Multilayer Perceptron (MLP)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.utils.class_weight import compute_class_weight
import re
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 Dense, Dropout, Embedding, Flatten
from tensorflow.keras.callbacks import EarlyStopping
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
```

In [2]:
```
# ==========================
# 0. Cargar dataset
# ==========================
df = pd.read_excel("datasetv2.xlsx")  # columnas: message, target
```

In [3]:
```
def limpiar_texto(texto):
    # Minúsculas
    texto = texto.lower()
    # Eliminar caracteres especiales pero mantener números y letras
    texto = re.sub(r"[^a-z0-9áéíóúñ ]", " ", texto)
    # Eliminar espacios múltiples
    texto = re.sub(r"\s+", " ", texto).strip()
    return texto
```

In [4]:
```
df["message"] = df["message"].astype(str).apply(limpiar_texto)
```

In [5]:
```
X = df["message"].astype(str)   # evitar errores con int
y = df["target"].values
```

In [6]:
```
# Codificar target (spam/legit → 0/1)
encoder = LabelEncoder()
y = encoder.fit_transform(df["target"])
```

In [7]:
```
# Asegurar que no haya NaN y que todo sea string
df["message"] = df["message"].astype(str)
```

In [8]:
```
# 2. Dividir dataset
# =======================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
```

In [9]:
```
# =======================
# 3. Tokenización y padding
# =======================
```

```python
max_words = 10000
max_len = 100

tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)

X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq  = tokenizer.texts_to_sequences(X_test)

X_train_pad = pad_sequences(X_train_seq, maxlen=max_len, padding="post")
X_test_pad  = pad_sequences(X_test_seq, maxlen=max_len, padding="post")
```

In [10]:
```python
# Balanceo con SMOTE
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train_pad, y_train)
```

In [11]:
```python
# =======================
# 4. Class Weights
# =======================
class_weights = compute_class_weight(
    class_weight="balanced",
    classes=np.unique(y_train),
    y=y_train
)
class_weights = dict(enumerate(class_weights))
print("Class weights:", class_weights)
```

```
Class weights: {0: np.float64(0.8653341114677561), 1: np.float64(1.18430511182108
63)}
```

In [12]:
```python
# =======================
# 5. Modelo MLP
# =======================
model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=64))
model.add(Flatten())  # convierte embeddings a un vector plano
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(32, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(1, activation="sigmoid"))

model.compile(
    loss="binary_crossentropy",
    optimizer="SGD",
    metrics=["accuracy"]
)
```

In [13]:
```python
#agregamos EarlyStopping
# Definimos el callback
early_stop = EarlyStopping(
    monitor='val_loss',       # Métrica a observar
    patience=3,               # Cuántas epochs esperar sin mejora
    restore_best_weights=True   # Recuperar los mejores pesos
)
```

In [14]:
```python
# =======================
# 6. Entrenamiento
# =======================
```

```
history = model.fit(
    X_train_res, y_train_res,
    epochs=50,
    batch_size=32,
    validation_data=(X_test_pad, y_test),
    class_weight=class_weights,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.5442 - loss: 0.6891 - val_accuracy: 0.4247 - val_loss: 0.6791
Epoch 2/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6517 - loss: 0.6317 - val_accuracy: 0.7157 - val_loss: 0.6085
Epoch 3/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7187 - loss: 0.5737 - val_accuracy: 0.7255 - val_loss: 0.5823
Epoch 4/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7294 - loss: 0.5575 - val_accuracy: 0.7184 - val_loss: 0.5681
Epoch 5/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7368 - loss: 0.5495 - val_accuracy: 0.7236 - val_loss: 0.5682
Epoch 6/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7407 - loss: 0.5394 - val_accuracy: 0.7267 - val_loss: 0.5562
Epoch 7/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7457 - loss: 0.5392 - val_accuracy: 0.7145 - val_loss: 0.5648
Epoch 8/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7437 - loss: 0.5290 - val_accuracy: 0.7188 - val_loss: 0.5501
Epoch 9/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7509 - loss: 0.5331 - val_accuracy: 0.7192 - val_loss: 0.5429
Epoch 10/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7482 - loss: 0.5267 - val_accuracy: 0.7271 - val_loss: 0.5397
Epoch 11/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7531 - loss: 0.5196 - val_accuracy: 0.7153 - val_loss: 0.5467
Epoch 12/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7520 - loss: 0.5150 - val_accuracy: 0.7287 - val_loss: 0.5320
Epoch 13/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7569 - loss: 0.5097 - val_accuracy: 0.7302 - val_loss: 0.5306
Epoch 14/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7626 - loss: 0.5037 - val_accuracy: 0.7283 - val_loss: 0.5371
Epoch 15/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7683 - loss: 0.4974 - val_accuracy: 0.7829 - val_loss: 0.4733
Epoch 16/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7778 - loss: 0.4784 - val_accuracy: 0.7511 - val_loss: 0.4851
Epoch 17/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7965 - loss: 0.4641 - val_accuracy: 0.8199 - val_loss: 0.4223
Epoch 18/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8083 - loss: 0.4393 - val_accuracy: 0.8219 - val_loss: 0.3961
Epoch 19/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8275 - loss: 0.4061 - val_accuracy: 0.8647 - val_loss: 0.3522
Epoch 20/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8503 - loss: 0.3760 - val_accuracy: 0.8525 - val_loss: 0.3390
```

