

UNIVERSIDAD SAN CARLOS DE GUATEMALA

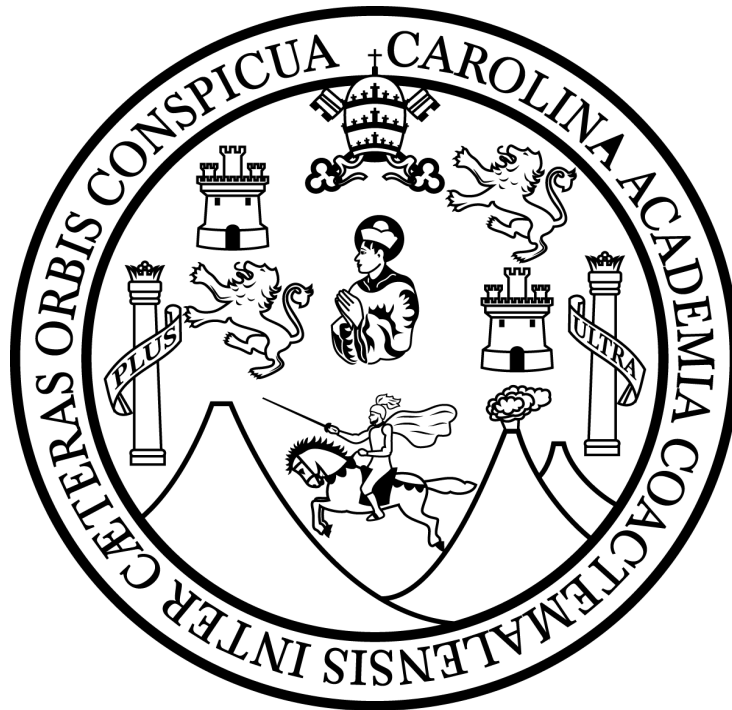
FACULTAD DE INGENIERIA

ESCUELA DE CIENCIAS Y SISTEMAS

INTELIGENCIA ARTIFICIAL 1

SECCIÓN A

ESCUELA DE VACACIONES, DICIEMBRE 2024



**PROYECTO ÚNICO – FASE #2**

**GRUPO #5**

POR:

3134216330901

MORALES XICARA, ERICK DANIEL

3146392170901

HERNÁNDEZ SAPÓN, LEVÍ ISAAC

3348212820901

SÁNCHEZ SANTOS, LUIS FERNANDO

GUATEMALA, QUETZALTENANGO, 10/12/2024

## INDICE

<b>MANUAL TÉCNICO.....</b>	<b>3</b>
<b>Descripción del Proyecto.....</b>	<b>3</b>
<b>Enfoque para el desarrollo de la IA.....</b>	<b>3</b>
Redes Neuronales Recurrentes (RNN).....	4
Intents.....	5
<b>Arquitectura para el desarrollo de la IA.....</b>	<b>6</b>
Implementación y Flujo.....	7
<b>Fase de entrenamiento para la IA.....</b>	<b>8</b>
Data Entrada.....	8
<b>Fase de entrenamiento para la IA mediante en Intents.....</b>	<b>9</b>
Procesamiento de la Entrada.....	9
Seleccionar respuesta aleatoria.....	9
Generar Vocabulario.....	10
Preprocesamiento de la data:.....	10
Creación del modelo.....	11
Entrenamiento del modelo.....	11
Predecir Respuesta.....	12
<b>Tecnologías Utilizadas.....</b>	<b>13</b>
Integración de Librerías.....	13
<b>BIBLIOGRAFÍA.....</b>	<b>15</b>

# **MANUAL TÉCNICO**

## **Descripción del Proyecto**

El primer proyecto asignado para el curso de Inteligencia Artificial 1, tiene como objetivo principal la creación de un modelo de inteligencia artificial, el cual sea capaz de poder procesar entradas de textos en idioma español y en inglés este pueda responder a las preguntas planteadas por el usuario de forma coherente, funcional y específica. Para la solución de dicho problema, se optará por utilizar tecnologías que han sido implementadas y basadas en el lenguaje de programación Python y este será entregado y desarrollado mediante un modelo incremental, dado que se le implementaran mejoras continuas al modelo de inteligencia artificial, lo cual garantiza que el resultado pueda llegar a cumplir de manera satisfactoria con los requisitos que ha impuesto el cliente en cada fase correspondiente al desarrollo del sistema.

