```
(venv) PS C:\Proyectos\NanoQuijote> # Descargar desde Gutenberg (usando Invoke-WebRequest)
>> Invoke-WebRequest -Uri "https://www.gutenberg.org/files/2000/2000-0.txt" -OutFile "quijote.txt"
>>
>> # Verificar descarga
>> Get-Content "quijote.txt" -TotalCount 5
>>
The Project Gutenberg eBook of Don Quijote, by Miguel de Cervantes Saavedra

This eBook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
(venv) PS C:\Proyectos\NanoQuijote> Set-Content -Path "entrenar.py" -Value @'
```

En la primera captura se utiliza el comando `Invoke-WebRequest` para descargar el libro del Quijote desde Project Gutenberg y guardarlo en un archivo llamado `quijote.txt`. Después se ejecuta `Get-Content -TotalCount 5` para mostrar las primeras líneas del archivo y verificar que la descarga se ha realizado correctamente. A continuación se crea el archivo `entrenar.py` mediante `Set-Content`, introduciendo en él el código encargado de preparar los datos y definir el modelo de lenguaje. Finalmente, se ejecuta `python entrenar.py`, lo que pone en marcha el proceso de entrenamiento usando el texto del Quijote como corpus.

```
(venv) PS C:\Proyectos\NanoQuijote> python entrenar.py
>>
Usando dispositivo: cuda
GPU: NVIDIA GeForce RTX 4070 Ti
VRAM disponible: 11.99 GB

Tamano del vocabulario: 105
Caracteres unicos:
 !"#$%'()*,-./0123456789:;?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]abcdefghijklmnopqrstuvwxyz¡«»¿ÁÉÍÑÓÚàáéíïñóùúü–
Total de tokens: 2,130,398
Tokens de entrenamiento: 1,917,358
Tokens de validacion: 213,040

Total de parametros del modelo: 10,819,689
Memoria estimada del modelo: 41.27 MB
```

En la segunda captura se observa la salida inicial del script `entrenar.py`. Primero se indica que se está utilizando el dispositivo `cuda` y la GPU disponible (NVIDIA GeForce RTX 4070 Ti) junto con la memoria de vídeo libre. A continuación se calcula el vocabulario de caracteres únicos del texto y se muestra el número total de tokens, diferenciando entre tokens de entrenamiento y de validación. También se imprime el número total de parámetros del modelo y una estimación de la memoria que ocupa. Por último comienza el bucle de entrenamiento, donde se muestran las iteraciones y los valores de `Train Loss` y `Val Loss`, lo que permite ver cómo el modelo va aprendiendo a medida que avanza el entrenamiento.

