FINE-TUNING DE UN MODELO DE BAJO COSTO PARA CORRECCIÓN AUTOMÁTICA DE FRASES EN ESPAÑOL

José Alejandro Torres Muñoz, Fiorella Valentina Trujillo Abello, Edwin Esteban Pineda Bociga, Javier Stiven Muñoz Cortes

¿CUÁL ES LA TAREA NLP?

- Tarea definida: Corrección automática de texto en español (Grammar Error Correction).
- Objetivo: Detectar errores de ortografía, acentos y estructura en frases cortas, y generar la versión corregida.

- Ejemplo:
- Entrada → "El nio comio manzana rroja ayer"
- Salida → "El niño comió manzana roja ayer"

¿POR QUÉ ESTA TAREA?

- Problema muy común en aplicaciones educativas o chatbots.
- El texto es corto: bajo costo computacional.
- Fácil de evaluar con ejemplos reales.
- Se puede resolver con modelos pequeños y económicos.

DATASET UTILIZADO

- Dataset propio construido manualmente.
- Pares input - output con frases incorrectas y corregidas.
- Formato JSONL.

- {"input": "El nio comio", "output": "El niño comió"}

CONSTRUCCIÓN DEL DATASET

- Se creo un dataset para esta tarea, debido a que no existe un dataset público grande para corrección ortográfica con la estructura deseada (Input erroneo → Output corregido). Por lo que se creo un script en Python el cual se alimente de diferentes fuentes de información publica generado errores de manera aleatoria.

input_text	target_text
Tras una infinidad de tanteos en la oscuriídád, empesó a encontrar respuestas.	Tras una infinidad de tanteos en la oscuridad, empezó a encontrar respuestas.
Tras una infinidad de la en tanteos oscuridad, empezó a eencontrar réspuesas.	Tras una infinidad de tanteos en la oscuridad, empezó a encontrar respuestas.
Tras una infinidad de tanteos eñ la oscruidad, empezó a énsontrar respuestas.	Tras una infinidad de tanteos en la oscuridad, empezó a encontrar respuestas.

- Limpieza y filtrado:
- Eliminación de secciones, tablas, listas
- Filtrado por longitud
- Estándar: frases 5–40 palabras, párrafos 25–120

- Fuentes usadas:
- Wikipedia (temas variados) → texto formal
- OpenSubtitles → lenguaje cotidiano
- Total inicial: decenas de miles de frases limpias

GENERACIÓN SINTÉTICA DE ERRORES

Tipos de errores introducidos:

- Errores fonéticos (b/v, c/s/z, g/j, ll/y...)
- Omisión o inserción aleatoria de letras
- Homófonos confusos (valla/vaya, echo/hecho...)
- Ruido de teclado
- Errores de acentuación
- Reordenamiento de palabras
- Uniones incorrectas ("del", "porla", etc.)

~200.000 pares (texto con errores - texto correcto)

El script genera tres frases errores por cada oración y de esta manera el modelo aprenda diferentes tipos de errores en las mismas frases, asi mismo el dataset se genera previo al entrenamiento del modelo por lo que el usuario lo puede ajustar a su necesidad.

GENERACIÓN SINTÉTICA DE ERRORES

```python
topics = [
    # Ciencia y tecnología
    "Inteligencia_artificial", "Aprendizaje_automático", "Red_neuronal",
    "Robótica", "Biotecnología", "Química_Orgánica", "Física_cuántica",
    "Teoría_de_la_relatividad", "Nanotecnología", "Ingeniería_genética",
    "Computadora", "Internet", "Ciberseguridad", "Criptografía",
    "Algoritmo", "Programación", "Sistemas_operativos", "Software_libre",
    "Base_de_datos", "Tecnología", "Electrónica", "Matemáticas",
    "Estadística", "Álgebra", "Cálculo", "Geometría",

    # Historia
    "Historia_de_Colombia", "Imperio_romano", "Antigua_Grecia",
    "Edad_Media", "Revolución_Francesa", "Primera_Guerra_Mundial",
    "Segunda_Guerra_Mundial", "Cristóbal_Colón", "Civilización_egipcia",
    "Civilización_maya", "Historiografía", "Arqueología",

    # Ciencia natural
    "Biología", "Zoología", "Botánica", "Genética", "Evolución",
    "Célula_(biología)", "Virus", "Bacteria", "Anatomía_humana",
    "Ecología", "Climatología", "Cambio_climático", "Geología",
    "Astronomía", "Sistema_solar", "Cosmología",

    # Medicina y salud
    "Medicina", "Farmacología", "Enfermedad", "Neurología",
    "Psiquiatría", "Psicología", "Nutrición", "Deporte", "Fisioterapia",
```

```python
# 1. homófonos
for a,b in HOMOPHONES:
    if base == a and random.random() < 0.3: return b
    if base == b and random.random() < 0.3: return a

# 2. quitar acentos
if random.random() < 0.50:
    base = unidecode(base)

# 3. fonética
for k,v in PHONETIC.items():
    if k in base and random.random() < 0.2:
        base = base.replace(k, v)

#  insertar "h" aleatoriamente
if random.random() < 0.1:
    pos = random.randint(0, len(base))
    base = base[:pos] + "h" + base[pos:]

# unir palabras pequeñas (ej: "de el" → "del")
if random.random() < 0.08:
    if base in ["de","a","por","para","con"]:
        base = base + random.choice(["l","la","lo","el"])
```

