Alma Mater Studiorum · Università di Bologna

Dipartimento di Fisica e Astronomia "Augusto Righi"
Corso di Laurea in Fisica

ResQNN: A Proof-of-Concept Quantum Generative Model for Fast Calorimeter Simulation of the ATLAS Detector

Relatore:
Prof. Matteo Franchini

Correlatore:
Dott. Emilio Apicella

Presentata da:
Francesco Coluccia

Anno Accademico 2025/2026

Ora affronterai il mare delle tenebre,
e ciò che in esso vi è di esplorabile.

— *. . . E tu vivrai nel terrore! L'aldilà* (Lucio Fulci, 1981)

Sommario

Negli esperimenti di fisica delle alte energie le simulazioni Monte Carlo sono indispensabili per confrontare i dati con le previsioni teoriche. In ATLAS la simulazione del rivelatore assorbe circa il 40% delle risorse computazionali e la parte più onerosa sono i calorimetri. Con High-Luminosity LHC la richiesta di eventi simulati crescerà più rapidamente delle risorse di calcolo. Per far fronte a questa pressione computazionale ATLAS utilizza AtlFast3, uno strumento di simulazione veloce dei calorimetri che affianca Geant4.
Parallelamente, si è iniziato a esplorare approcci ibridi classico-quantistici per capire se, quando le tecnologie quantistiche saranno più mature, sarà possibile sfruttarle per le simulazioni.
In questo contesto viene presentato ResQNN, un modello generativo ibrido per la simulazione veloce dei calorimetri basato sul Quantum Reservoir Computing. ResQNN divide la generazione in due stadi: un modello delle energie, che genera l'energia depositata in ciascun layer, e un modello delle forme, che la distribuisce fra i voxel del layer. Il modello, sviluppato su un dataset open source di pioni carichi (CaloChallenge), è stato ampliato in questa tesi con nuove funzionalità.
I test effettuati riguardano il modello delle energie. Tre funzioni costo sono state studiate: il caso dove si è aggiunta una funzione score basata sull'energia è stato il migliore. Inserire il rumore latente dopo il reservoir riduce molto il costo computazionale senza peggiorare la qualità degli eventi. Né più campioni nella funzione costo né un decoder più grande sembrano invece risolvere i limiti nella ripartizione dell'energia fra i layer e, rispetto a un gemello puramente classico, il reservoir quantistico non mostra un vantaggio significativo. I risultati infine suggeriscono quali parti del modello migliorare in futuro e quali test effettuare per capire se, e in quali condizioni, un reservoir quantistico possa contribuire alla simulazione dei calorimetri.

Indice

Introduzione **5**

1 Machine learning generativo classico nella fisica delle alte energie **7**
1.1 Machine Learning e Deep Learning 7
1.2 Algoritmi per il machine learning generativo 9
1.2.1 Normalizing Flow 9
1.2.2 Autoencoder variazionali 10
1.2.3 Reti generative avversarie 12
1.2.4 Modelli di diffusione 14
1.3 Machine learning generativo per la fisica delle alte energie 15

2 L'esperimento ATLAS e la simulazione veloce in ATLAS **17**
2.1 Il Large Hadron Collider 18
2.2 Il rivelatore ATLAS 21
2.3 Simulazione veloce dei calorimetri di ATLAS: AtlFast3 27
2.3.1 Motivazioni della simulazione veloce del rivelatore 27
2.3.2 Architettura di AtlFast3 27
2.3.3 Valutazione delle prestazioni 28

3 Machine learning generativo quantistico **30**
3.1 Reti neurali quantistiche 30
3.2 Circuiti quantistici variazionali 32
3.3 Modelli generativi quantistici 34
3.3.1 Quantum Circuit Born Machine 35
3.3.2 GAN quantistiche 36
3.3.3 Reti neurali invertibili quantistiche 38
3.4 Reservoir Computing e Quantum Reservoir Computing 39
3.5 Quantum computing per la fisica delle alte energie 42

4 ResQNN: un possibile approccio basato sul QRC per la simulazione veloce dei calorimetri **46**
4.1 Il dataset della CaloChallenge 46
4.2 Struttura generale di ResQNN 48
4.3 L'energy model 52
4.4 Lo shape model 53
4.5 Addestramento di ResQNN 54
4.6 Generazione degli shower con ResQNN 57
4.7 Test e validazione del modello 59
4.7.1 Confronto fra le funzioni costo 60
4.7.2 Confronto fra le configurazioni pre e post del rumore 70
4.7.3 Ottimizzazione dell'energy model 77
4.7.4 Confronto fra modello quantistico e classico 81

Conclusioni **85**

A Riepilogo degli iperparametri **87**

Introduzione

Nella fisica delle alte energie il passo più oneroso delle simulazioni Monte Carlo è, tipicamente, la simulazione dell'interazione fra le particelle e il rivelatore: nell'esperimento ATLAS questa assorbe circa il 40% delle risorse di calcolo complessive, soprattutto per i calorimetri, simulati con Geant4. Con l'avvio di High-Luminosity LHC la richiesta di eventi simulati crescerà più rapidamente delle risorse di calcolo.
Per far fronte a questa pressione computazionale ATLAS utilizza strumenti di simulazione veloce come AtlFast3, da 3 a 15 volte più veloce di Geant4, ed è proprio in questo contesto che i modelli generativi trovano una delle loro applicazioni più promettenti.
Parallelamente si stanno esplorando versioni quantistiche o ibride classico-quantistiche dei modelli generativi, per capire se, quando le tecnologie quantistiche saranno più mature, sarà possibile sfruttarle anche per le simulazioni degli esperimenti.
In questo contesto si inserisce questa tesi. Viene presentato ResQNN, un modello generativo ibrido basato sul Quantum Reservoir Computing: un sistema non addestrato, il reservoir, trasforma i dati in una rappresentazione più ricca, da cui una rete classica può estrarre più facilmente l'informazione necessaria. In ResQNN la generazione è divisa in due stadi: un energy model, che genera le energie depositate nei layer del calorimetro, e uno shape model, che distribuisce tale energia fra i voxel di ciascun layer. Entrambi sono costruiti a partire da blocchi ibridi, formati da un encoder classico, da un reservoir quantistico con parametri fissi e da un decoder classico: l'energy model ne contiene uno solo, lo shape model uno per ciascun layer. La casualità necessaria alla generazione è introdotta da un rumore latente, che può entrare nel blocco prima o dopo il reservoir. Il modello, ampliato in questa tesi con nuove funzionalità, è stato sviluppato su un dataset open source di pioni carichi (CaloChallenge).
Lo studio si è concentrato sull'energy model, attraverso quattro test. Il primo confronta tre diverse funzioni costo e mostra che le funzioni costo regressive, con cui il modello era stato inizialmente sviluppato, portano a ignorare il rumore latente, un problema che non si risolve reintroducendo a posteriori la variabilità mancante tramite un fattore casuale, mentre aggiungere una funzione score basata sull'energia, che premia esplicitamente la diversità fra gli eventi generati con la stessa energia incidente, permette al modello di apprendere la dispersione degli shower. Il secondo confronta due diversi punti in cui il rumore latente può entrare nel blocco ibrido e mostra che inserirlo dopo il reservoir ridu-

ce di circa M volte il costo di ogni passo di addestramento, senza peggiorare la qualità degli eventi generati, e rende molto più economica anche la generazione di più eventi con la stessa energia incidente. Il terzo verifica se un maggior numero di campioni per la valutazione della funzione costo o un decoder più grande risolvano i limiti che restano nella ripartizione dell'energia fra i layer, mentre il quarto confronta il modello con un gemello puramente classico, identico in tutto tranne che per l'assenza del circuito. Come vedremo, l'energia totale depositata è riprodotta bene, mentre la ripartizione dell'energia fra i layer resta il principale problema aperto: nessun modello riproduce i picchi stretti delle distribuzioni delle energie dei primi quattro layer, la dispersione dei singoli layer è sbagliata fino a un fattore 1.6 e i layer non sono mai esattamente vuoti. Infine, quando encoder e decoder hanno molti neuroni il gemello privo del circuito risulta migliore del modello ibrido, mentre riducendo il numero di neuroni le differenze diventano confrontabili con quelle dovute alla casualità dell'addestramento. Inoltre, con il circuito simulato su CPU, l'addestramento del modello quantistico richiede circa 300 volte il tempo di quello classico.

Questi risultati vanno letti entro i limiti dello studio svolto, che riguarda il solo energy model e si basa su pochi addestramenti, in alcuni casi di modelli piccoli addestrati su 6000 eventi, e su un'unica struttura del reservoir, simulata in modo esatto. Fra i possibili sviluppi futuri vi sono quindi uno studio più approfondito degli iperparametri e della funzione costo dell'energy model, la ripetizione dei test con più seed, il confronto con un reservoir classico, la valutazione dello shape model e dello shower completo e, in prospettiva, l'esecuzione del circuito con un numero finito di misure e su un computer quantistico reale.

La tesi è strutturata in quattro capitoli. Nel primo capitolo vengono introdotti i concetti di machine learning e deep learning e vengono descritti i principali modelli generativi classici, cioè normalizing flow, autoencoder variazionali, reti generative avversarie e modelli di diffusione, insieme alle loro applicazioni nella fisica delle alte energie.

Nel secondo capitolo vengono presentati l'acceleratore LHC e il rivelatore ATLAS, con particolare attenzione al sistema calorimetrico, e viene descritto AtlFast3, lo strumento di simulazione veloce dei calorimetri utilizzato da ATLAS.

Il terzo capitolo estende queste idee al calcolo quantistico: dopo aver introdotto le reti neurali quantistiche e i circuiti variazionali, insieme al problema dei barren plateau, vengono descritti i principali modelli generativi che ne fanno uso, il paradigma del reservoir computing nella sua versione classica e quantistica e infine le applicazioni del quantum computing alla fisica delle alte energie.

Nel quarto capitolo viene presentato ResQNN: dopo il dataset della CaloChallenge vengono descritti la struttura del modello, cioè il blocco ibrido, l'energy model e lo shape model, il suo addestramento e la generazione degli shower. Vengono infine presentati e discussi i quattro test effettuati sull'energy model. Seguono le conclusioni, in cui sono riassunti i risultati ottenuti, i limiti dello studio e i possibili sviluppi futuri, e un'appendice in cui sono riportati gli iperparametri del modello.

Capitolo 1

Machine learning generativo classico nella fisica delle alte energie

In questo capitolo inizieremo fornendo alcune definizioni utili a comprendere il contesto nel quale si inseriscono le tecniche di machine learning generativo. Successivamente verrà descritta la struttura dei principali modelli generativi classici ora in uso senza soffermarci su come vengono specificatamente scritti gli algoritmi. Infine vedremo quali sono le possibili applicazioni dei modelli generativi all'interno della fisica delle alte energie e perché oggigiorno sia necessario utilizzare tali tecniche o svilupparne di nuove.

1.1 Machine Learning e Deep Learning

Negli ultimi anni termini come Intelligenza Artificiale (AI), Machine Learning (ML) e Deep Learning (DL) sono diventati sempre più diffusi e non appartengono più esclusivamente al lessico degli specialisti del settore. Tuttavia, la loro crescente diffusione, soprattutto in contesti informali, ha spesso generato confusione, portando a un utilizzo improprio o indistinto di questi concetti. In questa sezione cercheremo quindi di chiarire il significato e le differenze tra questi termini.
Il termine intelligenza artificiale non ha una definizione stringente, parte dalla volontà di creare dei programmi in grado di mimare una generica forma di intelligenza naturale. Nel contesto delle AI classiche si utilizzano ancora algoritmi hard-coded. Un esempio semplice è un classificatore di numeri in pari o dispari.
Il machine learning è invece un sottogruppo delle AI che si occupa di algoritmi che hanno la capacità di acquisire "conoscenza" a partire da pattern, utili a svolgere il proprio compito, all'interno dei dati. In questo caso il programmatore si occupa di creare un ambiente e delle condizioni tali per cui l'algoritmo possa allenarsi da sé per ottimizzare i risultati.
Quando invece bisogna affrontare problemi più complessi e computazionalmente onerosi

si utilizza il deep learning, un sottogruppo del ML, come mostrato in fig. 1.1, in cui vengono utilizzate le reti neurali: un'architettura a strati per elaborare al meglio l'informazione. Le reti neurali sono dette profonde quando contengono diversi strati nascosti.