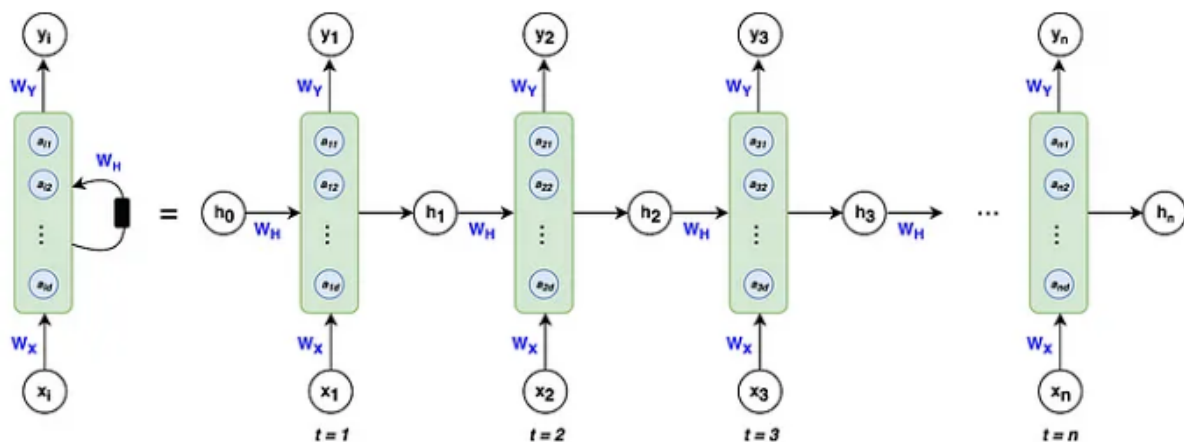
## **Enfoque para el desarrollo de la IA**

Para poder desarrollar el modelo de inteligencia artificial, se optará por implementar una arquitectura de Intenciones “Intents”, la cual está basada en redes neuronales recurrentes (RNN) con capas Long Short-Term Memory (LSTM). Dicho enfoque que será implementado es ideal para poder realizar procesamiento del lenguaje natural (NLP), en el cual se presenta una relación entre la secuencia de entrada (pregunta realizada por el usuario) y una secuencia de salida (respuesta brindada por el modelo de IA).

## Redes Neuronales Recurrentes (RNN)

Las redes neuronales recurrentes son un tipo especializado de red neuronal diseñado para poder ser utilizada en datos secuenciales. Su principal característica es la capacidad de poder recordar información previa en la entrada y utilizarla en las decisiones actuales, lo cual lo hace ideal para tareas donde el orden que requieren los datos es importante, así como el análisis del lenguaje natural, series temporales y reconocimiento de patrones en audio o video. Las redes neuronales funcionan de la siguiente manera:

- o Estado Oculto: Este estado funciona como una memoria que almacena información sobre las entradas previas en la secuencia, lo cual le permite a la red poder tomar decisiones informadas en datos pasados
- o Pesos Compartidos: Las RNN emplean pesos para cada elemento de la secuencia, lo que ayuda a reducir la complejidad del modelo y facilita la generación a diferentes tamaños de secuencias
- o Procesamiento secuencial: Cada salida depende no solo del elemento actual que esta siendo analizado, sino también del estado oculto, el cual refleja la información acumulada de entradas anteriores
- o Estructura en bucle: Las RNN tienen conexiones recurrentes que permiten poder pasar información de un estado al siguiente dentro de la misma red.



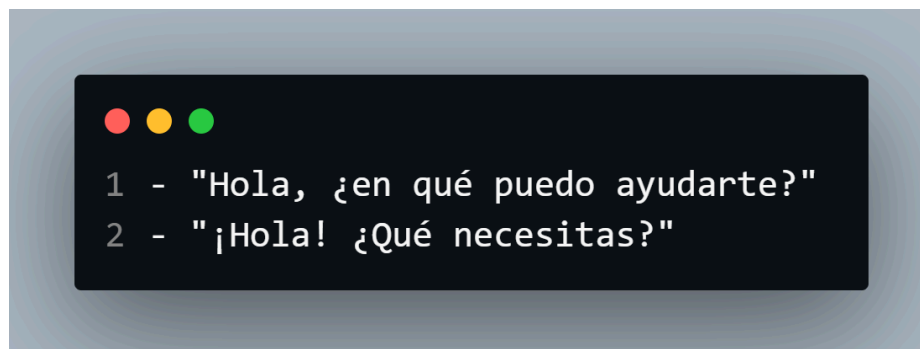
## Intents

Los Intents son aplicados en el ámbito del procesamiento del lenguaje natural, es una representación estructurada de la intención que puede llegar a tener el usuario al comunicarse con un sistema. Se utiliza principalmente al aplicarlo con chatbots, asistentes virtuales y sistemas de clasificación del texto. Para la maquetación de un Intent son necesarios los siguientes componentes:

