



Università Politecnica delle Marche

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e
dell'Automazione

CORSO DI DATA SCIENCE

**ProntoChef: Realizzazione di un Chatbot culinario
tramite framework Rasa**

Professore

Prof. Domenico Ursino

Studenti

Matteo Risolo

Niccolò De Pascali

ANNO ACCADEMICO 2025/2026

Indice

1	Introduzione	5
1.1	Contesto del progetto	5
1.2	Scopo e obiettivi del progetto	6
2	Il dataset "Italian Food Recipes"	8
2.1	Struttura del dataset	8
2.2	Pre-processing	9
2.3	EDA	13
3	Sviluppo del chatbot	20
3.1	Benvenuto e Onboarding	20
3.2	Ricerca per categoria	22
3.3	Ricerca per ingredienti	24
3.4	Proposta di una ricetta casuale	27
3.5	Ricerca per difficoltà	29
3.6	Visualizzazione ingredienti	31
3.7	Ricerca diretta e logica di visualizzazione	32
3.7.1	Ricerca diretta per nome	33
3.7.2	Selezione da una lista	33
3.7.3	Logica unificata	34
3.7.4	Visualizzazione della scheda della ricetta	34
3.8	Suggerimento ricette simili	34
3.9	Ricerca guidata	36

3.10 Gestione della lista della spesa	37
3.11 Gestione di fallback	39
4 Test del chatbot	40
4.1 Test di Benvenuto e Onboarding	41
4.2 Test ricerca per categoria	44
4.3 Test selezione ricetta	46
4.4 Test ricerca per ingredienti	48
4.5 Test proposta ricetta casuale	49
4.6 Test ricerca per difficoltà	50
4.7 Test visualizzazione ingredienti	51
4.8 Test ricerca diretta per nome	52
4.9 Test suggerimento ricette simili	54
4.10 Test della ricerca guidata	55
4.11 Test gestione della lista della spesa	57
4.12 Test di fallback	58

Capitolo 1

Introduzione

Negli ultimi anni, l'evoluzione delle tecnologie di Intelligenza Artificiale conversazionale ha portato a una crescente diffusione dei chatbot come strumenti di supporto all'utente in diversi ambiti applicativi, tra cui l'assistenza clienti, la ricerca di informazioni e la raccomandazione di contenuti. In particolare, i chatbot basati su Natural Language Processing (NLP) consentono agli utenti di interagire con i sistemi informatici utilizzando il linguaggio naturale, riducendo la complessità dell'interazione e migliorando l'esperienza complessiva.

In questo contesto si colloca il presente progetto, che ha come obiettivo lo sviluppo di **ProntoChef**, un chatbot conversazionale progettato per assistere l'utente nella ricerca e selezione di ricette di cucina in modo intuitivo, guidato e flessibile. Il chatbot è stato pensato come un assistente virtuale in grado di adattarsi alle esigenze dell'utente, fornendo suggerimenti personalizzati sulla base di preferenze esplicite, vincoli alimentari e disponibilità degli ingredienti.

1.1 Contesto del progetto

La scelta del dominio culinario non è casuale: la cucina rappresenta un ambito quotidiano e trasversale, in cui gli utenti spesso si trovano a dover risolvere problemi pratici come:

- decidere cosa cucinare in base agli ingredienti disponibili,
- ricercare una ricetta simile a qualcosa di già visto,
- selezionare piatti in base al livello di difficoltà,
- esplorare nuove ricette in modo casuale o guidato.

Tradizionalmente, queste operazioni richiedono la navigazione manuale di siti web o applicazioni, spesso caratterizzate da interfacce complesse o da una sovrabbondanza di informazioni. Un chatbot conversazionale permette invece di semplificare il processo decisionale, trasformando la ricerca in un dialogo naturale e progressivo.

Dal punto di vista tecnologico, il progetto si inserisce nel contesto dei framework open-source per chatbot, con particolare riferimento a Rasa, una piattaforma che consente di sviluppare assistenti conversazionali avanzati, mantenendo il controllo completo su modelli di NLP, gestione del dialogo, logica di business e integrazione con canali di messaggistica esterni.

1.2 Scopo e obiettivi del progetto

Lo scopo principale del progetto è la progettazione e implementazione di un chatbot intelligente capace di:

- Comprendere il linguaggio naturale dell'utente, riconoscendo intenzioni (intent) ed entità (entity).
- Gestire conversazioni multi-turno, mantenendo il contesto del dialogo.
- Guidare l'utente passo dopo passo nella definizione dei criteri di ricerca tramite un sistema di form conversazionali.
- Offrire funzionalità multiple, tra cui:

- avviare una ricerca guidata di ricette, durante la quale l’utente viene accompagnato passo dopo passo nella definizione delle proprie preferenze;
 - filtrare le ricette in base a criteri come categoria e livello di difficoltà;
 - effettuare una ricerca basata sugli ingredienti disponibili, riducendo sprechi e facilitando la scelta;
 - ricevere suggerimenti casuali per stimolare la scoperta di nuove ricette;
 - selezionare o approfondire ricette già proposte nel corso della conversazione scoprendone tutti i dettagli.
 - gestire e consultare una lista della spesa personale in base agli ingredienti delle ricette interessate
- Interagire attraverso una piattaforma di messaggistica reale, nello specifico Telegram, rendendo il chatbot utilizzabile in un contesto concreto e non puramente teorico.

Capitolo 2

Il dataset "Italian Food Recipes"

2.1 Struttura del dataset

Il dataset che è stato utilizzato come base di conoscenza per lo sviluppo del chatbot è denominato "**Italian Food Recipes**" ed è disponibile su Kaggle al seguente link: <https://www.kaggle.com/datasets/edoardoscarpaci/italian-food-recipes> Si tratta di una raccolta di ricette tipiche della cucina italiana pubblicata sulla piattaforma Kaggle da *edoardoscarpaci* come risorsa aperta per progetti di analisi dati, NLP e raccomandazioni di ricette.

Il dataset è composto da 7 colonne:

- **#:** Una colonna senza nome che contiene un id numerico univoco per identificare la ricetta all'interno del datatset
- **Nome:** Colonna che indica il nome completo della ricetta
- **Categoria:** Indica la categoria della ricetta tra una serie di categorie disponibili che saranno approfondite in seguito (es. dolci, primi piatti, antipasti, ecc.)
- **Link:** Un link ad un sito contenente vari dettagli della ricetta
- **Persone/Pezzi:** Colonna che indica il numero di persone per la quale è pensata e tarata la ricetta o il numero di pezzi se si tratta di una ricetta

che produce elementi finiti piuttosto che porzioni per un determinato numero di persone (es. biscotti, pasticcini, ecc.)

- **Ingredienti:** Colonna che riporta la lista degli ingredienti e la relativa quantità in maniera strutturata (es. [['Mascarpone', '750g'], ['Uova', '260g'], ['Savoiardi', '250g']])
- **Steps:** Colonna che contiene l'intero testo che descrive il procedimento per la preparazione della ricetta in questione

Il dataset contiene informazioni strutturate ma non troppo rigide, il che lo rende ideale per:

- estrazione di entità (ingredienti, categorie, etc.) con modelli NLP,
- filtri personalizzati basati sui criteri dell'utente,
- suggerimenti simili grazie alla comparazione degli ingredienti.

In altre parole, la struttura del dataset è particolarmente adatta per implementare le funzionalità di ProntoChef desiderate quali ricerca guidata, filtri per difficoltà, filtraggio per ingredienti e suggerimenti simili e creare così un chatbot in lingua italiana.

2.2 Pre-processing

La fase di preprocessing ha avuto come obiettivo principale quello di rendere il dataset adatto all'utilizzo all'interno del chatbot, garantendo:

- coerenza dei dati,
- assenza di valori mancanti nelle informazioni critiche,
- uniformità testuale per facilitare confronti e ricerche,
- una struttura degli ingredienti facilmente interrogabile,

- la derivazione di attributi utili al dialogo, come il livello di difficoltà.

Poiché il chatbot basa il proprio funzionamento sulla ricerca e sul filtraggio delle ricette, è stato necessario trasformare il dataset originale in una forma più strutturata e affidabile.

In una prima fase sono state eliminate tutte le righe contenenti valori nulli nelle colonne considerate fondamentali, ovvero: *Nome* (della ricetta), *Ingredienti* e *Steps* (procedimento).

```
1 recipes.dropna(subset=[‘Nome’, ‘Ingredienti’, ‘Steps’], inplace
                  =True)
```

Listing 2.1: Rimozione dei record incompleti dal dataset delle ricette

Questa scelta è motivata dal fatto che una ricetta priva di titolo, ingredienti o procedimento risulta inutilizzabile per il chatbot, che deve presentare il nome della ricetta all’utente, filtrare in base agli ingredienti ed eventualmente mostrare o riassumere i passaggi di preparazione. La rimozione preventiva di queste righe garantisce quindi una maggiore qualità complessiva del dataset.

Successivamente, alcune colonne testuali sono state normalizzate applicando rimozione degli spazi superflui e conversione in minuscolo.

```
1 recipes[“Nome”] = recipes[“Nome”].str.strip().str.lower()
recipes[“Categoria”] = recipes[“Categoria”].str.strip().str.
                           lower()
```

Listing 2.2: Normalizzazione del testo per le colonne Nome e Categoria

Questa operazione è fondamentale per evitare duplicazioni logiche (ad esempio “Dolci” vs “dolci”) e per facilitare confronti testuali, matching degli slot nel chatbot, e ricerche case-insensitive.

La colonna Ingredienti nel dataset originale è memorizzata come stringa che rappresenta una lista Python. Per rendere questi dati realmente utilizzabili, è stato implementato un parsing tramite `ast.literal_eval`.

```
import ast
```

```

2
def parse_ingredients(x):
4    try:
6        # Converte la stringa in una lista Python reale
8        parsed = ast.literal_eval(x)
10       return [
12           {
14               "nome": ing[0].strip().lower(),
16               "quantita": ing[1].strip().lower()
18           }
20           for ing in parsed
22       ]
24   except:
26       # Gestione degli errori per formati non validi
28   return []

```

Listing 2.3: Funzione di parsing e strutturazione degli ingredienti

Il risultato è una nuova colonna, `ingredienti_parsed`, contenente una lista di dizionari, ciascuno con nome dell'ingrediente e quantità associata. Questa struttura rende gli ingredienti semanticamente chiari e facilmente manipolabili, adatti a successive elaborazioni.

Per facilitare ulteriormente le operazioni di filtraggio e confronto, è stata creata una seconda rappresentazione semplificata degli ingredienti:

```

recipes["ingredienti_flat"] = recipes["ingredienti_parsed"].
2     apply(
3         lambda x: [i["nome"] for i in x]
4     )

```

Listing 2.4: Estrazione dei nomi degli ingredienti (Flattening)

La colonna `ingredienti_flat` contiene esclusivamente i nomi degli ingredienti, senza quantità. Questa scelta si rivelerà particolarmente utile per ricerche basate sugli ingredienti disponibili, suggerimenti di ricette simili e matching diretto con l'input dell'utente nel chatbot.

La colonna `Steps` è stata ripulita rimuovendo caratteri di nuova linea e spazi inutili:

```
1 # Rimozione dei caratteri di nuova riga e degli spazi bianchi  
    in eccesso  
2  
3     recipes["Steps"] = (  
4         recipes["Steps"]  
5             .str.replace("\n", " ")  
6             .str.strip()  
7     )
```

Listing 2.5: Pulizia e formattazione della colonna Steps

Questo rende il testo più uniforme, più leggibile e pronto per essere eventualmente mostrato o elaborato dal chatbot senza formattazioni indesiderate.