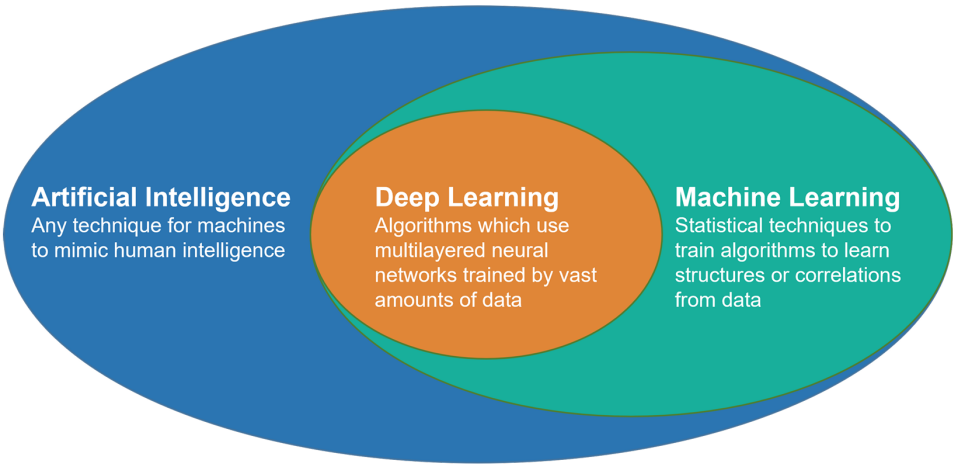


Figura 1.1: *Struttura gerarchica di intelligenza artificiale, machine learning, deep learning.* [1]

Nelle reti neurali i parametri del modello, detti weight, vengono ottimizzati in modo da minimizzare la funzione di perdita fino al raggiungimento di un minimo, auspicabilmente globale. La funzione di perdita, che può assumere forme molto diversificate, indica di quanto il modello stia sbagliando rispetto alla soluzione attesa o ottimale. Durante la fase di addestramento, per minimizzare la funzione di perdita viene utilizzato l'algoritmo di backpropagation: viene calcolato il gradiente della funzione di perdita rispetto ai parametri in ogni strato procedendo a ritroso, cioè si parte dallo strato di output e si arriva a quello di input passando per gli strati intermedi. La backpropagation permette quindi di calcolare i gradienti della funzione di perdita rispetto ai parametri della rete, che vengono successivamente aggiornati mediante un algoritmo di ottimizzazione.
La capacità di una rete neurale, dopo la fase di addestramento, di effettuare delle previsioni su dei dati che non ha mai visto viene detta generalizzazione.
In base al compito che bisogna svolgere, il processo di apprendimento viene distinto in supervisionato e non supervisionato. L'apprendimento supervisionato consiste nell'individuazione, da parte della macchina, di possibili relazioni fra i dati e delle etichette (in inglese label) ad essi associate. Nell'apprendimento non supervisionato invece si cerca di individuare delle strutture o dei pattern nei dati stessi. I modelli generativi rappresentano proprio una classe di tecniche d'apprendimento non supervisionato[1].

[1]Per precisare, in realtà esiste anche il cosiddetto apprendimento semi-supervisionato in cui solo un

1.2 Algoritmi per il machine learning generativo

I modelli generativi apprendono la distribuzione probabilistica di un dataset fornito e generano dei dati in accordo con tale distribuzione. Tra le tecniche più utilizzate abbiamo i Normalizing Flow, gli Autoencoder Variazionali (VAE), le Reti Generative Avversarie (GAN) e, più recentemente proposti, i Diffusion Model. Qui di seguito ne verrà data un'introduzione breve, non esaustiva ed incentrata più sui concetti che sul formalismo matematico su cui sono fondati tali modelli.

1.2.1 Normalizing Flow

I Normalizing Flow sono dei modelli generativi il cui funzionamento si basa sulla cosiddetta change of variables formula [2], in cui una distribuzione $p(\mathbf{x})$ viene sostituita da una distribuzione $\pi(\mathbf{z})$, con $\mathbf{x}, \mathbf{z} \in \mathbb{R}^D$, secondo la relazione

$$p(\mathbf{x}) = \pi(\mathbf{z}) \left|\det J_{f^{-1}}(\mathbf{x})\right| . \tag{1.1}$$

In questa equazione $\mathbf{z} = f^{-1}(\mathbf{x})$, f^{-1} è una funzione invertibile che mappa i punti $\mathbf{x}$ della funzione $p(\mathbf{x})$ nei punti $\mathbf{z}$ della funzione $\pi(\mathbf{z})$ e $J_{f^{-1}}(\mathbf{x})$ è la matrice Jacobiana definita come

$$J_{f^{-1}} = \begin{bmatrix} \frac{\partial f_1^{-1}}{\partial x_1} & \cdots & \frac{\partial f_1^{-1}}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_D^{-1}}{\partial x_1} & \cdots & \frac{\partial f_D^{-1}}{\partial x_D} \end{bmatrix} .$$

Supponiamo ora di lavorare con una sequenza di trasformazioni invertibili f_i. Applichiamo in modo sequenziale le trasformazioni invertibili e, utilizzando il teorema della funzione inversa, otteniamo dalla eq. (1.1)

$$p(\mathbf{x}) = \pi(\mathbf{z}_0 = f^{-1}(\mathbf{x})) \prod_{i=1}^{K} \left|J_{f_i}(\mathbf{z}_{i-1})\right|^{-1} .$$

Un esempio della trasformazione di una distribuzione Gaussiana in una distribuzione multimodale viene mostrato in forma grafica in fig. 1.2. Supponiamo ora di iniziare con una distribuzione normale $\pi(\mathbf{z}_0) = \mathcal{N}(\mathbf{z}_0|0)$, il logaritmo di $p(\mathbf{x})$ sarà

$$\ln p(\mathbf{x}) = \ln \mathcal{N}(\mathbf{z}_0|0) - \sum_{i=1}^{K} \ln \left|J_{f_i}(\mathbf{z}_{i-1})\right|.$$

Il logaritmo della densità $p(\mathbf{x})$ definisce la funzione utilizzata per l'addestramento del modello. In particolare, massimizzare $\ln p(\mathbf{x})$ equivale a massimizzare la verosimiglianza

piccolo quantitativo di dati è etichettato. Alcuni modelli generativi rientrano in questo tipo di schema

dei dati osservati. Il primo termine, $\ln \mathcal{N}(\mathbf{z}_0|0)$, misura quanto la variabile latente $\mathbf{z}_0 = f^{-1}(\mathbf{x})$ sia compatibile con una distribuzione Gaussiana standard. Il secondo termine, che coinvolge il determinante della Jacobiana, tiene conto della variazione di volume indotta dalla trasformazione invertibile ed è necessario per garantire la corretta normalizzazione della distribuzione risultante.

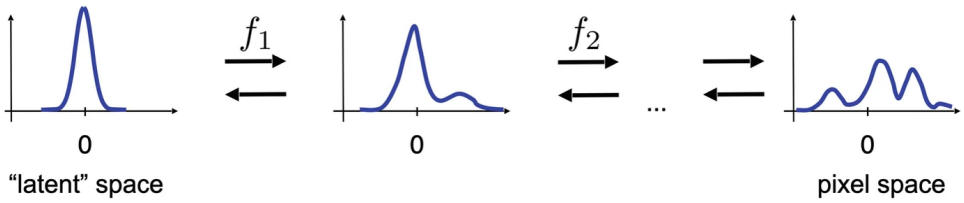


Figura 1.2: *Esempio della trasformazione di una distribuzione Gaussiana (lo spazio latente) in una distribuzione multimodale (lo spazio dei dati, per esempio lo spazio dei pixel di un'immagine)* [2]

Per modellare la trasformazione f è possibile usare delle reti neurali aventi però due proprietà: in primis vogliamo che la trasformazione sia invertibile per cui occorre usare una rete neurale invertibile (INN), inoltre vogliamo anche che il logaritmo della Jacobiana sia relativamente semplice da calcolare [2].
I normalizing flow offrono due modalità operative: il percorso inverso, che viene utilizzato per la stima della densità e l'ottimizzazione della trasformazione, ed il percorso diretto, che effettivamente funziona come modello generativo [3]. Nella direzione inversa, campioni provenienti dalla distribuzione complessa $p(\mathbf{x})$ vengono mappati nella distribuzione base $\pi(\mathbf{z})$ ottimizzando il processo. Nella direzione diretta, dei campioni vengono estratti dalla distribuzione base $\pi(\mathbf{z})$ e vengono mappati nello spazio dei dati $p(\mathbf{x})$ permettendo quindi, partendo da delle variabili casualmente estratte, di generare dei dati in accordo con la distribuzione $p(\mathbf{x})$ dei dati stessi.

1.2.2 Autoencoder variazionali

Per capire il funzionamento degli autoencoder variazionali occorre prima fornire una breve descrizione degli autoencoder (AE).
Un autoencoder è uno specifico tipo di rete neurale utilizzata per codificare un input in una rappresentazione compressa che successivamente viene decodificata cosicché l'input ricostruito sia il più simile possibile a quello originale [4]. Ogni autoencoder è formato da un Encoder che comprime la dimensionalità dei dati $\mathbf{x}$ nelle features $\mathbf{z}$ appartenenti al cosiddetto latent space, e dal Decoder che ricostruisce i dati a partire da $\mathbf{z}$ ottenendo $\hat{\mathbf{x}}$. Encoder e decoder sono tipicamente delle reti neurali. Se indichiamo con e la funzione

di codifica e con d quella di decodifica abbiamo

$$\begin{cases} \mathbf{z} = e(\mathbf{x}) \\ \hat{\mathbf{x}} = d(\mathbf{z}) \end{cases} \implies \hat{\mathbf{x}} = d(e(\mathbf{x})),$$

Quando il modello è ben addestrato, idealmente, si dovrebbe avere $\mathbf{x} = \hat{\mathbf{x}}$. Quindi per addestrare un autoencoder occorre minimizzare la distanza fra $\mathbf{x}$ e $\hat{\mathbf{x}}$ minimizzando la funzione di perdita [1]

$$l = ||\mathbf{x} - \hat{\mathbf{x}}||^2 = ||\mathbf{x} - d(e(\mathbf{x}))||^2.$$

Una rappresentazione grafica della struttura di un autoencoder è mostrata in fig. 1.3.

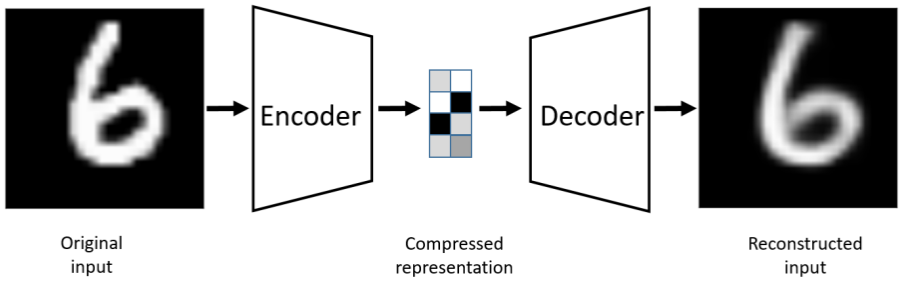


Figura 1.3: *Esempio di un autoencoder. Qui l'immagine in input viene prima compressa tramite l'encoder e poi ricostruita tramite il decoder.* [4]

In linea di principio dovrebbe esser possibile utilizzare, dopo la fase di addestramento, il decoder come modello generativo semplicemente campionando e decodificando punti dal latent space. Tuttavia, ciò non è possibile perché gli autoencoder soffrono di overfitting, infatti non impongono una struttura regolare allo spazio latente, e quindi non si raggiunge un livello di generalizzazione sufficiente a generare dei dati in modo soddisfacente.
Per raggiungere questa regolarizzazione ci vengono in aiuto gli autoencoder variazionali (VAE) che al posto di lavorare con singoli punti $\mathbf{z}$ dal latent space, lavorano con una distribuzione. La particolarità degli autoencoder variazionali sta quindi nel combinare l'apprendimento profondo con metodi probabilistici. In particolare l'idea alla base dei VAE consiste nel produrre tramite l'encoder non dei singoli punti ma una distribuzione normale fattorizzata. Precisamente l'encoder, a partire dal dataset, crea un vettore di medie $\boldsymbol{\mu}$ ed un vettore di deviazioni standard $\boldsymbol{\sigma}$ con cui costruire la distribuzione fattorizzata $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ a partire dalle distribuzioni $\mathcal{N}(\mu_i, \sigma_i)$. Il concetto alla base di tale processo è la Variational Inference (VI) che sceglie una famiglia di funzioni di densità di probabilità e trova quella più vicina alla densità di probabilità effettiva, spesso utilizzando come metrica di ottimizzazione la divergenza di Kullback-Leibler [5]. Frequentemente

si sceglie di lavorare con una distribuzione il più possibilmente vicina alla distribuzione normale standard, utilizzando quindi la funzione di perdita[2]

$$l = ||\mathbf{x} - d(\mathbf{z})||^2 + KL\left[\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}), \mathcal{N}(\mathbf{0}, \mathbf{1})\right].$$

Per generare la ricostruzione dei dati quindi si campionano dei punti dalle distribuzioni del latent space e si applica il decoder. In particolare durante la fase di campionamento si fa uso del reparameterization trick: non si campiona direttamente dalla distribuzione frutto del processo di encoding, ma si aggiunge un parametro $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ così che il campionamento di $\mathbf{z}$ segua la seguente

$$\mathbf{z} \sim \mathcal{N}'(\boldsymbol{\mu}, \boldsymbol{\sigma}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}.$$

Questa è una scelta pratica imposta dal dover calcolare i gradienti rispetto a $\boldsymbol{\mu}$ e $\boldsymbol{\sigma}$ durante la fase di addestramento. Il reparameterization trick rende differenziabile il campionamento.
Una visualizzazione grafica di un autoencoder variazionale è mostrata in fig. 1.4.

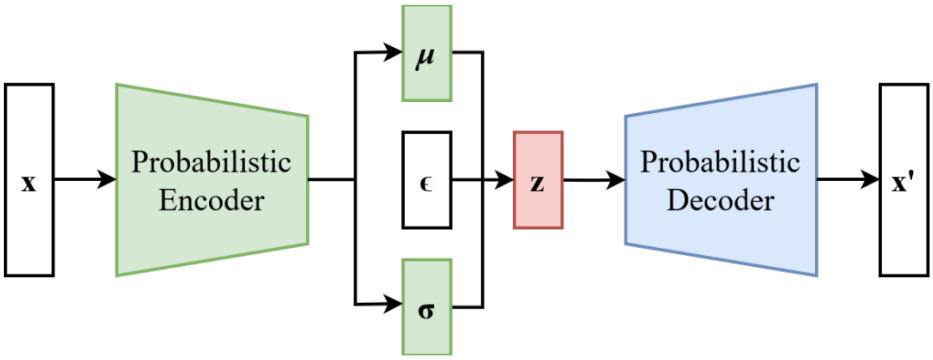


Figura 1.4: *Esempio di un autoencoder variazionale. Qui il dato ricostruito viene indicato con $\boldsymbol{x}'$, nella notazione utilizzata in questa tesi sarebbe $\hat{\boldsymbol{x}}$.* [5]

Una volta terminata la fase di addestramento dell'autoencoder variazionale sarà possibile utilizzare il decoder come generatore di dati.

1.2.3 Reti generative avversarie

Le reti generative avversarie sono un modello costituito da due reti addestrate simultaneamente: un Generatore $G(\mathbf{z}; \boldsymbol{\theta}_g)$ che prova ad apprendere la distribuzione dei dati ed un Discriminatore $D(\mathbf{x}; \boldsymbol{\theta}_d)$ che stima la probabilità che un campione appartenga ai dati

[2]In generale si usa una funzione di perdita della forma: funzione di perdita della ricostruzione + termine di regolarizzazione KL

di addestramento o a quelli generati da G [6].
Più specificatamente, G produce dei campioni sintetici $\mathbf{x}$ in accordo con la distribuzione p_g a partire da una variabile estratta da una distribuzione di rumore $\mathbf{z} \sim p_z$ e genera campioni cercando di renderli indistinguibili da quelli reali. D invece prova a capire se i dati che riceve sono reali (e quindi $\mathbf{x} \sim p_{data}$) o generati (con $\mathbf{x} \sim p_g$).
Il processo di addestramento avviene a turni alternati: in una prima fase il generatore G migliora la propria capacità di generare dati sempre più simili ai dati reali (variando i parametri $\boldsymbol{\theta}_g$ che caratterizzano G e tenendo fissi quelli che caratterizzano D), nella seconda fase il discriminatore D incrementa la propria abilità nel ben distinguere i dati generati da quelli reali (variando i parametri $\boldsymbol{\theta}_d$ della rete D e tenendo fissi quelli del generatore G). Le due fasi si alternano fino all'ottenimento di un risultato soddisfacente. L'addestramento sarà concluso con successo quando i dati del generatore mostrano una distribuzione molto simile, idealmente uguale, a quella dei dati reali e quindi il discriminatore non è più capace di classificare i dati correttamente.
Si definisce la seguente funzione $V(G, D)$ e le due reti vengono allenate in una sfida di massimizzazione e minimizzazione

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z} [\log (1 - D(G(\mathbf{z})))]$$

in cui abbiamo indicato con $D(\mathbf{x})$ la probabilità che $\mathbf{x}$ provenga dai dati reali e non da p_g. D viene allenato in modo tale da massimizzare la probabilità che la classificazione sia avvenuta correttamente, G viene invece allenato in modo tale che venga minimizzato $(1 - D(G(\mathbf{z})))$ e che quindi i dati generati risultino essere il più simili possibile, nel limite desiderato indistinguibili, ai dati reali [6].
Una rappresentazione grafica del funzionamento delle reti generative avversarie è mostrata in fig. 1.5.

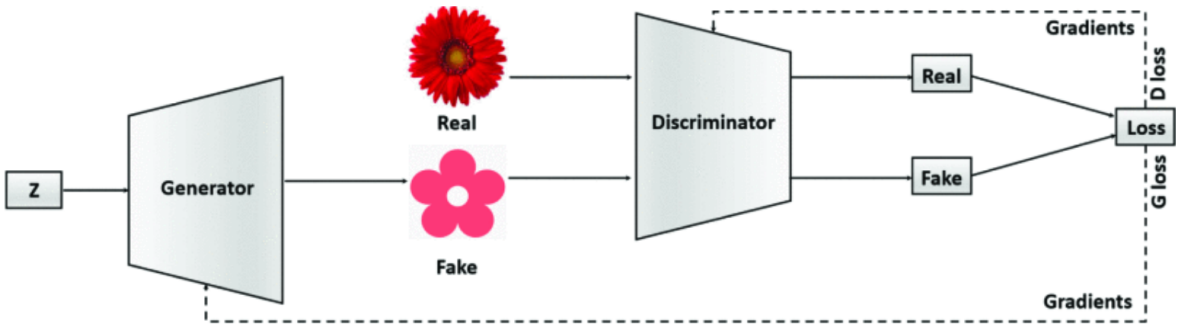


Figura 1.5: *Struttura di reti generative avversarie. Le linee tratteggiate indicano il processo di addestramento.* [7]

Una volta che il generatore ed il discriminatore sono stati addestrati correttamente, è possibile utilizzare la rete neurale G come generatore di dati fornendogli dei campioni estratti da del rumore.

1.2.4 Modelli di diffusione

I modelli di diffusione costituiscono tutta una famiglia di modelli generativi nati di recente e sviluppatisi molto velocemente in questi anni. Molti metodi sono stati sviluppati per migliorare i modelli di diffusione, sia soffermandosi su performance empiriche, sia estendendo le capacità del modello da un punto di vista teorico [8].
Esistono diversi formalismi per i modelli di diffusione, i tre principali sono: Denoising Diffusion Probabilistic Models (DDPM), Score Based Generative Models (SGM) e Stochastic Differential Equations (SDE). È comunque possibile dimostrare che questi tre formalismi, sotto particolari condizioni, sono matematicamente equivalenti. In particolare SDE "incapsula" al suo interno sia SGM sia DDPM [9]. Di seguito non tratteremo questi formalismi ma vedremo solo qual è l'idea che caratterizza tutti i modelli di diffusione.
In modo approssimativo, tutti i modelli di diffusione consistono in un processo forward ed un processo backward [10]. Durante il processo forward il modello distrugge un campione della distribuzione dei dati iniettando del rumore, tipicamente ciò avviene corrompendo sequenzialmente il campione con un rumore casuale Gaussiano. Nel processo backward una rete neurale è addestrata per rimuovere sequenzialmente il rumore ed ottenere un nuovo campione "ripulito".
Una rappresentazione grafica di tale processo viene mostrata in fig. 1.6.

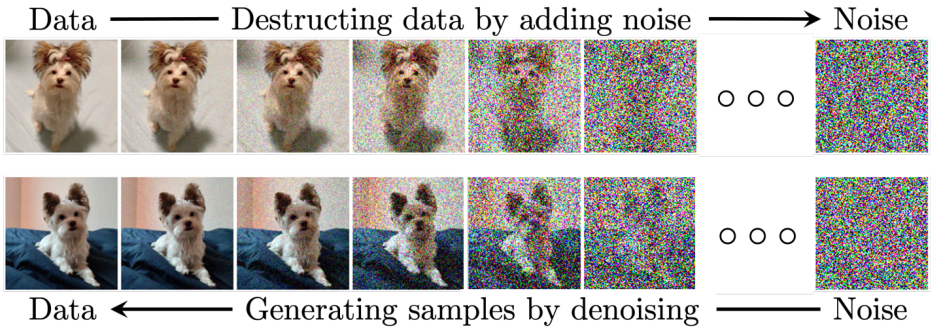


Figura 1.6: *Rappresentazione di come funzionano, in modo generale, i modelli di diffusione.* [8]

Una volta che il modello sarà addestrato con successo, sarà allora possibile utilizzare il processo backward per generare dei dati a partire da del rumore.

1.3 Machine learning generativo per la fisica delle alte energie

Le simulazioni Monte Carlo occupano un ruolo centrale in fisica delle alte energie: ogni misura di precisione o ricerca di nuova fisica al LHC richiede il confronto fra i dati sperimentali e le previsioni ottenute simulando l'intera catena che va dal processo di collisione fondamentale fino alla risposta del rivelatore mostrata in fig. 1.7. Il passo più oneroso dell'intera catena è, tipicamente, la simulazione dettagliata dell'interazione fra le particelle ed i detector che da sola può arrivare ad assorbire la maggior parte del budget di calcolo di un esperimento. Ad esempio nell'esperimento ATLAS l'80 % delle risorse computazionali destinate alle simulazioni MC vengono usate per la simulazione dei calorimetri [11]. La richiesta di eventi simulati aumenterà conseguentemente all'avvio dell'High-Luminosity LHC (HL-LHC), tuttavia le risorse di calcolo disponibili non sono attese crescere altrettanto rapidamente. In [12] vengono mostrate le proiezioni delle risorse necessarie sia alla computazione che allo stoccaggio dei dati per gli esperimenti ATLAS e CMS (i due esperimenti più grandi in LHC). È in questo contesto di crescente pressione computazionale che i modelli generativi trovano una delle loro applicazioni più naturali: uno strumento concreto per accelerare di diversi ordini di grandezza singoli anelli della catena di simulazione.

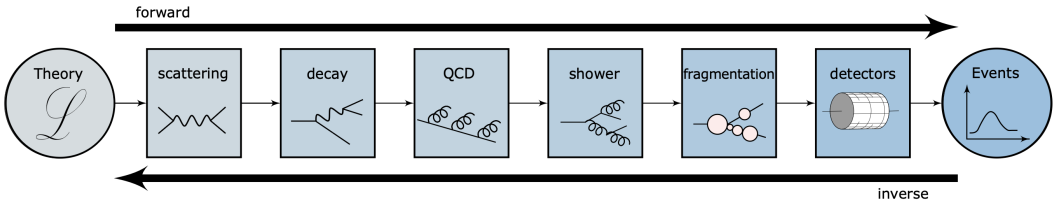


Figura 1.7: *Rappresentazione della catena di generazione in LHC.* [13]

Prima di passare in rassegna le applicazioni più rilevanti, è utile introdurre un criterio che aiuti a orientarsi fra di esse, distinguendo due situazioni concettualmente diverse in cui i modelli generativi vengono impiegati. Nella prima situazione la distribuzione di probabilità che si vuole campionare è simulata direttamente eseguendo i calcoli teorici al desiderato ordine perturbativo di approssimazione. Ad esempio la sezione d'urto differenziale di un processo di collisione è calcolabile, in linea di principio, a partire dai diagrammi di Feynman usando la teoria delle perturbazioni. In questo caso il modello generativo ha il solo compito di campionare in modo efficiente una funzione nota ma costosa da valutare puntualmente, in uno spazio delle fasi ad alta dimensionalità [14].
Nella seconda situazione, al contrario, la distribuzione bersaglio non è esprimibile in forma chiusa, o perché descrive un processo la cui fisica è troppo complessa per essere risolta analiticamente (l'evoluzione di uno sciame di particelle secondarie nel materiale

del rivelatore) [15], o perché appartiene a un regime della cromodinamica quantistica (QCD) non perturbativo per cui non esiste ancora una teoria predittiva (ad esempio l'adronizzazione dei partoni in adroni osservabili) [16], oppure per vari altri motivi. In entrambi i casi il modello generativo deve apprendere la distribuzione empiricamente, a partire da un campione di esempi prodotti da un simulatore di riferimento[3] o direttamente dai dati sperimentali.
Tale classificazione, per quanto utile, non è sempre netta: alcune tecniche nate per il secondo caso, in particolare i diffusion model e il flow-matching, sono state recentemente impiegate con successo anche per accelerare il campionamento dello spazio delle fasi, segno di una progressiva convergenza metodologica fra i due filoni [17].

Una volta disponibile un modello generativo capace di stimare, o campionare da, una densità di probabilità complessa, si aprono ulteriori applicazioni. Nell'unfolding, ad esempio, una rete generativa viene addestrata a invertire statisticamente la catena di simulazione, restituendo, per ogni evento osservato a livello di rivelatore, una distribuzione, a livello delle particelle, di una qualche quantità fisica di interesse [18]. Un'altra applicazione riguarda la ricerca di nuova fisica in modo indipendente dal modello: non si assume in anticipo quale forma debba avere il segnale cercato, si vuole solo scoprire se nei dati appare qualcosa di inatteso. Il metodo si basa su una variabile che, in presenza di una nuova particella, mostrerebbe un picco localizzato (per esempio una massa invariante). Lontano da quel picco, in zone dei dati dove siamo ragionevolmente certi che non ci sia alcun segnale, un modello generativo impara come è fatta la sola distribuzione di fondo. Questa forma appresa viene poi estesa anche alla zona centrale, dove un eventuale segnale si nasconderebbe, ottenendo così una previsione di quanti eventi di fondo aspettarsi in quella zona. Se i dati osservati superano questa previsione, l'eccesso osservato è un possibile segnale di nuova fisica [19]. Un'ultima applicazione è nata quando ci si è chiesti se gli eventi generati da un modello aggiungano davvero informazione statistica indipendente rispetto al campione di addestramento, o se siano solo una sua ricombinazione. La risposta, non scontata, è che un modello generativo ben calibrato può ridurre l'incertezza statistica al di sotto di quella del solo campione di addestramento [20].

