tft.drawString("a potential gas leak.", 45, 210);
        if (sound_on == 1) {
            beep(1000);
        }
    }
}

```

Figura 59: caso 3 - predizione non anomala, uguale a "lighter\_gas" e con predizione con valore di almeno 0.80

La terza condizione è progettata per affrontare una situazione in cui il sistema rileva un tipo di odore non anomalo corrispondente a "lighter\_gas" con un'accuratezza sulla predizione maggiore o uguale a 0,80. Il verificarsi di questa condizione potrebbe significare che il livello di lighter\_gas nell'aria ha superato una soglia critica e che potenzialmente potrebbe essere una perdita di gas. Quando questa condizione è soddisfatta il testo su Wio Terminal è visualizzato con il solito font ma di colore rosso (TFT\_RED) per indicare la criticità della situazione. Anche il valore dell'accuratezza della predizione massima ottenuta (max\_val) viene visualizzato in rosso.

In questo caso viene anche mostrato all'utente un messaggio di possibile criticità che afferma: "Attention, there could be a potential gas leak." Per quanto riguarda invece il font di questo testo, abbiamo utilizzato un tipo di carattere differente, "FreeSansOblique12pt7b", mentre il colore del testo è stato impostato su bianco (TFT\_WHITE).

Anche in questo caso il sistema utilizza Blynk.virtualWrite per inviare il tipo di odore rilevato (result.classification[max\_idx].label) e il valore dell'accuratezza associata (result.classification[max\_idx].value \* 100) ai widget corrispondenti (V1 e V2). In aggiunta, rispetto alle 2 condizioni precedenti, viene inviato il messaggio descrittivo "Attention, there could

be a potential gas leak." attraverso il virtual pin V8 per fornire ulteriori dettagli sulla situazione.

Ancora una volta, il sistema è stato programmato per mantenere la coerenza visiva tra Wio Terminal e Blynk quindi i widget con virtual pin V1 e V2 vengono settati a rosso con `Blynk.setProperty(V1, "color", "#FF0000")` e `Blynk.setProperty(V2, "color", "#D3435C")`. Anche in questo caso le tonalità di rosso sono leggermente differenti per cui varia l'HTML color code utilizzato nei due metodi.

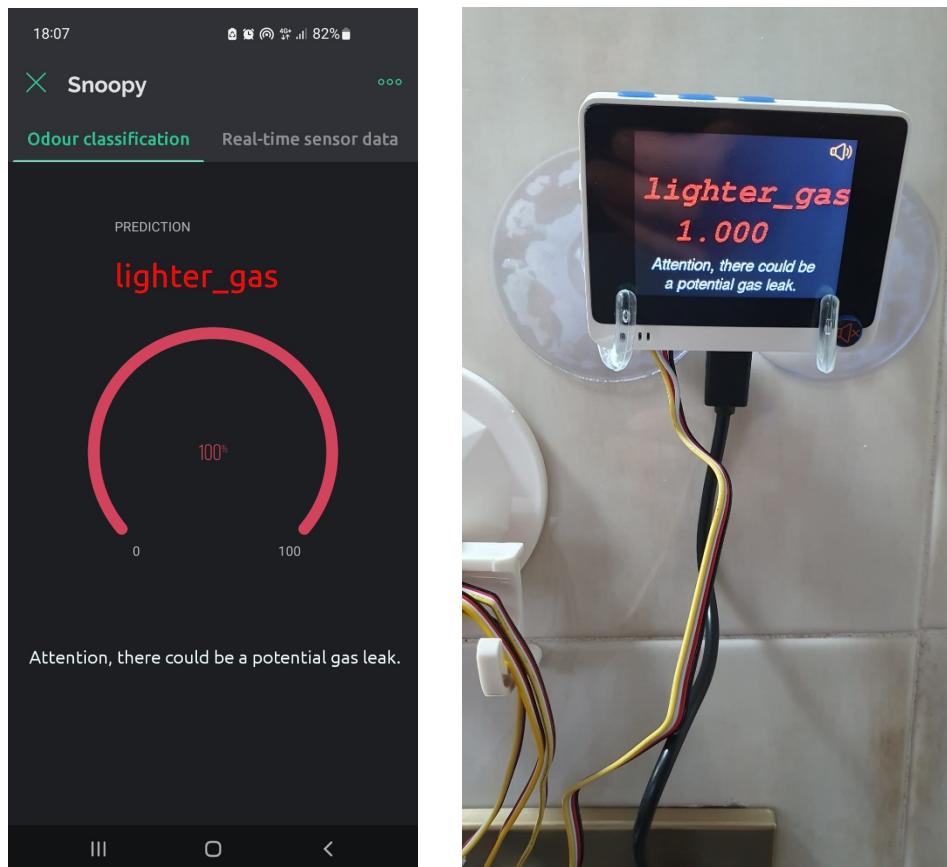


Figura 60: schermate di Wio Terminal ed app quando la predizione non è anomala, è "lighter\_gas" ed ha un valore della predizione massima di almeno 0.80

Anche all'interno di questa condizione viene verificato lo stato della variabile `sound_on`. Se `sound_on` è uguale a 1, ovvero se il suono è abilitato, allora viene attivato un allarme sonoro utilizzando la funzione `beep()`.

```

void beep(unsigned int hold = 1000) {
{
    analogWrite(WIO_BUZZER, 128);
    delay(hold);
    analogWrite(WIO_BUZZER, 0);
}

```

Figura 61: funzione beep()

La funzione beep() mostrata in figura 61 è progettata per emettere un suono simile ad un allarme. Prende un parametro opzionale hold che rappresenta la durata in millisecondi del suono. Essa è strutturata su 3 linee di codice:

- analogWrite(WIO\_BUZZER, 128): questa istruzione utilizza la funzione analogWrite() per attivare il buzzer presente su Wio Terminal. Imposta il valore a 128, il che significa che il buzzer produrrà un suono a metà intensità.
- delay(hold): serve a settare la durata del suono dell'allarme.
- analogWrite(WIO\_BUZZER, 0): questa istruzione serve ad interrompere il suono.

In sostanza, quando la funzione beep() viene chiamata, il buzzer emetterà un suono per la durata specificata da hold, poi si fermerà per 1 secondo e ripartirà ripetendosi con questo stesso andamento per tutta la durata di tempo in cui Wio Terminal rileva "lighter\_gas" nell'aria con un'accuratezza superiore a 0.80.

Ma non è tutto, perché sempre in relazione a questa terza condizione, viene inviata una notifica al dispositivo mobile dell'utente.

