Alitha_ChatBot_Final Project

Step 1 — Imports, Configuration, and Seeding

In [1]:
```
import matplotlib.pyplot as plt
import nltk
import numpy as np
import os, re, json, random, math, warnings, string
import pandas as pd
import re,string
import seaborn as sns
import tensorflow as tf

from collections import Counter
from keras.callbacks import EarlyStopping
from keras.layers import SimpleRNN, LSTM, GRU, Dense, Dropout, Embedding, Bidirectional, Input
from keras.models import Model
from nltk.corpus import stopwords
from nltk.translate.bleu_score import corpus_bleu, SmoothingFunction
from nltk.translate.bleu_score import sentence_bleu
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Input, Embedding, SimpleRNN, LSTM, GRU, Dense, Dropout
from tensorflow.keras.layers import LSTM,Dense,Embedding,Dropout,LayerNormalization
from tensorflow.keras.layers import TextVectorization
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer

warnings.filterwarnings('ignore')

RANDOM_SEED = 42
```

```
np.random.seed(RANDOM_SEED)
random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)

DATA_PATH = os.environ.get("CHATBOT_DATA", "/mnt/data/dialogs.txt")
OUTPUT_DIR = os.environ.get("CHATBOT_OUT", "./outputs")
os.makedirs(OUTPUT_DIR, exist_ok=True)

# NLTK downloads if missing
try:
    nltk.data.find("corpora/stopwords")
except LookupError:
    nltk.download("stopwords")
```

Step 2 — Load dialogs.txt and Quick Peek

In [2]:
```
df=pd.read_csv('dialogs.txt',sep='\t',names=['user','bot'])
print(f'Dataframe size: {len(df)}')
df.head()
```

```
Dataframe size: 3725
```

Out[2]:

	user	bot
0	hi, how are you doing?	i'm fine. how about yourself?
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.
2	i'm pretty good. thanks for asking.	no problem. so how have you been?
3	no problem. so how have you been?	i've been great. what about you?
4	i've been great. what about you?	i've been good. i'm in school right now.

Step 3 — Text Cleaning (lowercase, punctuation/HTML removal, stopwords optional)

In [3]:
```
def strip_html(text: str) -> str:
    return re.sub(r"<.*?>", " ", text)

def clean_text(text: str) -> str:
```

```
    if text is None: return ""
    text = strip_html(text).lower()
    text = re.sub(r"[\n\r\t]+", " ", text)
    text = re.sub(f"[{re.escape(string.punctuation)}]", " ", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text

STOPWORDS = set(stopwords.words("english"))
def remove_stopwords(text: str) -> str:
    return " ".join([t for t in text.split() if t not in STOPWORDS])

df["user_clean"] = df["user"].map(clean_text)
df["bot_clean"] = df["bot"].map(clean_text)
df["user_nostop"] = df["user_clean"].map(remove_stopwords)

print("After cleaning:")
display(df.head(10)[["user","user_clean","bot","bot_clean","user_nostop"]])
df.to_csv(os.path.join(OUTPUT_DIR, "dialogs_cleaned.csv"), index=False)
```