A prescindere dalla specifica applicazione, tutti i più grandi esperimenti di fisica delle alte energie hanno iniziato, e continuano nel presente, a verificare quali possono essere le applicazioni dei modelli generativi che possano portare ad un vantaggio in termini di efficienza, risoluzione, risorse computazionali utilizzate, tempo richiesto, ecc.
In questa tesi l'attenzione sarà rivolta alla simulazione degli shower calorimetrici. In particolare nel capitolo successivo vedremo quali sono i modelli utilizzati dall'esperimento ATLAS a tale scopo e nell'ultimo capitolo verrà analizzato un nuovo possibile modello.

[3]Tipicamente Geant4 per la risposta del rivelatore, generatori fenomenologici come Pythia per l'adronizzazione.

Capitolo 2

L'esperimento ATLAS e la simulazione veloce in ATLAS

La fisica delle particelle elementari si occupa di indagare i costituenti fondamentali della materia e le interazioni che li governano. Per poter osservare fenomeni che avvengono a scale così ridotte, dell'ordine di 10^{-17} cm per le particelle subnucleari, è necessario disporre di sonde con impulso sufficientemente elevato, come conseguenza diretta della relazione di De Broglie

$$\lambda = \frac{h}{p},$$

secondo cui la lunghezza d'onda associata a una particella è inversamente proporzionale al suo impulso. Inoltre c'è un'altra possibilità investigativa legata alla produzione nuove particelle massive: l'energia resa disponibile in una collisione può infatti convertirsi nella massa delle particelle prodotte secondo la relazione

$$E = mc^2.$$

Gli acceleratori sono gli strumenti utilizzati per lo studio della fisica atomica e subatomica perché consentono di accelerare particelle cariche a velocità prossime a quella della luce e a sprigionare grandi energie.

In generale un acceleratore è costituito da una sorgente di particelle, un campo accelerante e un campo guida, quest'ultimo necessario per costringere le particelle a muoversi lungo traiettorie predefinite. L'accelerazione avviene in condizioni di ultra-alto vuoto, in modo da minimizzare le collisioni con le molecole del gas residuo che destabilizzerebbero i fasci di particelle.
Una distinzione fondamentale è quella fra acceleratori a bersaglio fisso, in cui il fascio accelerato collide con un bersaglio fermo nel sistema di riferimento del laboratorio, ed i collider, in cui a scontrarsi sono due fasci con impulsi uguali e opposti.

Nei prossimi paragrafi descriviamo le caratteristiche principali del Large Hadron Collider (LHC), l'acceleratore attualmente più potente al mondo, e del rivelatore ATLAS. Nell'ultimo paragrafo vedremo invece quali sono gli strumenti utilizzati dal gruppo ATLAS per la simulazione veloce di shower calorimetri.

2.1 Il Large Hadron Collider

Il Large Hadron Collider (LHC) è un collider adronico circolare situato al CERN (Conseil Européen pour la Recherche Nucléaire), nei pressi di Ginevra, a cavallo del confine franco-svizzero. Si tratta ad oggi dell'acceleratore più grande e potente mai costruito, in grado di far collidere protoni, e in alcuni periodi di presa dati anche ioni pesanti, a energie nel centro di massa mai raggiunte prima.

Struttura e catena di accelerazione

L'LHC è ospitato in un tunnel circolare di circa 27 km di circonferenza, a una profondità media di 100 m, originariamente costruito per ospitare il Large Electron-Positron Collider (LEP) e successivamente adattato per accogliere l'attuale acceleratore. La scelta di sostituire un collider elettrone-positrone con uno protone-protone non è casuale, ma è dettata dal fenomeno della radiazione di sincrotrone: una particella carica che percorre una traiettoria circolare irraggia infatti una potenza come

$$\frac{dE}{dt} \propto \frac{E^4}{m^4 \rho^2},$$

indicando con E l'energia della particella, m la sua massa e ρ il raggio di curvatura. Viene quindi irraggiata una potenza inversamente proporzionale alla quarta potenza della sua massa (da notare che a parità di energia, una particella più leggera irraggia una potenza molto maggiore). Per questo motivo un elettrone, circa 2000 volte più leggero di un protone, perde per irraggiamento un'energia enormemente superiore a parità di traiettoria percorsa e ciò limitava l'energia massima raggiungibile da LEP a soli 209 GeV. Utilizzando protoni, invece, l'LHC riesce attualmente a raggiungere energie nel centro di massa fino a $\sqrt{s} = 13.6$ TeV [21].

Prima di essere iniettati nell'LHC, i protoni vengono prodotti ionizzando atomi di idrogeno e fatti accelerare attraverso una catena di acceleratori più piccoli. Il LINAC4 porta i protoni a un'energia di 160 MeV; questi vengono poi iniettati nel Proton Synchrotron Booster (PSB), dove raggiungono 2 GeV, e successivamente nel Proton Synchrotron (PS) e nel Super Proton Synchrotron (SPS), che accelerano i protoni rispettivamente fino a 26 GeV e 450 GeV. Solo a questo punto i due fasci, circolanti in direzioni opposte all'interno di due tubi mantenuti in condizioni di ultra-alto vuoto, vengono iniettati nell'anello principale dell'LHC come mostrato in fig. 2.1.

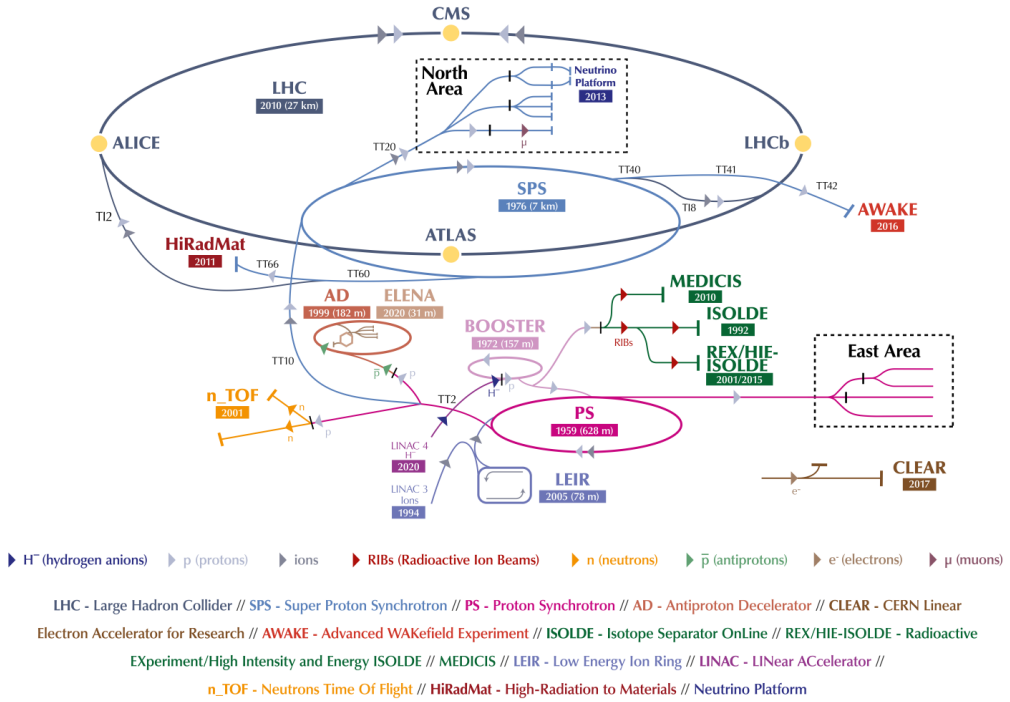


Figura 2.1: *Schema del complesso di acceleratori del CERN.* [22]

Per mantenere i fasci lungo orbite circolari stabili bisogna ricorrere a un potente sistema magnetico: l'LHC è dotato di 1232 magneti dipolari superconduttori (ciascuno lungo circa 15 m), raffreddati a 1.9 K, capaci di generare un campo magnetico di 8.3 T. A questi si affiancano 392 magneti quadrupolari, il cui compito è invece quello di focalizzare i fasci, riducendone le dimensioni trasversali e aumentando così la probabilità di collisione [23].

Luminosità ed energia nel centro di massa

Al di là dell'energia raggiunta, un secondo parametro cruciale per la fisica di un collider è la luminosità istantanea, cioè la grandezza che quantifica il numero di collisioni prodotte per unità di tempo. Dato un processo con sezione d'urto σ, il tasso di eventi attesi è infatti dato da

$$\frac{dN}{dt} = \sigma \mathcal{L},$$

dove $\mathcal{L}$ è la luminosità istantanea. Quest'ultima può essere espressa in funzione dei parametri caratteristici del fascio, indicando con f la frequenza di rivoluzione, con n_b il numero di pacchetti circolanti e con N_1, N_2 il numero di particelle contenute in ciascun

pacchetto, come

$$\mathcal{L} = \frac{f\, n_b\, N_1 N_2}{4\pi \sigma_x \sigma_y},$$

cioè in funzione anche delle dimensioni trasversali dei fasci σ_x e σ_y. Integrando la luminosità istantanea sul tempo di presa dati si ottiene la luminosità integrata

$$L = \int \mathcal{L}\, dt,$$

solitamente espressa in fb^{-1}, unità di misura tale per cui 1 b = 10^{-24} cm^2. Il numero totale di eventi attesi per un dato processo risulta quindi pari a $N = \sigma L$.

In sostanza, la luminosità quantifica quanto efficacemente si riesce a tradurre l'energia dei fasci in eventi osservabili. Massimizzare tale grandezza è essenziale dal momento che quanto maggiore è il numero di collisioni raccolte tanto più precise saranno le misure effettuate e tanto più sarà possibile investigare processi rari.
Nella tabella 2.1 sono riassunti i principali periodi di presa dati dell'LHC, noti come Run, con le rispettive energie nel centro di massa e luminosità integrate misurate dall'esperimento ATLAS.

Periodo	$\sqrt{s}$ [TeV]	L [fb^{-1}]
Run 2 (2015–2018)	13	140.1
Run 3 (2022–2024)	13.6	164
Run 3 (2022–2025)	13.6	≈ 250

Tabella 2.1: Energie nel centro di massa e luminosità integrate raccolte da ATLAS durante i principali periodi operativi dell'LHC. La luminosità integrata per il Run 3 (2022-2025) è quella attesa. I valori riportati sono stati presi da [24], [25], [26].

I quattro esperimenti principali

Lungo l'anello dell'LHC, in corrispondenza dei quattro punti di interazione in cui i fasci vengono fatti collidere, sono installati altrettanti esperimenti principali, ciascuno progettato per uno scopo fisico specifico:

1. **ATLAS** (A Toroidal LHC ApparatuS): rivelatore di tipo general-purpose, cioè progettato per coprire un ampio spettro di obiettivi fisici, fra cui lo studio delle proprietà del bosone di Higgs e la ricerca di fisica oltre il Modello Standard. È l'esperimento oggetto di questa tesi e viene descritto in dettaglio nel paragrafo successivo.

2. **CMS** (Compact Muon Solenoid): anch'esso un rivelatore general-purpose, condivide con ATLAS gran parte degli obiettivi scientifici, pur adottando soluzioni tecniche differenti, cioè un diverso design del sistema di magneti e dei sottorivelatori.

3. **ALICE** (A Large Ion Collider Experiment): dedicato principalmente allo studio delle collisioni fra ioni pesanti, con l'obiettivo di indagare le proprietà del plasma di quark e gluoni, uno stato della materia che si ritiene abbia caratterizzato l'universo primordiale.

4. **LHCb** (Large Hadron Collider beauty): specializzato nello studio degli adroni contenenti quark b e c per la ricerca dell'asimmetria fra materia e antimateria. A differenza di ATLAS e CMS, non circonda simmetricamente il punto di collisione, ma è concentrato sulle particelle emesse in una specifica direzione.

Complessivamente, oltre a questi quattro rivelatori principali, all'LHC sono installati anche esperimenti più piccoli e specializzati, come LHCf, TOTEM, MoEDAL-MAPP, FASER e SND@LHC [27].

2.2 Il rivelatore ATLAS

ATLAS (A Toroidal LHC ApparatuS) è uno dei due rivelatori general-purpose installati lungo l'anello dell'LHC. Possiede una struttura cilindrica a strati concentrici attorno al punto di collisione pensata per garantire una copertura angolare pressoché completa e per misurare, strato dopo strato, caratteristiche diverse delle particelle prodotte. Procedendo dall'interno verso l'esterno si incontrano: l'Inner Detector, il sistema calorimetrico e lo spettrometro per muoni. Tutto ciò è immerso in un campo magnetico generato da un sistema di magneti utile a modificare le traiettorie delle particelle cariche elettricamente.

Struttura generale e sistema di coordinate

Il rivelatore, mostrato in fig. 2.2 ha una lunghezza di circa 44 m, un diametro di 25 m e un peso di circa 7000 tonnellate.

Per descriverne la geometria si utilizza un sistema di coordinate cilindriche (z, ϕ, η) centrato nel punto di interazione: l'asse z coincide con la direzione del fascio, $\phi \in [0, 2\pi)$ è l'angolo azimutale nel piano trasverso e, al posto dell'angolo polare θ, si preferisce utilizzare la pseudorapidità

$$\eta = -\ln\left(\tan\frac{\theta}{2}\right).$$

Il rivelatore può essere suddiviso, in prima approssimazione, in tre macro-sistemi funzionali: l'Inner Detector, dedicato al tracciamento delle particelle cariche; il sistema calorimetrico, che ne misura l'energia, e lo spettrometro per muoni, sottosistema capace appunto di rilevare tali particelle. Vale la pena anticipare che, in vista dell'upgrade ad alta luminosità (HL-LHC), è già prevista una sostituzione dell'attuale Inner Detector con un nuovo tracciatore interamente al silicio (Inner Tracker, ITk), capace di operare fino

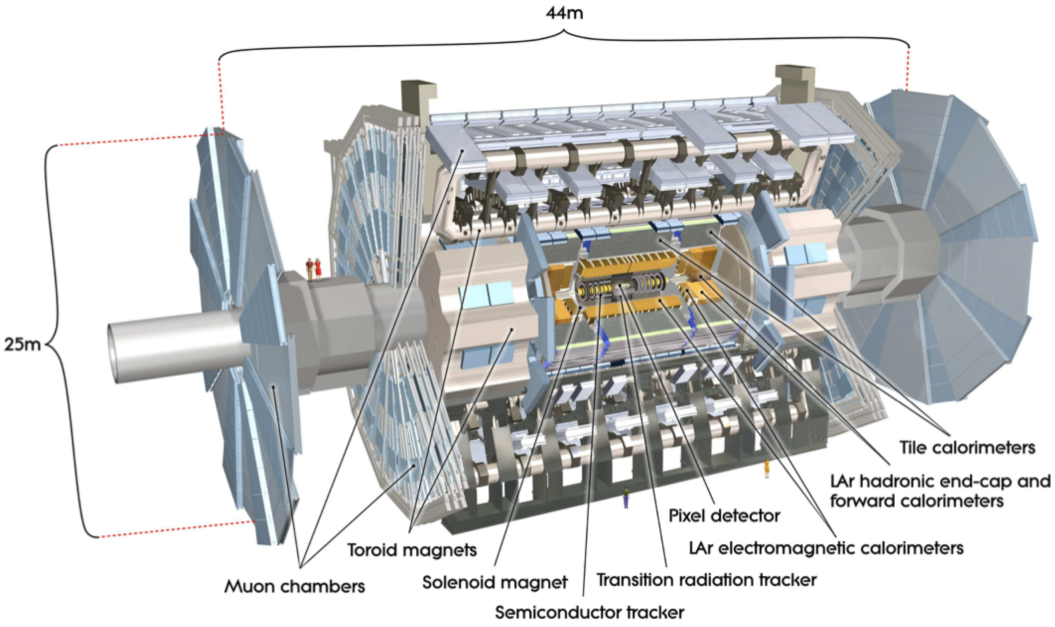


Figura 2.2: *Struttura del rivelatore ATLAS* [28]

a $|\eta| = 4.0$. Verranno anche modificati il rivelatore di muoni e il sistema d'acquisizione [29].

Sistema di magneti

Per poter modificare le traiettorie delle particelle cariche e misurarne carica elettrica e impulso trasverso bisogna ricorrere a un sistema di magneti superconduttori. L'impulso trasverso p_T, la carica q, il campo magnetico $\boldsymbol{B}$ e il raggio di curvatura ρ dipendono fra loro secondo la relazione

$$p_T = q|\boldsymbol{B}|\rho.$$

Il sistema è composto da tre magneti distinti, mantenuti a temperature criogeniche da un impianto a elio liquido:

1. **Solenoide centrale**: posto immediatamente attorno all'Inner Detector, è lungo circa 5.8 m con un diametro di circa 2.46 m e genera un campo assiale di 2 T.

2. **Toroidi barrel**: costituito da otto bobine superconduttrici disposte a petalo attorno all'asse z. Il valore medio del campo magnetico nella regione di tracciamento dei muoni è di circa 0.5–1 T.

3. **Toroidi di end-cap**: analoghi al toroide barrel ma di dimensioni minori. Estendono la copertura del sistema toroidale alle regioni più in avanti del rivelatore.

A differenza del solenoide, che agisce su tutte le particelle cariche, il sistema toroidale è progettato per curvare esclusivamente le traiettorie dei muoni, le uniche particelle cariche in grado di raggiungere lo spettrometro esterno [26]. In fig. 2.3 sono rappresentati i tre componenti che formano il sistema magnetico.

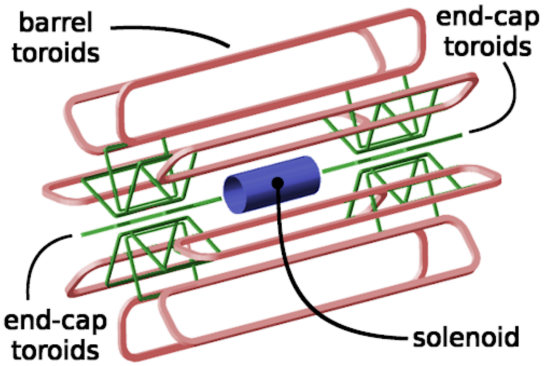


Figura 2.3: *Rappresentazione schematica del sistema di magneti di ATLAS. In blu il solenoide centrale, in rosso i toroidi barrel e in verde i toroidi di end-cap.*

Inner Detector

L'Inner Detector è il sottosistema più interno di ATLAS e copre la regione $|\eta| < 2.5$. Il suo compito è misurare traiettoria, impulso e carica elettrica delle particelle cariche prodotte nel vertice di collisione. È costituito da tre sottorivelatori concentrici, mostrati in fig. 2.4, ciascuno ottimizzato per un compito specifico:

1. **Pixel Detector**: il più vicino alla linea del fascio (a partire da circa 3.3 cm di distanza) e con la granularità più fine, con circa 92 milioni di canali di lettura complessivi. Il suo strato più interno, l'Insertable B-Layer (IBL), aggiunto durante il primo Long Shutdown, è fondamentale per la ricostruzione dei vertici secondari.

2. **Semiconductor Tracker (SCT)**: costituito da oltre 4000 moduli di microstrisce di silicio (circa 6.3 milioni di canali complessivi), disposti in quattro strati cilindrici nella regione centrale e in dischi nelle regioni di end-cap; garantisce una risoluzione spaziale di circa 20 μm nel piano trasverso.

3. **Transition Radiation Tracker (TRT)**: il più esterno dei tre, è costituito da centinaia di migliaia di tubi a deriva del diametro di 4 mm, riempiti con una miscela di gas a base di xenon. Oltre a fornire numerosi punti aggiuntivi di misura della traiettoria, permette un primo livello di identificazione degli elettroni: la probabilità di produzione della radiazione di transizione dipende dal fattore di

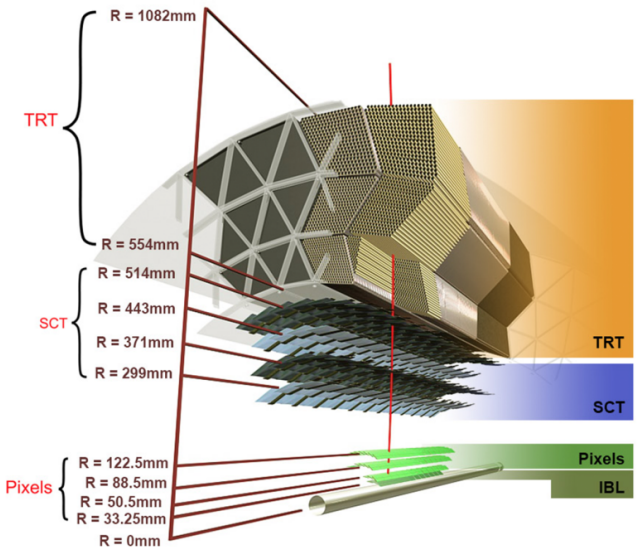


Figura 2.4: *Sezione trasversale dell'Inner Detector* [30]

Lorentz della particella e aumenta significativamente per γ dell'ordine di 10^3, fino a raggiungere una regione di saturazione. Questo comportamento permette di distinguere gli elettroni dagli adroni carichi a parità di impulso [26].

Calorimetri

Il sistema calorimetrico, mostrato in fig. 2.5, circonda l'Inner Detector ed è costituito da un calorimetro elettromagnetico (ECAL) e uno adronico (HCAL), entrambi calorimetri a campionamento, cioè formati da strati alternati di materiale assorbente (che favorisce lo sviluppo di sciami di particelle secondarie) e materiale attivo (che ne misura l'energia depositata).
L'ECAL, che utilizza argon liquido come mezzo attivo (da cui il nome Liquid Argon Calorimeter, LAr) e piombo come assorbitore, ha una geometria che ne garantisce la copertura angolare completa; la sua parte centrale (EMB) copre la regione $|\eta| < 1.475$, mentre gli end-cap (EMEC) coprono $1.375 < |\eta| < 3.2$.
L'HCAL è invece diviso in un Tile Calorimeter a scintillatori, che copre la regione centrale ($|\eta| < 1.7$), e in calorimetri adronici di end-cap e forward (HEC ed FCal) che estendono la copertura fino a $|\eta| < 4.9$.
Elettroni e fotoni depositano praticamente tutta la loro energia già nell'ECAL, mentre gli adroni, interagendo anche fortemente, vi rilasciano solamente circa il 60% della propria energia: il restante 40% viene quindi campionato dall'HCAL, motivo per cui i due calo-

rimetri devono necessariamente lavorare in combinazione per una corretta ricostruzione dell'energia adronica [26].

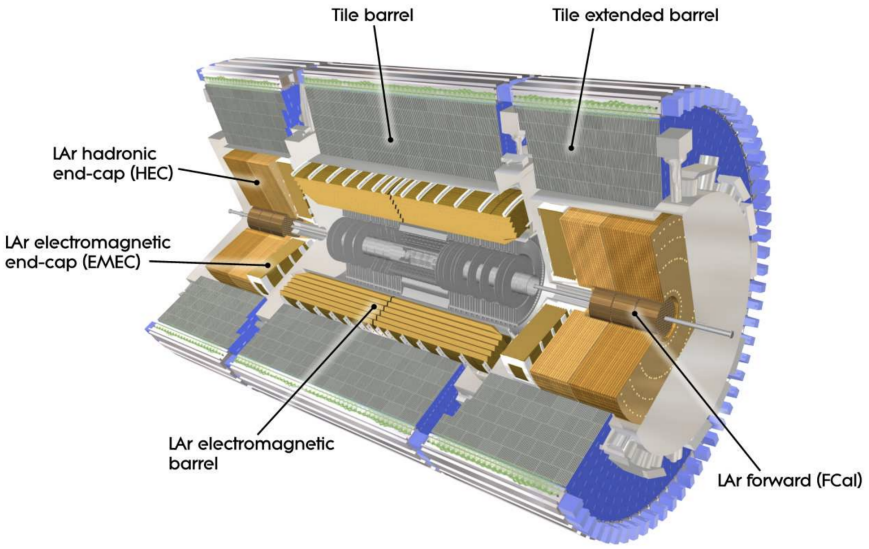


Figura 2.5: *Rappresentazione del sistema calorimetrico di ATLAS.* [28]

Spettrometro per muoni

Lo spettrometro per muoni costituisce lo strato più esterno di ATLAS e sfrutta la curvatura della traiettoria nel campo magnetico per ricostruire con precisione l'impulso dei muoni, unica particella carica capace di attraversare l'intero sistema calorimetrico, insieme ai neutrini (neutri) che pure lo attraversano indisturbati. Le tecnologie impiegate variano a seconda della regione del rivelatore:

1. **Regione centrale (barrel)**: le misure di precisione sono affidate alle Monitored Drift Tubes (MDT), mentre il trigger è fornito dalle Resistive Plate Chambers (RPC).

2. **Regioni in avanti (end-cap)**: le misure di precisione sono garantite dalle cosiddette New Small Wheels (NSW), che integrano rivelatori Micromegas e small-strip Thin Gap Chambers (sTGC). Ulteriori Thin Gap Chambers (TGC), poste più esternamente, forniscono invece il segnale di trigger in questa regione [26].

Complessivamente, lo spettrometro, mostrato in fig. 2.6, garantisce una copertura in pseudorapidità fino a $|\eta| \sim 2.7$ per il tracciamento di precisione.

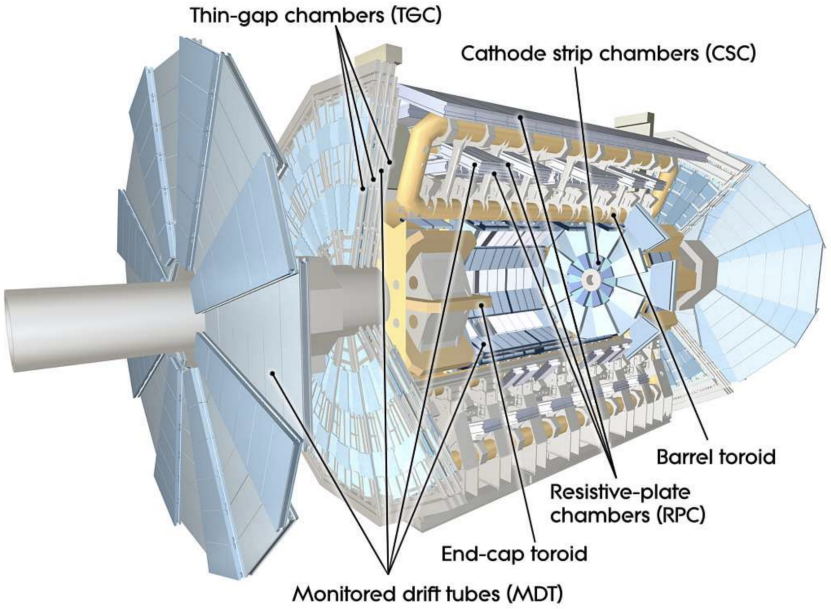


Figura 2.6: *Rappresentazione dello spettrometro per muoni di ATLAS. Vengono anche riportati i toroidi barrel ed i toroidi di end-cap* [28]

Sistema di trigger e acquisizione dati

Data l'enorme quantità di collisioni prodotte ogni secondo, dell'ordine dei miliardi, è necessario un sistema in grado di selezionare in tempo reale solo gli eventi di reale interesse fisico. Tale selezione avviene in due passaggi: un primo livello hardware (L1), che sfrutta informazioni dai soli calorimetri e dallo spettrometro per muoni per prendere una decisione in meno di 2.5 μs, e un secondo livello software, l'High-Level Trigger (HLT), che opera su una farm di circa 40000 core di CPU per effettuare un'analisi più dettagliata degli eventi selezionati dal L1 [26]. A titolo esemplificativo: su circa un miliardo di collisioni, soltanto poche centinaia vengono effettivamente salvate per l'analisi offline.
Un fattore che complica notevolmente il lavoro del sistema di trigger, così come quello degli algoritmi di ricostruzione, è il cosiddetto pile-up, cioè il numero di interazioni protone-protone che avvengono simultaneamente a ogni incrocio di pacchetti. Indicando con $\langle\mu\rangle$ il pile-up medio, esso può essere espresso come

$$\langle\mu\rangle = \mathcal{L}\,\frac{\sigma_{pp}}{f_{rev}\,n_b},$$

dove $\sigma_{pp} \simeq 80$ mb è la sezione d'urto anelastica protone-protone [24]. Durante il Run 2 ATLAS ha operato con $\langle\mu\rangle \simeq 30$, valore salito a circa 60 nel Run 3. Tale crescita, se da un lato permette di raccogliere più dati nello stesso tempo, dall'altro peggiora

la risoluzione delle misure e complica sensibilmente la ricostruzione degli oggetti fisici, rendendo necessarie tecniche dedicate di sottrazione del pile-up [26].

2.3 Simulazione veloce dei calorimetri di ATLAS: AtlFast3

2.3.1 Motivazioni della simulazione veloce del rivelatore

La produzione di eventi Monte Carlo (MC) rappresenta una delle voci più onerose nel bilancio computazionale dell'esperimento ATLAS: la sola simulazione del passaggio delle particelle nel rivelatore assorbe circa il 40% delle risorse di calcolo complessive [11]. All'interno della sola simulazione, a dominare il consumo di risorse è la modellizzazione dettagliata degli sciami elettromagnetici e adronici nel sistema calorimetrico che avviene tramite Geant4. A causa della complessa geometria, tale simulazione può arrivare a occupare circa l'80% del tempo totale di simulazione per processi come la produzione di coppie $t\bar{t}$ [11].

Bisogna inoltre considerare che il fabbisogno computazionale di ATLAS è destinato a crescere ulteriormente nei prossimi anni, sia durante l'attuale Run 3 sia in vista del Run 4, il primo del programma ad alta luminosità (HL-LHC) [31]. Per far fronte a questo problema, sono stati sviluppati nel tempo diversi strumenti di simulazione veloce. Si tratta di programmi in grado di riprodurre la risposta del rivelatore molto più rapidamente di Geant4, mantenendo al contempo una perdita di accuratezza minima.
Il primo di questi strumenti, AtlFastII, è stato impiegato per generare oltre 32 dei 52 miliardi di eventi simulati per le analisi del Run 2. Tale strumento presentava tuttavia dei limiti evidenti che sono stati superati da AtlFast3, analizzato qui di seguito [11].

2.3.2 Architettura di AtlFast3

AtlFast3 è un tool che permette la generazione diretta dei depositi di energia nel sistema calorimetrico di ATLAS. Lo strumento è costituito da due sottosistemi, corrispondenti a due approcci differenti al problema:

- **FastCaloSimV2**: parametrizza separatamente lo sviluppo longitudinale e laterale dello sciame. Lo sviluppo longitudinale, cioè la frazione di energia depositata in ciascuno strato del calorimetro, è fortemente correlato fra strati adiacenti e, per gestire questa correlazione, si utilizza una Principal Component Analysis (PCA) in due passaggi. Vengono prima classificati gli sciami in intervalli in base all'energia, pseudorapidità e tipo di particella e poi per ogni intervallo viene generata una rappresentazione non correlata e approssimativamente Gaussiana (viene quindi creata

una mappatura fra intervalli e distribuzioni). In fase di simulazione vengono selezionati in modo casuale dei bin, vengono generati dei numeri casuali e, tramite la trasformazione inversa, si generano delle energie totali e le rispettive distribuzioni nei vari layer. Lo sviluppo laterale è invece parametrizzato tramite funzioni di densità di probabilità bidimensionali [31].

- **FastCaloGANV2**: si tratta di un sistema basato su Generative Adversarial Networks (GAN). Vengono addestrate simultaneamente due reti neurali, un generatore e un discriminatore: il primo cerca di produrre sciami il più possibile simili a quelli generati da Geant4, mentre il secondo cerca di distinguere gli sciami reali da quelli generati. Le due reti migliorano iterativamente le proprie prestazioni finché non si raggiunge un buon equilibrio. A quel punto il generatore è in grado di simulare sciami calorimetrici in tempi molto ridotti rispetto a Geant4, mantenendo una buona accuratezza. Nello specifico, FastCaloGANV2 impiega una Wasserstein GAN con gradient penalty (WGAN-GP), una variante che rende l'addestramento più stabile [31].

La scelta fra i due strumenti non è arbitraria, ma dipende dalla regione del rivelatore, dal tipo di particella e dalla sua energia. In fig. 2.7 sono riportati i regimi operativi dei modelli. In generale, Geant4 continua a essere utilizzato per tutte le particelle nell'Inner Detector, per i muoni e per gli adroni a bassa energia nel calorimetro. FastCaloGANV2 copre prevalentemente le basse energie e le regioni più in avanti del rivelatore, invece FastCaloSimV2 viene preferito alle energie più elevate nella regione centrale. Un caso a parte è rappresentato dal cosiddetto muon punch-through, cioè la parte di particelle che raggiungono lo spettrometro a muoni a seguito di interazioni tardive di adroni ad alta energia: questo fenomeno viene modellato con l'uso congiunto di una particolare parametrizzazione (diversa da quella usata in FastCaloSimV2) e di Geant4 [11].

2.3.3 Valutazione delle prestazioni

AtlFast3 rappresenta oggi il miglior compromesso disponibile fra velocità di calcolo e fedeltà nella riproduzione della fisica di Geant4.
Dal punto di vista computazionale, la versione di AtlFast3 attualmente in produzione per il Run 3 risulta da 3 a 15 volte più veloce di Geant4, con il guadagno osservato esser maggiore per processi con particelle ad alta energia. Complessivamente, tenendo conto anche della simulazione dell'Inner Detector (che rimane a carico di Geant4 che è molto dispendioso), AtlFast3 richiede in media solo circa il 20% della CPU necessaria alla simulazione completa tramite Geant4 [31].
Per quanto riguarda l'accuratezza fisica, AtlFast3 riproduce con buona fedeltà l'energia del cluster calorimetrico e, per la maggior parte delle osservabili utilizzate nelle analisi fisiche, AtlFast3 e Geant4 concordano entro pochi punti percentuali, con un miglioramento evidente rispetto al suo predecessore AtlFastII. Un ulteriore vantaggio di AtlFast3

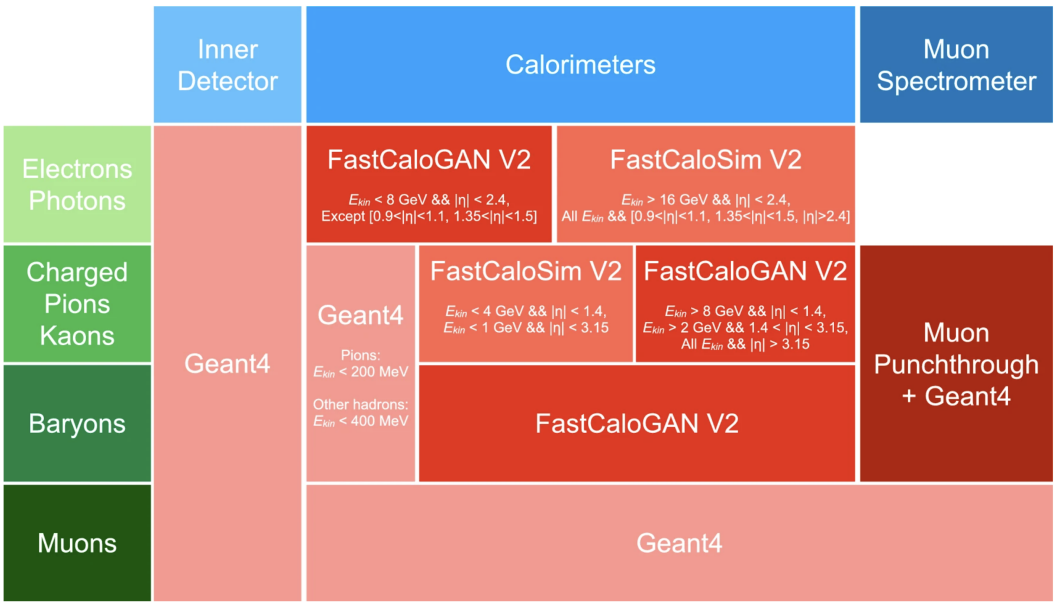


Figura 2.7: *Rappresentazione grafica dei regimi operativi di Geant4, FastCaloSimV2 e FastCaloGANV2 in AtlFast3 in termini di energia cinetica della particella E_k e pseudo-rapidità η.* [32]

è la possibilità di utilizzarlo praticamente per ogni processo fisico, sia di segnale sia di fondo.
Queste caratteristiche sono tutte rilevanti in vista del Run 4 per cui si stima che oltre il 90-95% delle analisi dovrà necessariamente affidarsi alla simulazione veloce [31].

Capitolo 3

Machine learning generativo quantistico

Negli ultimi anni è emerso e sta crescendo rapidamente il quantum machine learning (QML), un nuovo ambito di ricerca in cui si combinano tecniche di machine learning con le potenzialità offerte dalle tecnologie quantistiche [33].
In questo capitolo vedremo prima quali devono essere le caratteristiche di una rete neurale quantistica e vedremo un possibile candidato per la realizzazione sperimentale di questa. Successivamente passeremo in rassegna tre modelli generativi quantistici, fra i più studiati e applicati fino ad ora: le Quantum Circuit Born Machine, le reti neurali avversarie quantistiche e le reti neurali invertibili quantistiche. Verrà poi presentato il paradigma del Reservoir Computing, con la sua controparte quantistica, e infine vedremo come può essere applicato il quantum computing a problemi di HEP

3.1 Reti neurali quantistiche

Fin dagli albori del quantum machine learning, l'idea di trovare una controparte quantistica per le reti neurali (NN) ha mobilitato numerose menti. Per capire al meglio come avviene l'estensione delle NN alle reti neurali quantistiche (QNN) qui viene seguito l'approccio usato in [34], cioè introduciamo un formalismo per le NN e successivamente lo estendiamo alle QNN.
Ogni NN è formata da neuroni, un oggetto che a partire dagli input che gli vengono forniti e dai pesi, produce un output. Qui trattiamo le reti neurali feed forward, cioè quelle reti in cui gli input dei neuroni di uno strato sono gli output dei neuroni dello strato precedente. Il neurone j-esimo dello strato l della rete prenderà in input $a_k^{(l-1)}$ e lo moltiplicherà con il peso $w_{jk}^{(l)}$; l'output dello stesso neurone verrà invece indicato come $a_j^{(l)}$ (Per chiarezza: j indicizza il neurone, l lo strato d'appartenenza e k l'input). Il funzionamento del neurone viene descritto tramite due funzioni, una prima lineare in

cui viene calcolato

$$z_j^{(l)} = \sum_{k=1}^{n} w_{jk}^{(l)} a_k^{(l-1)},$$

supponendo che il neurone abbia n input, e una seconda tipicamente non lineare detta funzione di attivazione. Tra le funzioni di attivazione più comuni abbiamo: ReLU, sigmoide, tangente iperbolica e Heaviside. A titolo esemplificativo consideriamo quest'ultima, è possibile quindi calcolare l'output del neurone

$$a_j^{(l)} = \begin{cases} 1 & \text{se } z_j^{(l)} \geq 0.5 \\ 0 & \text{altrimenti .} \end{cases}$$

Per costruire il corrispettivo quantistico del neurone bisogna necessariamente svolgere due step: render il processo prima reversibile e poi unitario. Per la reversibilità bisogna fare in modo che il numero di input sia uguale al numero di output. Indicando con in_i l'input i-esimo, descriviamo il neurone classico come una funzione $(\text{in}_1, \text{in}_2, \ldots \text{in}_n) \to \text{out}$ e creiamo una porta classica reversibile

$$(\text{in}_1, \text{in}_2, \ldots \text{in}_n, 0) \to (\text{in}_1, \text{in}_2, \ldots \text{in}_n, \text{out}).$$

Tale operazione può essere sempre rappresentata con una matrice di permutazione P. Questa porta classica reversibile deve esser resa una porta quantistica unitaria

$$(|\Psi_{\text{in}}\rangle\,|0\rangle) \to (|\Psi_{\text{out}}\rangle\,|\text{out}\rangle).$$

Per addestrare le NN con il metodo della discesa del gradiente bisogna definire una funzione costo, tipicamente della forma [34]:

$$\mathcal{L} = \frac{1}{2}\left|\left|\boldsymbol{y}^{(L)} - \boldsymbol{a}^{(L)}\right|\right|^2,$$

dove $\boldsymbol{y}^{(L)}$ è il vettore degli output desiderati e $\boldsymbol{a}^{(L)}$ è il vettore degli output effettivi dello strato L. Per minimizzare la funzione costo bisogna modificare i pesi dei neuroni di ogni strato nella direzione opposta a quella del gradiente di $\mathcal{L}$. Ogni peso $w_{jk}^{(l)}$ viene incrementato di

$$\delta w_{jk}^{(l)} = -\eta \frac{\partial \mathcal{L}}{\partial w_{jk}^{(l)}}.$$

Per le QNN si può usare un approccio simile: si ha che ogni unitaria U_N, agente su N qubit, può esser espressa tramite le matrici di Pauli σ_j con $j = 1, 2, 3$, la matrice identità 2×2 σ_0 e 4^N parametri reali $a_{j_1, j_2, \ldots j_N}$:

$$U_N = \exp\left[i \left(\sum_{j_1=0}^{3} \sum_{j_2=0}^{3} \cdots \sum_{j_N=0}^{3} a_{j_1, j_2, \ldots j_N} (\sigma_{j_1} \otimes \sigma_{j_2} \otimes \cdots \otimes \sigma_{j_N}) \right) \right].$$

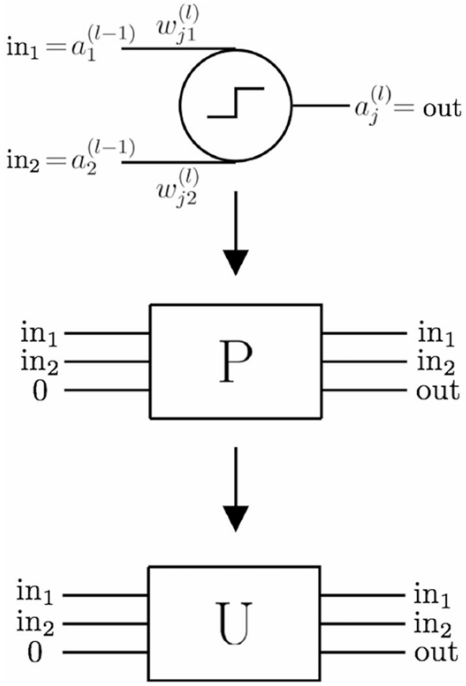


Figura 3.1: *Schema del processo di costruzione del neurone quantistico a partire dal neurone classico irreversibile.* [34]

Il processo di addestramento consiste dunque nel modificare tali parametri con incrementi pari a

$$\delta a_{j_1,j_2,\ldots j_N} = -\eta \frac{\partial \mathcal{L}}{\partial a_{j_1,j_2,\ldots j_N}},$$

in modo tale per cui lo stato del qubit di output sia il risultato aspettato. In sostanza quindi abbiamo che la controparte quantistica del neurone è funzionalmente analoga a un operatore unitario U. Una rappresentazione degli step seguiti per ottenere il neurone quantistico è mostrata in fig. 3.1.

3.2 Circuiti quantistici variazionali

Una classe copiosa e promettente di algoritmi frutto di ricerche per implementare tecniche di QML è costituita dagli algoritmi variazionali quantistici (VQA), tutti messi in pratica tramite i circuiti variazionali quantistici (VQC). A incrementare l'interesse generale è il fatto che i VQC sono ritenuti i candidati più promettenti per essere la controparte quantistica delle reti neurali classiche [35].
Un VQC è un circuito quantistico formato da porte parametrizzate, cioè delle porte

logiche che effettuano una modifica dello stato di un qubit in base al valore di un parametro θ. Le porte parametrizzate più comunemente usate sono quelle rotazionali, le quali permettono di effettuare una rotazione di angolo θ dello stato di un qubit attorno a un determinato asse.
Nelle applicazioni dei VQC si definisce una funzione costo $\mathcal{L}$ che, quando minimizzata, assicura che il nostro circuito riesca a svolgere in modo appropriato il compito per cui è stato creato. Per cui, denotando con $\boldsymbol{\theta}$ i parametri del circuito e con $U(\boldsymbol{\theta})$ il VQC, il processo di ottimizzazione dei parametri può esser descritto come

$$\min_{\boldsymbol{\theta}} \mathcal{L}(U(\boldsymbol{\theta})).$$

Una delle peculiarità dei VQC è che le prestazioni di questi dipendono pesantemente dall'ansatz, cioè dalla specifica struttura del circuito, come mostrato in [36]. Formalmente è possibile scrivere $U(\boldsymbol{\theta})$, che ricordiamo esser unitaria, come il prodotto di L unitarie

$$U(\boldsymbol{\theta}) = U_L(\theta_L) \dots U_2(\theta_2) U_1(\theta_1)$$

in cui indichiamo con θ_i il sottoinsieme i-esimo di $\boldsymbol{\theta}$. La scelta dell'ansatz consiste proprio nel determinare la struttura delle unitarie $U_i(\theta_i)$ in base alle specifiche esigenze del problema, ricordando sempre che ciascuna matrice unitaria può poi esser decomposta in una sequenza di porte, parametrizzate o meno, sui qubit [37]. Sono quindi stati sviluppati i quantum architecture search (QAS) algorithm, cioè degli algoritmi che ottimizzano anche la struttura stessa del VQC [38]. Alla luce di ciò, il processo di ottimizzazione degli algoritmi QAS può esser descritto come

$$\min_{s,\boldsymbol{\theta}} \mathcal{L}(U_s(\boldsymbol{\theta})),$$

denotando con s la struttura del VQC [39]. Lo schema di un circuito variazionale quantistico viene mostrato in fig. 3.2.

Nelle reti neurali classiche è noto che una maggiore profondità della rete, cioè un maggior numero di strati, porta a una migliore espressività [40, 41]. Definendo l'espressività come la capacità di un circuito di generare stati che siano ben rappresentativi dello spazio di Hilbert (Nel caso di un singolo qubit, corrisponde alla capacità del circuito di esplorare tutta la sfera di Bloch [42]), ci potremmo aspettare che anche per i VQC valga una proprietà analoga. Tale prospettiva incontra un limite teorico analizzato in [43], in cui si dimostra che per un'ampia classe di circuiti quantistici parametrizzati si verifica l'effetto Barren Plateau, un fenomeno in cui l'ottimizzazione del circuito si blocca. Precisamente il Barren Plateau è una regione nello spazio dei parametri in cui il gradiente della funzione costo è esponenzialmente piccolo e quindi risulta fondamentalmente piatto, rendendo impossibile l'aggiornamento dei parametri (In [43] si mostra che la probabilità che il gradiente sia diverso da zero diminuisce esponenzialmente con il numero di qubit).

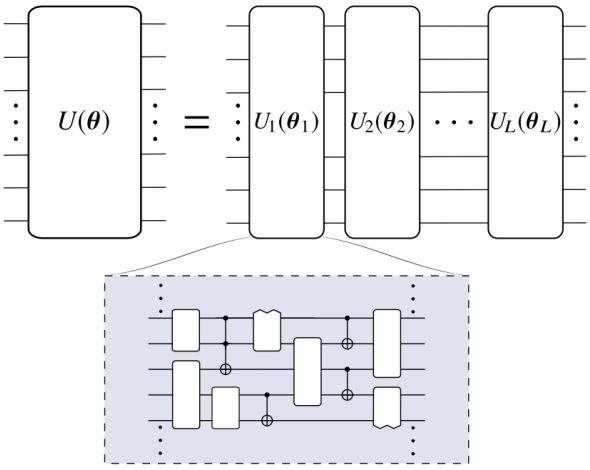


Figura 3.2: *Struttura generale di un VQC. Qui viene mostrata la decomposizione in porte solo di un unico $U_i(\theta_i)$.* [37]

Esistono diversi tipi di Barren Plateau classificati in base alla loro causa d'origine, alcuni di questi vengono trattati in [44, 45, 46], ma sono comunque tutti distruttivi per il processo di ottimizzazione. Questo decadimento del gradiente della funzione costo in funzione del numero di qubit (o della profondità del circuito), che possiamo formalmente esprimere come

$$\nabla \mathcal{L} \sim e^{-n}$$

indicando con n il numero di qubit (o la profondità del circuito), limita di molto la gamma dei possibili ansatz ottimizzabili.

3.3 Modelli generativi quantistici

L'obiettivo dei modelli generativi quantistici, come per la controparte classica, è di allenarsi su dei campioni estratti da una distribuzione data per apprendere tale distribuzione e generare nuovi campioni.
In generale, molti di questi modelli generativi quantistici sono costruiti in modo simile alla loro controparte classica (quando presente), sostituendo alcune parti del modello con metodi quantistici. Nella maggior parte dei casi a esser sostituite sono le reti neurali, rimpiazzate dai VQC di cui si vanno ad ottimizzare i parametri che definiscono le porte rotazionali [37].
Molte architetture sono state proposte per i modelli generativi quantistici nel corso degli anni, qui daremo un'introduzione a: Quantum Circuit Born Machine (QCBM), Quantum Generative Adversarial Network (QGAN) e Quantum Invertible Neural Network (QINN).

3.3.1 Quantum Circuit Born Machine

Le Quantum circuit Born Machine (QCBM) costituiscono un'ampia classe di modelli generativi quantistici non aventi alcun analogo nei modelli classici [47]. In sostanza una QCBM è costituita da un VQC addestrato per essere un modello generativo.
Dato un dataset $D = \{\boldsymbol{y}\}$ costituito da campioni estratti da una distribuzione $\pi(\boldsymbol{y})$, l'obiettivo è costruire una QCBM capace di generare campioni appartenenti alla distribuzione $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ simile alla distribuzione del dataset. La QCBM prende dei qubit, ciascuno nello stato $|0\rangle$, e li modifica tramite l'utilizzo delle porte unitarie di un VQC avente parametri $\boldsymbol{\theta}$, fino ad arrivare allo stato $|\Psi_{\boldsymbol{\theta}}\rangle$. Seguendo la regola di Born e indicando con $|x\rangle$ uno stato della base, il processo di misura dello stato finale produce un campione $\boldsymbol{x}$ con probabilità

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = |\langle \boldsymbol{x}|\Psi_{\boldsymbol{\theta}}\rangle|^2.$$

A titolo esemplificativo, in fig. 3.3 viene mostrato il circuito adoperato nella QCBM in [47].