Questa funzione è particolarmente utile quando l'utente è fuori casa e non può supervisionare direttamente la situazione. In questo modo, l'utente rimane costantemente informato e può intraprendere le azioni necessarie, come contattare immediatamente le autorità competenti o verificare personalmente la situazione.

Abbiamo potuto realizzare questa funzionalità grazie a Blynk Console che permette di creare eventi che vengono utilizzati dalle applicazioni Blynk per tracciare, registrare e gestire eventi importanti che si verificano sui dispositivi connessi. Gli eventi vengono anche utilizzati per le notifiche, che possono essere inviate via email, recapitate come notifiche push sullo smartphone dell'utente o inviate come SMS.

Noi abbiamo deciso di inviarle come notifiche push.

Nel nostro caso l'evento chiaramente si attiva quando i nostri sensori rilevano una quantità di gas superiore alla soglia critica. Nella creazione dell'evento sono state settate varie cose come il nome dell'evento, il codice dell'evento da usare su Arduino per creare il collegamento, il tipo di suono della notifica e il testo da mostrare all'utente quando visualizza la notifica.

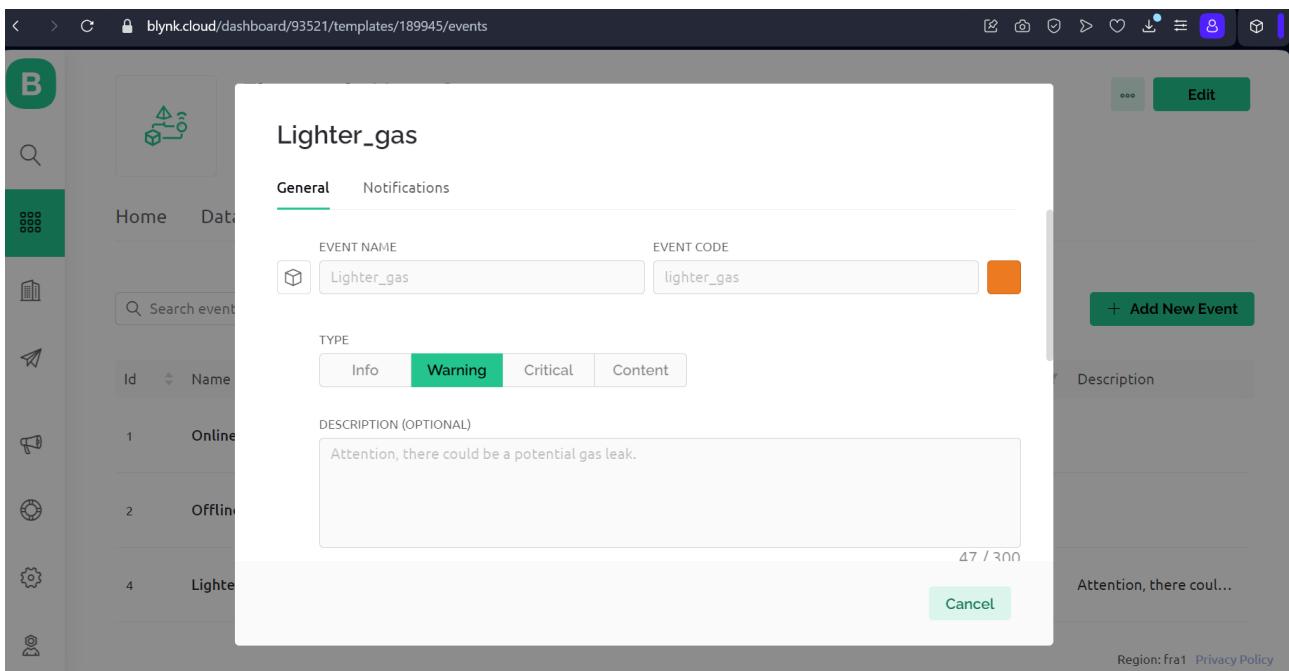


Figura 62: creazione eventi con Blynk Console

Nel codice Arduino l'invio della notifica viene settato in questo modo:

```
Blynk.logEvent("lighter_gas");
```

Figura 63: invio notifica

Nella figura 62 è mostrato come appare la notifica (anche dopo la sua apertura) sullo smartphone.

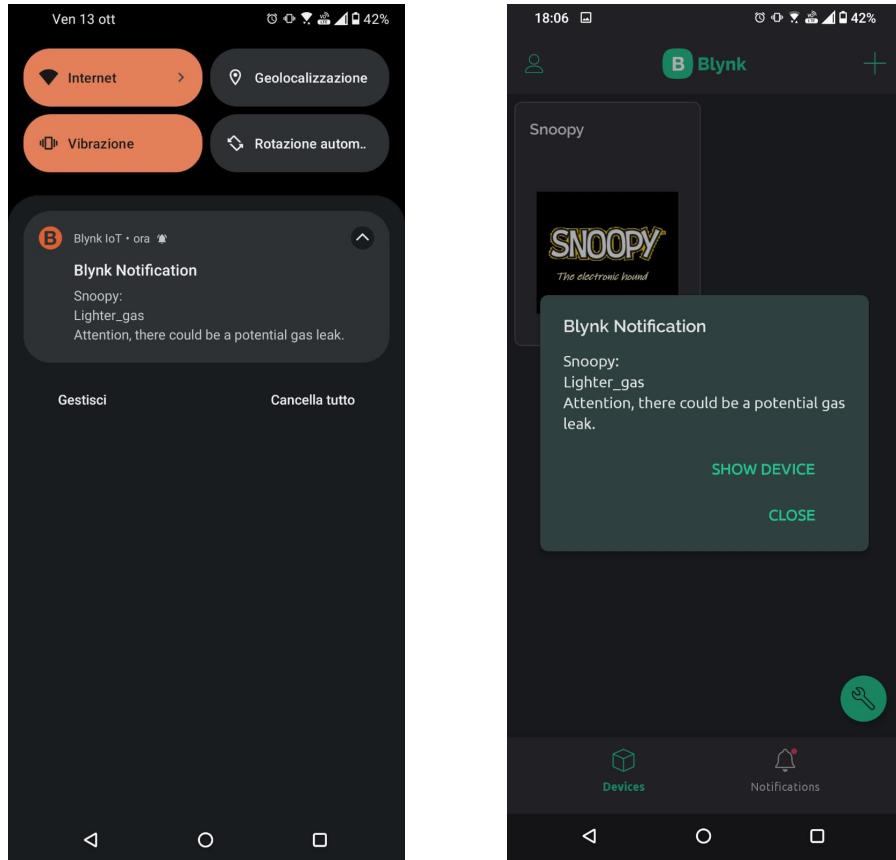


Figura 64: notifica

A questo punto è rimasta da analizzare la quarta ed ultima condizione ossia quella in cui la predizione è individuata come anomala.

```

} else {
    tft.setFreeFont(&FreeMonoBoldOblique24pt7b);
    tft.setTextColor(TFT_GREEN);
    tft.drawString("Unknown", 20, 60);
    sprintf(str_buf, "%.3f", result.anomaly);
    tft.drawString(str_buf, 60, 120);
    sound_on = sound(sound_on);
}

```

Figura 65: caso 4 – predizione anomala

In questa situazione, il nostro naso elettronico non è in grado di distinguere chiaramente quale sia l'odore nell'aria poiché la rilevazione risulta essere anomala. Di conseguenza, il sistema non può prendere una decisione definitiva e chiara sulla natura dell'odore. Come risposta

a questa incertezza, mostriamo la stringa "Unknown" sul display del Wio Terminal in verde. Questo indica che, nonostante l'anomalia nella predizione, il sistema è in uno stato stabile e non è stata rilevata una situazione critica. Per quanto riguarda invece il font, è lo stesso delle prime 3 condizioni e il testo è visualizzato in verde anche sull'applicazione.

Come abbiamo già detto l'app è organizzata in 2 tab: la schermata denominata "Odour classification" e quella denominata "Real-time sensor data". Abbiamo già ampiamente descritto come avviene il popolamento della prima schermata. Ora resta che mostrare il codice di come abbiamo popolato i 6 gauge visualizzati nella seconda sezione.

```
Blynk.virtualWrite(v0, bme_temp);
Blynk.virtualWrite(v3, bme_hum);
Blynk.virtualWrite(v4, sgp_co2);
Blynk.virtualWrite(v5, gm_no2_v);
Blynk.virtualWrite(v6, gm_eth_v);
Blynk.virtualWrite(v7, gm_co_v);
```

Figura 66: invio sul cloud dei valori chimici dell'aria

Come è possibile osservare dalla figura 66, per inviare i dati dei sei componenti rilevati (temperatura, umidità, anidride carbonica, diossido di azoto, etanolo e monossido di carbonio) in tempo reale dai nostri 3 sensori, utilizziamo anche in questo caso la funzione `Blynk.virtualWrite()` passando come parametri il virtual pin e la variabile che contiene i valori richiesti (le abbiamo descritte nel capitolo "Raccolta dati ed addestramento del modello").

La barra progressiva all'interno di ciascun gauge si muove in tempo reale per indicare la variazione dei dati rilevati da ciascun sensore. Questo permette agli utenti di vedere immediatamente se un certo parametro è stabile o sta subendo delle variazioni.

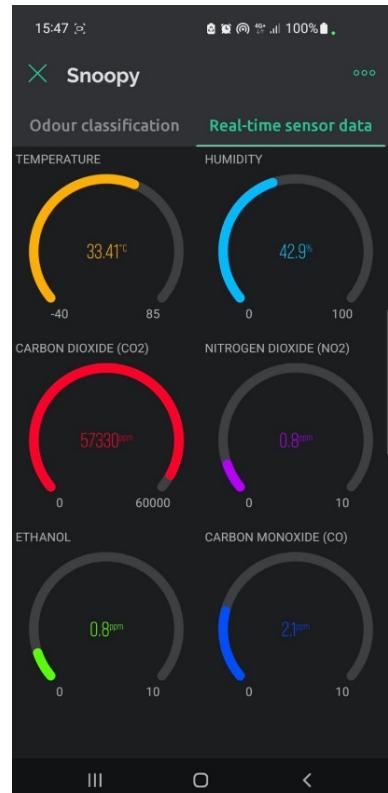


Figura 67: tab "Real-time sensor data"

Dopo aver completato la realizzazione del nostro naso elettronico e dell'app Blynk associata, abbiamo sentito il desiderio di estendere il nostro progetto aggiungendo una nuova e intrigante funzionalità. Il nostro obiettivo era quello di rendere l'interazione con il nostro naso elettronico più coinvolgente ed utile per gli utenti. Volevamo creare un meccanismo che consentisse di ottenere informazioni dal sistema anche in situazioni in cui non si avesse immediatamente accesso allo smartphone o ci si trovasse a una certa distanza dal naso elettronico. La soluzione che ci è venuta in mente è stata quella di implementare un'applicazione basata sulla modalità di interazione vocale, permettendo all'utente di porre domande e ricevere risposte utili basate sui dati raccolti dai sensori.

Ormai le nostre case sono sempre più "smart" e la presenza di dispositivi come Amazon Echo o Google Hub sono la normalità. Da qui l'idea di sviluppare una Skill di Alexa, ossia un'applicazione con comandi vocali che sfrutta l'assistente vocale Alexa.

Le Skill consentono ad Alexa di eseguire una vasta gamma di azioni e fornire informazioni in base alle richieste degli utenti. Possono essere sviluppate per ampliare le capacità di Alexa e personalizzarle per rispondere a esigenze specifiche degli utenti o per interagire con servizi e dispositivi esterni.

Per implementare l'applicazione abbiamo utilizzato la piattaforma Voiceflow[9].

Voiceflow è una piattaforma di sviluppo di voice user interface (VUI) che consente agli sviluppatori di creare applicazioni vocali interattive senza richiedere conoscenze avanzate di programmazione. Questa piattaforma offre un ambiente visuale basato su cloud che semplifica la progettazione, la creazione e la gestione di conversazioni. Essa consente agli sviluppatori di progettare flussi di conversazione, definire



Figura 68: dispositivi Amazon Echo

le risposte vocali e gestire l'interazione con gli utenti attraverso comandi vocali.

Una volta completato il processo di sviluppo in Voiceflow, è possibile collegare l'assistente vocale Alexa all'Alexa Skill creata e consentire agli utenti di interagire con esso attraverso comandi vocali su dispositivi compatibili con Alexa, come gli altoparlanti Echo.

Per la creazione di Skill di Alexa sulla piattaforma Voiceflow viene utilizzata una programmazione a blocchi. La prima operazione per la creazione dell'applicazione è stata quella di assegnare un nome al nostro assistente vocale, e abbiamo scelto "Snoopy Voice". Per invocare l'apertura dell'applicazione su un dispositivo Alexa, gli utenti devono pronunciare la frase "Alexa, open Snoopy Voice".

Nella figura 69 è possibile osservare la struttura dell'assistente vocale nel suo complesso.

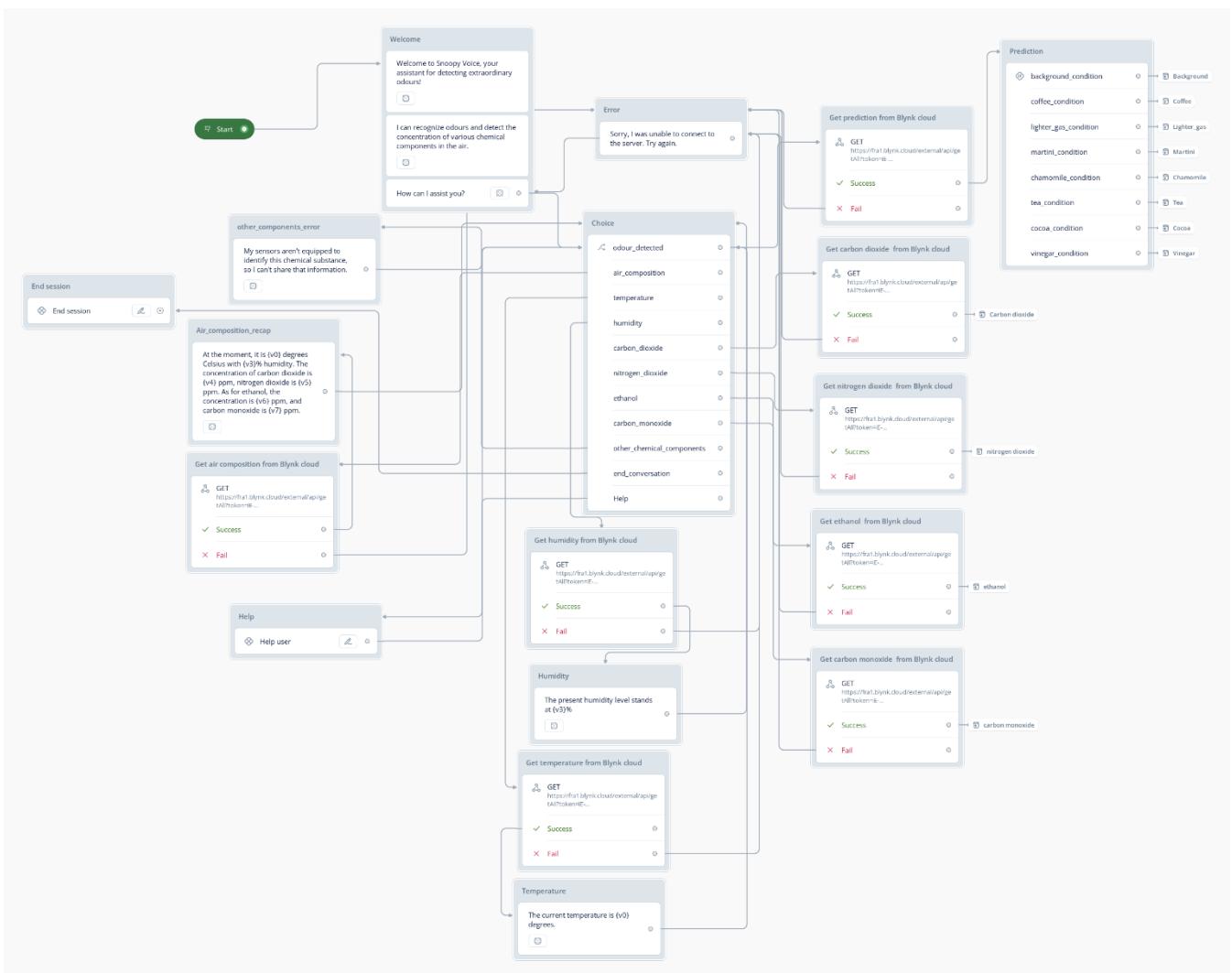


Figura 69: struttura di Snoopy Voice

Quando l'assistente viene invocato, esegue una sequenza di azioni predefinite. Innanzitutto, saluta l'utente con un messaggio di benvenuto. Successivamente, informa l'utente sulle sue capacità e chiede se può essere d'aiuto, rimanendo in ascolto di una possibile risposta dell'utente.

Queste tre espressioni dell'assistente vocale sono implementate utilizzando blocchi di tipo "speak". Ogni blocco "speak" richiede di fornire un testo da riprodurre vocalmente quando il flusso degli eventi raggiunge quel punto. In particolare, il primo blocco "speak" contiene il testo "Welcome to Snoopy Voice, your assistant for detecting extraordinary odors", il secondo contiene "I can recognize odors and detect the concentration of various chemical components in the air", e il terzo contiene "How can I assist you?". Tuttavia, per rendere l'assistente vocale più naturale e non monotono, abbiamo fornito diverse varianti di ciascuna frase in modo che l'assistente possa dire cose differenti ma mantenendo comunque lo stesso significato.

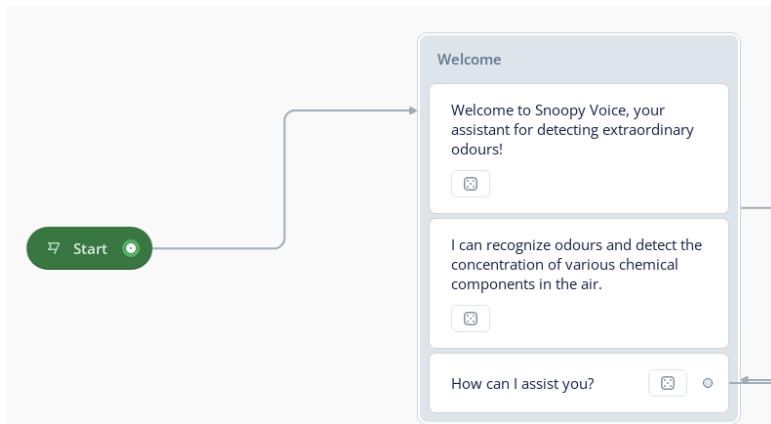


Figura 70: blocco di benvenuto

Un elemento cruciale dell'applicazione è il grande blocco chiamato "Choice". Questo blocco è di tipo "choice" il che significa che l'assistente arrivato in questo punto resterà in ascolto in attesa che l'utente pronunci utteranze associate alle diverse possibili scelte. Esso contiene 11 sotto-blocchi, ciascuno dei quali rappresenta una possibile scelta che l'utente può fare pronunciando frasi specifiche che noi abbiamo dovuto stabilire. Gli utenti non sono limitati a dover pronunciare esattamente le frasi inserite manualmente da noi, perché

Voiceflow offre la possibilità di addestrare un modello di Natural Language Understanding (NLU) che arricchisce le possibili utteranze che noi abbiamo stabilito. L'assistente sarà quindi in grado di fornire una risposta adeguata anche a frasi non precedentemente specificate.