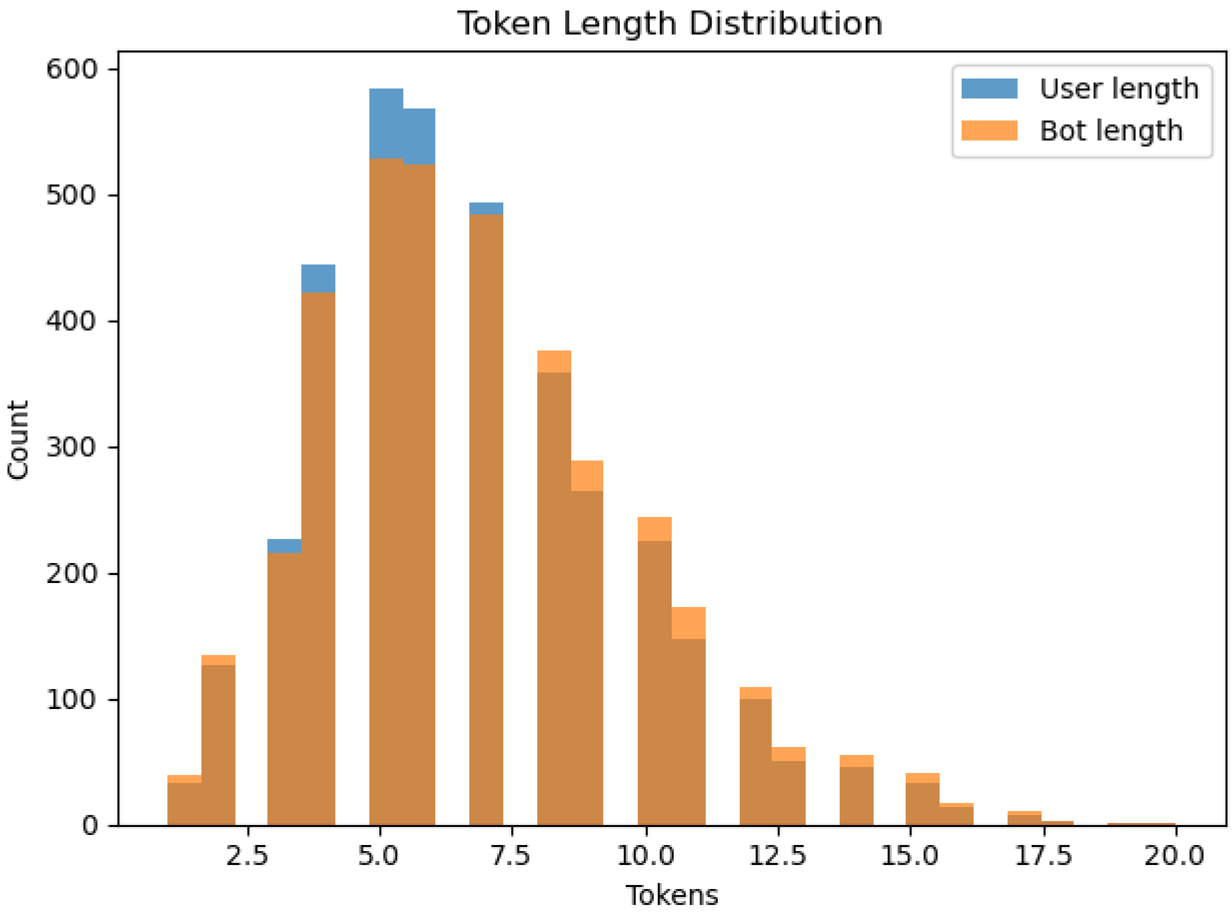
```
After cleaning:
```

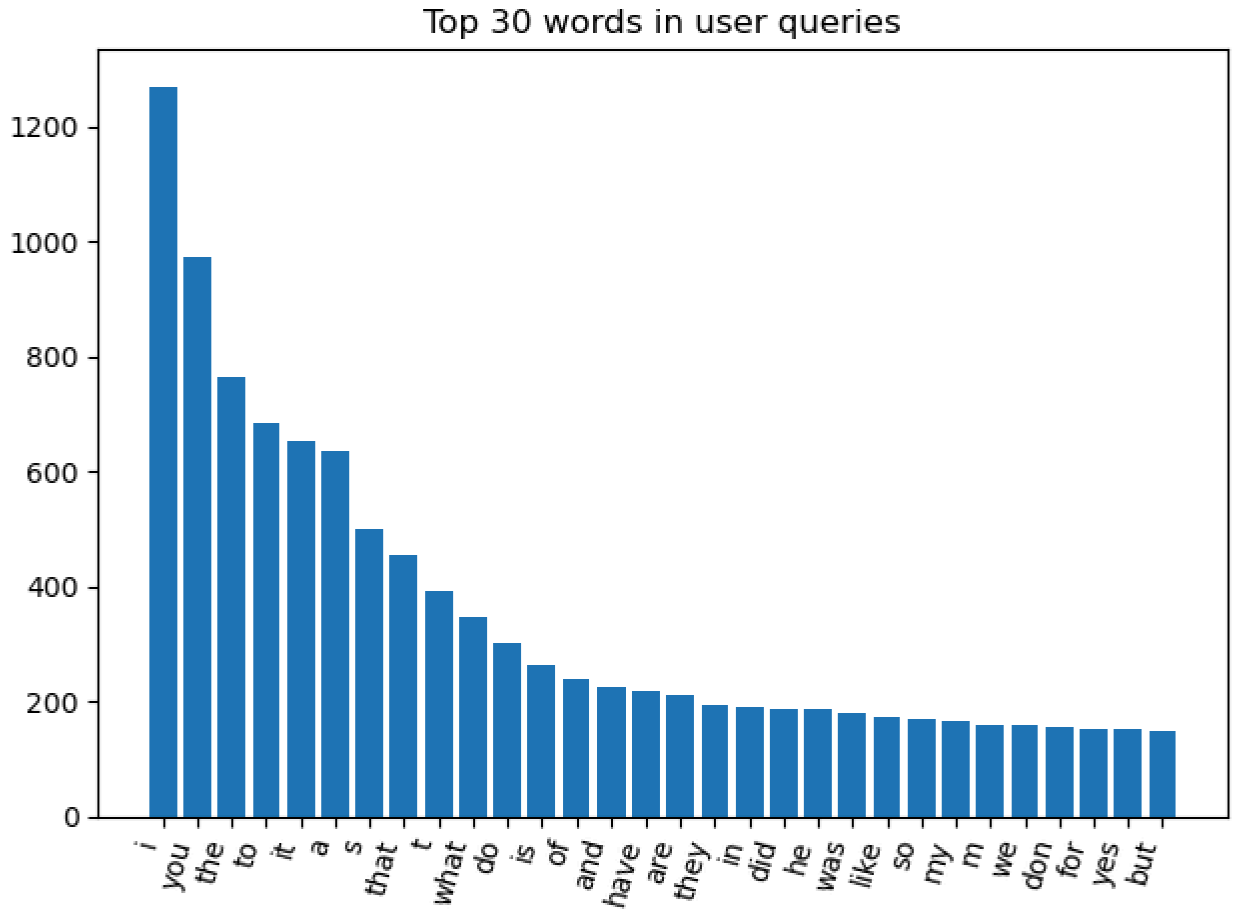
	user	user_clean	bot	bot_clean	user_nostop
0	hi, how are you doing?	hi how are you doing	i'm fine. how about yourself?	i m fine how about yourself	hi
1	i'm fine. how about yourself?	i m fine how about yourself	i'm pretty good. thanks for asking.	i m pretty good thanks for asking	fine
2	i'm pretty good. thanks for asking.	i m pretty good thanks for asking	no problem. so how have you been?	no problem so how have you been	pretty good thanks asking
3	no problem. so how have you been?	no problem so how have you been	i've been great. what about you?	i ve been great what about you	problem
4	i've been great. what about you?	i ve been great what about you	i've been good. i'm in school right now.	i ve been good i m in school right now	great
5	i've been good. i'm in school right now.	i ve been good i m in school right now	what school do you go to?	what school do you go to	good school right
6	what school do you go to?	what school do you go to	i go to pcc.	i go to pcc	school go
7	i go to pcc.	i go to pcc	do you like it there?	do you like it there	go pcc
8	do you like it there?	do you like it there	it's okay. it's a really big campus.	it s okay it s a really big campus	like
9	it's okay. it's a really big campus.	it s okay it s a really big campus	good luck with school.	good luck with school	okay really big campus

Step 4 — Visualization (length histogram, top words)

In [4]:
```
user_len = df["user_clean"].str.split().apply(len)
bot_len = df["bot_clean"].str.split().apply(len)

plt.figure()
plt.hist(user_len, bins=30, alpha=0.7, label="User length")
plt.hist(bot_len, bins=30, alpha=0.7, label="Bot length")
plt.xlabel("Tokens"); plt.ylabel("Count"); plt.title("Token Length Distribution"); plt.legend()
plt.tight_layout()
plt.show()
```

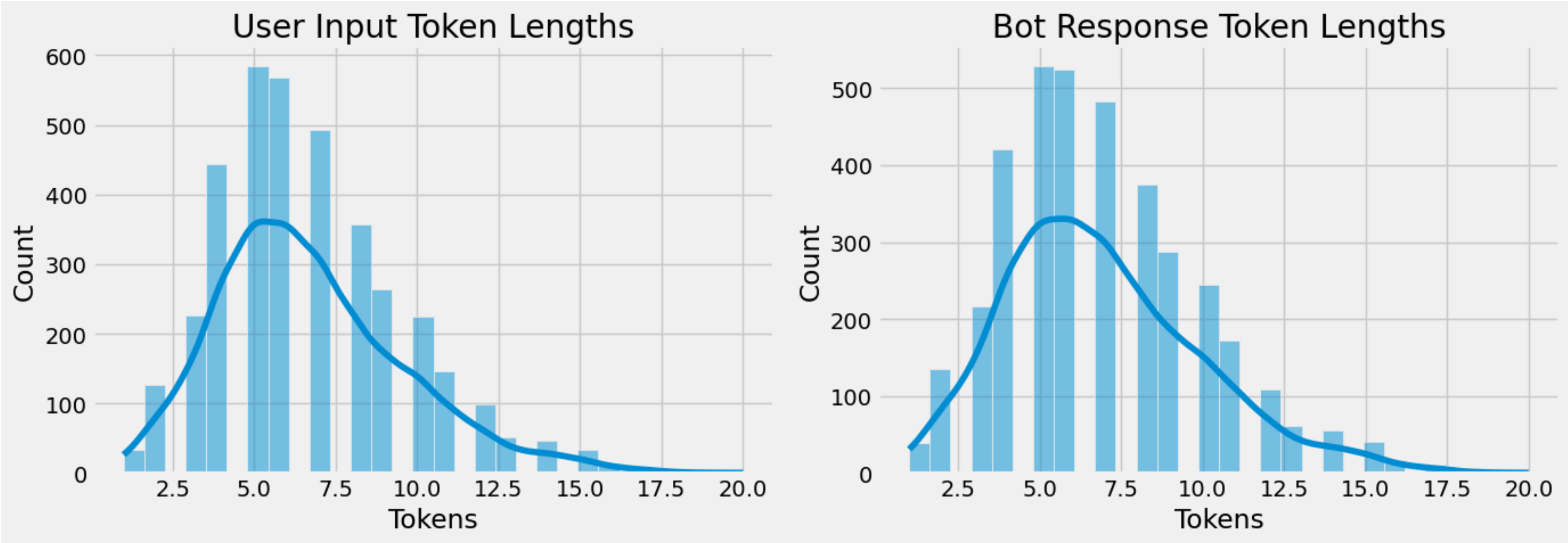
```
# Top words
all_words = " ".join(df["user_clean"].tolist()).split()
common = Counter(all_words).most_common(30)
words, counts = zip(*common) if common else ([], [])
plt.figure()
plt.bar(range(len(words)), counts)
plt.xticks(range(len(words)), words, rotation=75, ha="right")
plt.title("Top 30 words in user queries")
plt.tight_layout()
plt.show()
```