Sono poi state eliminate tutte le ricette con un valore non valido (minore o uguale a zero) nella colonna Persone/Pezzi:

```
recipes = recipes[recipes['Persone/Pezzi'] > 0]
```

Listing 2.6: Filtraggio dei record basato sulla validità delle porzioni

Questo protegge da eventuali errori presenti nel dataset e garantisce che ogni ricetta abbia un numero di porzioni significativo, informazione importante per le risposte del chatbot e gestire correttamente lo slot relativo al numero di persone che si approfondirà più avanti.

Poiché il dataset originale non forniva sempre un livello di difficoltà esplicito, è stato deciso di stimare la difficoltà della ricetta in modo automatico, basandosi sul numero di ingredienti:

```
1 def stima_difficoltà(row):  
2     # Calcolo della cardinalità della lista degli ingredienti  
3     n_ing = len(row["ingredienti_flat"])  
4  
5     if n_ing <= 6:  
6         return "facile"  
7     elif n_ing <= 10:
```

```

    return "media"
9 else:
    return "difficile"

```

Listing 2.7: Logica di stima della difficoltà basata sul numero di ingredienti

Il criterio adottato è il seguente:

- facile: fino a 6 ingredienti
- media: da 7 a 10 ingredienti
- difficile: più di 10 ingredienti

Questa euristica, pur semplice, risulta efficace per un chatbot orientato all’utente finale, che necessita di una distinzione chiara e immediata del livello di complessità.

Al termine del *pre-processing*, la distribuzione delle ricette per livello di difficoltà risulta la seguente:

- **difficile**: 2861 ricette
- **media**: 2317 ricette
- **facile**: 757 ricette

Questa distribuzione evidenzia una prevalenza di ricette di difficoltà medio-alta, ma garantisce un numero comunque elevato anche per ricette di difficoltà *facile*, segno che l’euristica può essere considerata buona.

con quest’ultima operazione si conclude la fase di *pre-processing*

2.3 EDA

Dopo la fase di preprocessing, il dataset finale risulta composto da 5935 ricette. Questo numero indica che, nonostante le operazioni di pulizia e filtraggio (rimozione di valori nulli e righe non valide), il dataset mantiene una dimensione

sufficientemente ampia. L'analisi delle categorie mostra una distribuzione non uniforme, ma rappresentativa della tradizione culinaria italiana:

- dolci: 1742
- primi piatti: 1313
- antipasti: 887
- secondi piatti: 830
- lievitati: 329
- piatti unici: 271
- contorni: 175
- torte salate: 105
- salse e sughi: 91
- bevande: 71
- marmellate e conserve: 62
- insalate: 57

Questa distribuzione evidenzia come il dataset sia fortemente orientato verso dolci e primi piatti, che rappresentano le categorie più numerose, seguite da antipasti e secondi piatti. Ciò implica che alcune categorie saranno naturalmente suggerite più spesso di altre.

Il numero medio di ingredienti per ricetta risulta pari a circa **11** ingredienti. Questo valore fornisce un'indicazione importante sulla complessità media delle ricette presenti nel dataset e cioè che molte ricette richiedono un numero non trascurabile di ingredienti. Questa osservazione giustifica ulteriormente l'introduzione di un livello di difficoltà e la possibilità per l'utente di filtrare le ricette.

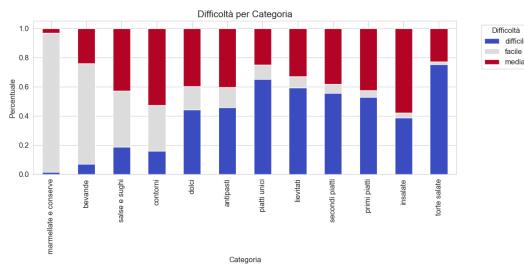


Figura 2.3: Difficoltà per categoria

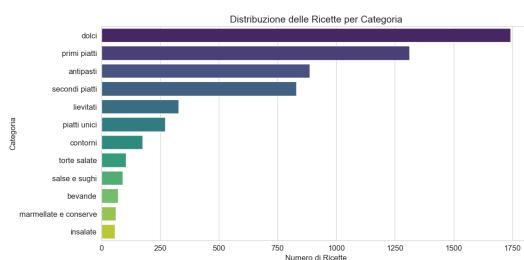


Figura 2.1: Distribuzione delle categorie nel dataset

Di seguito anche la distribuzione della difficoltà già discussa in precedenza e la distribuzione della difficoltà in relazione alla categoria:

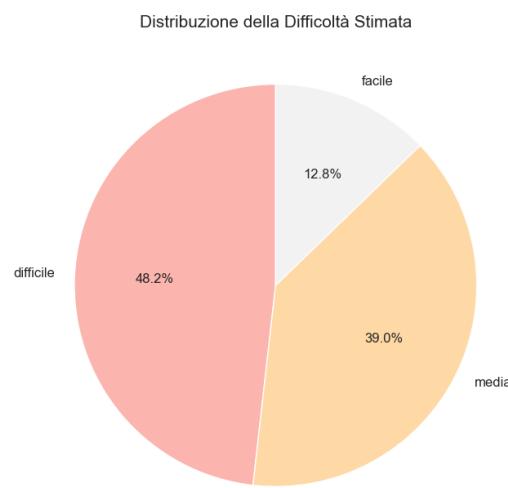


Figura 2.2: Distribuzione della difficoltà

La seguente è invece la top 20 degli ingredienti più presenti tra tutte le ricette:

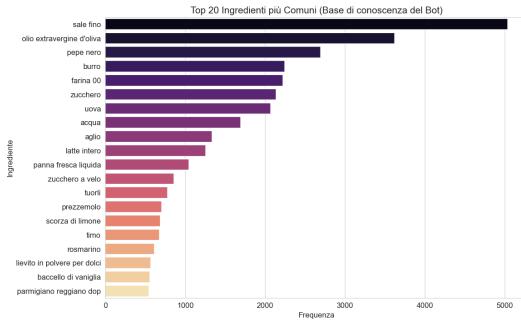


Figura 2.4: Top 20 ingredienti

Il grafico seguente mostra invece un istogramma con una curva di densità sovrapposta. Vediamo una distribuzione asimmetrica a destra. Ciò significa che mentre la maggior parte delle ricette si concentra attorno a un valore medio, c'è una "coda lunga" di ricette molto complesse con tantissimi ingredienti (fino a oltre 50), mentre non si può ovviamente scendere sotto lo zero. Il picco (moda) sembra attestarsi intorno ai 10-11 ingredienti. Questa è la complessità "standard" delle ricette nel dataset, come era già emerso. In figura sono visibili anche le soglie (le linee tratteggiate) che dividono le tre difficoltà.

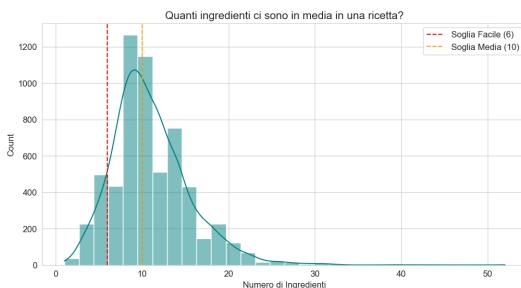


Figura 2.5: Distribuzione ingredienti e difficoltà

La scelta delle classi di difficoltà tramite euristica basata sul numero di ingredienti, trova Validazione tramite lunghezza del testo di preparazione. Il boxplot mette in relazione le tre classi di difficoltà (calcolate in base agli ingredienti) con la lunghezza del testo delle istruzioni (numero di parole). Serve a rispondere alla domanda: "Usare il numero di ingredienti è un buon metodo per stimare la difficoltà reale?". Il grafico evidenzia una chiara tendenza

crescente. La classe "**Facile**" (Viola) ha un box compatto e posizionato in basso. La mediana si attesta intorno alle 130 parole. Questo conferma che pochi ingredienti corrispondono quasi sempre a procedure brevi e veloci.

Nella classe "**Media**" (Magenta): Si nota un incremento, con una mediana verso le 160 parole.

Nella classe "**Difficile**" (Arancione) il salto è evidente. La mediana sale a circa 230 parole e, soprattutto, il box è molto più alto, con "baffi" che si estendono fino a 500 parole.

Il grafico valida la logica del chatbot. Anche se l'algoritmo usa solo il conteggio degli ingredienti per decidere la difficoltà, il boxplot dimostra che questa metrica è un ottimo indicatore indiretto per la complessità della preparazione. Le ricette classificate come "Difficili" per via degli ingredienti si rivelano essere effettivamente quelle che richiedono spiegazioni più lunghe e articolate. Questo conferma che la logica di business del bot è robusta e coerente con la realtà dei dati.

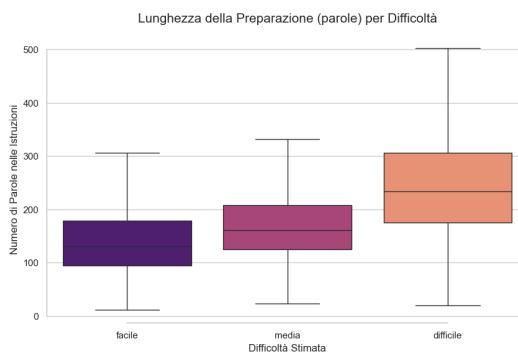


Figura 2.6: Boxplot per la lunghezza della preparazione per difficoltà

Oltre alle statistiche descrittive di base, è stata condotta un'analisi testuale sui titoli delle ricette, con l'obiettivo di individuare le combinazioni di parole più ricorrenti. Questa analisi permette di comprendere quali concetti, ingredienti o tipologie di piatti emergano con maggiore frequenza già dal nome della ricetta. Prima della vettorizzazione, è stata definita manualmente una lista di stop words italiane, comprendente articoli, preposizioni semplici e articola-

te, congiunzioni e termini poco informativi nel contesto culinario (ad esempio “ricetta”, “base”). Lo scopo di questa fase è ridurre il rumore e concentrare l’analisi sulle parole realmente discriminanti, evitando che combinazioni molto comuni ma poco significative dominino i risultati. Per l’analisi è stato utilizzato `CountVectorizer` della libreria `scikit-learn`, configurato per estrarre bigrammi (coppie di parole consecutive). I titoli delle ricette sono stati quindi trasformati in una rappresentazione vettoriale basata sulla frequenza delle coppie di parole. Dopo la trasformazione è stata calcolata la frequenza totale di ciascun bigramma nel dataset, i risultati sono stati ordinati in modo decrescente ed è stato poi creato un DataFrame contenente bigrammi e frequenze. Successivamente, sono stati selezionati i 15 bigrammi più frequenti, visualizzati tramite un grafico a barre orizzontali:

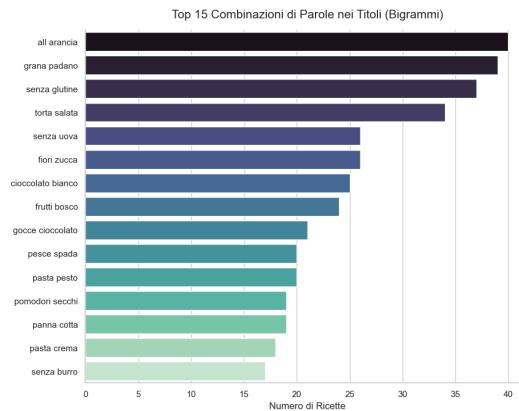


Figura 2.7: Top 15 bigrammi

Questo invece il codice:

```

from sklearn.feature_extraction.text import CountVectorizer
# Utilizzo CountVectorizer per trovare le coppie di parole (bigrammi)
# Definisco le parole da rimuovere prima per pulire i risultati (articoli, preposizioni)
stop_words_it = [
    'di', 'a', 'e', 'da', 'in', 'con', 'su', 'per', 'tra', 'fra', 'il', 'lo', 'le', 'i', 'gli', 'le',
    'un', 'uno', 'una', 'e', 'che', 'ai', 'allo', 'alla', 'ai', 'agli', 'ala', 'dell', 'dello',
    'della', 'così', 'degli', 'della', 'molti', 'tanti', 'molte', 'tante', 'mai', 'magli', 'malle', 'sul',
    'tutte', 'sulle', 'sui', 'anghi', 'solida', 'I', 'd', 'r', 'ricetta', 'base'
]

# Impiego del CountVectorizer per cercare coppie di 2 parole (ngram_range=(2,2))
c_vec = CountVectorizer(stop_words=stop_words_it, ngram_range=(2, 2))

# Trasformazione titoli
ngrams = c_vec.fit_transform(recipes['Nome'])

# Conteggio frequenza
count_values = ngrams.toarray().sum(axis=0)
vocab = c_vec.vocabulary_

# Creazione DataFrame per il grafico
df_bigram = pd.DataFrame(sorted([(count_values[i], k) for k, i in vocab.items()]), reverse=True).rename(columns={0: 'Frequenza', 1: 'Bigrammo'})

```

Figura 2.8: Codice Top 15 bigrammi

Infine, l'ultima visualizzazione riguarda una heatmap per categorie e difficoltà:

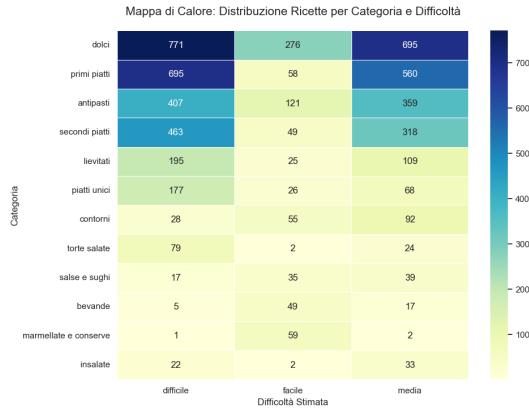


Figura 2.9: Heatmap

Capitolo 3

Sviluppo del chatbot

3.1 Benvenuto e Onboarding

La fase di onboarding del chatbot è stata progettata per offrire all'utente un ingresso immediato e intuitivo all'interno del sistema, riducendo la necessità di istruzioni esterne e favorendo un'interazione naturale fin dal primo messaggio.

Il chatbot riconosce una vasta gamma di espressioni di saluto attraverso l'intent `greet`. Questo intent include sia saluti tradizionali (“ciao”, “buongiorno”, “salve”), sia messaggi meno strutturati o tipici dell'avvio di una conversazione (“start”, “iniziamo”, “chi sei”). Dal punto di vista conversazionale, ciò consente al chatbot di intercettare correttamente il primo messaggio dell'utente anche quando non viene espresso in modo formale, migliorando la robustezza dell'interazione iniziale. Alla rilevazione dell'intent di saluto, una regola dedicata attiva sempre la risposta di benvenuto, indipendentemente dallo stato della conversazione. Questa scelta garantisce che il chatbot sia in grado di rispondere correttamente anche in caso di riavvio o di messaggi fuori contesto.

Il messaggio di benvenuto è stato progettato per presentare l'identità del chatbot (ProntoChef), definire immediatamente il contesto culinario e invitare l'utente all'azione con una domanda aperta. Per evitare risposte ripetitive e rendere l'interazione più naturale, sono state definite più varianti di messaggi

di benvenuto, selezionate in modo casuale. Questa scelta contribuisce a dare al chatbot una personalità più dinamica e meno meccanica.

Dal punto di vista tecnico, i messaggi sfruttano la formattazione HTML supportata dal canale Telegram (ad esempio grassetto ed emoji), migliorando la leggibilità e l'impatto visivo del testo.

Oltre al semplice saluto, è stata implementata una funzionalità di assistenza accessibile in qualsiasi momento tramite l'intent `help`. Questo intent raccoglie richieste esplicite di supporto (“aiutami”, “cosa sai fare”, “istruzioni”) e messaggi che indicano smarrimento o incertezza da parte dell'utente. Quando viene riconosciuto l'intent di aiuto, il chatbot fornisce una risposta strutturata che descrive in modo chiaro e dettagliato le principali funzionalità disponibili.

Il messaggio di aiuto è suddiviso logicamente in sezioni, distinguendo tra:

- azioni che l'utente può compiere immediatamente (ricerca per ingredienti, categoria, difficoltà, ricerca guidata);
- azioni disponibili dopo aver visualizzato un elenco di ricette (selezione di una ricetta, suggerimenti simili, gestione della lista della spesa).

Questa suddivisione permette all'utente di comprendere non solo cosa il chatbot può fare, ma anche quando determinate funzionalità diventano disponibili all'interno del flusso conversazionale.

L'onboarding non è stato concepito come una semplice introduzione testuale, ma come una vera e propria guida all'utilizzo del chatbot. Grazie alla combinazione di messaggi di benvenuto, intent di aiuto sempre accessibili e risposte formattate in modo chiaro, l'utente è messo in condizione di iniziare subito l'interazione senza dover consultare documentazione esterna.

Questa fase iniziale è fondamentale per garantire un'esperienza utente fluida e per incentivare l'esplorazione delle funzionalità offerte dal chatbot.

```

rules:
  - rule: Greet user anytime
    steps:
      - intent: greet
      - action: utter_greet

  - rule: Help user anytime
    steps:
      - intent: help
      - action: utter_help

```

Figura 3.1: Rules per gli intent di greet e help

3.2 Ricerca per categoria

Una delle principali modalità di interazione offerte dal chatbot è la ricerca di ricette per categoria, che consente all’utente di esplorare il dataset culinario a partire dal tipo di piatto desiderato (ad esempio dolci, primi piatti, secondi piatti, ecc.).

Questa funzionalità è pensata per supportare un comportamento molto comune negli utenti: sapere che tipo di piatto si vuole preparare, senza avere in mente una ricetta specifica.

La categoria del piatto è gestita tramite lo slot `category`, definito come slot testuale e con `influence_conversation` impostato a `true`, in modo da permettere al valore dello slot di influenzare le predizioni successive del dialogo.

Il mapping dello slot è stato progettato con una logica a priorità:

- **Estrazione da entità (from_entity):** Quando l’utente utilizza un intent esplicito legato alla ricerca per categoria (ad esempio `search_by_category`), il valore dello slot viene popolato direttamente dall’entità `category` riconosciuta nel messaggio. Questo approccio consente una gestione robusta dei casi in cui l’utente esprime chiaramente la propria richiesta, come ad esempio: “voglio un dolce” oppure “mostrami dei primi piatti”.

- **Fallback testuale** (`from_text`) nel contesto del form. Nel caso della ricerca guidata, se il chatbot non riesce a riconoscere un'entità strutturata, lo slot può essere popolato con il testo inserito dall'utente, ma solo quando il form è attivo e lo slot richiesto è `category`. Questa soluzione permette di mantenere flessibilità nell'interazione senza interferire con altre funzionalità del chatbot al di fuori del form.

Questa doppia strategia consente di bilanciare precisione e robustezza, evitando conflitti tra intenti e garantendo che il valore corretto venga salvato nello slot appropriato.

La ricerca per categoria è attivata tramite l'intent `search_by_category`, addestrato con numerosi esempi che coprono sia categorie standard (dolci, primi piatti, contorni), sia sinonimi o formulazioni alternative (dessert, sughi, stuzzichini). Questa varietà di esempi consente al modello NLU di generalizzare meglio e riconoscere correttamente la richiesta dell'utente anche in presenza di espressioni meno canoniche.

Quando l'utente esprime un intent di ricerca per categoria, il flusso conversazionale segue una sequenza ben definita:

- riconoscimento dell'intent `search_by_category`;
- invio di un messaggio di feedback all'utente (azione `utter_searching`);
- esecuzione dell'azione custom `action_search_by_category`.

Questa struttura rende il dialogo più naturale, fornendo all'utente un riscontro immediato mentre il chatbot elabora la richiesta.

L'azione `action_search_by_category` ha il compito di recuperare e presentare all'utente un insieme di ricette appartenenti alla categoria richiesta. Il funzionamento dell'azione può essere riassunto nei seguenti passaggi:

1. recupero del valore dello slot `category`;

2. verifica della sua presenza: se lo slot non è valorizzato, il chatbot richiede esplicitamente all’utente di specificare una categoria;
3. interrogazione del dataset tramite una funzione dedicata che filtra le ricette in base alla categoria;
4. gestione del caso in cui non siano presenti ricette per la categoria indicata;
5. selezione casuale di un sottoinsieme di ricette, per evitare risposte sempre identiche;
6. presentazione dei risultati all’utente in forma di elenco.

Al termine della risposta, la lista delle ricette mostrate viene salvata nello slot `last_recipes`, in modo da poter supportare interazioni successive, come la selezione di una ricetta specifica o la richiesta di suggerimenti simili.

```
class ActionSearchByCategory(Action):
    def name(self) -> str:
        return "action_search_by_category"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: dict):
        category = tracker.get_slot("category")

        if not category:
            dispatcher.utter_message(response="utter_ask_category")
            return []

        recipes = get_recipes_by_category(recipes_df, category)

        if not recipes:
            dispatcher.utter_message(response="utter_no_category_results", category=category)
            return []

        response = f"Ecco alcune ricette per la categoria {category}:\n"
        response += "\n".join(f"\n{r}" for r in recipes)

        dispatcher.utter_message(response)
        dispatcher.utter_message(response="utter_ask_select_from_list")

        return [SlotSet("last_recipes", recipes)]
```

Figura 3.2: Custom Action per la ricerca per categoria

3.3 Ricerca per ingredienti

Una delle funzionalità principali del chatbot è la ricerca di ricette basata sugli ingredienti disponibili, comunemente detta svuotafrigo. Questa modalità con-

sente all’utente di ottenere suggerimenti culinari a partire dagli ingredienti che ha già a disposizione, riducendo sprechi e semplificando la scelta del piatto.

Dal punto di vista conversazionale, questa funzionalità presenta diverse criticità: l’utente può esprimere gli ingredienti in modo libero, incompleto o ambiguo, e spesso senza un intent esplicito. Per questo motivo è stata progettata una gestione robusta dello slot `ingredients`.

Lo slot `ingredients` è definito come una lista di stringhe e ha `influence_conversation` impostato a `true`, poiché il suo contenuto influenza direttamente la ricerca delle ricette e le risposte successive del chatbot. Il mapping dello slot è stato progettato in modo contestuale e protetto, per evitare interferenze con altre parti della conversazione.

1. Mapping esplicito per l’intent `search_by_ingredients`: Quando l’utente utilizza un intent esplicito di ricerca per ingredienti (ad esempio: “ho uova e pancetta”), lo slot viene popolato direttamente dalle entità ingredient riconosciute dal modello NLU. Questo caso rappresenta il flusso ideale, in cui l’intent dell’utente è chiaro e le informazioni sono strutturate.
2. Mapping controllato per intenti generici (`inform`, `guided_search`). Per evitare che ingredienti vengano erroneamente estratti durante altre interazioni (ad esempio all’interno di un form o durante la richiesta di difficoltà o categoria), il mapping da entità è consentito solo se il chatbot si trova all’interno del form `guided_search_form` (che sarà illustrato in seguito) o se lo slot richiesto è effettivamente `ingredients`. Questa condizione protegge il sistema da effetti collaterali, come il popolamento dello slot ingredienti con parole non pertinenti pronunciate in altri contesti della conversazione.
3. Fallback testuale nel contesto del form. Nel caso in cui il riconoscimento delle entità fallisca durante la ricerca guidata, lo slot può essere popolato

direttamente con il testo inserito dall’utente, ma esclusivamente quando il form è attivo e sta richiedendo lo slot `ingredients`.

Questo approccio garantisce flessibilità senza compromettere la stabilità globale del dialogo.