Le 11 possibili opzioni considerate nel blocco "Choice" sono:

1. Chiedere quale odore è stato rilevato nell'aria.
  2. Chiedere qual è la composizione chimica dell'aria in questo momento.
  3. Chiedere qual è la temperatura corrente.
  4. Chiedere qual è la percentuale di umidità nell'ambiente.
  5. Chiedere qual è la concentrazione di anidride carbonica nell'aria.
  6. Chiedere qual è la concentrazione di diossido di azoto nell'aria.
  7. Chiedere qual è la concentrazione di etanolo nell'aria.
  8. Chiedere qual è la concentrazione di monossido di carbonio nell'aria.
  9. Chiedere quale è la concentrazione di un qualsiasi altro componente chimico non rilevabile dai nostri sensori.
10. Chiedere di terminare la conversazione.
11. Chiedere supporto tecnico.

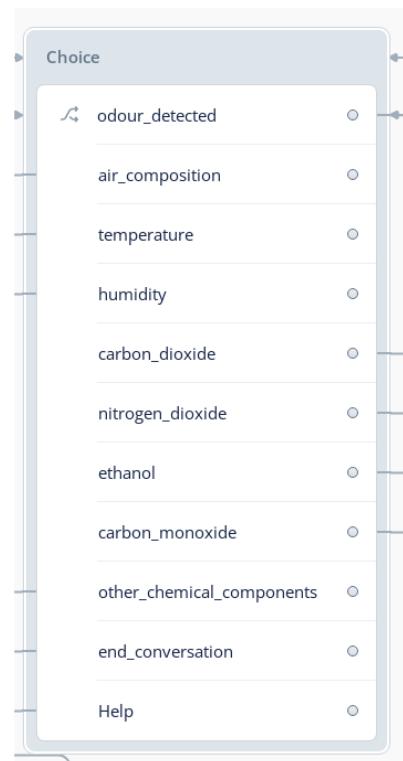


Figura 71: blocco Choice

Espletata una qualsiasi di queste richieste (tranne la 10), il flusso è ricondotto sul blocco "Choice" cosicché l'utente può effettuare nuove richieste senza che venga interrotta la sessione.

A seconda dell'utteranza pronunciata dall'utente, "Snoopy Voice" selezionerà l'azione corrispondente. Se la richiesta dell'utente rientra nelle opzioni da 1 a 8, l'assistente dovrà effettuare una chiamata GET al cloud di Blynk (a cui il nostro naso elettronico invia costantemente in

tempo reale i dati dei sensori) per recuperare le informazioni specifiche richieste dall'utente.

Tuttavia, nei casi 9, 10 e 11, l'assistente non effettuerà alcuna richiesta al cloud. Di conseguenza, se la richiesta dell'utente riguarda un componente chimico non rilevabile dai nostri sensori (9), l'assistente si sposterà in un nuovo blocco di tipo "speak" chiamato "other\_components\_error" e comunicherà all'utente verbalmente un messaggio del tipo: "My sensors aren't equipped to identify this chemical substance, so I can't share that information."

Se la richiesta dell'utente è quella di terminare la conversazione (10), l'assistente chiuderà la sessione corrente spostandosi su un particolare tipo di blocco predefinito chiamato "exit" per avvisare l'utente che la sessione è stata chiusa con successo.

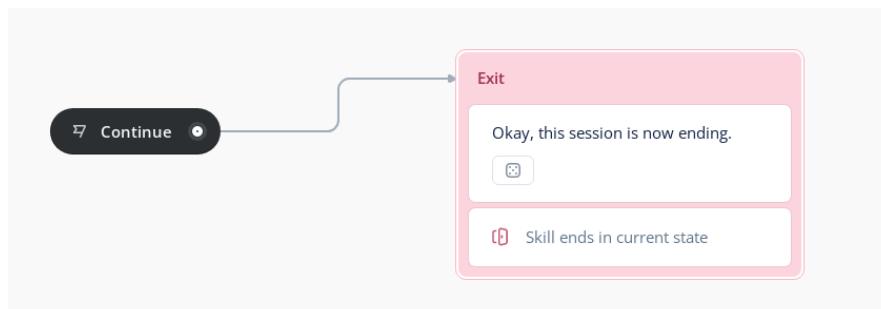


Figura 72: blocco Exit

Se l'utente richiede supporto tecnico (11), l'assistente risponderà verbalmente ricapitolando tutte le funzioni che è in grado di eseguire.

Nei casi da 1 a 8, la situazione diventa leggermente più complessa. Come già menzionato in precedenza, l'assistente deve prima recuperare le informazioni memorizzate nel cloud. Questi dati sono salvati sul server sotto forma di valori associati ai virtual pin numerati da V0 a V7.

In questo senso Voiceflow offre la possibilità di creare dei blocchi di tipo "API" che consentono di effettuare richieste HTTP. Quando effettuiamo una chiamata GET al server, otteniamo in risposta un file JSON contenente diverse chiavi, tra cui quelle da V0 a V7, che rappresentano i virtual pin con i relativi valori.

Per utilizzare queste informazioni all'interno delle frasi che desideriamo far pronunciare all'assistente, dobbiamo creare delle variabili in cui inserire i valori ottenuti dalle chiamate al server. Abbiamo scelto di denominare queste variabili come i virtual pin, quindi per accedere a queste informazioni all'interno delle frasi dell'assistente, utilizziamo la seguente sintassi: "{Vi}", dove "i" varia da 0 a 7. Le parentesi graffe indicano l'intenzione di richiamare una variabile. In questo modo, possiamo inserire dinamicamente i valori dei virtual pin nelle risposte dell'assistente, consentendo agli utenti di ottenere informazioni aggiornate in tempo reale direttamente dal nostro naso elettronico.

Per quanto riguarda il caso 1 "odour\_detected", esso conduce ad un blocco di tipo "condition" chiamato "Prediction". Questo blocco verifica il valore contenuto all'interno della variabile vlottenuto della chiamata al server. Se è uguale a "background" parte il blocco "Background", se è uguale a "coffee" parte il blocco "Coffee" e così via.