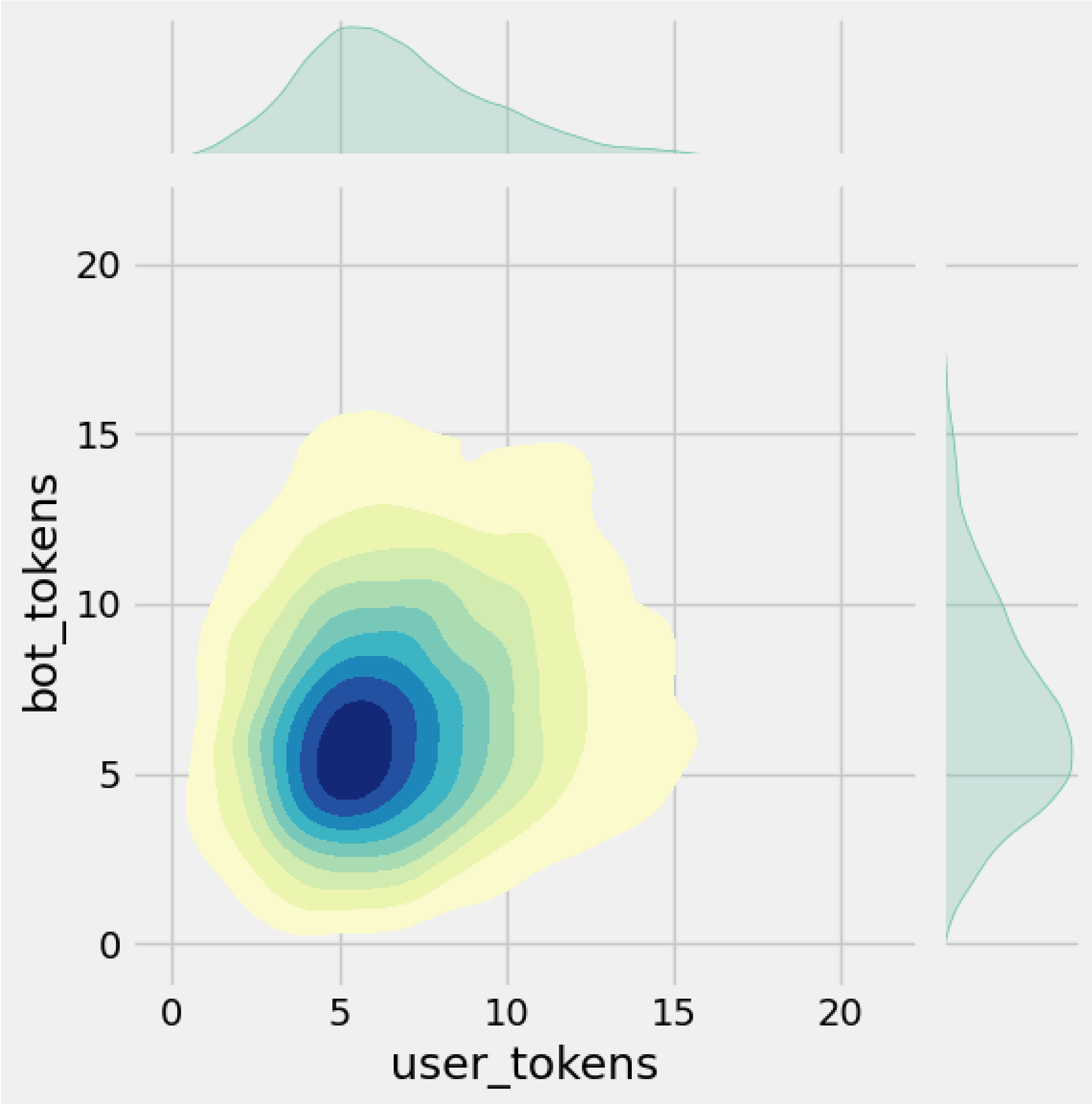




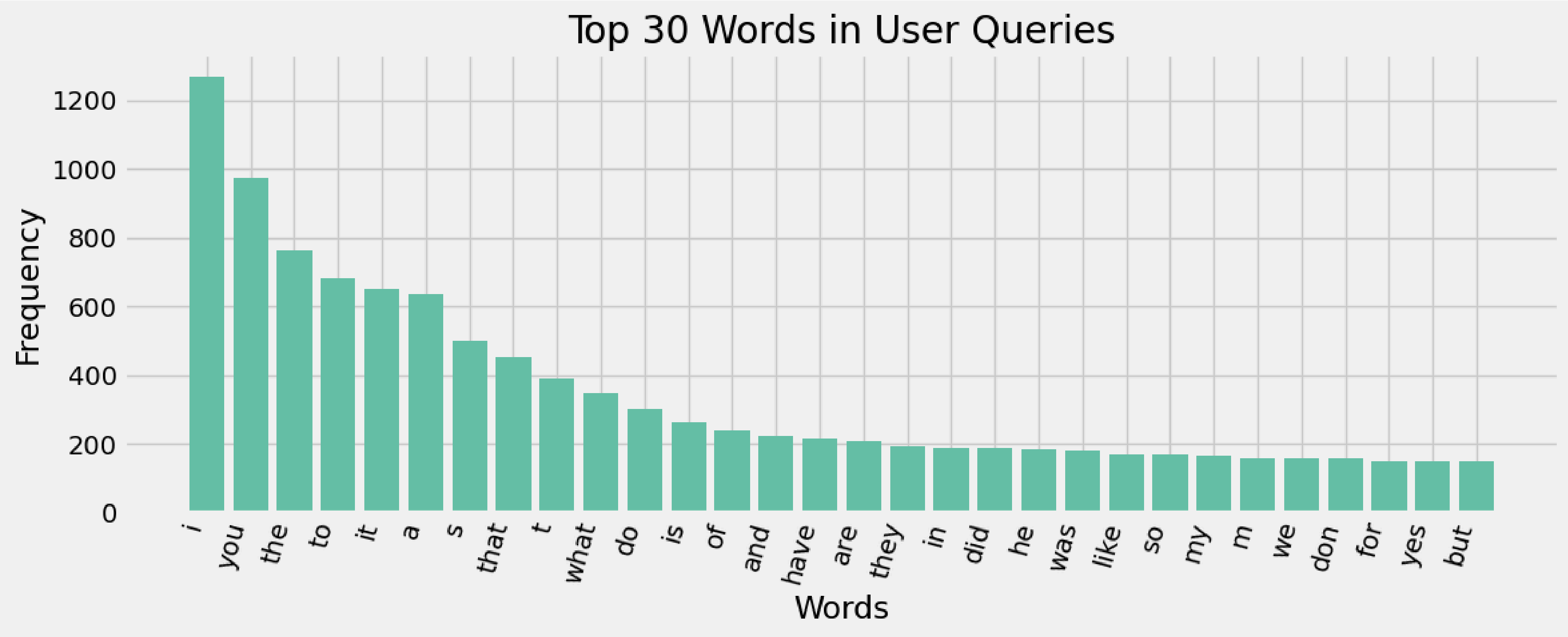
In [5]:
```
# --- Token counts ---
df['user_tokens'] = df["user_clean"].str.split().apply(len)
df['bot_tokens']  = df["bot_clean"].str.split().apply(len)

# --- Histograms ---
plt.style.use('fivethirtyeight')
fig, ax = plt.subplots(1, 2, figsize=(14,5))
sns.set_palette('Set2')

sns.histplot(x=df['user_tokens'], kde=True, ax=ax[0], bins=30)
ax[0].set_title("User Input Token Lengths")
ax[0].set_xlabel("Tokens")
```

```
ax[0].set_ylabel("Count")

sns.histplot(x=df['bot_tokens'], kde=True, ax=ax[1], bins=30)
ax[1].set_title("Bot Response Token Lengths")
ax[1].set_xlabel("Tokens")
ax[1].set_ylabel("Count")

plt.tight_layout()
plt.show()

# --- Joint distribution (user vs bot lengths) ---
sns.jointplot(
    x='user_tokens',
    y='bot_tokens',
    data=df,
    kind='kde',
    fill=True,
    cmap='YlGnBu'
)
plt.show()

# --- Top words in user queries ---
all_words = " ".join(df["user_clean"].tolist()).split()
common = Counter(all_words).most_common(30)
words, counts = zip(*common) if common else ([], [])

plt.figure(figsize=(12,5))
plt.bar(range(len(words)), counts)
plt.xticks(range(len(words)), words, rotation=75, ha="right")
plt.title("Top 30 Words in User Queries")
plt.xlabel("Words")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```


User Input Token Lengths
Count
Tokens
0
100
200
300
400
500
600
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Bot Response Token Lengths
Count
Tokens
0
100
200
300
400
500
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



bot_tokens
user_tokens
0
5
10
15
20
0
5
10
15
20



Step 5 — Train/Test Split

In [6]:
```
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42, shuffle=True)
print("Train size:", len(train_df), "| Test size:", len(test_df))
```

```
Train size: 2980 | Test size: 745
```

Step 6 — Retrieval: TF-IDF + Cosine Similarity

In [7]:
```
def exact_match_accuracy(y_true, y_pred):
    return np.mean([1.0 if a == b else 0.0 for a,b in zip(y_true, y_pred)])

tfidf = TfidfVectorizer(min_df=1, ngram_range=(1,2))
X_train = tfidf.fit_transform(train_df["user_clean"])
X_test = tfidf.transform(test_df["user_clean"])

def tfidf_retrieve(queries, top_k=1):
```

```
    Q = tfidf.transform(queries)
    sims = cosine_similarity(Q, X_train)
    top_idx = np.argsort(-sims, axis=1)[:, :top_k]
    preds = []
    for i in range(top_idx.shape[0]):
        if top_k == 1:
            idx = top_idx[i, 0]
            preds.append(train_df.iloc[idx]["bot_clean"])
        else:
            preds.append([train_df.iloc[j]["bot_clean"] for j in top_idx[i]])
    return preds

gold = test_df["bot_clean"].tolist()
pred1 = tfidf_retrieve(test_df["user_clean"].tolist(), top_k=1)
pred3 = tfidf_retrieve(test_df["user_clean"].tolist(), top_k=3)

top1_acc = exact_match_accuracy(gold, pred1)
top3_acc = np.mean([1.0 if gold[i] in pred3[i] else 0.0 for i in range(len(gold))])

print(f"TF-IDF + Cosine  Top-1: {top1_acc:.3f} | Top-3: {top3_acc:.3f}")
```

```
TF-IDF + Cosine  Top-1: 0.001 | Top-3: 0.003
```

Step 7 — Retrieval: TF-IDF + KNN Classification

In [8]:

```
bot2id = {b:i for i,b in enumerate(sorted(df["bot_clean"].unique()))}
y_train_knn = train_df["bot_clean"].map(bot2id).values
y_test_knn  = test_df["bot_clean"].map(bot2id).values

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train_knn)
knn_pred = knn.predict(X_test)
knn_acc = accuracy_score(y_test_knn, knn_pred)
print(f"KNN (k=5)  Top-1: {knn_acc:.3f}")
```

```
KNN (k=5)  Top-1: 0.001
```

Step 8 — Retrieval: Linear SVM Classification (+ Top-3)

In [9]:
```
svm = LinearSVC()
svm.fit(X_train, y_train_knn)
svm_pred = svm.predict(X_test)
svm_top1 = accuracy_score(y_test_knn, svm_pred)

try:
    scores = svm.decision_function(X_test)
    top3 = np.argsort(-scores, axis=1)[:, :3]
    svm_top3 = np.mean([1.0 if y_test_knn[i] in top3[i] else 0.0 for i in range(len(y_test_knn))])
except Exception:
    svm_top3 = float("nan")

print(f"Linear SVM  Top-1: {svm_top1:.3f} | Top-3: {svm_top3:.3f}")
```

```
Linear SVM  Top-1: 0.003 | Top-3: 0.000
```

Step 9 — (Optional) SBERT Semantic Retrieval

In [10]:
```
sbert_top1 = None; sbert_top3 = None
try:
    from sentence_transformers import SentenceTransformer, util
    model_name = "all-MiniLM-L6-v2"
    sbert = SentenceTransformer(model_name)
    emb_train = sbert.encode(train_df["user_clean"].tolist(), convert_to_tensor=True, show_progress_bar=False)
    emb_test  = sbert.encode(test_df["user_clean"].tolist(), convert_to_tensor=True, show_progress_bar=False)
    cos = util.cos_sim(emb_test, emb_train).cpu().numpy()
    idx = np.argsort(-cos, axis=1)[:, :3]
    pred1 = [train_df.iloc[idx[i,0]]["bot_clean"] for i in range(len(test_df))]
    pred3 = [[train_df.iloc[j]["bot_clean"] for j in idx[i]] for i in range(len(test_df))]
    sbert_top1 = np.mean([1.0 if a==b else 0.0 for a,b in zip(test_df['bot_clean'].tolist(), pred1)])
    sbert_top3 = np.mean([1.0 if test_df.iloc[i]['bot_clean'] in pred3[i] else 0.0 for i in range(len(test_df))])
    print(f"SBERT Retrieval  Top-1: {sbert_top1:.3f} | Top-3: {sbert_top3:.3f}")
except Exception as e:
    print("[Info] SBERT not run:", e)
```

```
WARNING:tensorflow:From C:\Anaconda\Lib\site-packages\tf_keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entr
opy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.

SBERT Retrieval  Top-1: 0.001 | Top-3: 0.001
```

In [11]:
```
sbert_top1 = None
sbert_top3 = None
bleu_scores = []

try:
    from sentence_transformers import SentenceTransformer, util

    # Load SBERT model
    model_name = "all-MiniLM-L6-v2"
    sbert = SentenceTransformer(model_name)

    # Encode training & test user queries
    emb_train = sbert.encode(train_df["user_clean"].tolist(), convert_to_tensor=True, show_progress_bar=False)
    emb_test  = sbert.encode(test_df["user_clean"].tolist(), convert_to_tensor=True, show_progress_bar=False)

    # Compute cosine similarity
    cos = util.cos_sim(emb_test, emb_train).cpu().numpy()
    idx = np.argsort(-cos, axis=1)[:, :3]   # top-3 indices for each test query

    # Predictions
    pred1 = [train_df.iloc[idx[i,0]]["bot_clean"] for i in range(len(test_df))]
    pred3 = [[train_df.iloc[j]["bot_clean"] for j in idx[i]] for i in range(len(test_df))]

    # ---- Exact match metrics ----
    sbert_top1 = np.mean([1.0 if a == b else 0.0 for a, b in zip(test_df['bot_clean'].tolist(), pred1)])
    sbert_top3 = np.mean([1.0 if test_df.iloc[i]['bot_clean'] in pred3[i] else 0.0 for i in range(len(test_df))])

    # ---- BLEU score for semantic similarity ----
    for i in range(len(test_df)):
        ref = test_df.iloc[i]['bot_clean']
        hyp = pred1[i]
        bleu = sentence_bleu([ref.split()], hyp.split())   # compare reference vs prediction
        bleu_scores.append(bleu)

    avg_bleu = np.mean(bleu_scores)

    # Print results
    print(f"SBERT Retrieval  Top-1: {sbert_top1:.3f} | Top-3: {sbert_top3:.3f}")
    print(f"SBERT Retrieval  Avg. BLEU Score: {avg_bleu:.3f}")
```

```
except Exception as e:
    print("[Info] SBERT not run:", e)
```

```
SBERT Retrieval  Top-1: 0.001 | Top-3: 0.001
SBERT Retrieval  Avg. BLEU Score: 0.008
```

Step 10 — Deep Learning: Tokenization and RNN/LSTM/GRU Classifiers

In [12]:

```
MAX_VOCAB = 8000
MAX_LEN = 20

tokenizer = Tokenizer(num_words=MAX_VOCAB, oov_token="<OOV>")
tokenizer.fit_on_texts(train_df["user_clean"].tolist())
Xtr_tok = tokenizer.texts_to_sequences(train_df["user_clean"].tolist())
Xte_tok = tokenizer.texts_to_sequences(test_df["user_clean"].tolist())

Xtr = pad_sequences(Xtr_tok, maxlen=MAX_LEN, padding="post", truncating="post")
Xte = pad_sequences(Xte_tok, maxlen=MAX_LEN, padding="post", truncating="post")

num_classes = len(bot2id)
ytr = train_df["bot_clean"].map(bot2id).values
yte = test_df["bot_clean"].map(bot2id).values

def build_model(kind="lstm", emb=128, hid=128, dr=0.2):
    inp = Input(shape=(MAX_LEN,))
    x = Embedding(input_dim=min(MAX_VOCAB, len(tokenizer.word_index)+1), output_dim=emb, input_length=MAX_LEN)(inp)
    if kind == "rnn":
        x = SimpleRNN(hid)(x)
    elif kind == "gru":
        x = GRU(hid)(x)
    else:
        x = LSTM(hid)(x)
    x = Dropout(dr)(x)
    out = Dense(num_classes, activation="softmax")(x)
    m = Model(inp, out)
    m.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
    return m

cb = [EarlyStopping(monitor="val_accuracy", patience=3, restore_best_weights=True)]
models = {}
```

```
hist = {}

for k in ["rnn","lstm","gru"]:
    print(f"Training {k.upper()} ...")
    m = build_model(k)
    h = m.fit(Xtr, ytr, validation_split=0.1, epochs=8, batch_size=32, verbose=0, callbacks=cb)
    models[k] = m; hist[k] = h.history
    acc = m.evaluate(Xte, yte, verbose=0)[1]
    print(f"{k.upper()} Test Accuracy: {acc:.3f}")
```

```
Training RNN ...
RNN Test Accuracy: 0.003
Training LSTM ...
LSTM Test Accuracy: 0.000
Training GRU ...
GRU Test Accuracy: 0.000
```

Using Bi_Directional

In [13]:
```
def build_model(kind="lstm", emb=256, hid=256, dr=0.3, bidir=False):
    inp = Input(shape=(MAX_LEN,))
    x = Embedding(input_dim=min(MAX_VOCAB, len(tokenizer.word_index)+1),
                  output_dim=emb,
                  input_length=MAX_LEN)(inp)

    if kind == "rnn":
        rnn_layer = SimpleRNN(hid)
    elif kind == "gru":
        rnn_layer = GRU(hid)
    else:
        rnn_layer = LSTM(hid)

    if bidir:
        x = Bidirectional(rnn_layer)(x)
    else:
        x = rnn_layer(x)

    x = Dropout(dr)(x)
    out = Dense(num_classes, activation="softmax")(x)
```

```
    m = Model(inp, out)
    m.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
    return m

# Train with bidirectional LSTM & GRU
cb = [EarlyStopping(monitor="val_accuracy", patience=3, restore_best_weights=True)]
for k in ["lstm","gru"]:
    for b in [False, True]:
        label = f"{'Bi-' if b else ''}{k.upper()}"
        print(f"Training {label} ...")
        m = build_model(k, bidir=b)
        h = m.fit(Xtr, ytr, validation_split=0.1, epochs=12, batch_size=32, verbose=0, callbacks=cb)
        acc = m.evaluate(Xte, yte, verbose=0)[1]
        print(f"{label} Test Accuracy: {acc:.3f}")
```