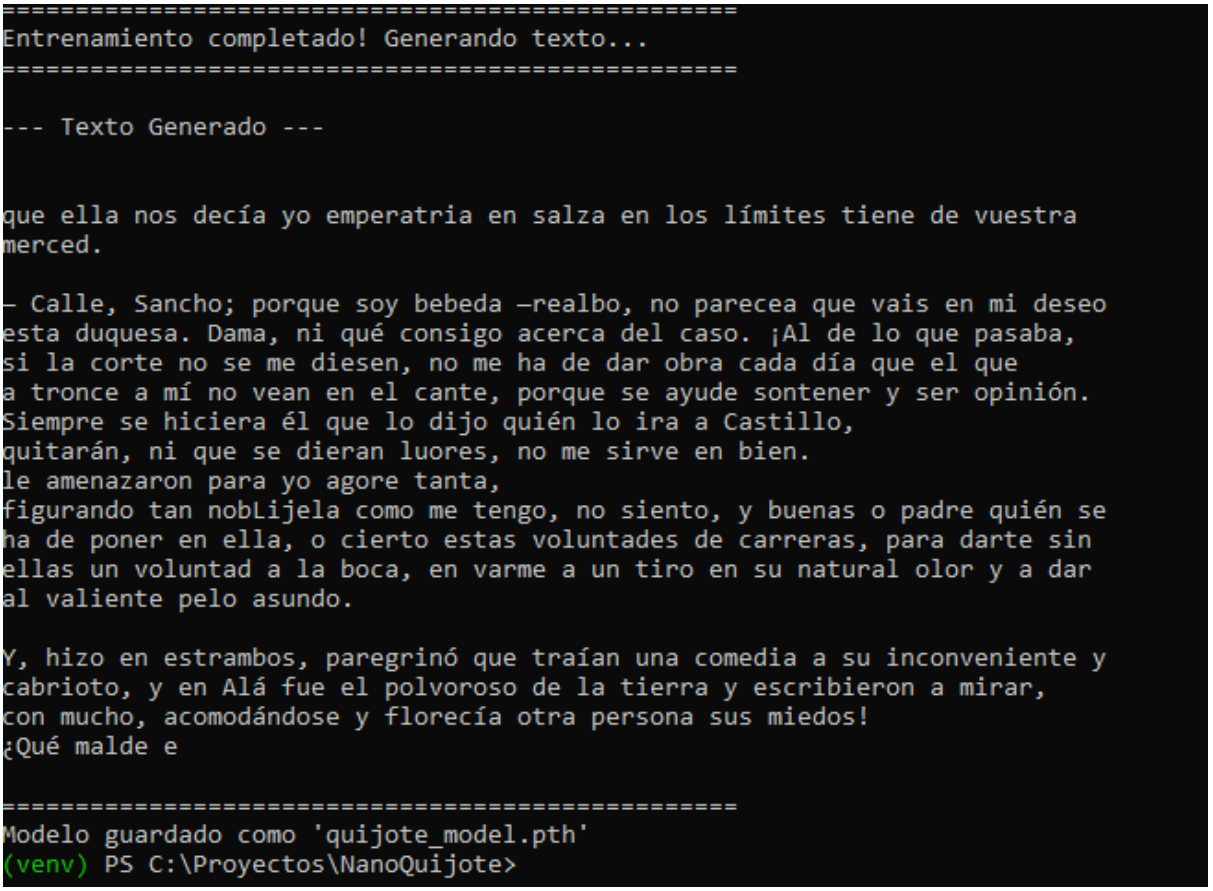
```
Iniciando entrenamiento...
==================================================
Iteracion     0 | Train Loss: 4.8939 | Val Loss: 4.8911
Iteracion   500 | Train Loss: 1.9575 | Val Loss: 2.1379
Iteracion  1000 | Train Loss: 1.5650 | Val Loss: 1.8125
Iteracion  1500 | Train Loss: 1.3951 | Val Loss: 1.6715
Iteracion  2000 | Train Loss: 1.2993 | Val Loss: 1.6186
Iteracion  2500 | Train Loss: 1.2396 | Val Loss: 1.5892
Iteracion  3000 | Train Loss: 1.1945 | Val Loss: 1.5414
Iteracion  3500 | Train Loss: 1.1596 | Val Loss: 1.4876
Iteracion  4000 | Train Loss: 1.1294 | Val Loss: 1.4827
Iteracion  4500 | Train Loss: 1.1032 | Val Loss: 1.4579
Iteracion  5000 | Train Loss: 1.0805 | Val Loss: 1.4244
Iteracion  5500 | Train Loss: 1.0609 | Val Loss: 1.4023
Iteracion  6000 | Train Loss: 1.0393 | Val Loss: 1.3917
Iteracion  6500 | Train Loss: 1.0230 | Val Loss: 1.3739
Iteracion  7000 | Train Loss: 1.0049 | Val Loss: 1.3594
Iteracion  7500 | Train Loss: 0.9910 | Val Loss: 1.3721
Iteracion  8000 | Train Loss: 0.9752 | Val Loss: 1.3516
Iteracion  8500 | Train Loss: 0.9588 | Val Loss: 1.3725
Iteracion  9000 | Train Loss: 0.9445 | Val Loss: 1.3645
Iteracion  9500 | Train Loss: 0.9306 | Val Loss: 1.3548

==================================================
Entrenamiento completado! Generando texto...
--------------------------------------------------
```

En esta captura se muestra el bucle de entrenamiento del modelo, donde periódicamente se imprime el estado de la pérdida. Se indica el número de iteración y los valores de `Train Loss` y `Val Loss` cada 500 iteraciones, comenzando en una pérdida de entrenamiento cercana a 4,89 y descendiendo progresivamente hasta valores en torno a 0,93. La pérdida de validación también disminuye desde aproximadamente 4,89 hasta alrededor de 1,35. Esta evolución refleja que el modelo está aprendiendo patrones del texto del Quijote y reduciendo el error tanto en los datos de entrenamiento como en los de validación a medida que avanzan las iteraciones.

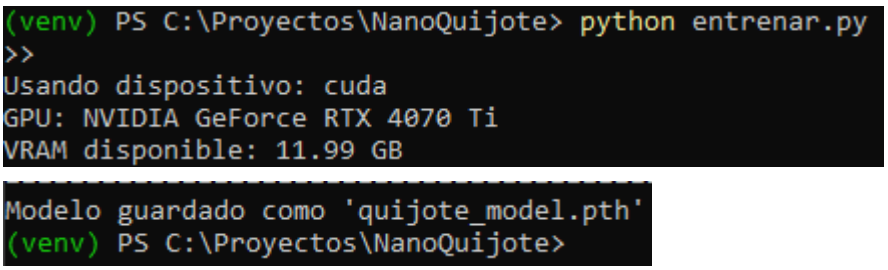
```python
# --- Hiperparametros (optimizados para RTX 4070 Ti) ---
batch_size = 64      # Aumentado para aprovechar los 12GB VRAM
block_size = 256     # Contexto mas largo (el doble que el original)
max_iters = 10000    # Mas iteraciones para mejor aprendizaje
eval_interval = 500
learning_rate = 3e-4
device = 'cuda' if torch.cuda.is_available() else 'cpu'
n_embd = 384         # Mayor capacidad (vs 256 original)
n_head = 6           # Mas cabezas de atencion
n_layer = 6          # Mas capas Transformer
dropout = 0.2
```

En esta captura se recogen los hiperparámetros definidos en el script `entrenar.py`. Entre ellos se encuentran `batch_size = 64` (tamaño de lote), `block_size = 256`

(longitud del contexto de entrada), `max_iters = 10000` (número máximo de iteraciones) y `eval_interval = 500` (frecuencia de evaluación). También se especifica `learning_rate = 3e-4` y el dispositivo de cómputo `device = 'cuda' if torch.cuda.is_available() else 'cpu'`. Además, se detallan los parámetros de la arquitectura Transformer: `n_embd = 384` (dimensión de los embeddings), `n_head = 6` (número de cabezas de atención), `n_layer = 6` (número de capas) y `dropout = 0.2`. En los comentarios se explica que estos valores se han ajustado para aprovechar los 12 GB de VRAM de la RTX 4070 Ti, permitir un contexto más largo y aumentar la capacidad del modelo respecto a una configuración básica.



En esta captura se ve el mensaje “Entrenamiento completado! Generando texto…” seguido de un bloque de texto generado por el modelo. El contenido utiliza vocabulario y estructuras propias del Quijote, aunque con frases parcialmente incoherentes debido al tamaño limitado del modelo y a que trabaja a nivel de caracteres. Este resultado demuestra que el modelo ha aprendido a imitar el estilo del corpus, combinando palabras arcaicas y giros lingüísticos característicos. Al final de la salida aparece el mensaje “Modelo guardado como 'quijote_model.pth'”, indicando que los pesos entrenados se han almacenado en un archivo para poder reutilizar el modelo posteriormente sin necesidad de reentrenarlo desde cero.



En la última captura se muestra una nueva ejecución de `python entrenar.py` dentro del mismo entorno virtual. De nuevo se indica que se está utilizando la GPU `cuda` con la NVIDIA GeForce RTX 4070 Ti y la VRAM disponible. A continuación se muestra de forma directa el mensaje “Modelo guardado como 'quijote_model.pth'”, sin detallar todo el proceso de entrenamiento. Esta captura refleja que el script puede ejecutarse varias veces para continuar el entrenamiento, ajustar parámetros o simplemente volver a guardar el modelo actualizado, manteniendo siempre el archivo `quijote_model.pth` como salida final del proceso.