ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA

Scuola di Ingegneria ed Architettura
Corso di Laurea in Ingegneria Elettronica

LARGE LANGUAGE MODELS SU RASPBERRY PI: DEPLOYMENT E ANALISI DELLE PRESTAZIONI

Elaborato in
Laboratorio Internet of Things

Tesi di Laurea di:
ALAN BAZZOCCHI

Relatore:
Prof. Ing.
ENRICO TESTI
Correlatore:
Prof. Ing.
ELIA FAVARELLI

SESSIONE III
ANNO ACCADEMICO 2024-2025

PAROLE CHIAVE

Large Language Models

Small Language Models

Raspberry Pi 2

Gemma 3 270M

Quantizzazione

Indice

Introduzione

L'obiettivo di questo elaborato è quello di analizzare una nuova frontiera dei large language models (LLM), ossia l'implementazione di modelli di dimensioni ridotte, gli *small language models* (SLM), laddove si hanno vincoli di latenza, costi operativi e limitazioni strutturali tali da non permettere l'utilizzo di LLM tradizionali. All'interno di questa categoria rientrano diversi modelli, tra cui Gemma 3 270M, che verrà descritto nel dettaglio nel corso dei successivi capitoli.

Questa classe di modelli di linguaggio trova importanti applicazioni nell'edge computing, inteso come l'elaborazione dei dati direttamente sui dispositivi che li generano (ad esempio sensori). Spesso le risorse di calcolo a disposizione su questi dispositivi sono ridotte, per questo l'obiettivo é quello di adottare modelli leggeri come gli SLM per svolgere l'analisi. Come esempio di dispositivo di edge computing, si é utilizzato per questo elaborato il Raspberry Pi 2 Model B.

La possibilità di eseguire modelli su dispositivi di questo tipo permette di ridurre il costo di accesso ai modelli LLM, uno dei principali ostacoli alla loro adozione su larga scala all'interno delle aziende. Il loro impiego offre anche altri vantaggi, tra cui il ridotto costo in termini computazionali ed energetici. Inoltre, la possibilitá di esecuzione on-device garantisce, oltre che a una maggiore tutela della privacy e della sicurezza dei dati, anche l'indipendenza dalla connessione Internet. Oltretutto, questi modelli presentano un'elevata facilitá di fine-tuning per domini specifici [1]. Di contro, risultano meno accurati rispetto a modelli di grandi dimensioni, ma possono comunque rappresentare un buon compromesso.

Nella prima parte dell'elaborato vengono indagati gli aspetti principali degli LLM, ponendo particolare attenzione a Gemma 3 270M e ad alcune delle sue quantizzazioni.

Si procede poi con un approfondimento sul Raspberry Pi 2 Model B, di cui vengono descritte le caratteristiche principali e le modalità con cui il modello è stato integrato e gestito all'interno del dispositivo.

Per concludere si analizzano le prestazioni del modello SLM su Raspberry Pi, evidenziando, tramite alcuni benchmark, le differenze di velocità e accuratezza delle risposte fra diverse quantizzazioni.

Capitolo 1

Large Language Models

Gli LLM sono modelli di intelligenza artificiale (IA) general-purpose, basati su reti neurali, con la capacità di ragionamento ed esecuzione di istruzioni. Vengono addestrati su una grande quantità di dati, rendendoli in grado di emulare i ragionamenti umani. Le porzioni di testo vengono spesso indicate anche con il termine *token*, che rappresenta una quantità elementare di testo a cui si fa riferimento all'interno dei modelli.

1.1 Caratteristiche degli LLM

Un tratto comune a tutti gli LLM è l'architettura *Transformer*, un meccanismo di *self-attention* che consente di pesare dinamicamente l'importanza di ogni token rispetto agli altri nella sequenza di testo considerata. Questo permette, in fase di generazione del token successivo, di valutare il contesto nel quale si trova (Figura 1.1). Tale meccanismo risolve il problema dell'ambiguità, ma causa un aumento drastico della complessità del modello. Il problema della complessità diventa ancora più critico nel caso di gestione di testi lunghi.

Gli LLM sono costituiti anche da una *matrice di embedding* dei token, che consente di associare a ciascun token un vettore numerico (Figura 1.1), usato come rappresentazione compatta del testo. Sfruttando questa rappresentazione numerica, il modello svolge poi dei calcoli algebrici.

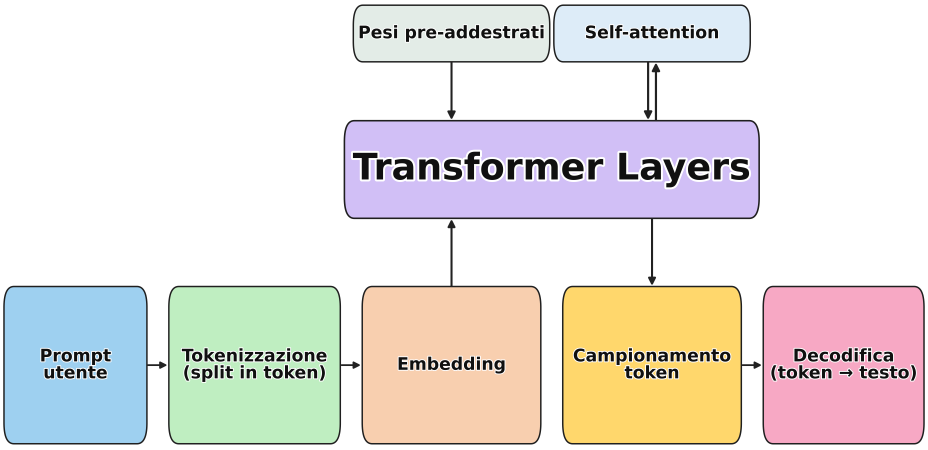


Figura 1.1: Principio di funzionamento degli LLM.

Un'importante caratteristica dei modelli è la quantità di *parametri* che li compongono. Con questo termine si intendono i pesi della rete neurale, che vengono ottimizzati durante la fase di addestramento del modello. Sono indicativi delle capacità espressive di un modello, ma anche del peso in termini di memoria occupata e quindi della velocità di generazione. Solitamente i parametri sono divisi fra quelli che costituiscono il blocco transformer e quelli assegnati alla matrice degli embedding.