SELECCIÓN DEL MODELO

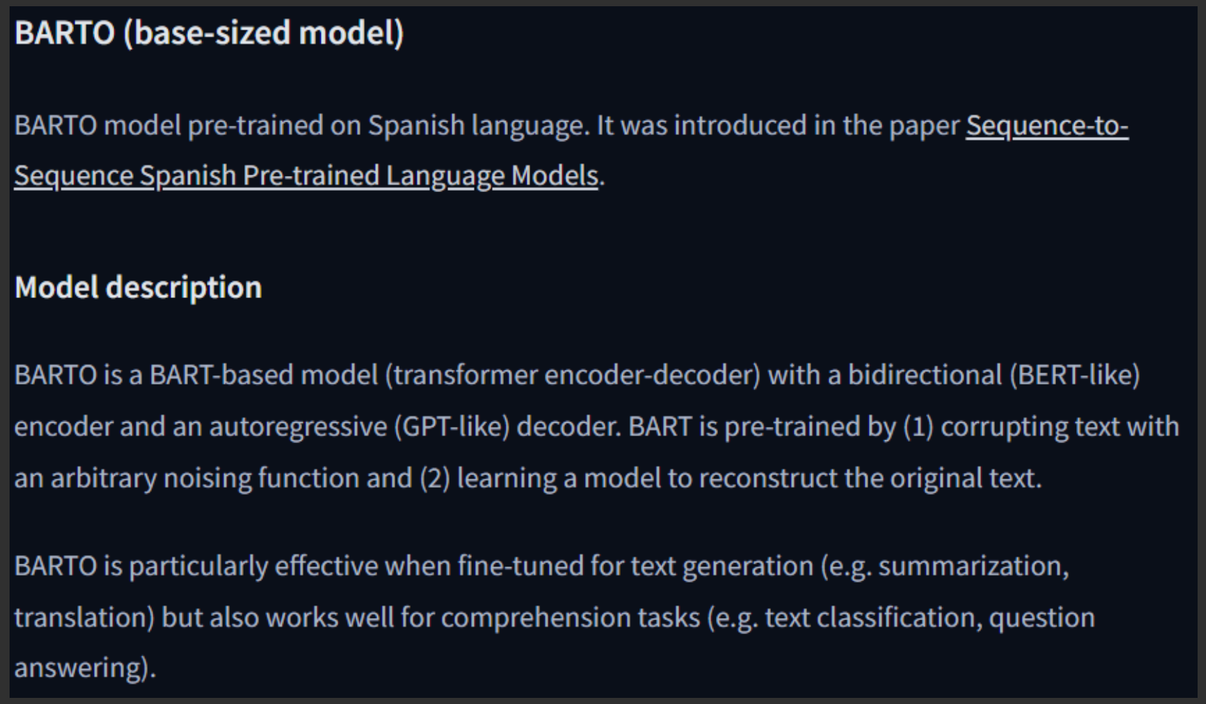
BART-base en español
Repositorio: vgaraujov/bart-base-spanish

- Arquitectura encoder–decoder: ideal para corrección
- Mejor comprensión bidireccional del texto
- Modelo liviano: barato de entrenar
- Preentrenado específicamente para español

El modelo no se entrena desde cero, se toma un BART ya preentrenado en español, y luego se entrena con datos nuevos sobre lo que ya conoce mediante fine-tuning.

Las capas iniciales del encoder y decoder contienen el “conocimiento estable” del idioma.
Si permites que se modifiquen:

- el modelo “olvida” español
- pierde comprensión base
- empeora en generalización



BARTO (base-sized model)

BARTO model pre-trained on Spanish language. It was introduced in the paper Sequence-to-Sequence Spanish Pre-trained Language Models.