La funzionalità è attivata tramite l’intent `search_by_ingredients`, addestrato con esempi che simulano frasi realistiche, includendo elenchi di ingredienti separati da congiunzioni, frasi colloquiali aderenti all’intent in questione e richieste con uno o più ingredienti. Questa varietà di esempi migliora la capacità del modello di riconoscere correttamente l’intent anche in presenza di formulazioni non standard.

La logica principale della funzionalità è invece implementata nell’azione custom `action_search_by_ingredients`.

Il comportamento dell’azione segue i seguenti passaggi:

1. Recupero degli ingredienti dallo slot: L’azione tenta innanzitutto di leggere lo slot `ingredients`, che rappresenta il canale preferenziale per l’informazione estratta dall’NLU.
2. Fallback sul testo dell’utente: Se lo slot risulta vuoto (ad esempio per mancato riconoscimento delle entità), l’azione analizza direttamente il testo dell’ultimo messaggio dell’utente. In questa fase viene rimossa la punteggiatura, viene applicata una lista di stop words per eliminare verbi, articoli e parole generiche e vengono mantenute solo parole significative, con lunghezza minima, interpretate come possibili ingredienti.
3. Gestione dei casi di input insufficiente: Se anche dopo il fallback non vengono individuati ingredienti validi, il chatbot richiede esplicitamente all’utente di fornire gli ingredienti disponibili.
4. Ricerca delle ricette: Una volta ottenuta una lista di ingredienti, l’azione interroga il dataset per recuperare le ricette compatibili, privilegiando quelle che contengono uno o più ingredienti forniti.

5. Presentazione dei risultati: Le ricette trovate vengono mostrate all'utente in forma di elenco, e il chatbot invita a selezionarne una per visualizzarne i dettagli.
6. Infine, l'elenco delle ricette proposte viene salvato nello slot `last_recipes`, rendendo possibile una continuità nella conversazione (ad esempio selezione di una ricetta o richiesta di alternative simili)

```
- intent: search_by_ingredients
examples:
  - ho [zucchine](ingredient) e [gambretti](ingredient)
  - cosa posso fare con [uova](ingredient) e [pancetta](ingredient)
  - ho solo [pasta](ingredient) e [pomodori](ingredient)
  - cosa cucino con [patate](ingredient), [cipolle](ingredient) e [salsiccia](ingredient)
  - ho [farina](ingredient), [uova](ingredient) e [zucchero](ingredient)
  - cosa posso preparare con [riso](ingredient)
  - idee con [melanzane](ingredient) e [mozzarella](ingredient)
  - ho in frigo [pollo](ingredient) e [limone](ingredient)
  - ricette con [tonno](ingredient) e [pasta](ingredient)
  - cosa posso fare usando [ricotta](ingredient)
```

Figura 3.3: Estratto di `nlu.yaml` per la ricerca per ingredienti

3.4 Proposta di una ricetta casuale

Il chatbot implementa una funzionalità di suggerimento casuale di una ricetta, pensata per supportare l'utente nei casi di indecisione o mancanza di ispirazione. Questa modalità permette di ottenere una ricetta senza specificare alcun criterio, delegando completamente la scelta al sistema. Dal punto di vista dell'esperienza utente, questa funzione simula il comportamento di un assistente umano che propone un'idea “al volo”, rendendo l'interazione più naturale e meno rigida.

La funzionalità è attivata tramite l'intent `surprise_me`, addestrato con un ampio insieme di esempi linguistici. Gli esempi includono richieste esplicite di casualità (“dammi una ricetta a caso”), espressioni colloquiali di indecisione (“non so cosa cucinare”) e richieste d’ispirazione (“stupiscimi”, “ispirami”).

Questa varietà di formulazioni consente al modello NLU di riconoscere correttamente l'intent anche in presenza di linguaggio spontaneo e poco strutturato.

La gestione del flusso è affidata invece a una rule dedicata, che garantisce una risposta immediata e deterministica. Quando viene rilevato l'intent `surprise_me`, il chatbot esegue direttamente l'azione `action_surprise_me`. In questo caso, l'uso di una rule, anziché di una story, evita ambiguità e assicura che la richiesta venga sempre soddisfatta nello stesso modo, indipendentemente dallo stato precedente della conversazione.

L'azione custom implementa l'intera logica di selezione e presentazione della ricetta casuale. Il comportamento dell'azione è articolato nei seguenti passaggi:

1. Verifica della disponibilità del dataset: L'azione controlla che il dataset delle ricette non sia vuoto, prevenendo errori in fase di esecuzione.
2. Selezione casuale della ricetta: Viene estratta una ricetta in modo casuale dal dataset tramite campionamento uniforme, garantendo varietà nelle proposte.
3. Recupero dei dettagli completi: A partire dal nome della ricetta selezionata, il chatbot recupera ingredienti strutturati, quantità, numero di persone/pezzi e istruzioni di preparazione.
4. Formattazione della risposta: La risposta è costruita in modo leggibile e coinvolgente
5. Aggiornamento degli slot di contesto: Al termine dell'azione, vengono aggiornati `recipe_name`, con il nome della ricetta proposta, `last_recipes`, contenente la ricetta suggerita e `recipe_index`, azzerato per evitare ambiguità. Questo permette all'utente di interagire successivamente con la ricetta proposta (ad esempio chiedendo ingredienti, ricette simili o aggiungendo elementi alla lista della spesa), come già visto in altri casi e come vedremo ancora.

3.5 Ricerca per difficoltà

Il chatbot supporta la ricerca di ricette in base al livello di difficoltà, stimata sul numero di ingredienti come già discusso nel capitolo di pre-processing, permettendo all’utente di filtrare le proposte in funzione delle proprie competenze culinarie o del tempo a disposizione. Questa funzionalità è particolarmente utile per adattare i suggerimenti sia a utenti principianti sia a utenti più esperti.

La difficoltà della ricetta è modellata tramite lo slot `difficulty`, di tipo `text`, che influenza il flusso della conversazione. Lo slot può essere valorizzato in due modalità ovvero come estrazione diretta da entità, quando l’utente esprime chiaramente la difficoltà (es. “voglio qualcosa di facile”) oppure acquisizione da testo libero, nel contesto del `guided_search_form` che si vedrà più avanti, come meccanismo di fallback nel caso in cui l’entità non venga riconosciuta correttamente. Questa configurazione rende il sistema più robusto rispetto a variazioni linguistiche e formulazioni non standard.

L’intent `filter_by_difficulty` intercetta le richieste di filtraggio basate sulla complessità della preparazione. Gli esempi di addestramento coprono livelli di difficoltà standard (facile, media, difficile), sinonimi (semplice, intermedio, avanzato, impegnativa), espressioni colloquiali e ingleismi (hard). Questa ampia copertura lessicale migliora la capacità del modello NLU di riconoscere correttamente l’intent anche in presenza di linguaggio informale o impreciso.

Anche in questo caso la gestione del flusso avviene tramite una rule dedicata in modo da evitare interferenze con altri percorsi conversazionali simili semanticamente.

L’azione custom implementa la logica di filtraggio e presentazione delle ricette in base alla difficoltà selezionata. Il funzionamento dell’azione è strutturato nei seguenti passaggi:

1. Recupero dello slot `difficulty`: Se lo slot non è valorizzato, il chatbot richiede esplicitamente all’utente di specificare il livello di difficoltà.
2. Filtraggio del dataset: Viene chiamata una funzione helper che filtra il dataset delle ricette confrontando il valore della difficoltà in modo case-insensitive.
3. Gestione dei casi senza risultati: Se non vengono trovate ricette compatibili, il chatbot informa l’utente dell’assenza di risultati per il livello richiesto.
4. Presentazione delle ricette: In caso di risultati validi, il chatbot restituisce un elenco di ricette, selezionate in modo casuale per aumentare la varietà delle proposte.
5. Aggiornamento del contesto: Le ricette suggerite vengono salvate nello slot `last_recipes`, consentendo interazioni successive (ad esempio la selezione di una ricetta specifica).

```
def get_recipes_by_difficulty(
    recipes_df,
    difficulty: str,
    limit: int = 5,
    shuffle: bool = True
):

    if not difficulty:
        return []

    difficulty = difficulty.lower().strip()

    filtered = recipes_df[
        recipes_df["difficoltà"].str.lower() == difficulty
    ]

    if filtered.empty:
        return []

    if shuffle:
        filtered = filtered.sample(frac=1)

    return filtered["Nome"].head(limit).tolist()
```

Figura 3.4: Funzione helper

3.6 Visualizzazione ingredienti

Il chatbot permette all’utente di visualizzare la lista degli ingredienti di una specifica ricetta, rispondendo a richieste dirette oppure a richieste contestuali basate su ricette suggerite in precedenza. Questa funzionalità è pensata per supportare sia una consultazione immediata sia l’uso pratico, ad esempio per la preparazione della lista della spesa.

L’intent `ask_recipe_ingredients` intercetta le richieste relative agli ingredienti di una ricetta. Gli esempi di addestramento includono richieste esplicite della lista ingredienti, formulazioni alternative e ricette con nomi lunghi e complessi, per aumentare la robustezza del riconoscimento dell’entità `recipe_name`.

Il nome della ricetta può essere individuato in tre modi:

- tramite entità esplicita (`recipe_name`) quando l’utente specifica chiaramente il nome;
- tramite riferimento posizionale, come “la prima”, “la seconda”, “l’ultima”, riferendosi alle ricette suggerite in precedenza;
- tramite contesto conversazionale, utilizzando lo slot `last_recipes`, che memorizza l’elenco delle ultime ricette proposte dal chatbot.

Questa strategia consente interazioni più naturali e riduce la necessità per l’utente di ripetere continuamente il nome della ricetta.

L’azione custom implementa la logica di recupero e visualizzazione degli ingredienti. Il flusso dell’azione è articolato nei seguenti passaggi:

1. Recupero del contesto: Viene letto lo slot `last_recipes` per gestire eventuali riferimenti posizionali e viene recuperato lo slot `recipe_name`, se presente.

2. Risoluzione dei riferimenti posizionali: Il testo dell’utente viene analizzato per individuare riferimenti come prima, seconda, ultima. Se trovato, il nome della ricetta viene risolto utilizzando l’indice corrispondente nella lista `last_recipes`.
3. Gestione dei casi ambigui: Se l’utente non specifica né un nome né una posizione, il chatbot chiede esplicitamente di indicare la ricetta desiderata. Se la ricetta non viene trovata nel dataset, viene restituito un messaggio di errore dedicato.
4. Recupero degli ingredienti: I dati della ricetta vengono ottenuti tramite una funzione helper. Gli ingredienti, già preprocessati in fase di preparazione del dataset, vengono formattati in modo leggibile per l’utente.
5. Risposta all’utente: Il chatbot restituisce la lista degli ingredienti, indicando anche il numero di persone o porzioni previste dalla ricetta.
6. Aggiornamento dello stato: Lo slot `recipe_name` viene sempre aggiornato con la ricetta corrente, consentendo richieste successive come la visualizzazione della preparazione.

La funzionalità è collegata a una story dedicata, che associa direttamente l’intent `ask_recipe_ingredients` all’azione custom. Questo garantisce un comportamento chiaro e coerente, indipendente dal percorso conversazionale precedente.

3.7 Ricerca diretta e logica di visualizzazione

Il chatbot consente all’utente di visualizzare la scheda completa di una ricetta, comprensiva di ingredienti e preparazione, sia tramite ricerca diretta per nome, sia tramite selezione da una lista di risultati precedentemente mostrata.

Questa funzionalità è stata progettata per gestire in modo fluido richieste esplicite, riferimenti impliciti e interazioni successive, mantenendo sempre coerenza conversazionale.