Un altro importante indice delle capacità di un LLM è la dimensione massima del *contesto*, che indica la quantità di token che il modello può gestire in una singola richiesta, ciò che è al di fuori non viene considerato nella fase generativa del token successivo. Una finestra di contesto di dimensioni elevate consente di effettuare ragionamenti basati su più dati e potenzialmente più articolati, mentre una finestra ridotta consente di ridurre il costo computazionale dell'esecuzione del modello.

Per valutare le prestazioni di un language model (LM) possono essere utilizzate varie metriche, fra cui:

- *Efficienza*, per cui si analizza, ad esempio, la velocità di generazione, che viene solitamente espressa in token/s, oppure il consumo energetico.

- *Qualità della risposta*, ottenuta mediante benchmark pubblici, che forniscono un set di prove tali da ottenere informazioni specifiche sul modello.

1.2 Gemma 3 270M

Il modello *Gemma 3 270M* è un SLM open source, quindi facilmente accessibile, che è stato sviluppato da Google con l'obiettivo di essere efficiente e capace di essere eseguito anche in locale su macchine con risorse di calcolo limitate. Può essere utilizzato per task semplici, quali generazione di testo e conversazioni. È stato addestrato su circa sei trilioni di token [2].

L'indicazione 270M indica il numero di parametri che costituiscono la struttura del modello, dove M si riferisce a milioni. A differenza dei modelli di grandi dimensioni, che hanno spesso decine di miliardi di parametri, questo ordine di grandezza lo rende utilizzabile anche su dispositivi con hardware limitato. Inoltre, date le dimensioni contenute, è adatto per essere ulteriormente addestrato con procedure di fine-tuning, così da aumentarne la specificità senza eccedere con la memoria occupata.

Tra i 270 milioni di parametri che costituiscono l'architettura del modello, 100 milioni sono allocati nei blocchi Transformer, mentre 170 milioni sono dedicati alla matrice di embedding [2]. Questa ripartizione è influenzata anche dall'ampiezza del vocabolario, pari a 256k token [2], che consente di gestire anche token più complessi. La differenza principale rispetto ad altri modelli di dimensioni simili è la distribuzione dei parametri all'interno della rete neurale.
Il modello ha una finestra di contesto pari a 32k token, un valore grande considerando le dimensioni ridotte del modello e più che sufficiente per i compiti a cui è destinato [2].

1.2.1 Quantizzazione

Con il termine *quantizzazione* si intende il processo di riduzione della precisione numerica dei parametri del modello. Il modello originale di riferimento utilizza numeri in virgola mobile a 16 bit per codificare i pesi, come si può notare nella tabella 1.1 con la sigla F16. Nel caso di quantizzazioni si possono ridurre il numero di bit con cui si codificano i pesi di un nodo, oppure si adatta la precisione dei parametri, passando da tipo float a rappresentazioni intere. Questa riduzione di dimensione consente di rendere il modello più leggero e meno esigente in termini di RAM occupata, a scapito della qualità delle risposte.

In questo elaborato si sono considerate diverse quantizzazioni di Gemma 3 270M, confrontate con il modello F16 su un dispositivo *Raspberry Pi 2 Model B*. Tra quelli disponibili su Hugging Face [3], si sono utilizzati i modelli indicati nella tabella 1.1. I modelli sono tutti in formato *.gguf*, compatibili per funzionare efficientemente su CPU poco potenti tramite strumenti come *llama.cpp*, che verrà trattato nel prossimo capitolo.

Modello	**Bit**	**Grado di Compressione**	**Schema di Quantizzazione**
`gemma-3-270m-it-UD-IQ2_M`	2	M	IQ (UD)
`gemma-3-270m-it-Q3_K_M`	3	M	K
`gemma-3-270m-it-Q4_K_M`	4	M	K
`gemma-3-270m-it-UD-Q6_K_XL`	6	XL	K (UD)
`gemma-3-270m-it-UD-Q8_K_XL`	8	XL	K (UD)
`gemma-3-270m-it-F16`	16	—	Float

Tabella 1.1: Modelli utilizzati con bit per peso, grado di compressione e schema di quantizzazione.

Come si può notare dal nome dei modelli, sono presenti diverse sigle che ne indicano le caratteristiche:

it: indica che il modello è *instruction tuned*, ovvero che è stato sottoposto ad un ulteriore addestramento su coppie istruzione-risposta, addestrandolo su istruzioni specifiche;

Qx: Q suggerisce che è un modello quantizzato, mentre x si riferisce alla famiglia di quantizzazione e indica il numero di bit medi per peso, quindi più basso è il numero più compresso è il modello;

K e IQ: precisano i metodi con cui si effettuano le quantizzazioni. Il primo raggruppa i parametri in blocchi che vengono successivamente compressi, mentre il secondo quantizza soltanto i parametri meno importanti, lasciando inalterati quelli più critici;

UD: evidenzia che ogni layer di parametri è stato quantizzato diversamente, cosí da massimizzare la qualità a parità di dimensione;

M e XL: indicano il grado di compressione del modello all'interno della stessa famiglia di quantizzazione.

Capitolo 2

Ambiente sperimentale: Raspberry Pi 2 Model B

Per lo svolgimento di questo elaborato è stato utilizzato, come piattaforma di edge computing, il Raspberry Pi 2 Model B (figura 2.1). Si tratta di un *computer in miniatura* a basso costo e con consumi ridotti che, nonostante l'hardware limitato, riesce a gestire una discreta mole di dati in locale.

Figura 2.1: Raspberry Pi 2 Model B.

2.1 Caratteristiche Hardware

Il Raspberry Pi non offre una grande potenza di calcolo. La scheda integra 1 GB di memoria RAM [4], che limita notevolmente la possibilitá di gestione di modelli di dimensioni superiori a quello selezionato. Ad esempio, la gestione della versione F16 del modello, occupa circa 700MB di RAM, lasciando un margine estremamente ridotto per sistema operativo ed altri processi.

Per quanto riguarda il processore integrato sulla scheda, si tratta di un SoC Broadcom BCM2836 con CPU quad-core ARM Cortex-A7 a 900MHz [4], decisamente poco prestante, anche per un modello piccolo come quello in esame.

La scheda, inoltre, presenta diverse interfacce di input e output. Di seguito vengono brevemente riportate quelle utilizzate in questo elaborato:

Porta ethernet: impiegata per la connessione di rete, necessaria per installazione e aggiornamento di pacchetti, e per l'accesso da remoto da pc tramite SSH;

Quattro porte USB 2.0: usate per il collegamento di periferiche come tastiera e mouse;

Porta micro-USB a 5 V: adibita all'alimentazione della scheda;

Porta HDMI: usata sia per il collegamento al monitor sia per l'uscita audio;

Slot microSD: supporto per la scheda microSD, su cui sono stati memorizzati i dati e il sistema operativo.

2.2 Procedure di Installazione

Per ridurre l'uso di risorse, si è installato su microSD il sistema operativo *Raspberry Pi OS Lite*, un derivato di Debian, distribuzione di Linux. Il sistema operativo è stato installato tramite l'applicazione Raspberry Pi Imager. Il termine *Lite* indica che la versione è minimale e senza interfaccia grafica, offrendo di fatto soltanto la linea di comando.

Dopo il primo avvio, sono stati installati, attraverso il prompt dei comandi, diversi pacchetti necessari al funzionamento della piattaforma e dei vari

modelli utilizzati. La principale installazione riguarda **llama.cpp**, che è stato clonato con *git* su Raspberry Pi e compilato tramite *CMake*. Llama.cpp è una libreria open-source che consente l'inferenza in locale di LLM, anche su dispositivi non specializzati. È stato scelto perchè leggero, quindi facilmente utilizzabile anche sull'hardware a disposizione, e privo di dipendenze esterne. Inoltre, è stato usato poichè permette di supportare i modelli in formato *.gguf*.

Llama-cli è l'eseguibile contenuto in llama.cpp, che consente di caricare il modello e generare del testo sfruttando solamente la CPU. Permette il completo controllo dei parametri di inferenza di un modello. All'avvio è possibile scegliere diverse caratteristiche tali da variare la risposta del modello, ad esempio modificando la lunghezza massima della risposta (con il parametro --n-predict), oppure il numero di thread della CPU utilizzate per l'operazione (ad esempio -t 4). Inoltre, llama-cli riporta automaticamente diverse sezioni di token/s al termine della conversazione, relative alle diverse fasi del dialogo (ad esempio generazione, valutazione, ecc.).

Per conversare con il modello, è possibile utilizzare il seguente comando da dentro la cartella llama.cpp, facendo riferimento alla directory di llama-cli come segue:

```
./build/bin/llama-cli \
  -m models/gemma3/gemma-3-270m-it-F16.gguf \
  --interactive-first
```

Con *-m* si indica il percorso del modello da caricare, mentre *--interactive-first* indica la modalitá interattiva per la chat. Se i parametri non sono specificati assumono un valore di default determinato da llama-cli.

2.2.1 Sintesi Vocale On-Device (TTS)

Per illustrare un esempio di utilizzo di questo sistema, si è implementata una conversione Text-to-Speech (TTS) on-device, installando i programmi necessari su Raspberry Pi. Data la risposta generata dal SLM, si procede sintetizzando un file audio mono di tipo *.wav* tramite *pico2wave*, che è il front-end del motore di sintesi vocale *SVOX Pico*. Successivamente, tramite lo strumento software *SoX*, si converte localmente il file audio mono in un file audio stereo a 48kHz, così che possa essere riprodotto su impianti HDMI tramite il comando *aplay* (sottosistema ALSA). Queste azioni sono state gestite tramite un codice Python. Il processo sequenziale appena descritto è mostrato nella figura 2.2.

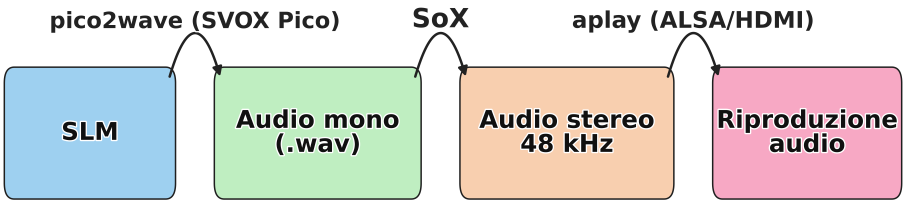


Figura 2.2: Pipeline TTS.

Capitolo 3

Numerical Results

In questo capitolo si analizzano i dati ottenuti dalle prove sperimentali. In particolare si confrontano: *i)* la velocità di risposta della scheda Raspberry Pi 2 Model B per i diversi modelli presi in considerazione; *ii)* l'accuratezza delle risposte generate, misurata utilizzando diversi benchmark.

3.1 Analisi Accuratezza

Per valutare in maniera completa i modelli presi in considerazione, sono stati scelti alcuni benchmark dalla letteratura in grado di valutare diversi aspetti del modello.

Per automatizzare l'analisi nel caso di benchmark generativi, si è realizzato uno script in Bash su Raspberry Pi. Il codice, dati in ingresso i prompt del benchmark considerato, genera un file di testo in cui per ciascuna riga c'è soltanto la risposta generata. Ad ogni riga viene avviata una nuova conversazione, così da evitare conflitti del contesto. Per eseguire i diversi benchmark selezionati, si è scelto di utilizzare un approccio greedy. Con questo termine si indica un metodo deterministico per cui la scelta del token da generare si basa sull'argmax, ovvero la scelta del token successivo ricade su quello che ha probabilità maggiore. Per questa modalità, i parametri con cui si eseguono i modelli sono:

```
--temp 0 --top-p 1 --top-k 0 --repeat-penalty 1
```

dove:

--temp: determina la creatività del modello, in questo caso niente sampling, ma puramente deterministico;

--top-p: tiene conto soltanto dell'insieme di token la cui somma di probabilità è maggiore di p, tagliando di fatto gli errori. È disattivata in quanto impostata a 1;

--top-k: considera solo i k token più probabili, a 0 è disattivato;

--repeat-penalty: penalizza i token recentemente usati, a 1 è disattivato.

La motivazione dietro queste impostazioni è quella di garantire la massima ripetibilità dei risultati, annullando ogni tipo di fenomeno stocastico. Inoltre, è una buona prassi per eseguire i confronti fra le diverse quantizzazioni.

3.1.1 MMLU

Il benchmark MMLU indaga sulle conoscenze generali del modello. Gli argomenti su cui viene valutato riguardano diverse discipline a diversi livelli di difficoltà (fisica, algebra, materie umanistiche, ecc.). Le domande poste ai modelli sono a risposta multipla, per cui ciascun prompt in ingresso al modello fornisce quattro alternative tra cui scegliere (A, B, C, D). Sono state utilizzate un totale di circa 14k domande fra quelle disponibili [5]. La valutazione è stata eseguita confrontando le risposte corrette fornite dal benchmark con quelle ottenute dal modello. Per la generazione è stato impostato il parametro --n-predict a 32 e approccio greedy.

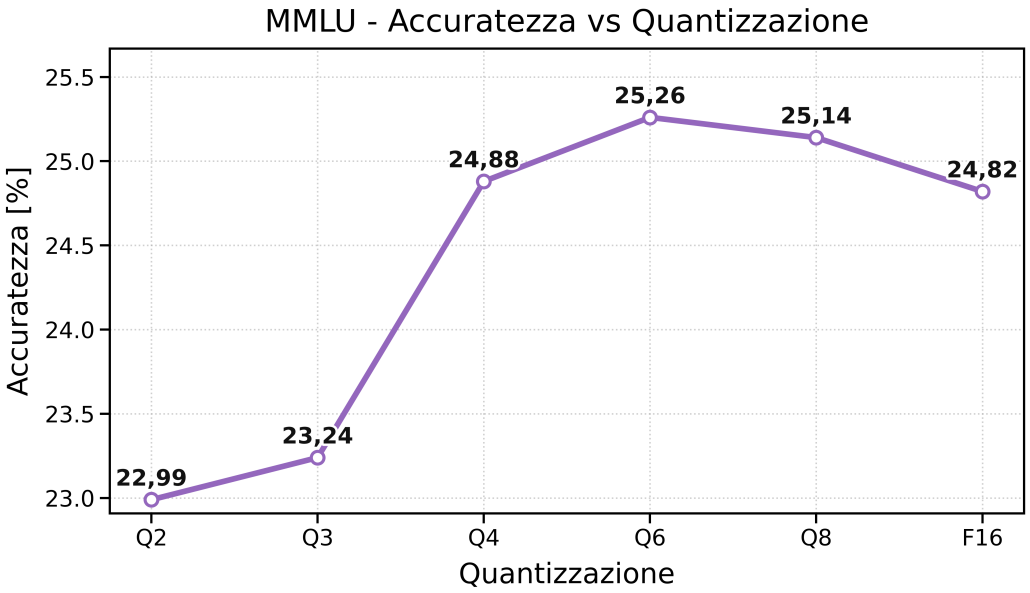


Figura 3.1: Accuratezza MMLU al variare del livello di quantizzazione (indicata con Qx, dove x è il numero di bit per peso). La compressione del modello diminuisce verso destra, fino al modello originale F16.

I risultati ottenuti su questo benchmark sono riportati nel grafico in figura 3.1. Come si può notare, ci sono differenze modeste fra i modelli da Q4 a F16. Precisamente, l'accuratezza varia in un intervallo di 0.44%. Questo andamento è probabilmente legato al modo in cui vengono effettuate le quantizzazioni. Infatti, le tecniche oggi in uso tendono, in generale, a preservare l'accuratezza del modello originale. In particolare, nel caso di SLM e di benchmark non particolarmente complessi, non si evidenzia una perdita di capacità notevole per quantizzazioni non aggressive. Le variazioni che si possono notare sono attribuibili al rumore stocastico del test, ossia alle fluttuazioni campionarie dovute alla dimensione finita del set di prova e ai casi di confine in cui due token hanno probabilità simili. Può accadere, inoltre, che la versione quantizzata del modello tenda ad essere meno verbosa di quello originale. Questo può causare un picco di accuratezza per i modelli quantizzati, tale da eguagliare o superare di poco F16. Si ottiene, infatti, un massimo per la quantizzazione Q6 pari a 25.26%. La differenza è però minima, rientrando quindi totalmente nell'effetto del rumore.

Diversamente da quanto descritto finora, osservando l'accuratezza dei modelli Q2 e Q3 si può notare un calo più significativo. Si passa, infatti, da 24.88% per la quantizzazione Q4, a un valore di 23.24% per Q3, fino a 22.99%

per Q2. Questo accade perché la quantizzazione tende a degradare maggiormente la precisione del modello al fine di ottenere livelli di compressione più elevati.

3.1.2 TruthfulQA-Gen

Il benchmark TruthfulQA-Gen è una variante generativa di TruthfulQA, che analizza la correttezza delle risposte generate dai modelli. In particolare, esprime la somiglianza con testi di riferimento forniti dal benchmark stesso. La comparazione viene eseguita sfruttando due metriche lessicali, BLEU e ROUGE, che permettono di valutare la qualità generativa. In particolare, le risposte vengono valutate tramite i seguenti indici:

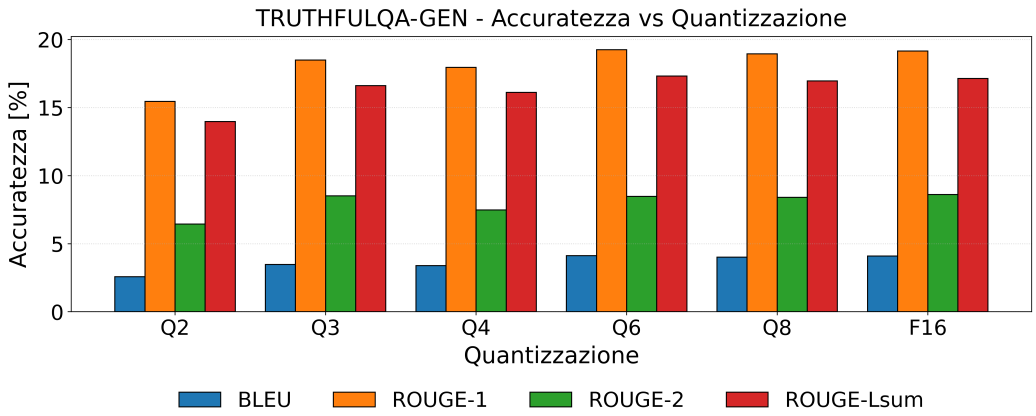
BLEU: misura la vicinanza lessicale al riferimento, basandosi su un piccolo insieme di parole, e valuta anche la lunghezza della risposta;

ROUGE - 1: esprime la percentuale di unigrammi (singole parole) dell'output che corrispondono a quelle presenti nella risposta di riferimento;

ROUGE - 2: simile a ROUGE-1, ma relativo a bigrammi (coppie di parole contigue), quindi più severo perché necessita anche dell'ordine corretto;

ROUGE - Lsum: indica l'aderenza con la risposta di riferimento sulla sequenza più lunga.

Per l'analisi sono state utilizzate 817 domande appartenenti a categorie diverse [6]. Anche in questo caso è stato utilizzato un approccio greedy con --n-predict fissato a 64.



	Q2	Q3	Q4	Q6	Q8	F16
BLEU	2.58	3.48	3.39	4.12	4.01	4.10
ROUGE-1	15.45	18.50	17.95	19.25	18.95	19.16
ROUGE-2	6.44	8.52	7.49	8.47	8.41	8.61
ROUGE-Lsum	13.97	16.61	16.12	17.31	16.96	17.14

Figura 3.2: Accuratezza TruthfulQA-Gen al variare del livello di quantizzazione (indicata con Qx, dove x è il numero di bit per peso). La compressione del modello diminuisce verso destra, fino al modello originale F16.

Come si può notare dal grafico in figura 3.2, anche per questo benchmark si possono ripetere le stesse considerazioni fatte per MMLU. I punti percentuali per cui differiscono le diverse quantizzazioni sono contenuti fino alla Q3. Per la Q2 si evidenzia, invece, un divario più marcato. Considerando ad esempio la metrica ROUGE-1, fino alla quantizzazione Q3 i valori passano da un minimo di 17.95% (per Q4) a un massimo di 19.25% (per Q6). Per Q2 si ha, invece, un calo importante di accuratezza fino a 15.45%. In sintesi, si può affermare che le quantizzazioni non estreme preservano buona parte della qualità generativa del modello.

3.1.3 IFEval

Il benchmark IFEval valuta quanto un modello rispetta in maniera precisa le istruzioni fornite (ad esempio maiuscole, punteggiatura, formato, ecc.). I vincoli vengono posti da un insieme di prompt standardizzati. Il punteggio è calcolato con uno script ufficiale contenuto nella repository [7], che valuta quanto il modello abbia rispettato i vincoli, indipendentemente dalla correttezza della risposta. Le percentuali che si ottengono si distinguono in:

prompt-level accuracy: indica la percentuale di prompt in cui le risposte hanno soddisfatto tutti i vincoli;

instruction-level accuracy: esprime la percentuale complessiva di vincoli soddisfatti.

Ciascuna di queste metriche si divide in *strict* (verifica rigida) e *loose* (verifica più tollerante). I valori riportati successivamente sono ottenuti da una media di questi due tipi di metriche.

Per questo benchmark si è scelto di analizzare i modelli anche con un approccio *sampling*, per effettuare un confronto con l'approccio greedy. In entrambi i casi si è impostato --n-predict a 1024, legato alla necessità di alcuni prompt di creare risposte più lunghe. L'approccio di sampling usato corrisponde alla configurazione ideale per l'inferenza (seguendo la documentazione di Gemma [8]). Il settaggio dei parametri utilizzato è il seguente:

```
--temp 1.0, --top-k 64, --top-p 0.95, --min-p 0.0
--repeat-penalty 1.0 --seed 123
```

dove:

--min-p: considera soltanto i token con probabilità >p;

--seed: imposta il seme del generatore casuale, rendendo le scelte ripetibili anche in questo caso di sampling.

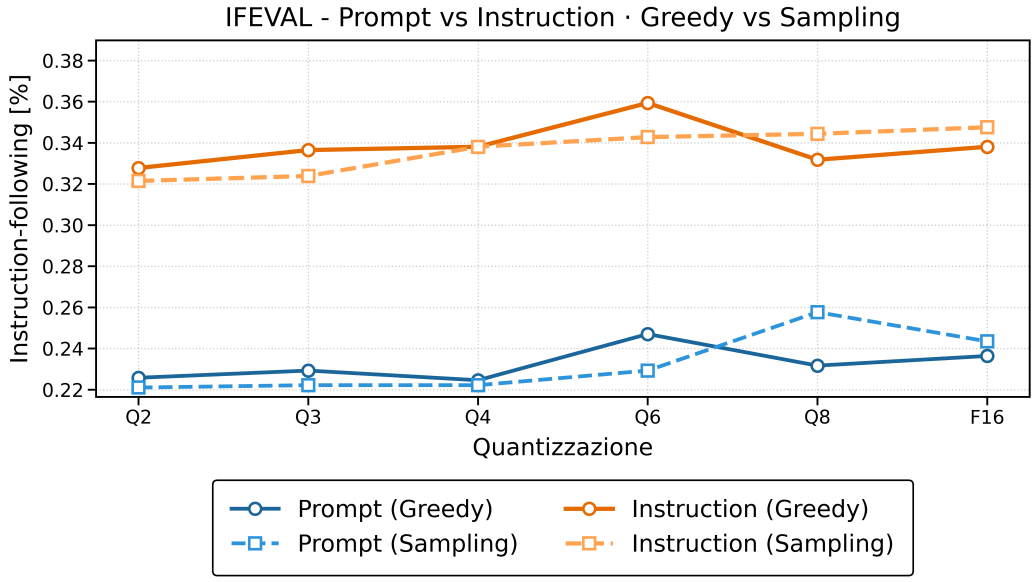


Figura 3.3: Instruction-following IFEval al variare del livello di quantizzazione (indicata con Qx, dove x è il numero di bit per peso) nel caso greedy e sampling. La compressione del modello diminuisce verso destra, fino al modello originale F16.

Quello che si nota dal grafico in figura 3.3 è che non ci sono differenze evidenti fra un approccio e l'altro. L'approccio sampling è più stocastico, quindi può migliorare su qualche vincolo e peggiorare su altri. Si può quindi dedurre che il rispetto delle istruzioni fornite non sembra dipendere dalla creatività del modello. Per quanto riguarda invece l'andamento dei valori per instruction-following, il comportamento è paragonabile a quanto descritto per i benchmark MMLU e TruthfulQA-Gen. Considerando l'approccio greedy e l'accuratezza a livello di instruction, infatti, si presenta sempre un massimo per la quantizzazione Q6 (pari a 36%), mentre si ha un minimo per Q2 (pari a 32.8%). Più in generale, le differenze tra i valori ottenuti per le diverse quantizzazioni, tralasciando il caso di Q6, non sono estremamente marcate. Questo indica che la quantizzazione non degrada eccessivamente la capacità del modello di instruction-following.

3.1.4 Perplexity

Il benchmark Perplexity valuta quanto il modello considera improbabile una parola in una sequenza di testo naturale. Se il valore ottenuto è alto indica che il modello trova inverosimile quella sequenza di token. La metrica viene calcolata con l'utility *llama-perplexity*, presente nelle versioni recenti di llama.cpp.

In questo caso non viene utilizzato l'approccio greedy, in quanto non c'è nessuna generazione da parte del modello. Infatti, il modello esegue un'analisi sul testo posto in input, denominato testset.txt. Il testo su cui viene eseguita la misurazione è di circa 40k token, ed è stato estratto dal sito Wikipedia relativo all'Italia [9].

Per avviare la misurazione viene utilizzato il seguente comando, variando ogni volta il modello considerato:

```
./bin/llama-perplexity
-m ../models/gemma3/gemma-3-270m-it-F16.gguf
-f ../testset.txt  -t 4  -c 256   -b 16   -ub 4
> ppl.txt 2> ppl.err
```

I parametri utilizzati non descritti precedentemente sono:

-f: file di test;

-c: dimensione del contesto, anche detto chunk;

-b: dimensione del batch logico, ovvero quanti token/chunk vengono eseguiti in parallelo;

-ub: suddivide il batch in sottoblocchi.

Al termine dell'esecuzione, vengono creati due file con i risultati ottenuti: ppl.txt e ppl.err. Per effettuare l'analisi, il file testset.txt viene suddiviso in token, per poi essere raggruppati in chunk contigui di lunghezza pari a 256 token (come indicato dal parametro -c). Per ogni chunk viene calcolata una stima del valore di perplexity parziale, che viene riportato all'interno del file ppl.txt. Nel file ppl.err, invece, viene riportata la perplexity media complessiva, indicata con l'intervallo "media ± errore". L'incertezza della misura, è generata dalla stima effettuata sui vari chunk. All'interno di ppl.err è riportato anche il valore medio di token/s valutati.

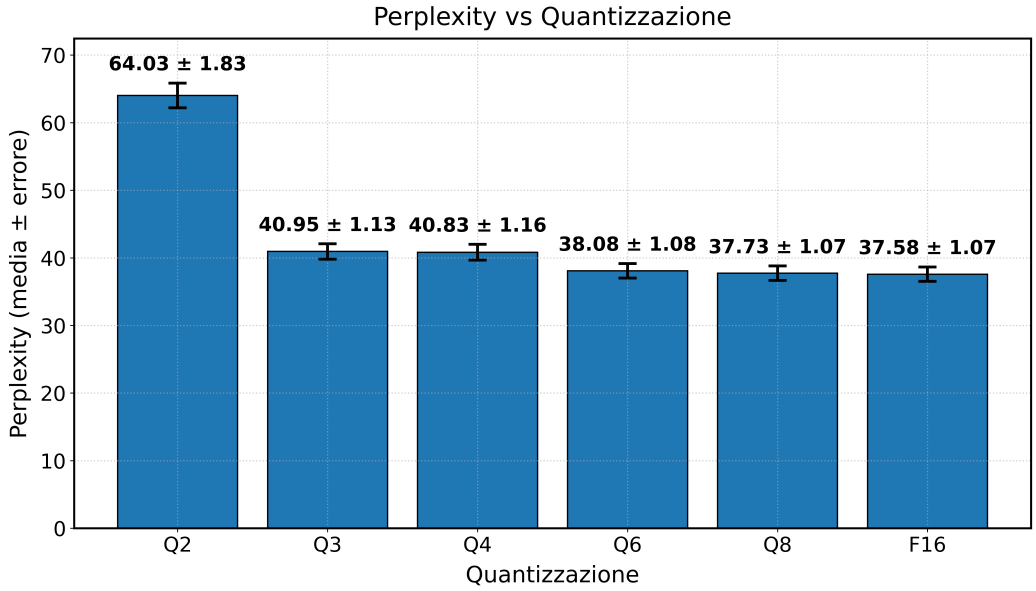


Figura 3.4: Perplexity al variare del livello di quantizzazione (indicata con Qx, dove x è il numero di bit per peso). La compressione del modello diminuisce verso destra, fino al modello originale F16.

Come si può notare dal grafico in figura 3.4, il valore di perplexity è minimo nel caso di modello non quantizzato ed è pari a 37.58 ± 1.07. Nel caso di Q8/Q6 si hanno dei valori simili al modello con massima precisione. Questo evidenzia ancora una volta la capacità di preservare la qualità della risposta nonostante la quantizzazione del modello. Si nota invece una prima differenza a partire dal modello Q4, per cui si passa a 40.83 ± 1.16, a salire fino a 64.03 ± 1.83 per il caso Q2, che è un valore decisamente alto. Si può quindi concludere che le quantizzazioni degradano maggiormente le prestazioni del modello su questo benchmark.

3.2 Analisi delle prestazioni

Come precedentemente indicato, l'eseguibile llama-cli restituisce tra i diversi valori di token/s, quello relativo alla fase generativa della risposta. Il calcolo viene svolto automaticamente, memorizzando il numero totale di token generati e il tempo di esecuzione totale, per poi svolgere la divisione. Questo calcolo viene esplicitato nella sezione *eval time* al termine della conversazione con il modello. I dati sono stati ottenuti aggiungendo una porzione di script allo stesso codice utilizzato per ottenere le risposte sui benchmark. La sua funzione è quella di salvare in ciascuna riga di un file .txt il valore di token/s generati dal Raspberry Pi per singolo prompt.

Facendo una media dei token/s generati da un modello per ciascun benchmark si è ottengono i grafici nelle figure 3.5 e 3.6.

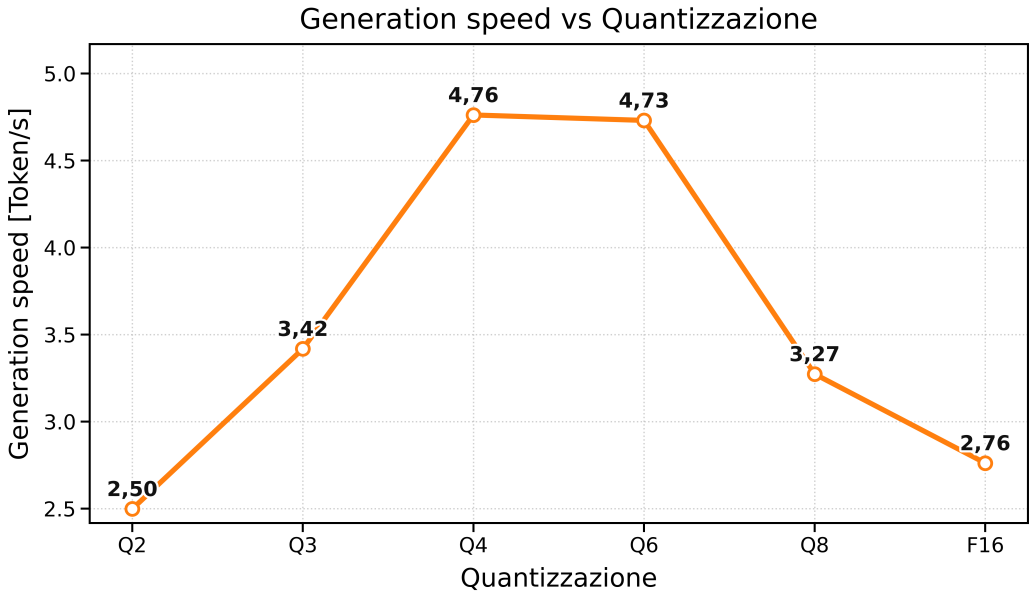


Figura 3.5: Generative speed al variare del livello di quantizzazione (indicato con Qx, dove x è il numero di bit per peso). La compressione del modello diminuisce verso destra, fino al modello originale F16.

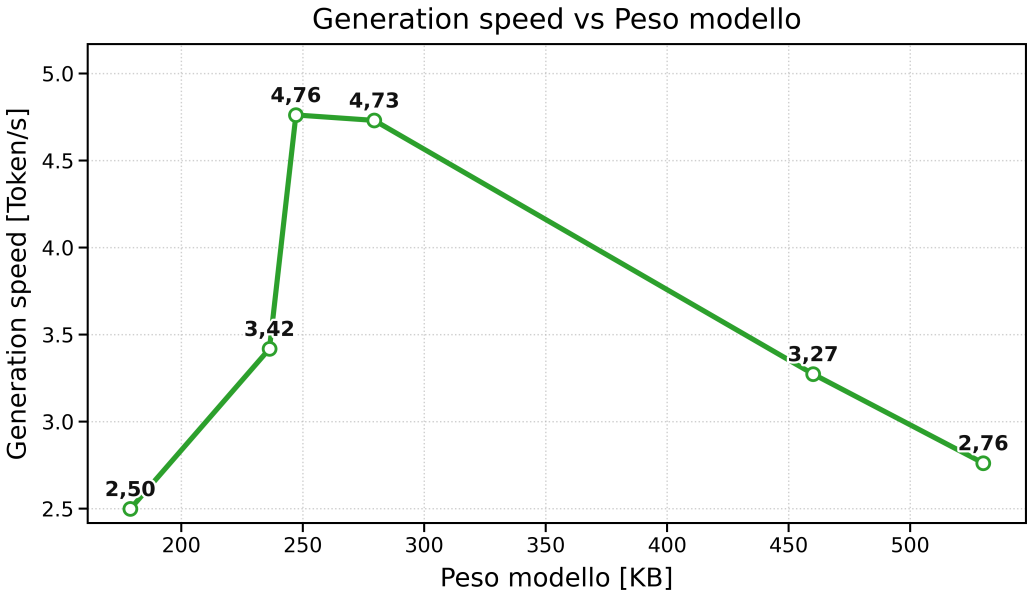


Figura 3.6: Generative speed al variare della dimensione del modello.

Da un'analisi superficiale si poteva pensare che le prestazioni del Raspberry Pi aumentassero con la compressione del modello in modo monotono. Invece, come si può notare dal grafico 3.5, il modello quantizzato Q4 ha un valore di *generative speed* maggiore del Q3 e del Q2. Questo risultato sperimentale trova spiegazione nella modalità di gestione dei modelli compressi, in particolare nell'overhead di de-quantizzazione. Infatti, un LLM quantizzato quando svolge l'inferenza, de-quantizza i pesi momentaneamente, spacchettandoli e riscalandoli da interi a virgola mobile, per eseguire i calcoli algebrici. In questo caso, essendo stato sottoposto a una quantizzazione spinta, il modello necessita di più operazioni per completare la de-quantizzazione, rallentando la generazione della risposta. La rapidità nel calcolo dipende anche dalle caratteristiche dell'hardware, infatti eseguendo i calcoli soltanto su CPU, è necessario molto tempo per completare la de-quantizzazione, penalizzando la velocità di generazione.

Osservando il grafico in figura 3.6, si può dedurre che in questo ambiente di lavoro, le quantizzazioni Q4 e Q6 sono il giusto compromesso fra quantizzazione e dimensione del modello. Infatti, utilizzando le quantizzazioni Q4 e Q6 si generano in media rispettivamente 4.76 e 4.73 token/s, contro i 3.42 token/s ottenibili usando Q3.

Il comportamento nel caso generativo si ripete anche nel caso valutativo. Come descritto nel paragrafo 3.1.4, al momento del calcolo del valore di perplexity vengono misurati anche i token valutati ogni secondo. Dai valori così ottenuti si giunge al grafico 3.7 riportato di seguito.

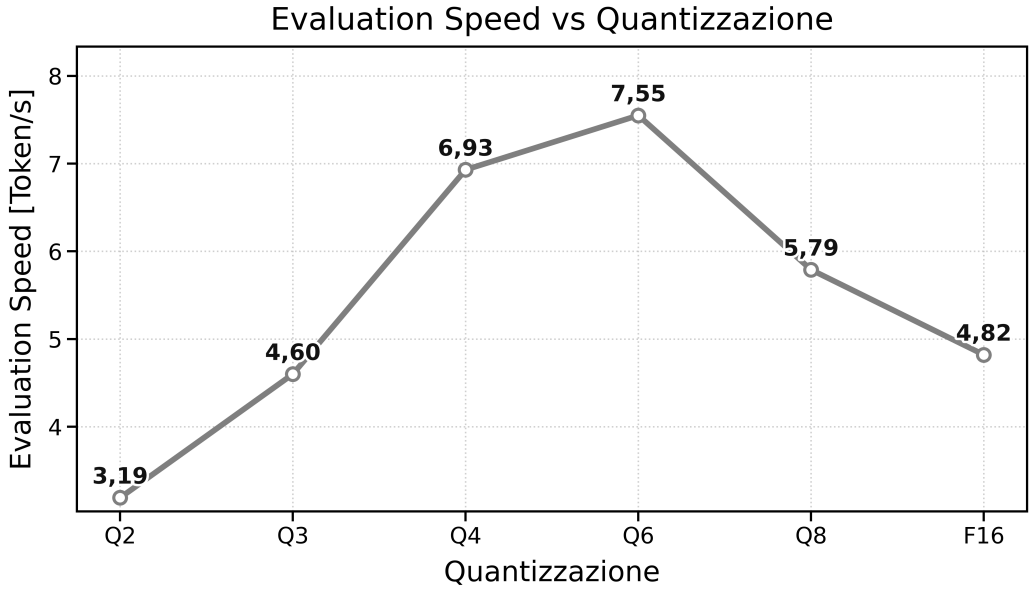


Figura 3.7: Evaluation speed valutata al variare della quantizzazione (indicata con Qx, dove x è il numero di bit per peso). La compressione del modello diminuisce verso destra, fino al modello originale F16.

Dal grafico 3.7 si evince quindi che l'andamento che si presenta in questo caso è simile a quello ottenuto al caso generativo, in quanto Q4 e Q6 sono le scelte più efficaci anche sotto questo aspetto. In particolare il modello quantizzato Q4 è in grado di valutare mediamente 6.93 token/s, decisamente elevato in confronto al modello Q3 che ha una velocità di valutazione pari soltanto a 4.60 token/s.

Capitolo 4

Conclusioni

Con questo elaborato è stata dimostrata la fattibilità dell'esecuzione on-device di un modello SLM, in particolare il caso di Gemma 3 270M su Raspberry Pi 2 Model B. La praticabilità applicativa in scenari di edge computing è stata dimostrata attraverso un sintetizzatore vocale TTS locale funzionante.

Sono state confrontate diverse quantizzazioni dello stesso modello, valutandole su benchmark eterogenei. Basandosi sui risultati ottenuti, si può affermare che le quantizzazioni Q4 e Q6 rappresentano il miglior compromesso tra accuratezza e prestazioni. In particolare, Q6 mostra un'accuratezza lievemente superiore a Q4 sui benchmark considerati (ad esempio MMLU: 25.26% vs 24.88%). In generazione, le velocità medie sono sostanzialmente allineate (4.76 token/s per Q4 vs 4.73 token/s per Q6), mentre in valutazione Q6 risulta leggermente più rapido (7.55 token/s vs 6.93 token/s). La scelta operativa dipende quindi dall'ambito d'uso: Q6 è preferibile quando si privilegia la qualità (e una valutazione più veloce); Q4 resta indicato quando il vincolo principale è la memoria occupata a fronte di prestazioni molto vicine. Le differenze rilevate per l'accuratezza sono comunque contenute, quindi sono attribuibili al rumore statistico.

L'analisi può essere estesa ad altri modelli, così da identificare quello più adatto all'hardware considerato. Ulteriori sviluppi possono includere l'impiego di benchmark differenti. Infine, può essere interessante valutare il comportamento di Gemma 3 270M dopo un fine-tuning mirato, evidenziando le principali differenze che insorgono dopo questo tipo di operazione.

Elenco delle figure

Bibliografia

[1] The Rise of Small Language Models

https://objectbox.io/the-rise-of-small-language-models/

[2] Introducing Gemma 3 270M

https://developers.googleblog.com/en/introducing-gemma-3-270m/

[3] Hugging Face Gemma 3 270M

https://huggingface.co/unsloth/gemma-3-270m-it-GGUF

[4] Specifiche Raspberry Pi 2 Model B

https://huggingface.co/unsloth/gemma-3-270m-it-GGUF

[5] Benchmark MMLU

https://huggingface.co/datasets/cais/mmlu

[6] Benchmark TruthfulQA-Gen

https://huggingface.co/datasets/domenicrosati/TruthfulQA

[7] Repository IFEval

https://github.com/oKatanaaa/ifeval

[8] Parametri Sampling

https://docs.unsloth.ai/models/gemma-3-how-to-run-and-fine-tune

[9] Pagina Wikipedia Italia

https://it.wikipedia.org/wiki/Italia