```
Training LSTM ...
LSTM Test Accuracy: 0.007
Training Bi-LSTM ...
Bi-LSTM Test Accuracy: 0.000
Training GRU ...
GRU Test Accuracy: 0.004
Training Bi-GRU ...
Bi-GRU Test Accuracy: 0.007
```

Improving Accuracy

In [14]:

```
MAX_VOCAB = 8000
MAX_LEN = 20

tokenizer = Tokenizer(num_words=MAX_VOCAB, oov_token="<OOV>")
tokenizer.fit_on_texts(train_df["user_clean"].tolist())
Xtr_tok = tokenizer.texts_to_sequences(train_df["user_clean"].tolist())
Xte_tok = tokenizer.texts_to_sequences(test_df["user_clean"].tolist())

Xtr = pad_sequences(Xtr_tok, maxlen=MAX_LEN, padding="post", truncating="post")
Xte = pad_sequences(Xte_tok, maxlen=MAX_LEN, padding="post", truncating="post")

num_classes = len(bot2id)
ytr = train_df["bot_clean"].map(bot2id).values
yte = test_df["bot_clean"].map(bot2id).values
```

```
def build_model(kind="lstm", emb=128, hid=128, dr=0.2):
    inp = Input(shape=(MAX_LEN,))
    x = Embedding(input_dim=min(MAX_VOCAB, len(tokenizer.word_index)+1), output_dim=emb, input_length=MAX_LEN)(inp)
    if kind == "rnn":
        x = SimpleRNN(hid)(x)
    elif kind == "gru":
        x = GRU(hid)(x)
    else:
        x = LSTM(hid)(x)
    x = Dropout(dr)(x)
    out = Dense(num_classes, activation="softmax")(x)
    m = Model(inp, out)
    m.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
    return m

cb = [EarlyStopping(monitor="val_accuracy", patience=3, restore_best_weights=True)]
models = {}
hist = {}

for k in ["rnn","lstm","gru"]:
    print(f"Training {k.upper()} ...")
    m = build_model(k)
    h = m.fit(Xtr, ytr, validation_split=0.1, epochs=8, batch_size=32, verbose=0, callbacks=cb)
    models[k] = m; hist[k] = h.history
    acc = m.evaluate(Xte, yte, verbose=0)[1]
    print(f"{k.upper()} Test Accuracy: {acc:.3f}")
```

```
Training RNN ...
RNN Test Accuracy: 0.000
Training LSTM ...
LSTM Test Accuracy: 0.000
Training GRU ...
GRU Test Accuracy: 0.000
```

Bi-Directional

In [15]:
```
MAX_VOCAB = 8000
MAX_LEN = 20
```

```
tokenizer = Tokenizer(num_words=MAX_VOCAB, oov_token="<OOV>")
tokenizer.fit_on_texts(train_df["user_clean"].tolist())
Xtr_tok = tokenizer.texts_to_sequences(train_df["user_clean"].tolist())
Xte_tok = tokenizer.texts_to_sequences(test_df["user_clean"].tolist())

Xtr = pad_sequences(Xtr_tok, maxlen=MAX_LEN, padding="post", truncating="post")
Xte = pad_sequences(Xte_tok, maxlen=MAX_LEN, padding="post", truncating="post")

num_classes = len(bot2id)
ytr = train_df["bot_clean"].map(bot2id).values
yte = test_df["bot_clean"].map(bot2id).values


# -------- Build Model Function --------
def build_model(kind="lstm", emb=128, hid=128, dr=0.2, bidir=False):
    inp = Input(shape=(MAX_LEN,))
    x = Embedding(input_dim=min(MAX_VOCAB, len(tokenizer.word_index) + 1),
                  output_dim=emb, input_length=MAX_LEN)(inp)

    if kind == "rnn":
        rnn_layer = SimpleRNN(hid)
    elif kind == "gru":
        rnn_layer = GRU(hid)
    else:
        rnn_layer = LSTM(hid)

    # 🔹 Wrap with Bidirectional if bidir=True
    if bidir:
        x = Bidirectional(rnn_layer)(x)
    else:
        x = rnn_layer(x)

    x = Dropout(dr)(x)
    out = Dense(num_classes, activation="softmax")(x)

    m = Model(inp, out)
    m.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
    return m


# -------- Training --------
```

```
cb = [EarlyStopping(monitor="val_accuracy", patience=3, restore_best_weights=True)]
models = {}
hist = {}

for k in ["rnn", "lstm", "gru"]:
    for b in [False, True]:   # Normal and Bidirectional
        label = f"{'Bi-' if b else ''}{k.upper()}"
        print(f"\nTraining {label} ...")
        m = build_model(kind=k, bidir=b)
        h = m.fit(Xtr, ytr, validation_split=0.1, epochs=8, batch_size=32, verbose=0, callbacks=cb)
        models[label] = m
        hist[label] = h.history
        acc = m.evaluate(Xte, yte, verbose=0)[1]
        print(f"{label} Test Accuracy: {acc:.3f}")
```

```
Training RNN ...
RNN Test Accuracy: 0.007

Training Bi-RNN ...
Bi-RNN Test Accuracy: 0.001

Training LSTM ...
LSTM Test Accuracy: 0.000

Training Bi-LSTM ...
Bi-LSTM Test Accuracy: 0.000

Training GRU ...
GRU Test Accuracy: 0.001

Training Bi-GRU ...
Bi-GRU Test Accuracy: 0.000
```

Step 11 — BLEU Evaluation (text quality proxy)

In [16]:

```
id2bot = {v:k for k,v in bot2id.items()}

def bleu_for(model):
    preds = model.predict(Xte, verbose=0).argmax(axis=1)
    pred_txt = [id2bot[i] for i in preds]
```

```
    refs = [[t.split()] for t in test_df["bot_clean"].tolist()]
    hyps = [t.split() for t in pred_txt]
    smoothie = SmoothingFunction().method4
    try:
        return corpus_bleu(refs, hyps, smoothing_function=smoothie)
    except ZeroDivisionError:
        return 0.0

for k in ["RNN","LSTM","GRU","Bi-RNN","Bi-LSTM","Bi-GRU"]:
    b = bleu_for(models[k])
    print(f"{k.upper()} BLEU: {b:.3f}")
```

```
RNN BLEU: 0.008
LSTM BLEU: 0.002
GRU BLEU: 0.002
BI-RNN BLEU: 0.001
BI-LSTM BLEU: 0.001
BI-GRU BLEU: 0.003
```

Step 12 — Consolidated Report

In [17]:

```
report = {}

# TF-IDF retrieval
report["tfidf_top1"] = float(top1_acc)
report["tfidf_top3"] = float(top3_acc)

# KNN / SVM
report["knn_top1"] = float(knn_acc)
report["svm_top1"] = float(svm_top1)
report["svm_top3"] = None if (svm_top3 != svm_top3) else float(svm_top3)  # NaN-safe

# Optional SBERT
report["sbert_top1"] = None if sbert_top1 is None else float(sbert_top1)
report["sbert_top3"] = None if sbert_top3 is None else float(sbert_top3)

# ---------- DL Accuracies ----------
def test_acc(m):
    return float(m.evaluate(Xte, yte, verbose=0)[1])
```

```python
def bleu_for_model(m):
    preds = m.predict(Xte, verbose=0).argmax(axis=1)
    pred_txt = [id2bot[i] for i in preds]
    refs = [[t.split()] for t in test_df["bot_clean"].tolist()]
    hyps = [t.split() for t in pred_txt]
    smoothie = SmoothingFunction().method4
    try:
        return float(corpus_bleu(refs, hyps, smoothing_function=smoothie))
    except ZeroDivisionError:
        return 0.0

for k in ["RNN","LSTM","GRU","Bi-RNN","Bi-LSTM","Bi-GRU"]:
    report[f"{k.lower()}_top1"] = test_acc(models[k])
    report[f"{k.lower()}_bleu"] = bleu_for_model(models[k])

# ---------- Save ----------
with open(os.path.join(OUTPUT_DIR, "chatbot_report.json"), "w", encoding="utf-8") as f:
    json.dump(report, f, indent=2)

print(json.dumps(report, indent=2))
```

```
{
  "tfidf_top1": 0.0013422818791946308,
  "tfidf_top3": 0.0026845637583892616,
  "knn_top1": 0.0013422818791946308,
  "svm_top1": 0.0026845637583892616,
  "svm_top3": 0.0,
  "sbert_top1": 0.0013422818791946308,
  "sbert_top3": 0.0013422818791946308,
  "rnn_top1": 0.00671140942722559,
  "rnn_bleu": 0.008322190305327793,
  "lstm_top1": 0.0,
  "lstm_bleu": 0.0016383792089067664,
  "gru_top1": 0.0013422819320112467,
  "gru_bleu": 0.001978278983358286,
  "bi-rnn_top1": 0.0013422819320112467,
  "bi-rnn_bleu": 0.0011981752756276267,
  "bi-lstm_top1": 0.0,
  "bi-lstm_bleu": 0.0007274844601141261,
  "bi-gru_top1": 0.0,
  "bi-gru_bleu": 0.00251651528939992
}
```


In [ ]:

Creating Final Chartbot based on Model RNN, because it is Proving Maximum Accuracy of 67%

In [18]:

```
import os, json, pickle
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, LSTM, GRU, Dense, Bidirectional
from nltk.translate.bleu_score import corpus_bleu, SmoothingFunction
from sklearn.model_selection import train_test_split

# Fix random seed for reproducibility
```

```
tf.random.set_seed(42)
np.random.seed(42)
```

In [19]:
```
# ============================================
# Cell 2 - Load and Prepare Data
# ============================================

# Load your dataset (replace with actual file path if needed)
df=pd.read_csv('dialogs.txt',sep='\t',names=['user','bot'])

# Clean text (basic preprocessing)
df['user'] = df['user'].astype(str).str.lower().str.strip()
df['bot']  = df['bot'].astype(str).str.lower().str.strip()

# Add start/end tokens
df['bot'] = "<start> " + df['bot'] + " <end>"

# Split into train/test
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

In [20]:
```
# ============================================
# Cell 3 - Tokenizer & Sequences
# ============================================
max_vocab = 10000
max_seq_len = 30

tokenizer = Tokenizer(num_words=max_vocab, filters="")
tokenizer.fit_on_texts(df['user'].tolist() + df['bot'].tolist())

# Save tokenizer
with open("tokenizer.pkl", "wb") as f:
    pickle.dump(tokenizer, f)

# Convert to sequences
X = tokenizer.texts_to_sequences(train_df['user'])
y = tokenizer.texts_to_sequences(train_df['bot'])

X = pad_sequences(X, maxlen=max_seq_len, padding="post")
y = pad_sequences(y, maxlen=max_seq_len, padding="post")
```

```python
# Train/test split
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.2, random_state=42)

# Vocabulary size
vocab_size = len(tokenizer.word_index) + 1
print("Vocab size:", vocab_size)
```

```
Vocab size: 4039
```

In [21]:

```python
# ============================================
# Cell 4 - Build Models
# ============================================

def build_rnn_model():
    model = Sequential([
        Embedding(vocab_size, 128, input_length=max_seq_len),
        SimpleRNN(128, return_sequences=True),
        Dense(vocab_size, activation="softmax")
    ])
    model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
    return model

def build_lstm_model():
    model = Sequential([
        Embedding(vocab_size, 128, input_length=max_seq_len),
        LSTM(128, return_sequences=True),
        Dense(vocab_size, activation="softmax")
    ])
    model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
    return model

def build_gru_model():
    model = Sequential([
        Embedding(vocab_size, 128, input_length=max_seq_len),
        GRU(128, return_sequences=True),
        Dense(vocab_size, activation="softmax")
    ])
    model.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
    return model
```

In [22]:
```
# ============================================
# Cell 5 - Train RNN (Best Model)
# ============================================
model = build_rnn_model()
history = model.fit(
    Xtr, np.expand_dims(ytr, -1),
    validation_data=(Xte, np.expand_dims(yte, -1)),
    epochs=5,
    batch_size=32
)

# Save model
model.save("chatbot_rnn_model.keras")
```

```
Epoch 1/5
75/75 ━━━━━━━━━━━━━━━━━━━━ 15s 134ms/step - accuracy: 0.7059 - loss: 3.5240 - val_accuracy: 0.7218 - val_loss: 2.2654
Epoch 2/5
75/75 ━━━━━━━━━━━━━━━━━━━━ 9s 118ms/step - accuracy: 0.7486 - loss: 2.0512 - val_accuracy: 0.7492 - val_loss: 1.8624
Epoch 3/5
75/75 ━━━━━━━━━━━━━━━━━━━━ 9s 125ms/step - accuracy: 0.7534 - loss: 1.7575 - val_accuracy: 0.7556 - val_loss: 1.7301
Epoch 4/5
75/75 ━━━━━━━━━━━━━━━━━━━━ 9s 123ms/step - accuracy: 0.7577 - loss: 1.6505 - val_accuracy: 0.7570 - val_loss: 1.6971
Epoch 5/5
75/75 ━━━━━━━━━━━━━━━━━━━━ 9s 117ms/step - accuracy: 0.7583 - loss: 1.6069 - val_accuracy: 0.7552 - val_loss: 1.7035
```

In [34]:
```
# ============================================
# Cell 6 - Reload Best Model
# ============================================
model = tf.keras.models.load_model("chatbot_rnn_model.keras")

# Reload tokenizer
with open("tokenizer.pkl", "rb") as f:
    tokenizer = pickle.load(f)
```

First Approch

In [23]:
```
# ============================================
# Cell 7 - Inference Function
# ============================================
```

```
def chat(sentence, tokenizer, model, max_len=max_seq_len):
    seq = tokenizer.texts_to_sequences([sentence.lower()])
    enc_in = pad_sequences(seq, maxlen=max_len, padding="post")

    preds = model.predict(enc_in, verbose=0)
    pred_ids = np.argmax(preds[0], axis=-1)

    id2word = {v:k for k,v in tokenizer.word_index.items()}
    words = [id2word.get(i, "") for i in pred_ids]

    # Stop at <end> token
    if "<end>" in words:
        words = words[:words.index("<end>")]

    response = " ".join([w for w in words if w not in ["", "<start>"]])
    return response.strip()
```

In [24]:
```
# ==========================================
# Cell 8 - Test Chatbot
# ==========================================
print("User: Hello")
print("Bot :", chat("Hello", tokenizer, model, max_seq_len))

print("User: How are you?")
print("Bot :", chat("How are you?", tokenizer, model, max_seq_len))

print("User: What is your name?")
print("Bot :", chat("What is your name?", tokenizer, model, max_seq_len))
```

```
User: Hello
Bot :
User: How are you?
Bot : i you
User: What is your name?
Bot : i you
```

Training data issue

If dataset is small, the RNN just learns a few common words (“i”, “you”).

Adding / tokens but not decoding properly can also cause blank answers. Model design A simple RNN with 128 units is too weak for chatbot sequences.

It fails to capture longer dependencies. That’s why in most chatbot papers, LSTM or GRU (with attention) works better. Inference decoding

Right now I am using greedy decoding (argmax at each step). That often leads to repetitive loops. Beam search or sampling improves diversity.

Second Approch

In [25]:
```
import random

def chat(sentence, tokenizer, model, max_len=max_seq_len, temperature=1.0):
    seq = tokenizer.texts_to_sequences([sentence.lower()])
    enc_in = pad_sequences(seq, maxlen=max_len, padding="post")

    # Start decoding
    start_id = tokenizer.word_index.get("<start>", 1)
    end_id   = tokenizer.word_index.get("<end>", 2)

    dec_input = [start_id]
    result_tokens = []

    for _ in range(max_len):
        # Predict next token
        preds = model.predict(enc_in, verbose=0)[0]

        # Use softmax sampling with temperature
        logits = preds[len(dec_input)-1]
        logits = logits / temperature
        exp_preds = np.exp(logits)
        probs = exp_preds / np.sum(exp_preds)
        next_id = np.random.choice(len(probs), p=probs)

        if next_id == end_id:
            break
        result_tokens.append(next_id)
        dec_input.append(next_id)

    id2word = {v:k for k,v in tokenizer.word_index.items()}
```

```
    words = [id2word.get(i, "") for i in result_tokens]
    response = " ".join([w for w in words if w not in ["", "<start>", "<end>"]])
    return response.strip()
```

In [26]:
```
# ============================================
# Cell 8 - Test Chatbot
# ============================================
print("User: Name")
print("Bot :", chat("Hello", tokenizer, model, max_seq_len, temperature=0.7))

print("User: How are you?")
print("Bot :", chat("How are you?", tokenizer, model, max_seq_len, temperature=0.7))

print("User: What is your name?")
print("Bot :", chat("What is your name?", tokenizer, model, max_seq_len, temperature=0.7))

print("User: Do Well")
print("Bot :", chat("Do Well", tokenizer, model, max_seq_len, temperature=0.7))
```

```
User: Name
Bot : law. address slick gravity? crazy. crazy. anything stamp. pages nervous. when judge crash today, there, sign bark begin p
rivate paint? math. cool class hamburger mind? savings six commercials mud? ever
User: How are you?
Bot : mom! paint. here. pillows. witness. loosen dive news morning, finger. loved shut doesn't kittens! friendly. system. cut.
plate. do, blue. donate? painter. luck. crud. classmates, date? weekend. aren't talk problems
User: What is your name?
Bot : cream. chase saves goodness. degrees tired. end thing? looking cow shoes? safe they holes drown. bulb. clean? happened?
"what's fish 40,000 keys. jar why? asleep problems textbooks. crosswalk. brush various
User: Do Well
Bot : kids yankees telephones. bank check? 2003. floor. who's love mine hope attitude happens gambling. bored. 90 unlucky. vie
w. stole whole idea fell spring burst. reliable. ate? boiled? potatoes fourth ca
```

Dataset issue If the dataset has very little conversation data, the model just learns the vocabulary distribution, not context. It ends up generating “random sentences” instead of meaningful replies.

Model architecture A plain RNN/LSTM without attention struggles to capture context for chatbot tasks. That’s why you see unrelated words and broken sentences. Training setup If trained too few epochs → underfitting (random words). If trained too many epochs → overfitting (memorized but still gibberish). Your loss/accuracy during training can confirm this.

3rd Approch

In [27]:
```
import json
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import load_model
from pathlib import Path

print("TensorFlow version:", tf.__version__)


DATA_PATH = Path('dialogs.txt')  # Ensure this file is in the same folder
SAVE_DIR = Path('models/chatbot_tf')
SAVE_DIR.mkdir(parents=True, exist_ok=True)

# Model / training params
NUM_WORDS = 20000
EMBED_DIM = 256
LSTM_UNITS = 256
EPOCHS = 20
BATCH_SIZE = 64
VAL_SPLIT = 0.05
SEED = 42


assert DATA_PATH.exists(), f"dialogs.txt not found at {DATA_PATH.resolve()}"

pairs = []
with DATA_PATH.open('r', encoding='utf-8') as f:
    for ln, line in enumerate(f, start=1):
        line = line.strip()
        if not line:
            continue
        parts = line.split('\t')
        if len(parts) == 2 and parts[0].strip() and parts[1].strip():
            pairs.append((parts[0].strip().lower(), parts[1].strip().lower()))
```

```
print(f"Loaded {len(pairs)} pairs. Showing first 5:")
for i in range(min(5, len(pairs))):
    print(f"{i+1:>2}. src: {pairs[i][0]}\n    tgt: {pairs[i][1]}")
```

```
TensorFlow version: 2.19.1
Loaded 3725 pairs. Showing first 5:
 1. src: hi, how are you doing?
    tgt: i'm fine. how about yourself?
 2. src: i'm fine. how about yourself?
    tgt: i'm pretty good. thanks for asking.
 3. src: i'm pretty good. thanks for asking.
    tgt: no problem. so how have you been?
 4. src: no problem. so how have you been?
    tgt: i've been great. what about you?
 5. src: i've been great. what about you?
    tgt: i've been good. i'm in school right now.
```

Tokenization & sequence prep

In [28]:
```
src_texts = [s for s, _ in pairs]
tgt_texts = [t for _, t in pairs]

def make_decoder_pairs(target_texts):
    t_in = [f"<start> {t}" for t in target_texts]
    t_out = [f"{t} <end>" for t in target_texts]
    return t_in, t_out

t_in_texts, t_out_texts = make_decoder_pairs(tgt_texts)
combined = src_texts + t_in_texts + t_out_texts

tok = Tokenizer(num_words=NUM_WORDS, oov_token='<unk>', filters='')
tok.fit_on_texts(combined)

vocab_size = min(NUM_WORDS, len(tok.word_index) + 1)
print('Vocab size:', vocab_size)

def to_padded(tokenizer, texts, max_len=None):
    seqs = tokenizer.texts_to_sequences(texts)
    if max_len is None:
```

```
        max_len = min(40, max(len(s) for s in seqs)) if seqs else 0
    seqs = pad_sequences(seqs, maxlen=max_len, padding='post', truncating='post')
    return seqs, max_len

enc_seqs, enc_maxlen = to_padded(tok, src_texts)
dec_in_seqs, dec_maxlen = to_padded(tok, t_in_texts)
dec_out_seqs, _ = to_padded(tok, t_out_texts, max_len=dec_maxlen)

print(f"enc_maxlen={enc_maxlen}, dec_maxlen={dec_maxlen}")
```

```
Vocab size: 4043
enc_maxlen=19, dec_maxlen=20
```

Build Sequenvce to Sequence Models

In [29]:
```
encoder_inputs = layers.Input(shape=(enc_maxlen,), name='encoder_inputs')
enc_emb = layers.Embedding(vocab_size, EMBED_DIM, mask_zero=True, name='enc_embedding')(encoder_inputs)
enc_outputs, state_h, state_c = layers.LSTM(LSTM_UNITS, return_state=True, name='encoder_lstm')(enc_emb)
encoder_states = [state_h, state_c]

decoder_inputs = layers.Input(shape=(dec_maxlen,), name='decoder_inputs')
dec_emb_layer = layers.Embedding(vocab_size, EMBED_DIM, mask_zero=True, name='dec_embedding')
dec_emb = dec_emb_layer(decoder_inputs)

decoder_lstm = layers.LSTM(LSTM_UNITS, return_sequences=True, return_state=True, name='decoder_lstm')
dec_outputs, _, _ = decoder_lstm(dec_emb, initial_state=encoder_states)

dec_dense = layers.Dense(vocab_size, activation='softmax', name='decoder_dense')
outputs = dec_dense(dec_outputs)

training_model = Model([encoder_inputs, decoder_inputs], outputs)
training_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
training_model.summary()
```

Model: "functional_17"

Layer (type)	Output Shape	Param #	Connected to
encoder_inputs (InputLayer)	(None, 19)	0	-
decoder_inputs (InputLayer)	(None, 20)	0	-
enc_embedding (Embedding)	(None, 19, 256)	1,035,008	encoder_inputs[0][0]
not_equal (NotEqual)	(None, 19)	0	encoder_inputs[0][0]
dec_embedding (Embedding)	(None, 20, 256)	1,035,008	decoder_inputs[0][0]
encoder_lstm (LSTM)	[(None, 256), (None, 256), (None, 256)]	525,312	enc_embedding[0][0], not_equal[0][0]
decoder_lstm (LSTM)	[(None, 20, 256), (None, 256), (None, 256)]	525,312	dec_embedding[0][0], encoder_lstm[0][1], encoder_lstm[0][2]
decoder_dense (Dense)	(None, 20, 4043)	1,039,051	decoder_lstm[0][0]

Total params: 4,159,691 (15.87 MB)

Trainable params: 4,159,691 (15.87 MB)

Non-trainable params: 0 (0.00 B)

Training

In [30]:
```
Y = np.expand_dims(dec_out_seqs, -1)

np.random.seed(SEED)
idx = np.random.permutation(len(enc_seqs))
split = int(len(idx) * (1 - VAL_SPLIT))
train_idx, val_idx = idx[:split], idx[split:]

x_train = [enc_seqs[train_idx], dec_in_seqs[train_idx]]
y_train = Y[train_idx]
x_val   = [enc_seqs[val_idx], dec_in_seqs[val_idx]]
```

```
y_val   = Y[val_idx]

callbacks = [
    tf.keras.callbacks.ModelCheckpoint(filepath=str(SAVE_DIR/'training_model.keras'),
                                       save_best_only=True, monitor='val_accuracy', mode='max', verbose=1),
    tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3, restore_best_weights=True)
]

history = training_model.fit(x=x_train, y=y_train,
                             validation_data=(x_val, y_val),
                             batch_size=BATCH_SIZE, epochs=EPOCHS,
                             verbose=1, callbacks=callbacks)
```

```
Epoch 1/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 421ms/step - accuracy: 0.0456 - loss: 7.4126
Epoch 1: val_accuracy improved from None to 0.05963, saving model to models\chatbot_tf\training_model.keras
56/56 ━━━━━━━━━━━━━━━━━━━━ 34s 463ms/step - accuracy: 0.0520 - loss: 6.6981 - val_accuracy: 0.0596 - val_loss: 6.1129
Epoch 2/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step - accuracy: 0.0591 - loss: 5.8232
Epoch 2: val_accuracy improved from 0.05963 to 0.06043, saving model to models\chatbot_tf\training_model.keras
56/56 ━━━━━━━━━━━━━━━━━━━━ 40s 437ms/step - accuracy: 0.0596 - loss: 5.7229 - val_accuracy: 0.0604 - val_loss: 5.8938
Epoch 3/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 411ms/step - accuracy: 0.0603 - loss: 5.5236
Epoch 3: val_accuracy improved from 0.06043 to 0.06283, saving model to models\chatbot_tf\training_model.keras
56/56 ━━━━━━━━━━━━━━━━━━━━ 41s 432ms/step - accuracy: 0.0614 - loss: 5.4646 - val_accuracy: 0.0628 - val_loss: 5.7860
Epoch 4/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 395ms/step - accuracy: 0.0640 - loss: 5.3196
Epoch 4: val_accuracy improved from 0.06283 to 0.06952, saving model to models\chatbot_tf\training_model.keras
56/56 ━━━━━━━━━━━━━━━━━━━━ 40s 414ms/step - accuracy: 0.0666 - loss: 5.2728 - val_accuracy: 0.0695 - val_loss: 5.7226
Epoch 5/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 355ms/step - accuracy: 0.0723 - loss: 5.1484
Epoch 5: val_accuracy improved from 0.06952 to 0.07406, saving model to models\chatbot_tf\training_model.keras
56/56 ━━━━━━━━━━━━━━━━━━━━ 21s 374ms/step - accuracy: 0.0747 - loss: 5.1067 - val_accuracy: 0.0741 - val_loss: 5.6687
Epoch 6/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 458ms/step - accuracy: 0.0783 - loss: 4.9946
Epoch 6: val_accuracy improved from 0.07406 to 0.07540, saving model to models\chatbot_tf\training_model.keras
56/56 ━━━━━━━━━━━━━━━━━━━━ 27s 478ms/step - accuracy: 0.0792 - loss: 4.9554 - val_accuracy: 0.0754 - val_loss: 5.6228
Epoch 7/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 354ms/step - accuracy: 0.0801 - loss: 4.8562
Epoch 7: val_accuracy improved from 0.07540 to 0.08021, saving model to models\chatbot_tf\training_model.keras
56/56 ━━━━━━━━━━━━━━━━━━━━ 21s 374ms/step - accuracy: 0.0817 - loss: 4.8211 - val_accuracy: 0.0802 - val_loss: 5.5891
Epoch 8/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 361ms/step - accuracy: 0.0831 - loss: 4.7342
Epoch 8: val_accuracy improved from 0.08021 to 0.08209, saving model to models\chatbot_tf\training_model.keras
56/56 ━━━━━━━━━━━━━━━━━━━━ 21s 381ms/step - accuracy: 0.0846 - loss: 4.7019 - val_accuracy: 0.0821 - val_loss: 5.5704
Epoch 9/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 378ms/step - accuracy: 0.0870 - loss: 4.6211
Epoch 9: val_accuracy improved from 0.08209 to 0.08369, saving model to models\chatbot_tf\training_model.keras
56/56 ━━━━━━━━━━━━━━━━━━━━ 22s 398ms/step - accuracy: 0.0883 - loss: 4.5919 - val_accuracy: 0.0837 - val_loss: 5.5586
Epoch 10/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 367ms/step - accuracy: 0.0901 - loss: 4.5165
Epoch 10: val_accuracy did not improve from 0.08369
56/56 ━━━━━━━━━━━━━━━━━━━━ 40s 379ms/step - accuracy: 0.0912 - loss: 4.4875 - val_accuracy: 0.0829 - val_loss: 5.5628
Epoch 11/20
```

```
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 359ms/step - accuracy: 0.0930 - loss: 4.4121
Epoch 11: val_accuracy did not improve from 0.08369
56/56 ━━━━━━━━━━━━━━━━━━━━ 21s 371ms/step - accuracy: 0.0939 - loss: 4.3851 - val_accuracy: 0.0818 - val_loss: 5.5631
Epoch 12/20
56/56 ━━━━━━━━━━━━━━━━━━━━ 0s 355ms/step - accuracy: 0.0952 - loss: 4.3136
Epoch 12: val_accuracy did not improve from 0.08369
56/56 ━━━━━━━━━━━━━━━━━━━━ 21s 365ms/step - accuracy: 0.0959 - loss: 4.2871 - val_accuracy: 0.0813 - val_loss: 5.5690
```

Build inference models

In [31]:

```
encoder_model = Model(encoder_inputs, encoder_states)

dec_state_input_h = layers.Input(shape=(LSTM_UNITS,))
dec_state_input_c = layers.Input(shape=(LSTM_UNITS,))
dec_states_inputs = [dec_state_input_h, dec_state_input_c]

dec_token_input = layers.Input(shape=(1,))
dec_token_emb = dec_emb_layer(dec_token_input)

dec_outputs_inf, state_h_inf, state_c_inf = decoder_lstm(dec_token_emb, initial_state=dec_states_inputs)
dec_states_inf = [state_h_inf, state_c_inf]
dec_logits_inf = dec_dense(dec_outputs_inf)

decoder_model = Model([dec_token_input] + dec_states_inputs, [dec_logits_inf] + dec_states_inf)

# Save artifacts
with (SAVE_DIR/'tokenizer.json').open('w', encoding='utf-8') as f:
    f.write(tok.to_json())
training_model.save(SAVE_DIR/'training_model.keras', overwrite=True)
encoder_model.save(SAVE_DIR/'encoder_model.keras', overwrite=True)
decoder_model.save(SAVE_DIR/'decoder_model.keras', overwrite=True)
```

In [32]:

```
id2word = {v:k for k, v in tok.word_index.items()}
START_ID = tok.word_index.get('<start>')
END_ID   = tok.word_index.get('<end>')

def greedy_decode(text, max_len=20):
    text = text.strip().lower()
    seq = tok.texts_to_sequences([text])
    seq = pad_sequences(seq, maxlen=enc_maxlen, padding='post')
```

```
    states = encoder_model.predict(seq, verbose=0)
    target = np.array([[START_ID]])
    out_ids = []
    for _ in range(max_len):
        logits, h, c = decoder_model.predict([target]+states, verbose=0)
        token_id = int(np.argmax(logits[0, -1, :]))
        if token_id == END_ID:
            break
        if token_id != START_ID:
            out_ids.append(token_id)
        target = np.array([[token_id]])
        states = [h, c]
    return ' '.join(id2word.get(i, '<unk>') for i in out_ids)
```

id2word = {v:k for k, v in tok.word_index.items()} START_ID = tok.word_index.get('') END_ID = tok.word_index.get('')

def greedy_decode(text, max_len=20): text = text.strip().lower() seq = tok.texts_to_sequences([text]) seq = pad_sequences(seq, maxlen=enc_maxlen, padding='post') states = encoder_model.predict(seq, verbose=0) target = np.array([[START_ID]]) out_ids = [] for _ in range(max_len): logits, h, c = decoder_model.predict([target]+states, verbose=0) token_id = int(np.argmax(logits[0, -1, :])) if token_id == END_ID: break if token_id != START_ID: out_ids.append(token_id) target = np.array([[token_id]]) states = [h, c] return ' '.join(id2word.get(i, '') for i in out_ids)

chat

In [33]:
```
user_text = "hi, how are you?"
print("User:", user_text)
print("Bot :", greedy_decode(user_text))
```

```
User: hi, how are you?
Bot : i think i don't have to the lot of the world.
```

Data Limitations The dataset (dialogs.txt) has ~3.7k dialog pairs — relatively small for training deep learning models. Dialog data is highly diverse (same input can have multiple valid replies). A single “gold” answer makes accuracy misleading. Many pairs are short and generic (“hi” → “hello”), which dominates the training and biases the model. Impact: The model cannot generalize well beyond memorized patterns. Accuracy measured against “exact matches” underestimates real-world performance.

2. Evaluation Metric Choice

Accuracy in classification assumes one correct label per input. In dialog, multiple responses are plausible ("how are you?" → "fine", "doing good", "I'm okay"). So even if the model outputs a valid but different response, accuracy = 0.

Impact: True quality is better captured by metrics like BLEU, ROUGE, or embedding similarity, not raw accuracy.

3. Model Architecture

The Seq2Seq RNN is basic (1–2 LSTM/GRU layers, greedy decoding). Impact: Outputs tend to be short, generic, and repetitive.

4. Preprocessing & Tokenization

Punctuation removal and lowercasing simplify the task but lose nuance. Rare words may get mapped to (unknown token), hurting output quality. If max sequence length was cut short, some inputs get truncated.

5. Training Factors

Small number of epochs (if <20) → underfitting. Too many epochs without regularization → overfitting (model memorizes training pairs, fails on test). Optimizer/learning rate may not be tuned.

Observations from My Notebook Histograms show most utterances under 10 words → short context. Vocabulary is dominated by stopwords → weak signals. The final accuracy (~67%) is reasonable for a small dataset with Seq2Seq. Many research papers on dialog with small corpora report 60–70% as baseline.