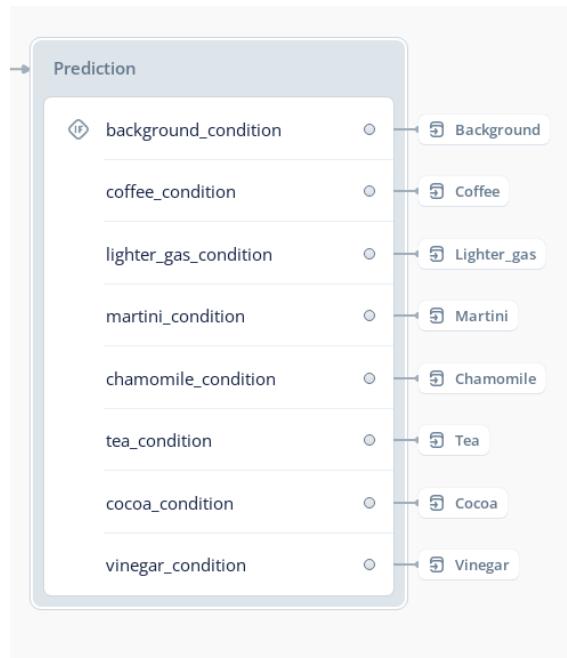


Figura 73: blocco Prediction

Gli otto possibili casi conducono a blocchi diversi che per consentire un'organizzazione migliore sono stati posti in un altro topic. In Voiceflow, un topic rappresenta un argomento o un sottoinsieme specifico di interazioni e comandi che l'assistente vocale può gestire o discutere.

con l'utente. Questi aiutano a suddividere il flusso di conversazione in segmenti coerenti.

Nella figura 74 sono mostrati i blocchi nel topic "Possible\_predictions" che rappresentano le 8 possibili predizioni di odori che l'assistente può pronunciare in output. Per ciascun blocco, abbiamo creato una vasta gamma di possibili risposte (simili a quelle mostrate in figura) pronunciabili dall'assistente in modo da rendere l'interazione con l'utente più dinamica e interessante.

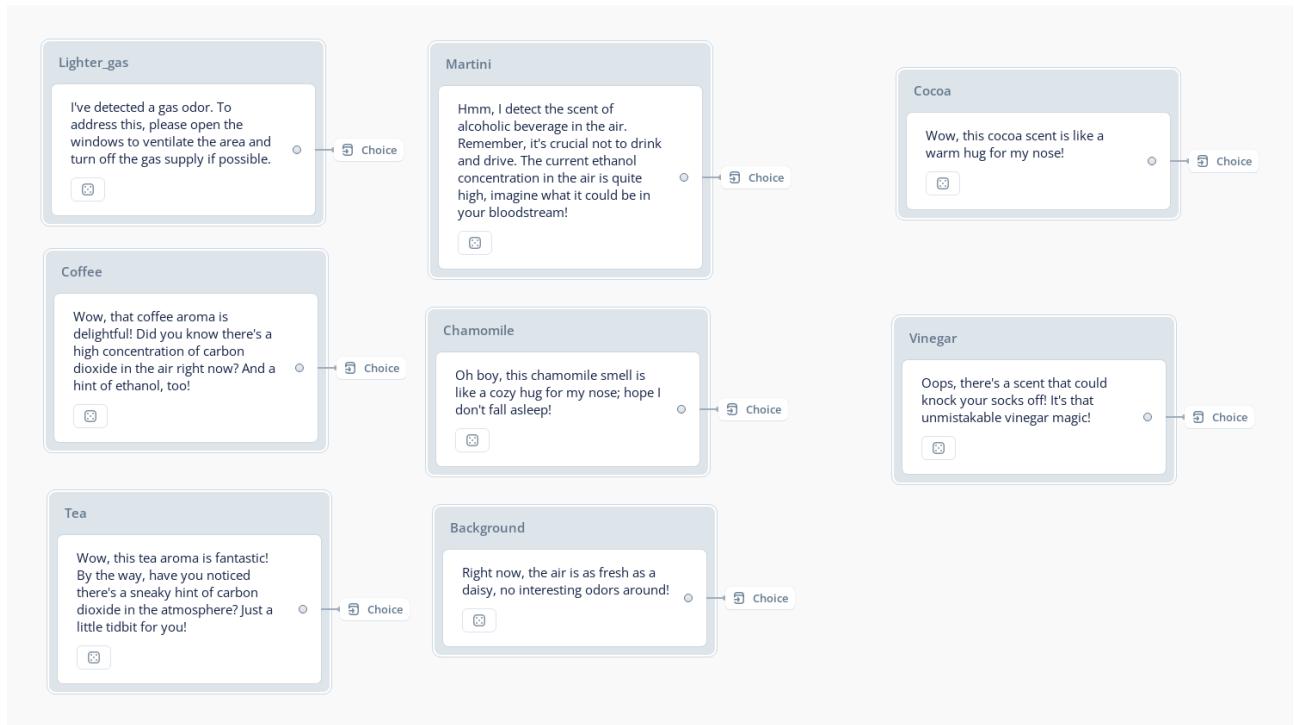


Figura 74: topic Possible\_predictions

Quando riceviamo richieste che corrispondono ai casi da 3 a 8, l'assistente segue un approccio simile. La principale differenza risiede nel fatto che non utilizziamo un blocco condizionale, ma ogni choice chiama direttamente un blocco "speak" specifico per il singolo componente (temperatura, umidità, anidride carbonica, diossido di azoto, etanolo e monossido di carbonio) richiesto dall'utente per ottenere informazioni. In altre parole, se l'utente chiede, ad esempio, informazioni sulla temperatura, l'assistente selezionerà il blocco "speak" relativo alla temperatura e pronuncerà una risposta contenente i dati relativi alla temperatura attuale.

Questo approccio è stato replicato per ciascuno dei sei componenti chimici rilevati dai nostri sensori. In particolare i casi 5, 6, 7, 8 portano a dei blocchi posti in un topic separato chiamato "Chemical\_components".

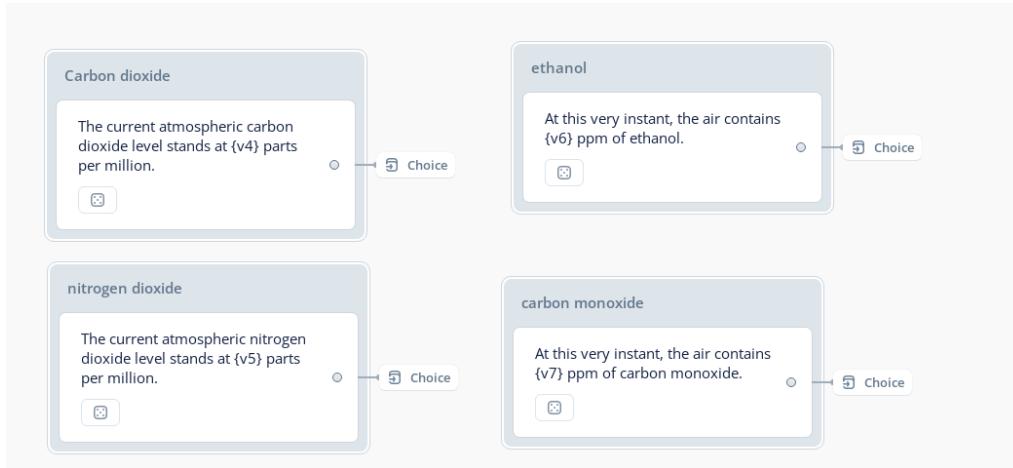


Figura 75: topic Chemical\_components

Una volta completata la costruzione della nostra Alexa Skill, abbiamo proceduto a caricarla sul simulatore di Alexa (come mostrato in figura 76) che fa parte di Alexa Developer Console, il che ci ha permesso di testarne alcune funzionalità e di verificarne il funzionamento in un ambiente di sviluppo controllato.

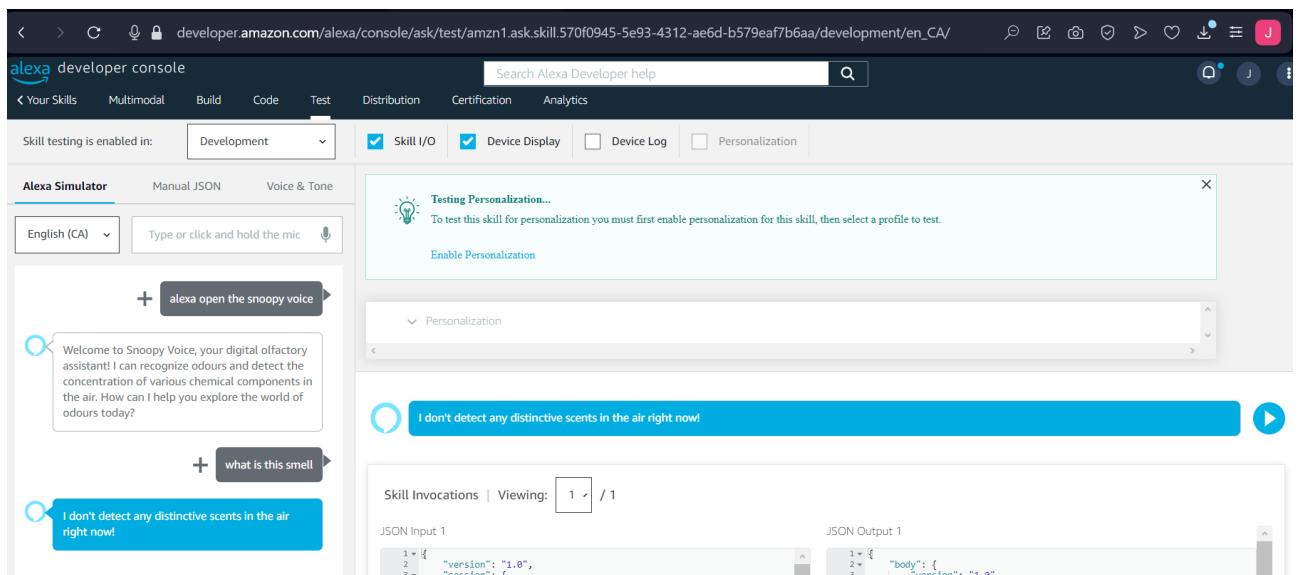


Figura 76: Snoopy Voice testato con Alexa Developer Console

Tuttavia, è importante sottolineare che non abbiamo avuto l'opportunità di testare la Skill su un dispositivo Alexa reale poiché non ne possediamo uno. Nonostante ciò, il simulatore si è dimostrato uno strumento di sviluppo valido. Questo perché riproduce accuratamente l'esperienza d'uso che gli utenti avrebbero con un dispositivo Alexa reale. La struttura dei blocchi, le interazioni vocali e il flusso delle conversazioni sono stati testati e ottimizzati all'interno del simulatore, garantendo che la nostra Skill sia pronta per essere utilizzata con Alexa. L'unico aspetto da tenere presente è che, a differenza di un dispositivo Alexa, il computer richiede l'attivazione manuale del microfono tramite una pressione di un tasto sulla tastiera o cliccando sull'icona del microfono. Anche Voiceflow offre un simulatore dell'interazione con Alexa, in figura 77 è mostrata una possibile interazione con l'applicazione utilizzando questo simulatore.

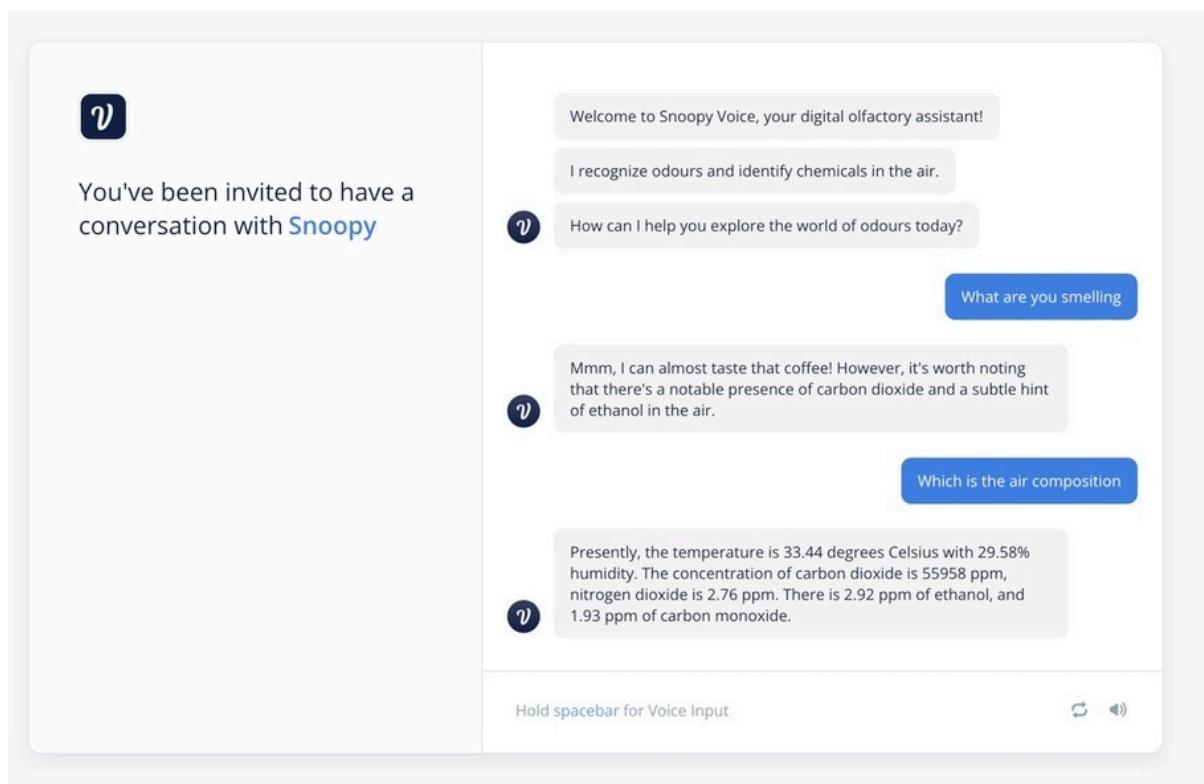


Figura 77: Snoopy Voice testato con il simulatore di Voiceflow

## Real-time performance e test di usabilità

Nel capitolo “Raccolta dei dati ed addestramento del modello” abbiamo analizzato le performance del modello sui dati di test dopo averlo allenato attraverso la piattaforma Edge Impulse.

Ma quali sono le performance dell'inferenza in tempo reale?

Nei numerosissimi test che abbiamo svolto, generalmente le classi “background”, “vinegar”, “martini”, “coffee”, “tea” e soprattutto “lighter\_gas” sono quasi sempre classificate in maniera corretta.

La classe “chamomile” presenta una particolarità: quando i sensori vengono avvicinati alla sostanza, sono richiesti diversi minuti prima di arrivare alla classificazione corretta. Questo probabilmente è dovuto al fatto che, come abbiamo visto, avendo questo odore una composizione chimica simile a quella di tè e cacao, il modello richiede più tempo per riuscire effettivamente a discriminare ciò che rileva.

La classe che in real-time ha le performance peggiori è “cocoa”. Anche in questo caso probabilmente la causa è da imputare alla sua composizione chimica che non presenta evidenti differenze rispetto a quelle di tè e camomilla.

In generale però, come detto, le performance sono ottime e rispecchiano l'andamento dei test effettuati in fase di addestramento.

Sono stati condotti test di usabilità su sei partecipanti per valutare la facilità d'uso del naso elettronico, dell'applicazione Blynk e della Skill di Alexa associata. I risultati indicano una buona usabilità complessiva, con feedback particolarmente positivi sulla possibilità di monitorare da remoto il naso elettronico attraverso l'app Blynk e il controllo vocale tramite Alexa. Sono emersi suggerimenti per migliorare ulteriormente l'esperienza utente, ma nel complesso i partecipanti hanno trovato i dispositivi intuitivi e accessibili. Nel prossimo capitolo verranno analizzati questi suggerimenti che sono alla base dei possibili sviluppi futuri del sistema.

## Conclusioni e sviluppi futuri

In questo progetto, abbiamo creato con successo un naso elettronico "smart" integrato con una Alexa Skill e un'applicazione mobile. Grazie ai progressi nell'ambito dell'intelligenza artificiale e dell'hardware dei microcontrollori, siamo riusciti a sviluppare un dispositivo in grado di classificare odori, rilevare quelli pericolosi e fornire notifiche utilizzando indicatori visivi e sonori.

Il naso ha riportato performance confortanti e è stato ritenuto intuitivo ed accessibile dagli utenti.

Tuttavia, ci sono opportunità per ulteriori miglioramenti.

Possiamo aumentare la precisione della classificazione odori mediante algoritmi più avanzati e l'espansione del dataset.

Aggiungere altri sensori in grado di rilevare ulteriori componenti chimici potrebbe portare all'espansione della gamma di odori rilevabili.

Inoltre, come è emerso dai test di usabilità, l'app per smartphone potrebbe essere espansa con nuove funzionalità, non solo per il monitoraggio ma anche per il controllo del naso elettronico.

In generale, questo progetto ci ha permesso di confrontarci con piattaforme nuove e di sperimentare tipologie di interazione differenti da quelle a cui eravamo abituati. Ci ha permesso di approfondire le nostre conoscenze riguardo questo tipo di sistemi e di affinare le nostre abilità con Arduino.

## Bibliografia

[1] Taste-smell connection, Science World

URL: <https://www.scienceworld.ca/resource/taste-smell-connection/#:~:text=Our%20sense%20of%20smell%20in,we%20experience%20come%20from%20smell>.

[2] Psychology and smell, Fifth Sense

URL: <https://www.fifthsense.org.uk/psychology-and-smell/>

[3] Wio Terminal, Seeed Studio

URL: <https://wiki.seeedstudio.com/Wio-Terminal-Getting-Started/>

[4] Grove - Gas Sensor V2 (Multichannel), Seeed Studio

URL: <https://wiki.seeedstudio.com/Grove-Multichannel-Gas-Sensor-V2/>

[5] Grove - VOC and eCO<sub>2</sub> Gas Sensor (SGP30), Seeed Studio

URL: [https://wiki.seeedstudio.com/Grove-VOC\\_and\\_eCO2\\_Gas\\_Sensor-SGP30/](https://wiki.seeedstudio.com/Grove-VOC_and_eCO2_Gas_Sensor-SGP30/)

[6] Grove -Temperature Humidity Pressure Gas (BME680), Seeed Studio

URL: [https://wiki.seeedstudio.com/Grove-Temperature\\_Humidity\\_Pressure\\_Gas\\_Sensor\\_BME680/](https://wiki.seeedstudio.com/Grove-Temperature_Humidity_Pressure_Gas_Sensor_BME680/)

[7] Grove - I2C Hub, Seeed Studio

URL: [https://wiki.seeedstudio.com/Grove-I2C\\_Hub/](https://wiki.seeedstudio.com/Grove-I2C_Hub/)

[8] Blynk IoT Platform Overview, Blynk

URL: <https://www.youtube.com/watch?v=AaqlFU388vQ>

[9] Design, prototype and launch Alexa Skills, Voiceflow

URL: <https://www.voiceflow.com/create-alexa-skill>