3.7.1 Ricerca diretta per nome

Quando l’utente indica esplicitamente il nome di una ricetta, l’intent `search_by_name` viene attivato e l’entità `recipe_name` viene estratta. L’azione cuore di questa parte è `action_smart_recipe_handler` gestisce la richiesta secondo due possibili scenari:

- una sola ricetta trovata con il nome esatto indicato: la scheda viene mostrata immediatamente;
- più ricette trovate che matchano con il nome indicato dall’utente: il chatbot presenta un elenco numerato di risultati e chiede all’utente di selezionarne una.

In caso di molti risultati, l’elenco viene limitato a un numero massimo (10) per evitare sovraccarico informativo, mantenendo comunque la naturalezza del dialogo.

3.7.2 Selezione da una lista

Se l’utente ha già ricevuto una lista di ricette, può selezionarne una usando riferimenti naturali come “la prima”, “la seconda”, “l’ultima” o numeri a parole o cifre. L’azione `ActionSelectByIndex` interpreta il riferimento numerico e aggiorna lo slot `recipe_name` con la ricetta selezionata, senza mostrare ancora i dettagli. Subito dopo, viene richiamata nuovamente `action_smart_recipe_handler`, che riconosce la selezione e mostra la scheda completa della ricetta.

3.7.3 Logica unificata

L'azione `action_smart_recipe_handler` funge da gestore intelligente centrale e opera in due fasi:

- Controllo del contesto: Se esiste una lista attiva (`last_recipes`), la selezione ha priorità assoluta. Lo slot `recipe_name` viene considerato affidabile solo se coerente con la lista corrente.
- Fallback a nuova ricerca: Se non c'è una lista valida o la richiesta è nuova, viene avviata una ricerca testuale nel dataset. Il chatbot decide automaticamente se mostrare subito la ricetta o proporre una selezione.

Questa struttura evita ambiguità, gestisce slot “sporchi” e permette all'utente di passare senza frizioni da una ricerca all'altra.

3.7.4 Visualizzazione della scheda della ricetta

Quando una ricetta viene selezionata, il chatbot mostra una scheda completa, composta da nome della ricetta, numero di persone o porzioni, lista degli ingredienti formattata, procedimento di preparazione. Lo slot `recipe_name` viene aggiornato con la ricetta visualizzata, consentendo anche in questo caso come per gli altri intent, interazioni successive come richiesta degli ingredienti, consultazione della preparazione e azioni future (es. lista della spesa) che saranno illustrate in seguito.

3.8 Suggerimento ricette simili

Il chatbot offre la possibilità di suggerire ricette simili a quella attualmente visualizzata, basandosi sulla somiglianza degli ingredienti. Questa funzionalità permette di proseguire l'esplorazione in modo naturale, senza richiedere all'utente una nuova ricerca esplicita.

Il suggerimento di ricette simili si basa sullo slot `recipe_name`, che contiene l'ultima ricetta visualizzata. Se questo contesto non è disponibile, il chatbot chiede all'utente di specificare una ricetta di riferimento, evitando suggerimenti fuori contesto.

Una volta individuata la ricetta di partenza:

1. viene richiamata la funzione `get_similar_recipes_by_ingredients`;
2. se vengono trovate ricette simili, il chatbot restituisce una lista di proposte;
3. l'elenco viene salvato nello slot `last_recipes`, permettendo all'utente di selezionare una ricetta tramite indice o nome, come nelle altre funzionalità del sistema.

La funzione `get_similar_recipes_by_ingredients` confronta la ricetta di riferimento con tutte le altre presenti nel dataset seguendo questi passaggi:

1. estrazione degli ingredienti della ricetta base e delle altre ricette;
2. rimozione di ingredienti troppo generici (es. sale, acqua, olio) tramite una lista di **stop ingredients**;
3. confronto tra gli insiemi di ingredienti usando una misura di similarità basata sull'intersezione tra gli insiemi;
4. selezione delle ricette che superano una soglia minima di similarità.

Le ricette vengono poi ordinate in modo deterministico in base al punteggio di similarità e vengono restituite solo le migliori N proposte sottoforma di elenco puntato in modo tale che l'utente può navigare con le modalità già viste.

```
STOP_INGREDIENTS = [
    "sale", "sale fino", "sale grosso", "pepe", "pepe nero",
    "acqua", "olio", "olio extravergine d'oliva", "olio evo", "olio di oliva"
]
```

Figura 3.5: Definizione degli stop ingredients

3.9 Ricerca guidata

Per gestire situazioni di indecisione dell’utente o richieste poco strutturate, è stata implementata una ricerca guidata basata su Form di Rasa. Questa modalità trasforma il chatbot in un vero assistente decisionale, capace di raccogliere progressivamente i criteri di ricerca, riassumendo in questo punto tutte le funzionalità già viste.

La ricerca guidata viene attivata tramite l’intent `guided_search`, che intercetta richieste esplicite di aiuto, frasi di indecisione e messaggi contenenti già uno o più parametri (categoria, difficoltà, ingredienti, numero di persone). All’attivazione, l’azione `ActionStartGuidedSearchForm` resetta tutti gli slot rilevanti e attiva il form `guided_search_form`, garantendo una ricerca pulita e coerente.

Il form richiede i seguenti slot:

- categoria del piatto
- difficoltà
- ingredienti disponibili
- numero di persone

Il form è flessibile: l’utente può anche saltare alcuni filtri indicando "qualsiasi" alla richiesta , mantenendo l’esperienza naturale e non rigida.

La classe `ValidateGuidedSearchForm` gestisce la validazione degli slot con logiche specifiche:

- Categoria e difficoltà: Vengono normalizzate e confrontate con i valori reali presenti nel dataset. È possibile indicare risposte come “qualsiasi” o “indifferente”.

- Ingredienti: Supporta input multipli, sia come entità che come testo libero. Gli ingredienti vengono normalizzati in una lista, consentendo un filtraggio flessibile.
- Numero di persone: Accetta numeri, parole (“due”, “quattro”) e gestisce casi di incertezza impostando un valore di default. Sono inoltre applicati controlli di validità per evitare valori non realistici.

Una volta compilati tutti gli slot, il form viene chiuso e viene eseguita l’azione `ActionSubmitGuidedSearch`, che:

1. recupera i filtri selezionati;
2. disattiva quelli impostati come “any”;
3. esegue la funzione `search_recipes_guided`, che combina tutti i criteri;
4. restituisce all’utente una lista di ricette compatibili.

I risultati vengono salvati nello slot `last_recipes`, permettendo di riutilizzare le stesse logiche di selezione già presenti nel chatbot.

3.10 Gestione della lista della spesa

Il chatbot include una lista della spesa persistente, che permette all’utente di salvare, consultare e gestire gli ingredienti necessari alle ricette visualizzate. La lista viene mantenuta nello slot `shopping_list`, uno slot custom che non influenza il flusso conversazionale, ma funge da memoria temporanea dell’utente.

Quando l’utente attiva l’intent `add_to_shopping_list`, il sistema avvia una procedura di aggiornamento dinamico strutturata in tre fasi:

1. **Recupero del contesto:** Il sistema identifica la ricetta attualmente discussa tramite lo slot `recipe_name`.

2. **Caricamento dati:** Vengono estratti gli ingredienti corrispondenti dal dataset normalizzato.
3. **Merge intelligente:** Gli ingredienti vengono integrati nella lista esistente dell'utente tramite la funzione `merge_shopping_lists`.

La funzione `merge_shopping_lists` è progettata per mantenere la lista concisa e leggibile, evitando duplicati ridondanti. La logica segue queste regole di business:

- **Somma quantitativa:** Se un ingrediente è già presente e le unità di misura sono compatibili (es. grammi con grammi), le quantità vengono sommate algebricamente.
- **Gestione incompatibilità:** Se le unità non sono confrontabili (es. "q.b." rispetto a "grammi"), il sistema evita calcoli errati e procede ad accodare le descrizioni in modo testuale.

La funzione `parse_quantity` consente di interpretare quantità scritte in modo naturale: 200g, 1.5 kg, 2 cucchiai come quantità numerica + unità oppure q.b. o testi non numerici gestiti come casi speciali. Questo permette al chatbot di fare calcoli quando possibile, senza perdere informazioni nei casi ambigui.

Con l'intent `show_shopping_list`, l'utente può visualizzare la lista della spesa in qualsiasi momento. La risposta mostra ogni ingrediente una sola volta, presenta le quantità sommate e pulite quando possibile e mantiene una formattazione leggibile e orientata all'uso pratico (spesa reale). Se la lista è vuota, il bot lo segnala in modo esplicito.

Tramite l'intent `clear_shopping_list`, l'utente può poi cancellare completamente la lista e ripartire da zero dopo aver completato la spesa. L'azione associata azzerà semplicemente lo slot `shopping_list`, garantendo un reset immediato e sicuro.

3.11 Gestione di fallback

Il fallback entra in gioco quando l’NLU non riesce a classificare l’intent con sufficiente confidenza oppure il messaggio dell’utente non rientra in nessun flusso gestito. In questi casi Rasa attiva automaticamente l’intent `nlu_fallback`.

La risposta arriva grazie all’azione `utter_default`:

```
utter_default:  
- text: "Scusa, non ho capito cosa intendi 😞. Posso cercare ricette, mostrarti il carrello o aiutarti a scegliere."  
- text: "Mmm, questo è fuori dalla mia portata culinaria 🍔. Prova a chiedermi una ricetta!"
```

Figura 3.6: Utter di default

Capitolo 4

Test del chatbot

Lo scopo della fase di testing è verificare il corretto funzionamento del chatbot in scenari realistici di utilizzo, valutando sia la corretta classificazione degli intenti sia il comportamento delle azioni personalizzate e la coerenza del flusso conversazionale. Per effettuare questa fase in un contesto reale di messaggistica, il sistema è stato collegato alla piattaforma **Telegram** tramite un bot dedicato. La connessione è avvenuta creando un bot Telegram attraverso il servizio **BotFather**, che fornisce un token di autenticazione utilizzato per autorizzare la comunicazione tra Telegram e il backend del chatbot.

Il chatbot, sviluppato con Rasa, è stato quindi configurato per utilizzare il Telegram Connector messo a disposizione dal framework. Tale connettore consente di ricevere i messaggi inviati dagli utenti su Telegram, inoltrarli al motore NLU e di dialogo, e restituire le risposte generate direttamente all'interno della chat.

Questa integrazione permette al chatbot di funzionare come un normale bot Telegram, mantenendo inalterata la logica conversazionale e rendendo il testing più fedele a un utilizzo reale da parte degli utenti.

Di seguito il test delle funzionalità descritte nel capitolo precedente

4.1 Test di Benvenuto e Onboarding

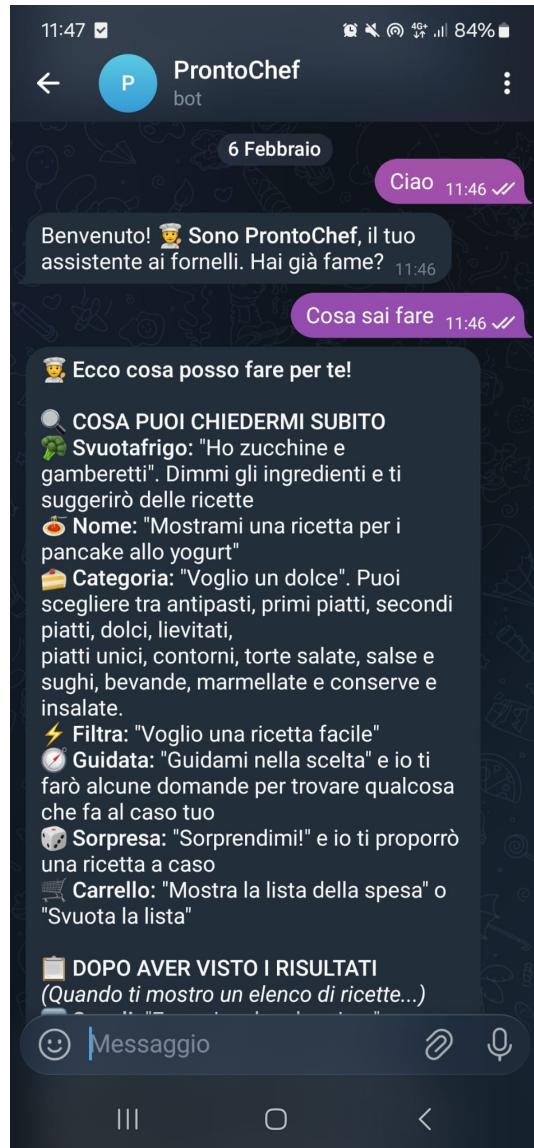


Figura 4.1: Saluto e Help (Pt.1)

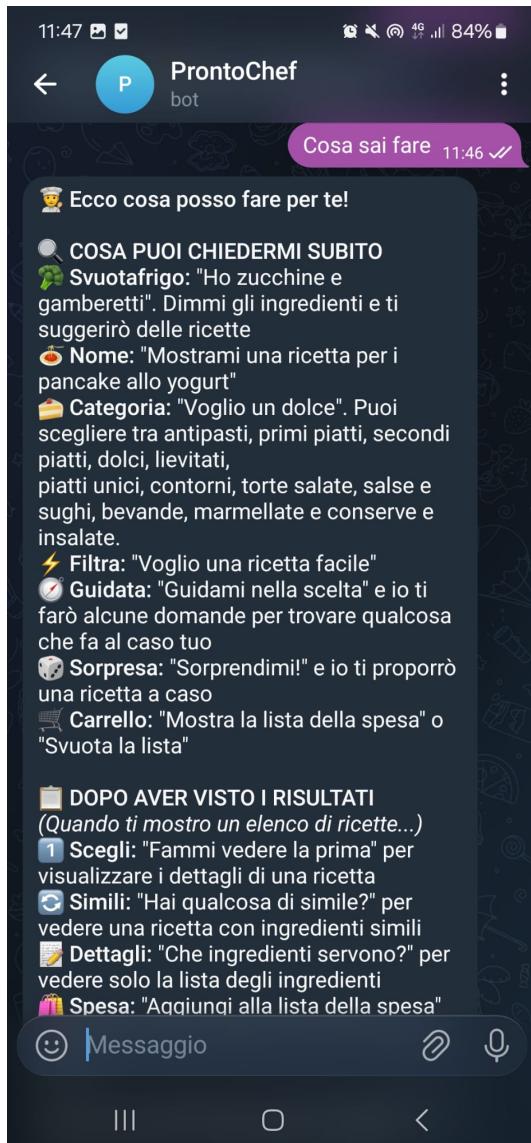


Figura 4.2: Saluto e Help (Pt.2)

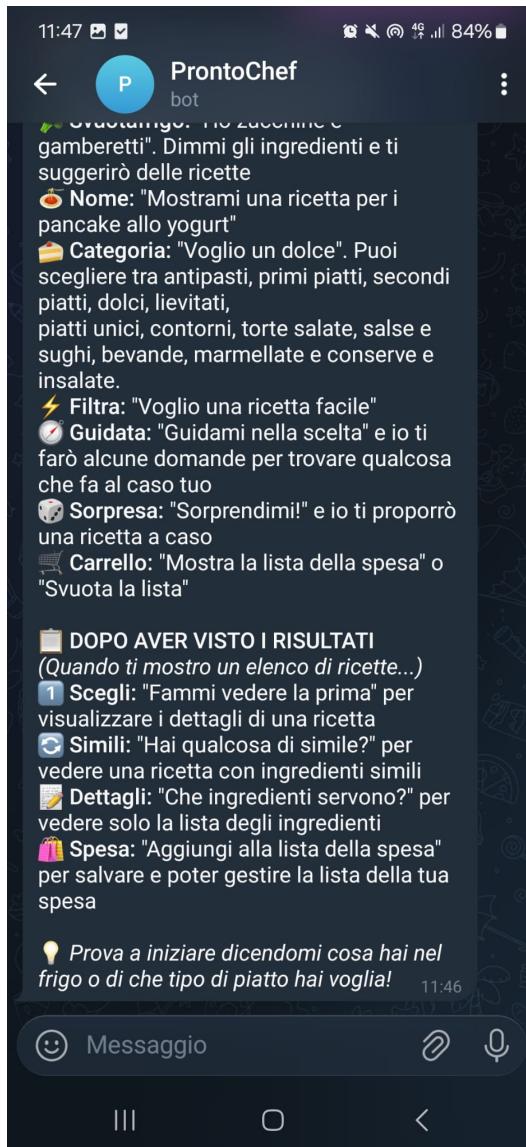


Figura 4.3: Saluto e Help (Pt.3)

Come si nota, il bot risponde al saluto presentandosi con una frase che punta a mettere subito a proprio agio l'utente, cercando di stimolare la sua curiosità nell'usare il bot. Alla richiesta di cosa sappia fare risponde invece con un elenco dettagliato, ma comunque chiaro, delle proprie funzionalità e formattato in modo schematico, dividendo l'output in due sezioni: "Cosa puoi chiedermi subito" e "Dopo aver visto i risultati". Inoltre l'uso delle emoji, che sarà frequente in tutte le conversazioni, punta a rendere ancora più efficace, chiara e accattivante la comunicazione con l'utente.

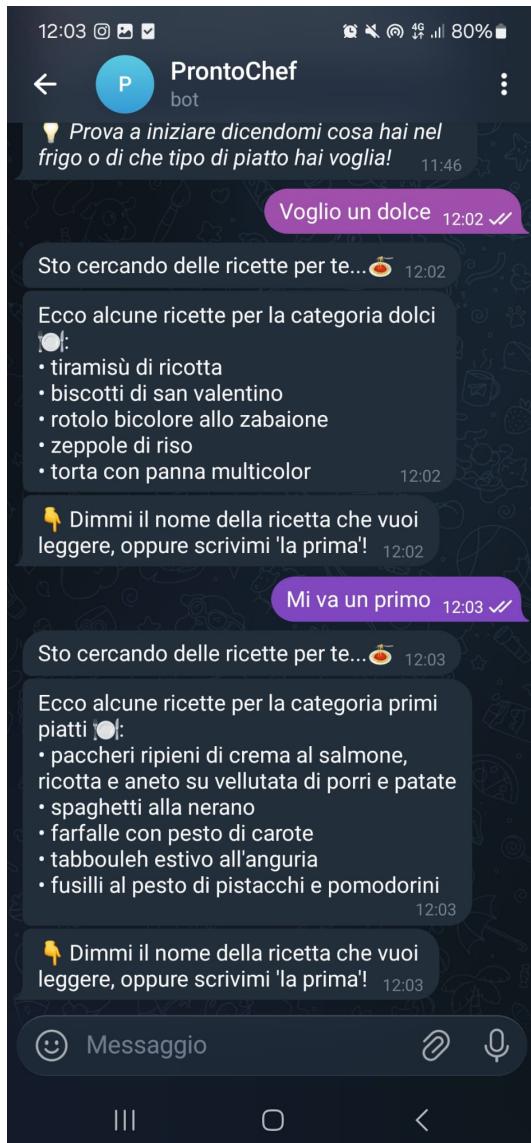


Figura 4.4: Ricerca per categoria

4.2 Test ricerca per categoria

La figura mostra due esempi per la ricerca per categoria. Il chatbot è in grado di capire e di estrarre correttamente l'entità `category` anche se ovviamente non matcha perfettamente con quella indicata nel dataset. Così anche se l'utente si limita a dire "primo", grazie anche all'uso di sinonimi in `nlu.yml`, il chatbot comprende che si tratta della categoria "primi piatti". In ogni caso la risposta è composta da tre messaggi:

- Messaggio che annuncia l'avvio della ricerca, per fornire subito un feedback immediato all'utente, evitando l'impressione di mancata risposta nel tempo che trascorre tra la richiesta e il recupero delle informazioni dal dataset e quindi la risposta;
- Messaggio che illustra un elenco di 5 ricette appartenenti a quella categoria sottoforma di elenco puntato;
- Messaggio conclusivo che invita l'utente a rispondere in modo corretto per selezionare una ricetta (indicando il nome o la posizione), evitando possibili errori.

4.3 Test selezione ricetta

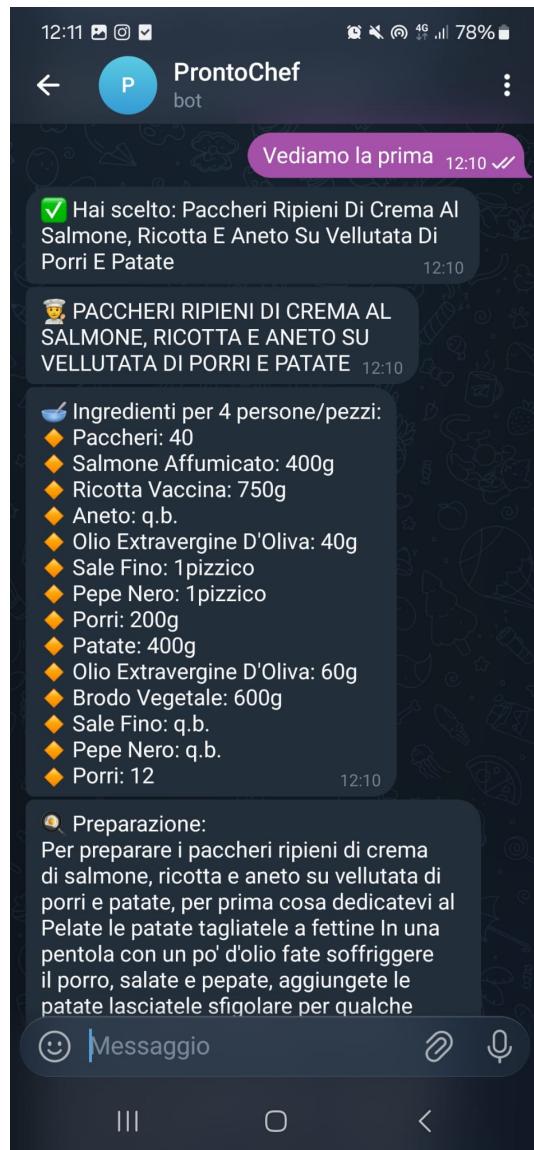


Figura 4.5: Selezione ricetta per posizione

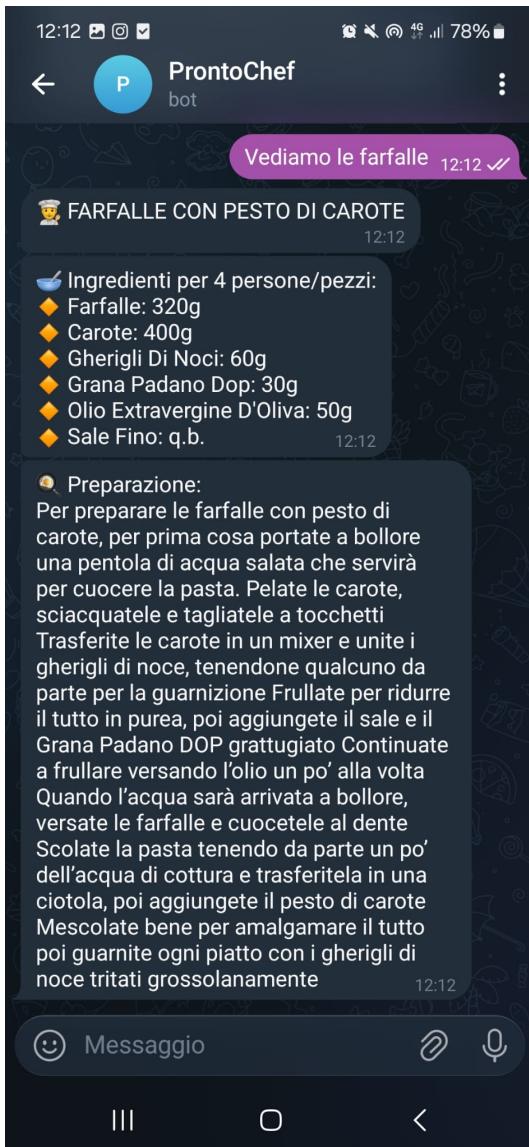


Figura 4.6: Selezione ricetta per nome

Dopo che il chatbot propone un qualsiasi elenco di ricette derivante da una qualsiasi ricerca, l'utente può visualizzare la scheda della ricetta selezionandola in due modi: per posizione e per nome. La prima figura mostra la funzionalità per posizione che permette subito all'utente di indicare la ricetta senza dover specificare il nome. Il chatbot risponde con un messaggio che conferma la selezione e indica il nome completo della ricetta selezionata, per dare un feedback della selezione all'utente. Dopodiché il dettaglio della ricetta viene fornita in tre messaggi: titolo, ingredienti formattati in modo schematico e con

riferimento al numero di persone/pezzi e poi l'intera preparazione.

La seconda figura mostra invece come, anche subito dopo aver visualizzato una ricetta, grazie al mantenimento dello stato della risposta con `last_recipes`, è possibile continuare a navigare tra le ricette proposte. In questo caso la selezione avviene per nome ed è importante notare come non sia necessario specificare il nome intero per la selezione, ma basta anche una o più parole contenute nel titolo, sufficienti a disambiguare la selezione. Anche in questo caso il bot risponde col feedback sulla selezione e la scheda completa.

4.4 Test ricerca per ingredienti

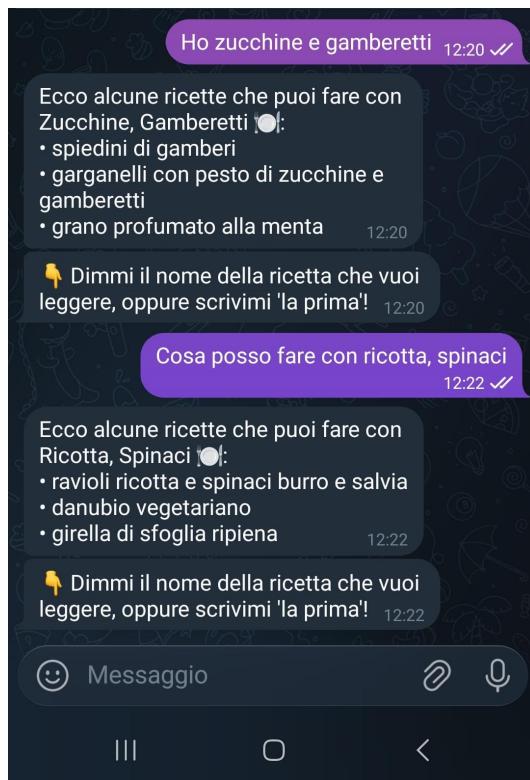


Figura 4.7: Ricerca per ingredienti

La figura mostra due esempi di ricerca per ingredienti. Anche in questo caso la risposta prevede un elenco di ricette che contiene tutti gli ingredienti specificati e che possono essere selezionate con i modi già visti. La risposta include il

riepilogo degli ingredienti che il chatbot ha compreso, per dare un feedback all'utente che sarà in grado di capire se il chatbot è riuscito a comprendere correttamente il messaggio inviato.

4.5 Test proposta ricetta casuale

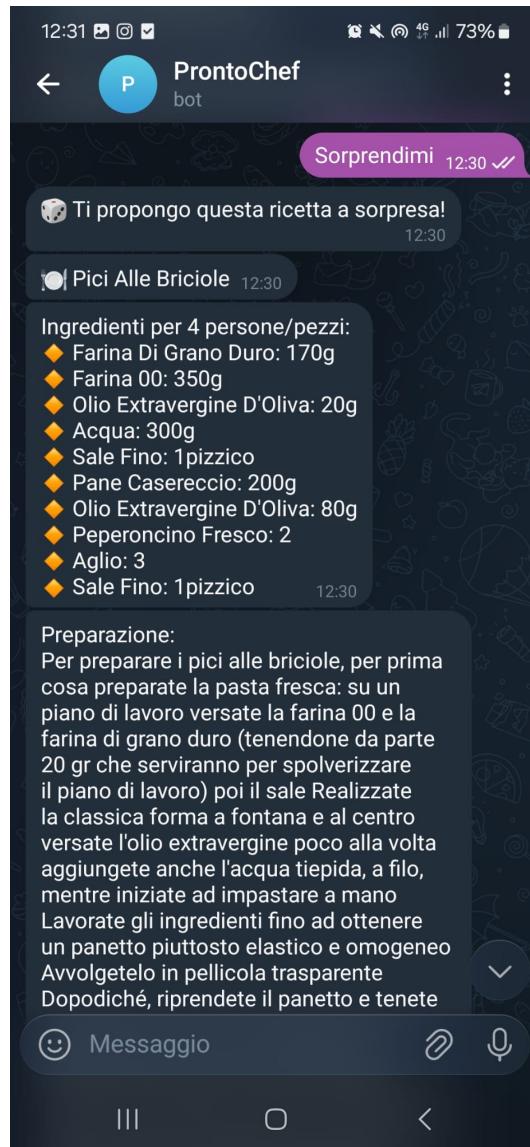


Figura 4.8: Proposta ricetta casuale

Quando l'utente scrive "Sorprendimi" o qualcosa di assimilabile ad una ricetta casuale, il bot recupera una ricetta dal dataset e risponde con la scheda

completa, indicando prima "Ti propongo questa ricetta a sorpresa!"

4.6 Test ricerca per difficoltà

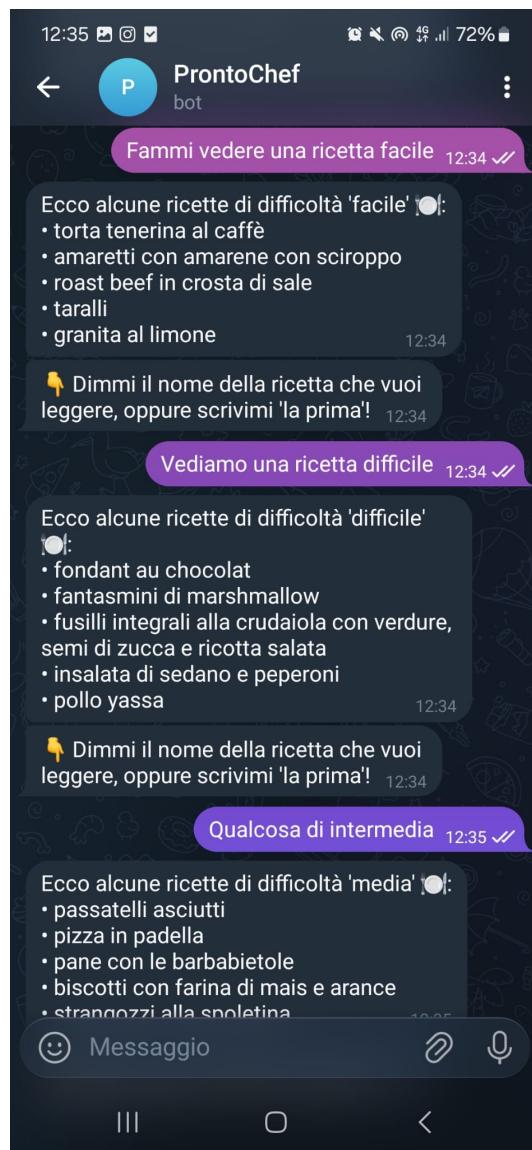


Figura 4.9: Ricerca per difficoltà

La figura mostra la ricerca per difficoltà. Si noti come anche in questo caso, l'uso dei sinonimi nel codice garantisce la possibilità di ottenere il risultato senza problemi. La risposta avviene sottoforma di elenco come per tutti gli altri casi.

4.7 Test visualizzazione ingredienti

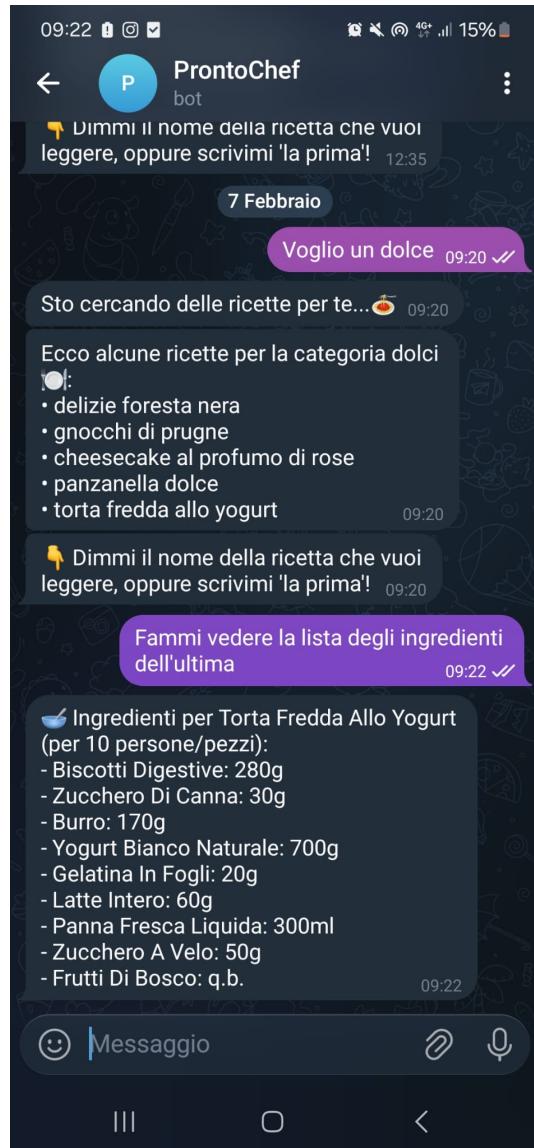


Figura 4.10: Visualizzazione ingredienti

L'immagine mostra come sia possibile anche richiedere semplicemente la lista degli ingredienti di una ricetta, anziché visualizzare la scheda completa, evitando di ricevere una risposta eccessivamente lunga dal bot.

4.8 Test ricerca diretta per nome

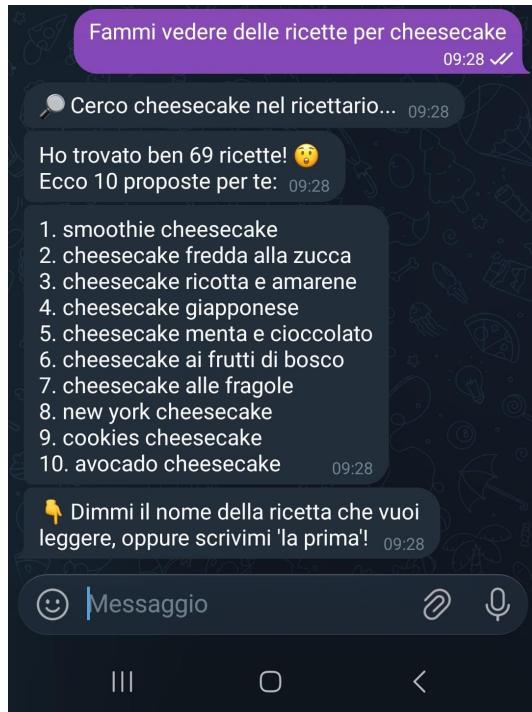


Figura 4.11: Ricerca per nome, caso 1

La figura mostra la ricerca diretta per nome nel caso in cui il nome indicato non sia però il nome esatto di una ricetta nel dataset. Quello che succede è che l'action già precedentemente descritta ricerca comunque qualcosa che matchi nel ricettario con quello che richiede l'utente, in questo caso "cheesecake", e risponde riepilogando quante proposte di questo tipo ha trovato (69 nell'esempio), visualizzando nella lista solo 10 di queste, casualmente, per facilità di navigazione.