Model description

BARTO is a BART-based model (transformer encoder-decoder) with a bidirectional (BERT-like) encoder and an autoregressive (GPT-like) decoder. BART is pre-trained by (1) corrupting text with an arbitrary noising function and (2) learning a model to reconstruct the original text.

BARTO is particularly effective when fine-tuned for text generation (e.g. summarization, translation) but also works well for comprehension tasks (e.g. text classification, question answering).

“vgaraujov/bart-base-spanish · Hugging Face,” Huggingface.co, Mar. 16, 2023. https://huggingface.co/vgaraujov/bart-base-spanish (accessed Nov. 24, 2025).

CONFIGURACIÓN DEL ENTRENAMIENTO

```
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)

training_args = TrainingArguments(
    output_dir="./bart-correccion",
    learning_rate=3e-5,
    per_device_train_batch_size=16,
    gradient_accumulation_steps=1,
    num_train_epochs=4,
    fp16=True,
    eval_strategy="epoch",
    logging_steps=50,
    save_strategy="epoch",
    save_total_limit=3,
    report_to="none"
)
```

Estrategia usada

- Congelar todo el encoder
- Entrenar solo las últimas 2 capas del decoder
- Esto reduce:
- tiempo
- costo
- riesgo de olvidar el preentrenamiento

Hiperparámetros clave

- LR: 3e-5
- Batch size: 16
- Épocas: 4
- FP16 activado
- Beam search en inferencia

ENTRENAMIENTO

[39404/39404 44:48, Epoch 4/4]

Epoch	Training Loss	Validation Loss
1	0.766500	0.581606
2	0.669000	0.510618
3	0.606000	0.480798
4	0.578500	0.471860

Pipeline

- Tokenización de pares input: target
- DataCollator para seq2seq
- Entrenamiento con Trainer
- Evaluación por época
- Guardado del mejor checkpoint

PRUEBA DE MODELO

```
text = "Yo qiero hir al parke con migo amijo.."
inputs = tokenizer(text, return_tensors="pt").to("cuda")
outputs = model.generate(**inputs, num_beams=4, max_length=64)
print("\nTexto original:", text)
print("Corregido:", tokenizer.decode(outputs[0], skip_special_tokens=True))
```

```
Texto original: Yo qiero hir al parke con migo amijo..
Corregido: Yo quiero ir al parke con migo amijo.
```

```
text = "La caza de mi papa ez mu grande i bonita."
inputs = tokenizer(text, return_tensors="pt").to("cuda")
outputs = model.generate(**inputs, num_beams=4, max_length=64)
print("\nTexto original:", text)
print("Corregido:", tokenizer.decode(outputs[0], skip_special_tokens=True))
```

```
Texto original: La caza de mi papa ez mu grande i bonita.
Corregido: La casa de mi papa es muy grande i bonita.
```

```
text = "Ella disen que nadien saves aser eso."
inputs = tokenizer(text, return_tensors="pt").to("cuda")
outputs = model.generate(**inputs, num_beams=4, max_length=64)
print("\nTexto original:", text)
print("Corregido:", tokenizer.decode(outputs[0], skip_special_tokens=True))
```

```
Texto original: Ella disen que nadien saves aser eso.
Corregido: Ella dice que nadien saben hacer eso.
```

```
text = "Tu hermana es mui intelijente i zabe resolver problemas."
inputs = tokenizer(text, return_tensors="pt").to("cuda")
outputs = model.generate(**inputs, num_beams=4, max_length=64)
print("\nTexto original:", text)
print("Corregido:", tokenizer.decode(outputs[0], skip_special_tokens=True))
```

```
Texto original: Tu hermana es mui intelijente i zabe resolver problemas.
Corregido: Tu hermana es muy inteligente i sabe resolver problemas.
```

CONCLUSIONES

- Al realizar el modelo para correccion en español, la complejidad aumenta debido a la cantidad de palabras del idioma, estas cuantan con tildes, acentos, letra “ñ”, asi como la h.
- De igual forma, al tratar de buscar datasets en español y modelos en español era más complejo, dificultando corrección de palabras como “i” o algunas otras que tienen sentido idiomático en ingles u otros idiomas
- Al crear un dataset sintetico, se intento agregar la mayor cantidad de errores generando tres posibles incorrectas con errores aleatorios, pero no es posible generar todos los posibles errores, por lo que el modelo siempre estara limitado a corregir solo los errores que conoce y no generalizar bien.
- Si se desea mejor el modelo a futuro, es recomendable tratar de mejorar/cambiar el modelo y asi este aprenda a generalizar los tipos de errores y no solo corregir los que sabe.

```
Texto original: vamoz a ugar a la play.
Corregido: Vamos a jugar a la play.
```

gracias por su atención