```
Epoch 21/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8655 - loss: 0.3469 - val_
accuracy: 0.8801 - val_loss: 0.2988
Epoch 22/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8801 - loss: 0.3234 - val_
accuracy: 0.8883 - val_loss: 0.2816
Epoch 23/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8869 - loss: 0.3047 - val_
accuracy: 0.9139 - val_loss: 0.2455
Epoch 24/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8963 - loss: 0.2775 - val_
accuracy: 0.9186 - val_loss: 0.2285
Epoch 25/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9075 - loss: 0.2659 - val_
accuracy: 0.9186 - val_loss: 0.2246
Epoch 26/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9141 - loss: 0.2514 - val_
accuracy: 0.9257 - val_loss: 0.2069
Epoch 27/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9184 - loss: 0.2363 - val_
accuracy: 0.9217 - val_loss: 0.2065
Epoch 28/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9211 - loss: 0.2323 - val_
accuracy: 0.9221 - val_loss: 0.2120
Epoch 29/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9260 - loss: 0.2190 - val_
accuracy: 0.9245 - val_loss: 0.2026
Epoch 30/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9276 - loss: 0.2093 - val_
accuracy: 0.9351 - val_loss: 0.1875
Epoch 31/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9326 - loss: 0.2004 - val_
accuracy: 0.9312 - val_loss: 0.1991
Epoch 32/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9374 - loss: 0.1934 - val_
accuracy: 0.9418 - val_loss: 0.1766
Epoch 33/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9411 - loss: 0.1889 - val_
accuracy: 0.9320 - val_loss: 0.1888
Epoch 34/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9421 - loss: 0.1839 - val_
accuracy: 0.9363 - val_loss: 0.1733
Epoch 35/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9441 - loss: 0.1730 - val_
accuracy: 0.9363 - val_loss: 0.1752
Epoch 36/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9486 - loss: 0.1668 - val_
accuracy: 0.9438 - val_loss: 0.1681
Epoch 37/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9513 - loss: 0.1576 - val_
accuracy: 0.9414 - val_loss: 0.1769
Epoch 38/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9535 - loss: 0.1496 - val_
accuracy: 0.9430 - val_loss: 0.1662
Epoch 39/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9552 - loss: 0.1492 - val_
accuracy: 0.9355 - val_loss: 0.1687
Epoch 40/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9548 - loss: 0.1440 - val_
accuracy: 0.9422 - val_loss: 0.1609
```

```
Epoch 41/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9573 - loss: 0.1380 - val_
accuracy: 0.9398 - val_loss: 0.1800
Epoch 42/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9618 - loss: 0.1307 - val_
accuracy: 0.9446 - val_loss: 0.1671
Epoch 43/50
215/215 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9624 - loss: 0.1269 - val_
accuracy: 0.9080 - val_loss: 0.2443
```

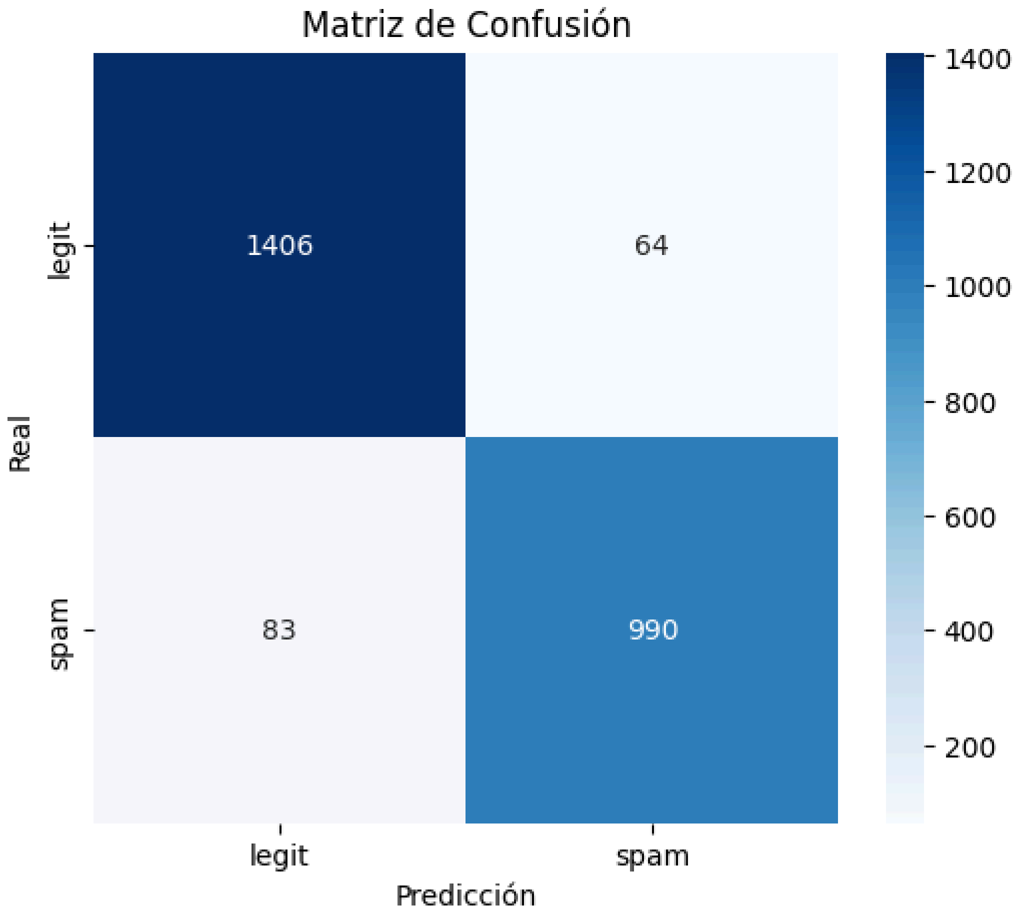
In [15]:

```python
# =======================
# 7. Evaluación
# =======================
y_pred = (model.predict(X_test_pad) > 0.5).astype("int32")

print("\nClassification Report:\n", classification_report(y_test, y_pred, target
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

Classification Report:
               precision    recall  f1-score   support

       legit       0.94      0.96      0.95      1470
        spam       0.94      0.92      0.93      1073

    accuracy                           0.94      2543
   macro avg       0.94      0.94      0.94      2543
weighted avg       0.94      0.94      0.94      2543


Confusion Matrix:
 [[1406   64]
 [  83  990]]
```

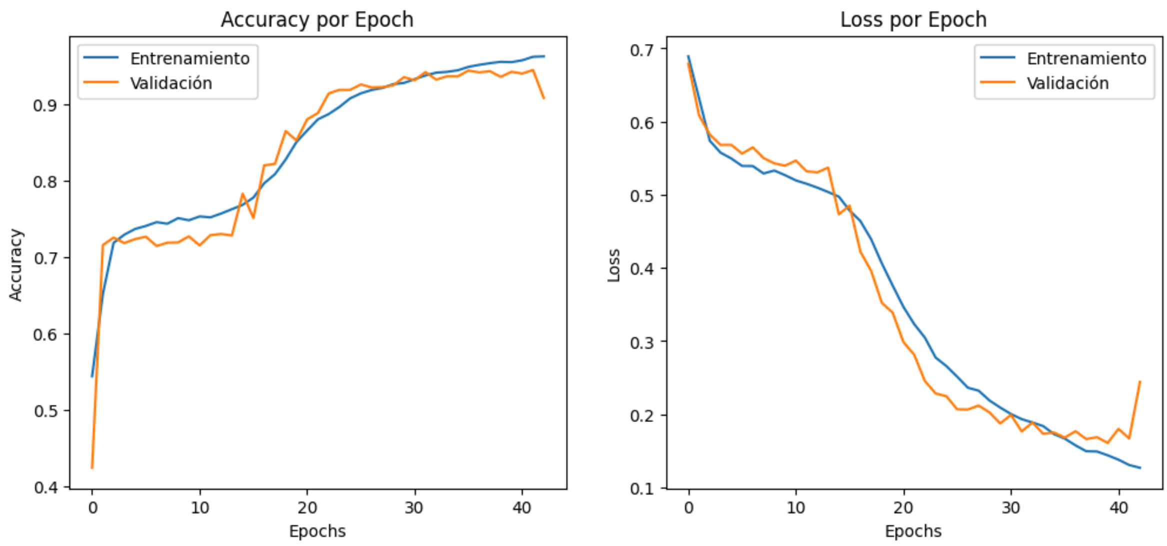
In [16]:

```python
# =======================
# 9. Matriz de confusión
# =======================
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["legit", "spam"]
plt.title("Matriz de Confusión")
plt.xlabel("Predicción")
plt.ylabel("Real")
plt.show()
```



In [17]:

```
# ========================
# 8. Gráfico de entrenamiento
# ========================
plt.figure(figsize=(12,5))

# Accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Accuracy por Epoch')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Loss por Epoch')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```


Accuracy por Epoch
Entrenamiento
Validación
Accuracy
Epochs
0.4
0.5
0.6
0.7
0.8
0.9
0
10
20
30
40
Loss por Epoch
Entrenamiento
Validación
Loss
Epochs
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
10
20
30
40

In [ ]: