Telegram Bot for Cisco Error Messages

DEMONSTRATION AND OVERVIEW WITH PROMPT ENGINEERING INSIGHTS

Introduction

- This presentation showcases a Telegram bot designed to provide support for Cisco error messages.

- Highlighting the importance of prompt engineering in bot development.

Features

- The bot provides error code details, regional support contacts, and detailed descriptions of error messages.
- Effective prompts are used to enhance user interaction.

Technologies Used

- Technologies:
Python, Telegram Bot API, Pandas, FuzzyWuzzy.

- Prompt engineering is crucial for guiding user interactions and bot responses.

Telegram Bot Setup

- Steps to set up the Telegram Bot API and obtain a token.
- Creating effective prompts for user interaction.

Code Overview

- Main script sections: imports, configuration, and core functions.

- **Prompt Engineering Insight**: How prompts guide the bot's behavior and improve user interactions.

```python
import logging
from telegram import Update, InlineKeyboardButton, InlineKeyboardMarkup, ReplyKeyboardMarkup
from telegram.ext import Updater, CommandHandler, CallbackQueryHandler, MessageHandler, Filters, CallbackContext
from fuzzywuzzy import fuzz
import pandas as pd

# Configure logger
logging.basicConfig(level=logging.INFO)

# Tokens and API keys
TELEGRAM_TOKEN = "TELEGRAM_TOKEN"
EXCEL_PATH = 'output.xlsx'
```

Loading and Normalizing Data

- How data is loaded from an Excel file.
- Normalizing error IDs for effective searches.

```python
# Load Excel data function
def load_excel_data(excel_path):
    try:
        df = pd.read_excel(excel_path)
        return df
    except Exception as e:
        logging.error(f"Error loading Excel file: {e}")
        return pd.DataFrame()  # Return empty DataFrame if loading fails

# Load data from Excel
df = load_excel_data(EXCEL_PATH)

# Function to normalize error ID
def normalize_error_id(error_id):
    if isinstance(error_id, str):
        return error_id.replace("EID", "").replace("WID", "").replace("-", "").strip()
    else:
        return str(error_id).replace("EID", "").replace("WID", "").replace("-", "").strip()

```

Error Search Functions

- Functions to find errors by ID, description, and message.

- **Prompt Engineering Insight:** How these functions use prompts to improve accuracy.

```python
# Find error by ID
def find_error_by_id(df, error_id):
    if df.empty:
        return None
    normalized_id = normalize_error_id(error_id)
    row = df[df['Error or Warning ID'].apply(normalize_error_id) == normalized_id]
    return row if not row.empty else None

# Find error by description
def find_error_by_description(df, description):
    if df.empty:
        return None
    row = df[df['Description'].str.contains(description, case=False, na=False)]
    return row if not row.empty else None

# Find error by message
def find_error_by_message(df, message):
    if df.empty:
        return None
    row = df[df['Error or Warning Message'].str.contains(message, case=False, na=False)]
    return row if not row.empty else None
```

Display Contacts Function

- Function to display regional support contacts.
- **Prompt Engineering Insight:** How regional prompts enhance user experience.

```python
# Function to display contacts
def display_contacts(region):
    contacts = {
        "North America": (
            "United States\n"
            "1 800 553 2447\n"
            "1 408 526 7209\n\n"
            "Canada\n"
            "English:\n"
            "1 800 553 2447\n"
            "1 408 526 7209\n"
            "French: (Interpreter service available upon request.)\n"
            "1 800 553 6387 - Option 3"
        ),
        "Africa": (
            "Africa & Indian Ocean Islands\n"
            "+32 2 704 5555\n\n"
            "South Africa\n"
            "0800 982650"
        ),
        und soweiter .............
    }
    return contacts.get(region, "Contacts not found for this region.")
```

Description Command Handler

- Command handler to display error and warning descriptions.

- **Prompt Engineering Insight:** How the description prompt provides valuable information.

```python
# Description command handler
def show_description(update: Update, context: CallbackContext):
    description_text = (
        "This chapter lists the Cisco ONS 15454, ONS 15454 SDH, ONS 15600, ONS 15327 and ONS 15310-CL error messages. "
        "Two types of messages: error messages (EID-nnnn) and warning messages (WID-nnnn). "
        "Error messages are an alert that an unexpected or undesirable operation has occurred that either indicates the risk
        "or an inability to properly manage devices in the network. Warnings are an alert that the requested operation could
        "Warnings are sometimes used to convey important information."
    )
    update.message.reply_text(description_text)
```

Start Command Handler

- Initial user interaction with the bot.

- **Prompt Engineering Insight:** How the start command guides user choices.

```python
# Start command handler
def start(update: Update, context: CallbackContext):
    keyboard = [
        ["New Query", "Cisco support in your Region", "Show Description"]
    ]
    reply_markup = ReplyKeyboardMarkup(keyboard, one_time_keyboard=True)
    update.message.reply_text('Please choose an option:', reply_markup=reply_markup)
```

Main Message Handler

- Handling user queries and interactions.

- **Prompt Engineering Insight:** How prompts guide users through various options.

```python
# Main handler for user messages
def handle_message(update: Update, context: CallbackContext):
    text = update.message.text
    chat_id = update.message.chat_id
    if text == "New Query":
        update.message.reply_text('Please enter the error code or description:')
    elif text == "Cisco support in your Region":
        regions = [
            ["North America", "Africa"],
            ["Asia Pacific", "Europe"],
            ["Latin America", "Middle East / Central Asia"]
        ]
        reply_markup = ReplyKeyboardMarkup(regions, one_time_keyboard=True)
        update.message.reply_text('Please choose your region:', reply_markup=reply_markup)
    elif text in ["North America", "Africa", "Asia Pacific", "Europe", "Latin America", "Middle East / Central Asia"]:
        contacts = display_contacts(text)
        send_message(chat_id, contacts, context)
        start(update, context)  # Show the main menu again after displaying contacts
    elif text == "Show Description":
        show_description(update, context)
    else:
        # Search for error by ID or description
        row_by_id = find_error_by_id(df, text)
        row_by_desc = find_error_by_description(df, text)

        if row_by_id is not None:
            result = row_by_id.iloc[0].to_dict()
            update.message.reply_text(f"Error found: {result}")
        elif row_by_desc is not None:
            result = row_by_desc.iloc[0].to_dict()
            update.message.reply_text(f"Error found: {result}")
        else:
            update.message.reply_text("Sorry, I didn't understand that command.")
```

Code Overview

- Main function to start the bot.
- **Prompt Engineering Insight:** Ensuring seamless bot operation.

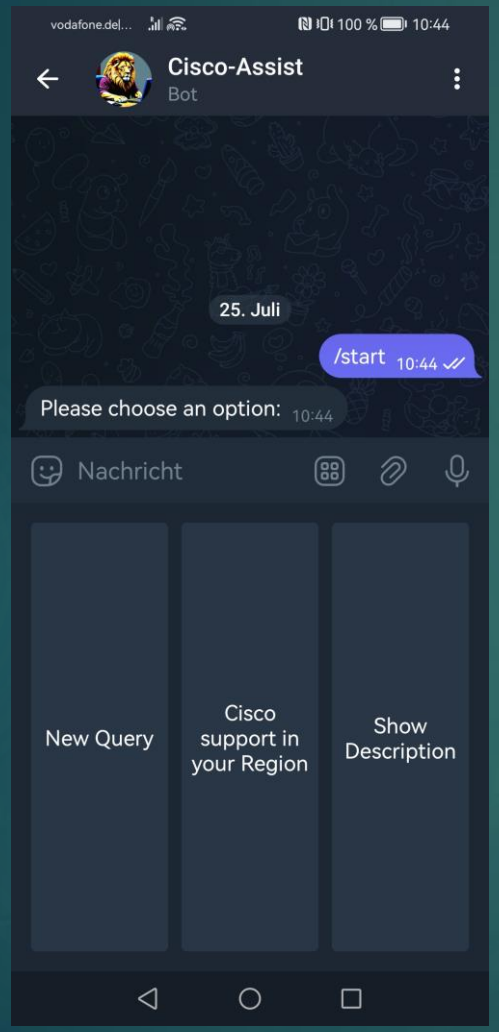
```python
# Main function to start the bot
def main():
    updater = Updater(TELEGRAM_TOKEN, use_context=True)
    dp = updater.dispatcher

    dp.add_handler(CommandHandler("start", start))
    dp.add_handler(MessageHandler(Filters.text & ~Filters.command, handle_message))

    updater.start_polling()
    updater.idle()

if __name__ == '__main__':
    main()
```

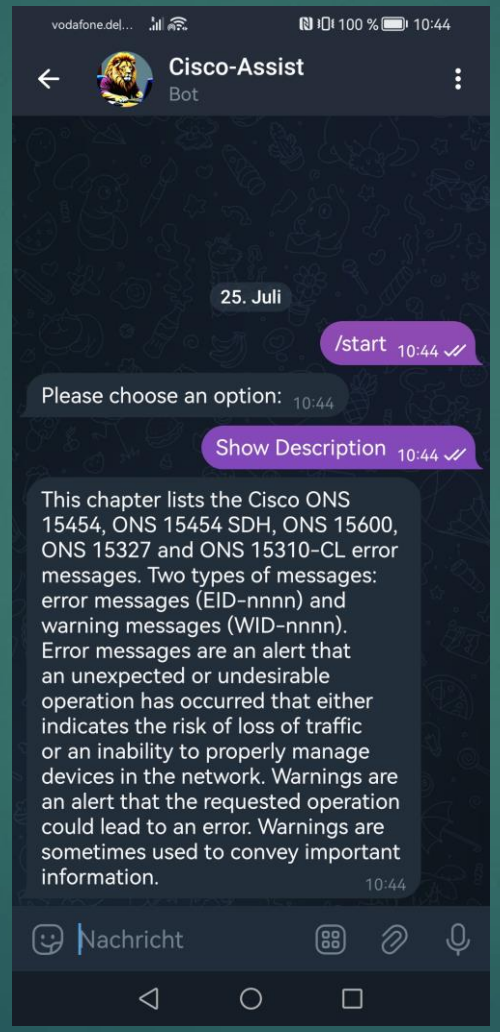
Bot Demonstration

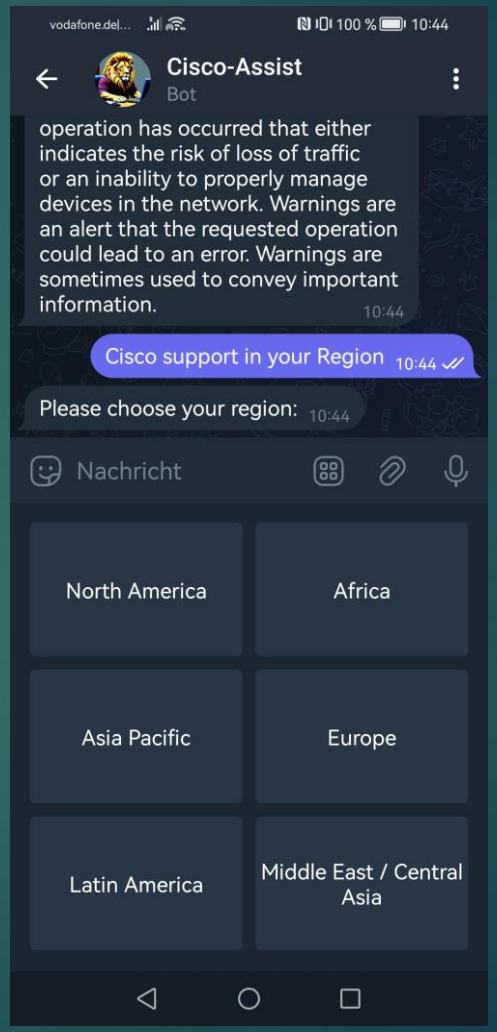
- Showcase of the bot in action.

- Screenshots of various interactions:
- Starting the bot.
- Entering an error code.
- Receiving error details.
- Getting regional support contacts.

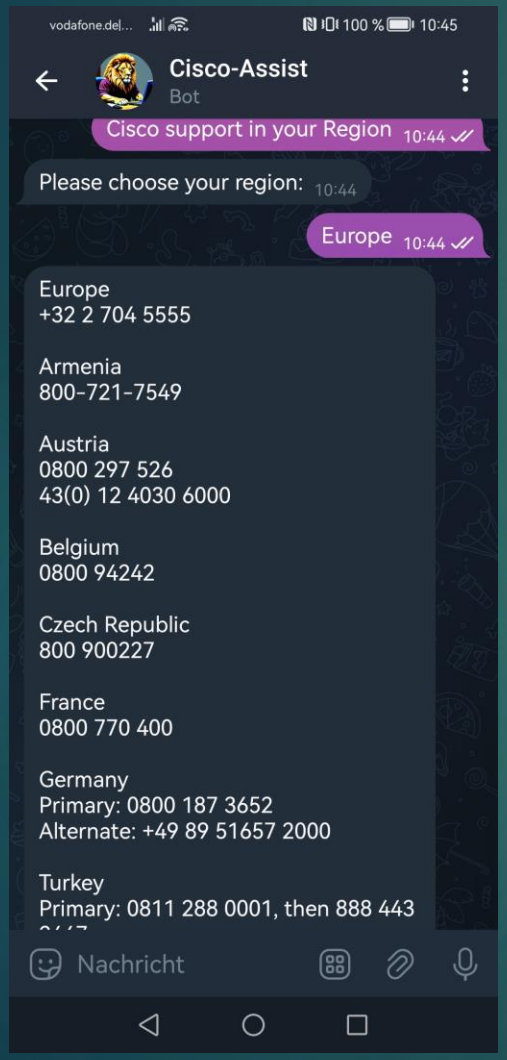
Bot Demonstration

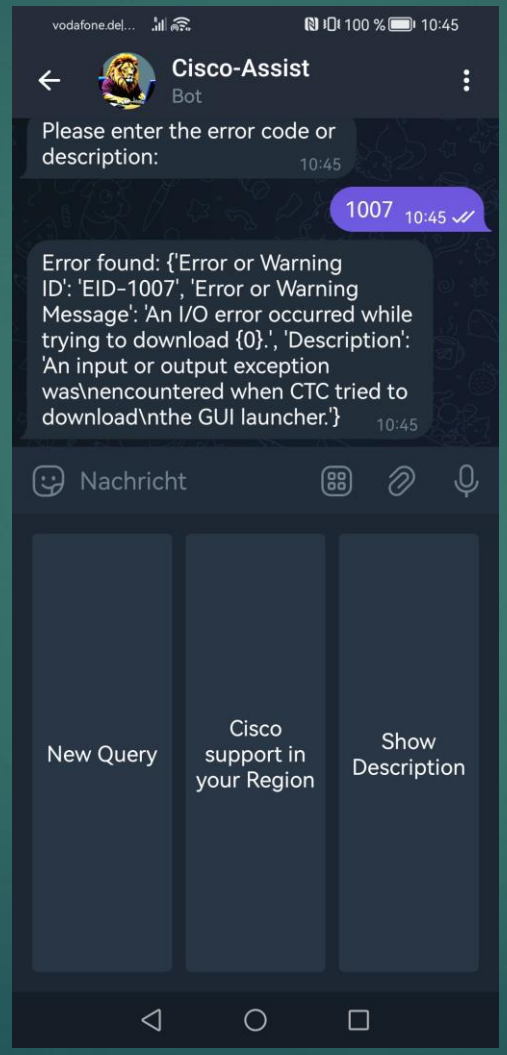


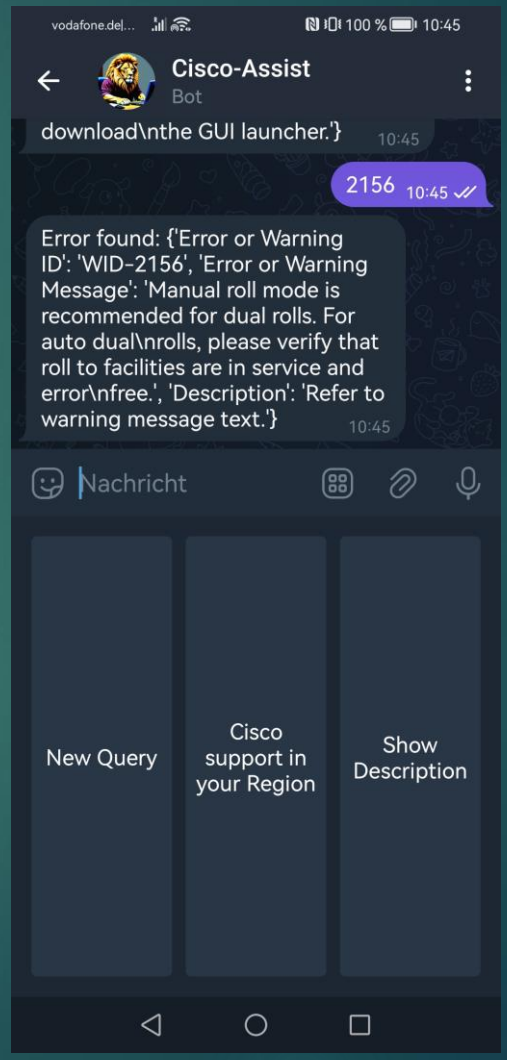




Bot Demonstration







Conclusion and Summary

- **Achieved Results:**
- **Effective Solution:** We developed a Telegram bot that successfully helps users find information about Cisco errors and support contact details.
- **Integration of Technologies:** Utilizing Python, Pandas, and FuzzyWuzzy enabled the creation of a functional and responsive tool.
- **User-Friendly Interface:** Interactive keyboards and command handling make the bot convenient to use.
- **Prospects of Prompt Engineering:**
- **Enhanced AI Interaction:** Prompt engineering allows for creating more accurate and useful AI responses, improving the quality of interactions.
- **Adaptive Algorithms:** Prompt engineering can help adapt the bot to various user needs and specific scenarios.
- **Expanding Functionality:** Future additions may include handling more complex queries and integration with other systems for real-time data retrieval.

What's the next?

- **Future Steps:**
- **Further Testing and Improvements:** Continuous testing and enhancement of the bot based on user feedback.
- **Expanding Knowledge Base:** Updating and expanding the database of errors and contact information.
- **Integration of New Technologies:** Exploring and implementing new technologies and approaches in prompt engineering to increase efficiency.

Thank you for your attention!