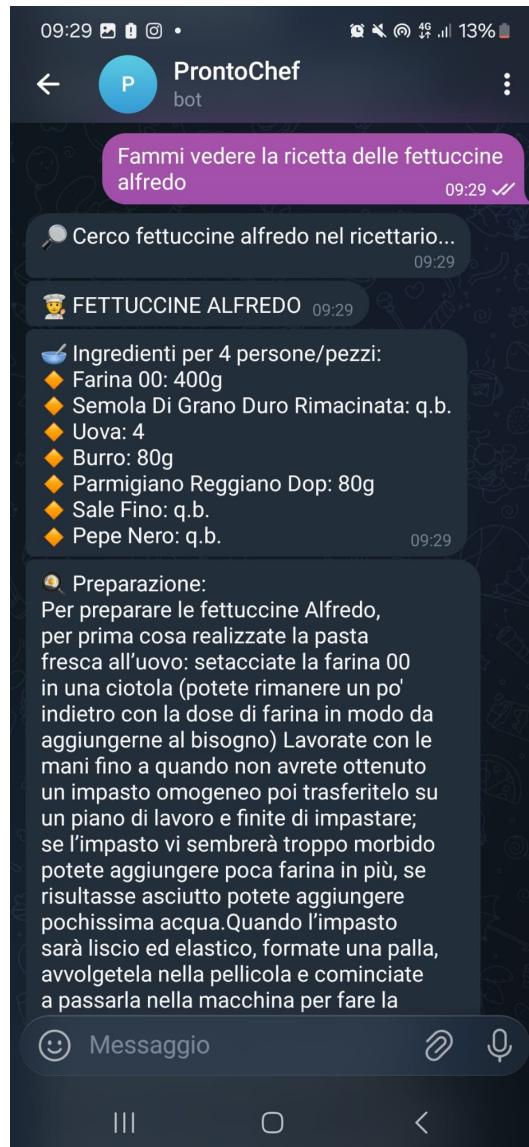


Figura 4.12: Ricerca per nome, caso 2

In questo caso l'utente ha specificato il nome completo di una ricetta presente nel ricettario, pertanto la ricerca non è ambigua e dà come esito esattamente la ricetta cercata che viene subito visualizzata.

4.9 Test suggerimento ricette simili

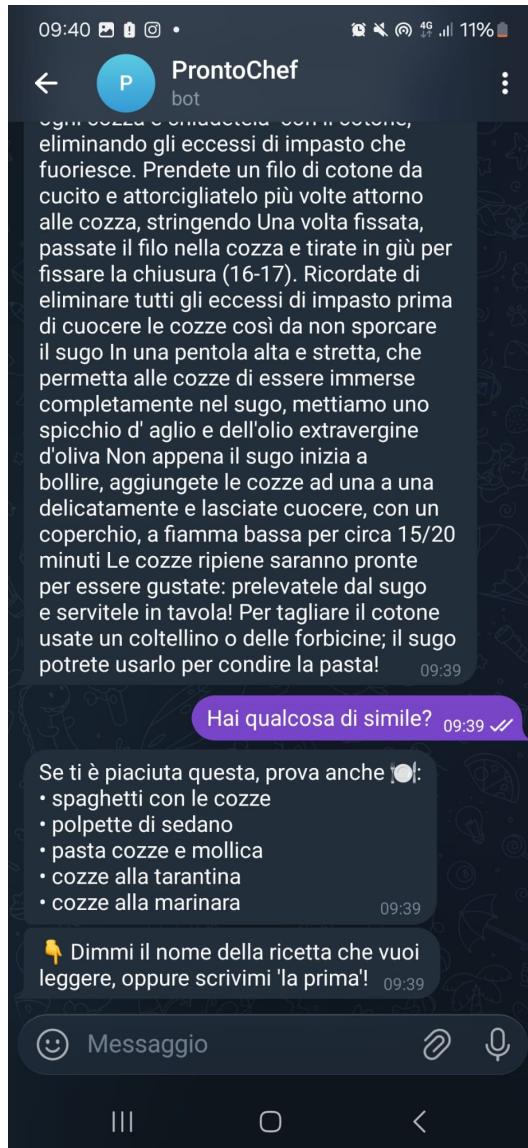


Figura 4.13: Ricerca ricette simili

La figura mostra il risultato della richiesta di ricette simili dopo aver visualizzato precedentemente la ricetta "Cozze ripiene". La lista di ricette proposte sono ricette simili a testimonianza della funzionalità dell'intent. L'unica ricetta che ad occhio potrebbe sembrare poco correlata è "polpette di sedano", ma la sua presenza si spiega con il fatto che tale ricetta presenta praticamente tutti

gli ingredienti delle "cozze ripiene", ad eccezione proprio delle cozze. Pertanto viene comunque classificata simile.

4.10 Test della ricerca guidata

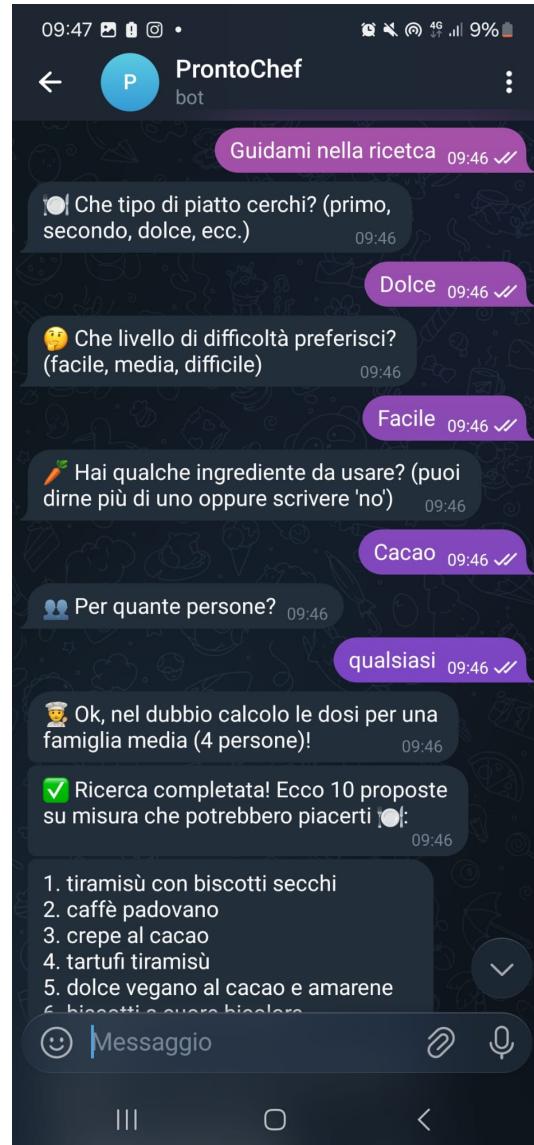


Figura 4.14: Ricerca guidata (Pt.1)

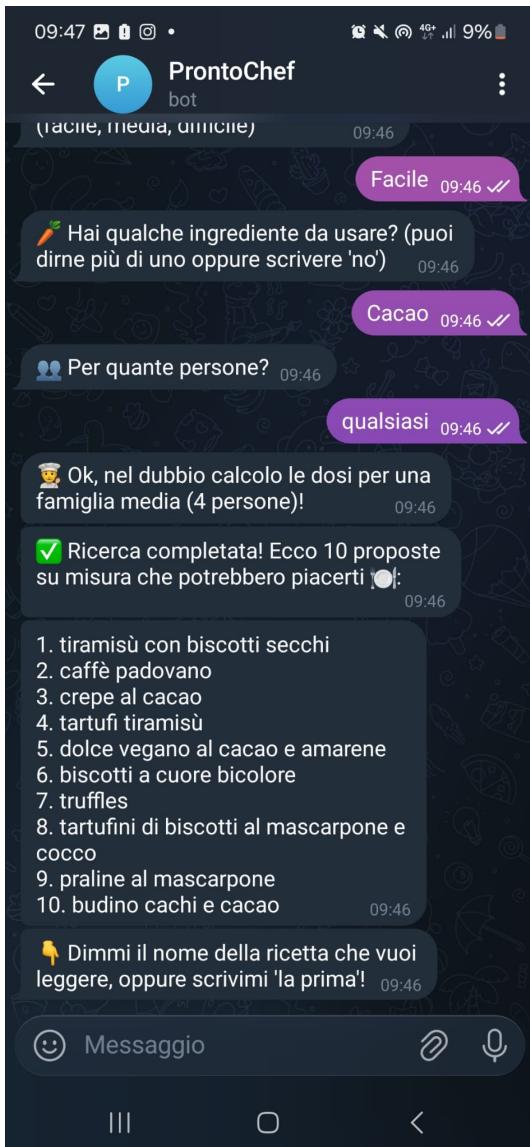


Figura 4.15: Ricerca guidata (Pt.2)

Le figure mostrano il `rasa form` in funzione. Questo viene invocato con "Guidami nella ricerca" e inizia a porre le domande per riempire gli slot già discussi in fase di sviluppo. Da notare che anche se non viene specificato il numero di persone/pezzi, il bot calcola automaticamente 4 persone, comunicandolo all'utente. La ricerca è mostrata in una lista classica e contiene una selezione di ricette che sono contemporaneamente dolci, facili, con cacao e per 4 persone (base).

4.11 Test gestione della lista della spesa



Figura 4.16: Lista della spesa (Pt.1)

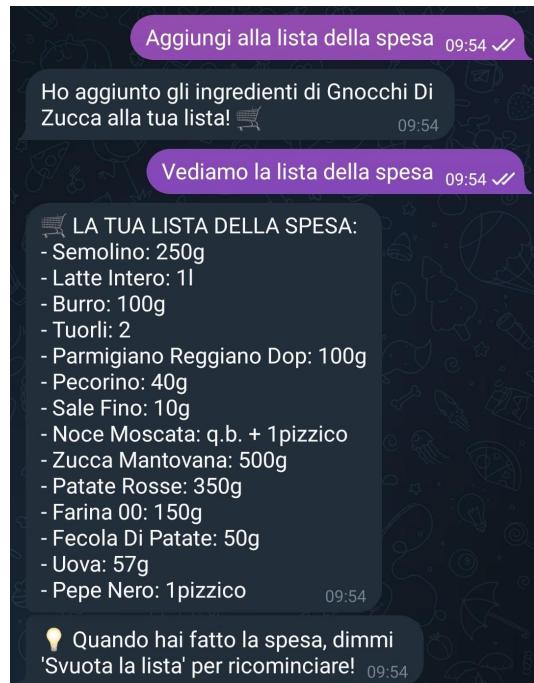


Figura 4.17: Lista della spesa (Pt.2)



Figura 4.18: Lista della spesa (Pt.3)

Le tre figure mostrano la gestione completa della lista della spesa. Si richiede di aggiungere due ricette alla lista e di visualizzare la stessa subito dopo. Si vede che la lista si riempie con gli ingredienti necessari. Infine alla richiesta di svuotamente la lista risulta poi vuota.

4.12 Test di fallback



Figura 4.19: Meccanismo di fallback

In figura si nota come il bot non sospenda la conversazione non dando risposta, ma piuttosto specifica che non sta comprendendo l'utente, quando quest'ultimo scrive qualcosa di incomprensibile al bot. Si attiva così la risposta di fallback come in figura.