- o Nombre del Intent: Una etiqueta o identificador único que describe la intención que pueda llegar a tener el usuario.
- o Patrones de entrada: Estas son representaciones de frases que presenten ejemplos de cómo un usuario podría llegar a expresar una intención, como podría ser en la siguiente entrada:



- o Respuestas: Estas son salidas generadas por el sistema para poder identificar de forma adecuada un intent, como pudiese llegar a ser las pertenecientes a un saludo:



- o Datos adicionales: Se le puede asignar a estos modelos información extra que el sistema pudiese llegar a necesitar para poder cumplir con la intención que pueda

llegar a pretender el usuario que se le presente, como pueden ser parámetros o entidades.

Para poder cumplir con las especificaciones de la creación del modelo, es posible utilizar **Tensorflow**, el cual es utilizado para poder construir y entrenar modelos que puedan clasificar intents. El modelo pretende asociar patrones de texto (entrada) con un intent específico (salida), para poder cumplir con dicho proceso general implica:

1. Procesamiento de Data: Se procede a convertir cada una de las frases de entrada en un entorno de formato que el modelo pueda entender (vectores numéricos)
2. Definición del modelo: Se procede a diseñar una red neuronal que clasifique las frases en intents.
3. Entrenamiento del modelo: Se ajusta el modelo usando datos de entrenamiento
4. Predicción: Se utiliza el modelo entrenado para poder clasificar nuevas frases de entrada.

### **Arquitectura para el desarrollo de la IA**

La arquitectura se divide en dos componentes principales: el codificador y el decodificador, los cuales trabajan en conjunto con la finalidad de poder comprender el contexto de la entrada y poder generar una respuesta coherente basada en el contexto de la pregunta realizada por el usuario. Adicionalmente fueron implementadas capas de embeddings para poder representar palabras como vectores densos en un espacio semántico y capas densas para la predicción de tokens en la secuencia de salida.

El codificador procesa la entrada (pregunta) y genera un conjunto de estados internos que representan el contexto de la secuencia. El proceso que realiza para poder llevar a cabo dicha secuencia es la siguiente:

- o Input Layer: Recibe secuencias tokenizadas y las convierte en embeddings utilizando una capa de tipo Embedding.
- o LSTM Layer: Genera representaciones contextuales y produce los estados finales de la celda (state\_h, state\_c) que son pasados al decodificador.

- o Output: Los estados finales del codificador (state\_h, state\_c) que encapsulan la información semántica de la entrada.

El decodificador genera una respuesta basada en los estados recibidos del codificador y los datos de entrada del decodificador. El proceso que realiza para poder llevar a cabo dicha secuencia es la siguiente:

- o Input Layer: Recibe las secuencias de inicio (<START>) como entrada inicial.
- o Embedding Layer: Similar al codificador, convierte tokens en vectores densos.
- o LSTM Layer: Genera una secuencia de estados y predicciones para cada token de la respuesta.
- o Dense Layer: Aplica una función de softmax para producir una distribución de probabilidad sobre el vocabulario, prediciendo la siguiente palabra.

## Implementación y Flujo

### 1. Procesamiento de datos

- ❖ Las preguntas y respuestas fueron extraídas de archivos YAML y limpiadas mediante expresiones regulares para normalizar texto.
- ❖ Se agregaron etiquetas especiales <START> y <END> a las respuestas, definiendo puntos claros para el inicio y fin de las secuencias de salida.

### 2. Tokenización:

- ❖ Se utilizó un tokenizer para convertir palabras en índices enteros que representan la posición de cada palabra en el vocabulario.
- ❖ La tokenización incluyó palabras de preguntas y respuestas para garantizar una representación uniforme del vocabulario.

### 3. Entrenamiento del Modelo:

- ❖ Entrada del Codificador: Estas se brindan a través de las secuencias de preguntas preprocesadas.
- ❖ Entrada del Decodificador: Estas son las respuestas que poseen la etiqueta <START>.

- ❖ Salida Esperada: Estas respuestas son tokenizadas para poder ser empleadas por el modelo de IA

#### 4. Inferencia

- ❖ Se creó un modelo separado para el codificador, el cual toma una pregunta y devuelve los estados contextuales.
- ❖ El decodificador se implementó como un modelo independiente para generar respuestas palabra por palabra, utilizando los estados iniciales del codificador.
- ❖ El flujo de predicción utiliza una secuencia vacía que es actualizada iterativamente con las palabras generadas, hasta alcanzar <END> o el límite de longitud máxima.

### Fase de entrenamiento para la IA

#### Data Entrada

Para que se pueda realizar el entrenamiento de la IA, se optó por utilizar archivos JSON, los cuales tienen una estructura clara y se emplearon para almacenar datos de conversaciones con categorías temáticas específicas y preguntas/respuestas asociadas. Estos archivos contienen dos elementos principales:

```
[
  {
    "input": "hola",
    "output": [
      "hola como estas",
      "hola que tal",
      "buenos días como estas"
    ]
  },
  {
    "input": "bien",
    "output": [
      "me alegra saberlo, dime en que puedo ayudarte, estoy entrenado para temas de programacion ",
      "que alegría, puedo ayudarte en distintos temas, estoy entrenado para temas de programacion",
      "me alegra escuchar eso, puedo ayudarte con el tema de programacion"
    ]
  },
]
```




## Fase de entrenamiento para la IA mediante en Intents

### Procesamiento de la Entrada

Para que el modelo sea entrenado, es necesario convertir un texto de entrada en un vector binario (array de ceros y unos) según un vocabulario predefinido, para ello es necesario realizar dicha conversión y se usaran los siguientes elementos:

- o `input.toLowerCase().split(' ')`: Convierte el texto a minúsculas y lo divide en palabras usando los espacios como delimitadores.
- o `new Array(vocabulary.length).fill(0)`: Crea un vector del mismo tamaño que el vocabulario, inicializado con ceros.
- o `words.forEach(...)`: Este método se encarga de iterar cada palabra sobre el texto; si la palabra esta en el vocabulario, marca el índice correspondiente con un 1; si la palabra no está en el vocabulario, imprime una advertencia.



```
1 function preprocessInput(input: string, vocabulary: string[]): number[] {
2   const words = input.toLowerCase().split(' ');
3   const vector = new Array(vocabulary.length).fill(0);
4
5   words.forEach((word) => {
6     const index = vocabulary.indexOf(word);
7     if (index !== -1) {
8       vector[index] = 1;
9     } else {
10      console.warn(`Palabra desconocida: ${word}`);
11    }
12  });
13
14  return vector;
15 }
```

### Seleccionar respuesta aleatoria

Se procede a seleccionar una respuesta aleatoria en base a una lista de plausibles respuestas. Al proporcionar respuestas variadas, el chatbot parece menos predecible, lo que mejora la experiencia del usuario y permite que cada intent tenga múltiples respuestas predefinidas, lo que hace que el sistema sea más adaptable y versátil.:

```

1 function getRandomResponse(responses: string[]): string {
2   const randomIndex = Math.floor(Math.random() * responses.length);
3   return responses[randomIndex];
4 }

```

### Generar Vocabulario

Se procede a crear un vocabulario único a partir de los datos de entrada para definir las palabras conocidas que el modelo puede interpretar. El vocabulario actúa como una referencia común para vectorizar las frases de entrada. Sin un vocabulario único, las palabras no se mapearían correctamente a índices consistentes. El vocabulario refleja las palabras más relevantes para la tarea, permitiendo al modelo enfocarse en datos específicos.

```

1 function generateVocabulary(data: { input: string; output: string[] }[]): string[] {
2   const allWords = data
3     .map((item) => item.input.toLowerCase().split(' '))
4     .reduce((acc, words) => acc.concat(words), []);
5   return Array.from(new Set(allWords));
6 }

```

### Preprocesamiento de la data:

```

const connectors = [
  "y", "que", "o", "u", "pero", "porque", "aunque", "si", "cuando",
  "como", "por", "a", "de", "en", "con", "para", "el", "la", "los", "las",
  "un", "una", "unos", "unas", "al", "del"
];

```

Se intentó puedes normalizar mejor la data, a la hora de pasar por el modelo para predecir, eliminando la información “basura” y poder pasarle un mejor contexto al modelo

## Creación del modelo

Para definir la arquitectura de una red neuronal que clasifica entradas textuales (representadas como vectores) en una de varias categorías (intents). Se debe de redistribuir de la siguiente manera:

- o Capa de Entrada - `inputShape: [vocabularyLength]`: Asegura que cada entrada tiene el mismo tamaño que el vocabulario. Cada vector representa la presencia/ausencia de palabras.
- o Capas Ocultas – *units: 16 y 8*: Estas capas realizan transformaciones no lineales, capturando patrones complejos en los datos.
- o Capas de salida:
  - o `units: outputLength`: Representa el número de intents posibles.
  - o `activation: 'softmax'`: Genera probabilidades para cada intent. El modelo predice el intent con la probabilidad más alta.
- o Compilación del modelo:
  - o `optimizer: 'adam'`: Algoritmo eficiente para ajustar los pesos del modelo.
  - o `loss: 'categoricalCrossentropy'`: Calcula qué tan diferente es la predicción del modelo respecto al objetivo.

```
1 function getRandomResponse(responses: string[]): string {
2   const randomIndex = Math.floor(Math.random() * responses.length);
3   return responses[randomIndex];
4 }
5 function generateVocabulary(data: { input: string; output: string[] }[]): string[] {
6   const allWords = data
7     .map((item) => item.input.toLowerCase().split(' '))
8     .reduce((acc, words) => acc.concat(words), []);
9   return Array.from(new Set(allWords));
10 }
```

## Entrenamiento del modelo

Entrenar el modelo para que aprenda a asociar entradas (frases vectorizadas) con salidas (intents). Las entradas (inputs) son vectores que representan frases de entrada y las salidas

(labels) son vectores one-hot que representan los intents. Es necesario que al modelo se le asigne un modelo supervisado ajusta sus pesos internos para minimizar la diferencia entre sus predicciones y los valores esperados. El modelo es almacenado y entrenado permite su reutilización para predicciones futuras sin necesidad de volver a entrenarlo.

```
1 async function trainModel(): Promise<void> {  
2   const vocabulary = generateVocabulary(trainingData);  
3   const model = createModel(vocabulary.length, trainingData.length);  
4  
5   const inputs = trainingData.map((data) => preprocessInput(data.input, vocabulary));  
6   const labels = trainingData.map((_, i) => {  
7     const labelVector = new Array(trainingData.length).fill(0);  
8     labelVector[i] = 1;  
9     return labelVector;  
10  });  
11  
12  const xs = tf.tensor2d(inputs);  
13  const ys = tf.tensor2d(labels);  
14  
15  await model.fit(xs, ys, {  
16    epochs: 200,  
17    batchSize: 15,  
18    shuffle: true,  
19  });  
20  
21  await model.save('localStorage://chatbot-model');  
22  console.log('Modelo entrenado y guardado.');
```

### Predecir Respuesta

Una vez que el modelo haya sido entrenado para poder clasificar una nueva entrada y seleccionar una respuesta apropiada, para ello se recupera el modelo previamente entrenado y guardado. Para poder llegar a predecir el intent, se transforma la frase en un vector y calcula la probabilidad de cada intent y selecciona el intent con la mayor probabilidad. Para que la respuesta se pueda generar, se mapea intent predicho a un conjunto de posibles respuestas y selecciona una de ellas.

```

1 async function predictResponse(input: string): Promise<string> {
2   const vocabulary = generateVocabulary(trainingData);
3   const model = await tf.loadLayersModel('localStorage://chatbot-model');
4
5   const processedInput = preprocessInput(input, vocabulary);
6   const tensorInput = tf.tensor2d([processedInput], [1, vocabulary.length]);
7
8   const prediction = model.predict(tensorInput) as tf.Tensor;
9   const predictedIndex = prediction.argmax(-1).dataSync()[0];
10
11  const possibleResponses = trainingData[predictedIndex].output;
12  return getRandomResponse(possibleResponses);
13 }

```

## Tecnologías Utilizadas

- o **Frontend:** Angular 12.x utilizando TypeScript
- o **Desarrollo de Modelo IA:** Python 3.12.x

## Integración de Librerías

- o Numpy: Es una biblioteca fundamental para cálculos numéricos en Python, proporcionando herramientas eficientes para trabajar con matrices, tensores y operaciones matemáticas de bajo nivel. Durante el entrenamiento del modelo, las secuencias de entrada (preguntas) y las salidas (respuestas) se representan como matrices numéricas. Numpy permite transformar las listas tokenizadas en arreglos multidimensionales listos para el procesamiento por TensorFlow.
- o Pandas: Es una biblioteca de análisis de datos diseñada para manipular y explorar estructuras tabulares o series de datos. Durante la etapa de preprocesamiento, Pandas se utiliza para estructurar y validar preguntas y respuestas extraídas de los archivos YAML. Permite verificar la consistencia de los datos antes de tokenizarlos.
- o Tensorflow: Es un framework de aprendizaje automático ampliamente utilizado para construir, entrenar y desplegar modelos de deep learning. Este proporciona las capas Embedding, LSTM y Dense que conforman el modelo Seq2Seq, cuyas capas LSTM manejan el aprendizaje de patrones secuenciales, mientras que la capa densa con activación softmax genera predicciones. Esto nos permite compilar el modelo con el

optimizador RMSprop y la pérdida categórica cruzada, garantizando una convergencia efectiva.

- o Pickle: Es una biblioteca de serialización que convierte objetos de Python en flujos de bytes para almacenarlos o transmitirlos. El tokenizer, junto con el tamaño máximo de las secuencias (`maxlen_questions` y `maxlen_answers`), se guarda en un archivo para reutilizarse durante la inferencia. Lo cual se utiliza para deserializar los objetos almacenados previamente, facilitando el despliegue del modelo sin necesidad de reentrenarlo.
- o PyYAML: Es una biblioteca para leer y escribir archivos en formato YAML, ampliamente utilizado para configurar datos estructurados. Se usa para cargar conversaciones de archivos YAML. Cada archivo contiene preguntas y respuestas en un formato estructurado, que se procesa para generar los datos de entrenamiento. Esto nos permite procesar archivos con diferentes estructuras dentro de una misma carpeta, garantizando que todos los datos relevantes sean incorporados al modelo.
- o Gensim: Es una biblioteca especializada en procesamiento de lenguaje natural y modelado semántico, particularmente útil para análisis y representación vectorial de vocabularios. Complementa al tokenizer proporcionando herramientas para crear y explorar el vocabulario derivado de las preguntas y respuestas procesadas. Este facilita la tokenización avanzada y la limpieza del texto, asegurando que las palabras clave sean representadas correctamente en el modelo.

## BIBLIOGRAFÍA

- Medium (2024), *Recurrent Neural Networks (RNN) from Basic to Advanced*  
[https://medium-com.translate.google/@sachinsoni600517/recurrent-neural-networks-rnn-from-basic-to-advanced-1da22aafa009?\\_x\\_tr\\_sl=en&\\_x\\_tr\\_tl=es&\\_x\\_tr\\_hl=es&\\_x\\_tr\\_pto=tc&\\_x\\_tr\\_hist=true](https://medium-com.translate.google/@sachinsoni600517/recurrent-neural-networks-rnn-from-basic-to-advanced-1da22aafa009?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc&_x_tr_hist=true)
- Medium (2020), *Understanding Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM)*.  
<https://medium.com/analytics-vidhya/undestanding-recurrent-neural-network-rnn-and-long-short-term-memory-lstm-30bc1221e80d>
- Medium (2024), *Recurrent Neural Networks (RNN) from Basic to Advanced*.  
<https://medium.com/@sachinsoni600517/recurrent-neural-networks-rnn-from-basic-to-advanced-1da22aafa009>
- Medium (2023), *Introduction to Long Short-Term Memory (LSTM)*.  
<https://medium.com/analytics-vidhya/introduction-to-long-short-term-memory-lstm-a8052cd0d4cd>
- Medium (2021), *Long Short-Term Memory Networks*.  
<https://medium.com/analytics-vidhya/long-short-term-memory-networks-23119598b66b>
- Medium (2024), *Understanding Long Short-Term Memory (LSTM) in Deep Learning: A Comprehensive Guide with Sample Code*.  
<https://medium.com/@hkabhi916/understanding-long-short-term-memory-lstm-in-deep-learning-a-comprehensive-guide-with-sample-72d884be4470>
- Medium (2024), *Sequence-to-Sequence Models*.  
<https://medium.com/@calin.sandu/sequence-to-sequence-models-603920ce9e96#:~:text=Seq2Seq%20models%20have%20been%20instrumental,text%20summarization%2C%20and%20speech%20recognition.&text=The%20encoder%20in%20a%20Seq2Seq,vector%20or%20a%20hidden%20state>

- Medium (2021), *Encoder-Decoder Seq2Seq Models, Clearly Explained*.  
<https://medium.com/analytics-vidhya/encoder-decoder-seq2seq-models-clearly-explained-c34186bf49b>
- Medium. (2023). *Understanding Sequence-to-Sequence (Seq2Seq) Models and their Significance*.  
<https://medium.com/aimonks/understanding-sequence-to-sequence-seq2seq-models-and-their-significance-d2f0fd5f6f7f>