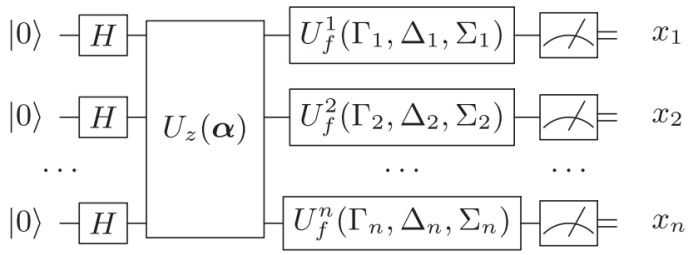


Figura 3.3: *Struttura del VQC utilizzato nella costruzione della QCBM in* [47]. *Per una completa comprensione dei parametri si rimanda all'articolo di provenienza.*

Nella maggior parte delle formulazioni, quando si lavora con un circuito a n qubit, il processo di misura avviene nella base computazionale. In questi casi si avrà quindi $\boldsymbol{x} = (x_1, \ldots, x_n)$ con $x_i \in \{0, 1\}$ e la distribuzione $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ sarà discreta. In [1] viene presentato questo esempio: se si vuole generare un valore discreto per l'energia in $[1, 512]$ sarà necessario usare 9 qubit, infatti $2^9 = 512$. Se invece si volesse generare un'immagine composta da otto pixel, servirebbero $9 \cdot 8 = 72$ qubit. Quindi in generale per generare dati un minimo più complessi con una QCBM sono necessari un elevato numero di qubit. Esiste tuttavia la possibilità di estendere le QCBM per distribuzioni continue. In [48] viene presentata una continuous variable Born machine, costruita su una formulazione alternativa del quantum computing basata sulle variabili continue.
Tipicamente per addestrare il circuito si sceglie un approccio basato sulla discesa del gradiente della funzione costo, definibile diversamente in base al problema specifico trattato. In fig. 3.4 viene mostrato lo schema d'addestramento usato in [49]: il campione $\boldsymbol{x}$

prodotto dal VQC viene confrontato con uno estratto da $\pi(\boldsymbol{y})$ utilizzando una funzione costo di cui ne viene calcolato il gradiente. Un ottimizzatore calcola il $\Delta\boldsymbol{\theta}$, cioè la modifica da effettuare ai parametri del circuito quantistico. Questo processo viene ripetuto fino alla fine dell'addestramento. L'addestramento sarà terminato con successo quando le distribuzioni $\pi(\boldsymbol{y})$ e $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ saranno sufficientemente simili.

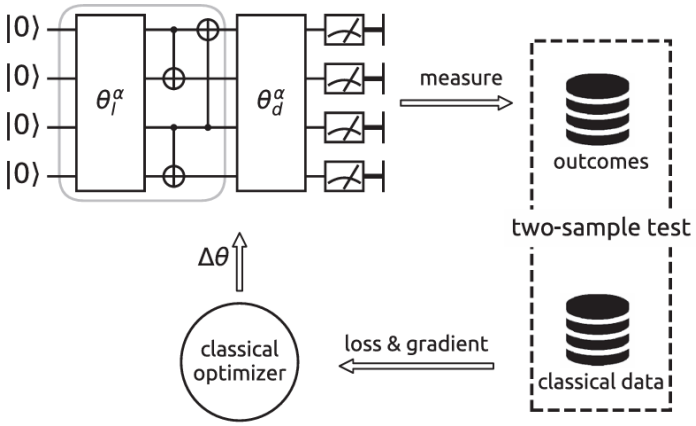


Figura 3.4: *Illustrazione dello schema d'addestramento della QCBM presentato in* [49].

3.3.2 GAN quantistiche

Le reti generative avversarie quantistiche (qGAN), sono il corrispettivo quantistico delle classiche GAN. Proposte inizialmente in [50], in cui si mostrava anche che le qGAN possono mostrare un vantaggio esponenziale rispetto alle classiche GAN in specifici casi di spazi ad alta dimensionalità, in meno di una decade sono state ampiamente studiate e applicate a molti casi d'uso specifici. Le qGAN si possono dividere in due categorie: qGAN ibride classiche-quantistiche, quando alcune parti che costituiscono il modello sono VQC e altre invece reti neurali classiche (tipicamente si sceglie di promuovere il generatore a un'architettura quantistica) e qGAN complete quando sia il generatore che il discriminatore sono dei VQC. Il principio di funzionamento delle qGAN è lo stesso delle GAN classiche tratte in precedenza.
Consideriamo il caso più inedito: una qGAN completa in cui vengono anche usati dei dati quantistici. Introduciamo una etichetta per i dati $|\lambda\rangle$ e supponiamo di avere una sorgente R di dati quantistici che fornisce una matrice densità ρ_λ^R a n qubit, cioè

$$R(|\lambda\rangle) = \rho_\lambda^R.$$

Definiamo il generatore G tramite un VQC, composto da porte parametrizzate dai parametri $\boldsymbol{\theta}_G$, che dovrà imitare la sorgente R. Il compito di G è, presi in input l'etichetta

$|\lambda\rangle$ e uno stato casuale iniziale $|z\rangle$, produrre uno stato quantistico

$$G(\boldsymbol{\theta}_G, |\lambda, z\rangle) = \rho_\lambda^G(\boldsymbol{\theta}_G, z).$$

Tale stato quantistico verrà fornito al discriminatore D, creato anch'esso tramite un VQC avente parametri $\boldsymbol{\theta}_D$. D dovrà capire se lo stato che riceve in input sia stato prodotto dalla sorgente R, in questo caso deve dare in output $|\text{reale}\rangle$, o prodotto da G, restituendo quindi $|\text{falso}\rangle$. Proprio come accadeva nelle GAN classiche, per le qGAN si può definire il processo di ottimizzazione tramite la funzione

$$\min_{\vec{\boldsymbol{\theta}}_G} \max_{\vec{\boldsymbol{\theta}}_D} \frac{1}{\Lambda} \sum_{\lambda=1}^{\Lambda} \Pr\Bigg(\big(D(\vec{\boldsymbol{\theta}}_D, |\lambda\rangle, R(|\lambda\rangle)) = |\text{reale}\rangle\big) \cap \\ \big(D(\vec{\boldsymbol{\theta}}_D, |\lambda\rangle, G(\vec{\boldsymbol{\theta}}_G, |\lambda, z\rangle)) = |\text{falso}\rangle\big), \Bigg)$$

in cui il generatore e il discriminatore si scontrano in una sfida di minimizzazione e massimizzazione [51].
Supponiamo ora di voler usare una qGAN completa ma di dover lavorare con dei dati classici. In questo caso, si apre un'importante questione riguardo a come convertire i dati classici, formati da stringhe di bit, in dati quantistici. Le tecniche di quantum encoding giocano un ruolo fondamentale nel rappresentare in modo appropriato i dati classici all'interno dello spazio di Hilbert, cioè nello spazio degli stati. Tre sono le principali tecniche di encoding [52]:

1. Basis Encoding: in questa tecnica il dato viene prima convertito in intero, poi in binario e ogni cifra binaria viene poi convertita in uno stato di un qubit. Data una stringa x di n bit che rappresenta un unico dato, si ottiene il dato quantistico

   $$|x\rangle = |b_1\rangle \otimes |b_2\rangle \otimes \cdots \otimes |b_n\rangle$$

   in cui b_i sono le cifre binarie.

2. Rotation Encoding: i dati classici x vengono convertiti tramite la rotazione di un qubit. Bisogna prima convertire il dato in un angolo θ, semplicemente riscalandolo $\mathbb{R} \to [0, 2\pi]$ (in [53] è mostrato il processo di encoding a partire da delle energie) e successivamente si codifica θ tramite dei gate rotazionali, ad esempio tramite RX:

   $$\text{RX}(\theta)\,|0\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle - i\sin\left(\frac{\theta}{2}\right)|1\rangle\,.$$

3. Amplitude encoding: i dati vengono codificati tramite l'ampiezza associata degli stati quantistici. Per codificare n dati x_i:

   $$|x\rangle = \frac{x_1}{\sqrt{\sum_{i=1}^n x_i^2}}\,|0\rangle + \frac{x_2}{\sqrt{\sum_{i=1}^n x_i^2}}\,|1\rangle + \cdots + \frac{x_n}{\sqrt{\sum_{i=1}^n x_i^2}}\,|n-1\rangle\,.$$

Come mostrato per amplitude e angle encoding in [54], un risultato estendibile anche per le altre tecniche, il metodo scelto per codificare i dati influenza di molto l'accuratezza di un modello implementato tramite un VQC.

Consideriamo ora una qGAN ibrida (generatore quantistico e discriminatore classico) con dati classici. Questa è fra le configurazioni più comuni, sia perché tecnologicamente più facile da realizzare e da addestrare avendo un unico VQC, sia perché nella stragrande maggioranza dei casi si lavora con dataset classici. Il generatore applicherà le porte parametrizzate a dei qubit fino a ottenere uno stato finale che dovrà subire un processo di misura, producendo quindi una stringa di bit come dato sintetizzato. Il discriminatore in questa configurazione è tipicamente una rete neurale profonda e il processo di ottimizzazione segue lo stesso principio delle GAN classiche e delle qGAN complete. La grande differenza fra le qGAN ibride e quelle complete è che in quest'ultime il discriminatore ha la possibilità di accedere direttamente allo stato dei qubit prodotti dal generatore.

3.3.3 Reti neurali invertibili quantistiche

Il principio dei normalizing flows come generatore può essere usato anche in ambito quantistico implementandolo tramite le Quantum Invertible Neural Network (QINN). Per creare un'architettura basata su un VQC invertibile conviene dividere questo in tre parti (codificazione dei dati, modello con le sue porte parametrizzate e fase di misurazione) e verificare che ciascuna sia invertibile. È necessario utilizzare un'architettura ad hoc per il modello che soddisfi l'invertibilità, come ad esempio quella adoperata in [55]. Supponiamo di avere un dato $x = (x_1, \dots x_n)^T$ e di doverlo codificare in uno stato quantistico $|x\rangle$. Se si vuole costruire una QINN bisogna assicurarsi che tale processo di codifica risulti esser invertibile. Se consideriamo ad esempio l'angle encoding, e supponiamo che sia già avvenuto il riscalamento per cui $x_i \in [0, \pi]$, definiamo l'operatore di preparazione dello stato tramite RY(x):

$$|x\rangle = \mathrm{RY(x_i)}\,|0\rangle^{\otimes n} = \bigotimes_{i=1}^{n} \left(\cos\left(\frac{x_i}{2}\right)|0\rangle + \sin\left(\frac{x_i}{2}\right)|1\rangle\right).$$

Tale processo è chiaramente invertibile, infatti misurando la probabilità P_i che l'i-esimo qubit sia nello stato $|1\rangle$ si ottiene una funzione invertibile

$$P(1)_i = \sin^2\left(\frac{x_i}{2}\right) \implies x_i = 2\arcsin\left(\sqrt{P(1)_i}\right).$$

Tale risultato non è estendibile a tutti i metodi di encoding, ad esempio la normalizzazione presente nell'amplitude encoding impedisce tale invertibilità.
Il comportamento di un VQC è rappresentabile tramite il processo $|x\rangle \mapsto U(\boldsymbol{\theta})\,|x\rangle = |y\rangle$.

Se consideriamo un'architettura a strati formata dall'applicazione sequenziale di porte quantistiche

$$U(\boldsymbol{\theta}) = U_L(\theta_L) \dots U_2(\theta_2) U_1(\theta_1),$$

il modello può essere invertito applicando l'aggiunto a ciascuna unitaria in ordine inverso [55]:

$$U^\dagger(\boldsymbol{\theta}) = U_1^\dagger(\theta_1) \dots U_{L-1}^\dagger(\theta_{L-1}) U_L^\dagger(\theta_L).$$

La fase più delicata è quella della misura dello stato del qubit, che sappiamo essere un processo probabilistico. Indicando con y il risultato misurato, non è direttamente possibile invertire il processo di misurazione tramite una trasformazione g

$$g : y \mapsto |y\rangle \,,$$

infatti molti stati diversi possono portare allo stesso valore d'aspettazione. L'idea per risolvere questa situazione è: fissata una trasformazione g, si addestra il modello in modo tale per cui $|y\rangle$ e $|\widetilde{y}\rangle = g(y)$ siano il più simili possibile. Precisamente si definisce la fedeltà come

$$F = \langle \widetilde{y} |\, y \rangle$$

e i parametri ottimali del VQC saranno quelli per cui $F \simeq 1$. In [55] si mostra che è favorevole (in termini della capacità di generazione del modello) implementare tale vincolo ragionando non sugli stati $|y\rangle$ e $|\widetilde{y}\rangle$, ma sui dati x e $\widetilde{x}$, indicando con $\widetilde{x}$ i dati che quando forniti in input al VQC fanno produrre lo stato $|\widetilde{y}\rangle$. Quindi si definisce una funzione di perdita $\mathcal{L}_{QINN}$ che parametrizzi non solo una generazione in accordo con i dati, ma anche la volontà di aver $x \simeq \widetilde{x}$.

3.4 Reservoir Computing e Quantum Reservoir Computing

L'idea di base del Reservoir Computing (RC) è di fare in modo che parte dello sforzo richiesto a una rete neurale durante la fase di addestramento per compiere una determinata task, venga ridotto usando una rappresentazione intermedia altodimensionale tramite il reservoir. Il reservoir è sostanzialmente un sistema, in alcuni casi fisico, fonte di correlazione caotica e non addestrabile, i cui output diventano gli input di una rete neurale di elaborazione.
Per capirne il funzionamento analizziamo matematicamente un modello composto da una rete neurale formata da un unico layer di lettura in coda a un reservoir: supponiamo di avere un input n-dimensionale $\boldsymbol{u}_{train}(t)$ e di volerlo trasformare in un output

m-dimensionale $\widetilde{\boldsymbol{y}}(t)$. I dati vengono direttamente caricati all'interno del reservoir così che il suo stato $\boldsymbol{X}$ al tempo t_i sia

$$\boldsymbol{X}(t_i) = \boldsymbol{f}(\boldsymbol{X}(t_{i-1}), \boldsymbol{u}(t_i))$$

in cui $\boldsymbol{f}$ è una funzione. Un insieme sufficientemente grande di variabili di stato del reservoir $\boldsymbol{x}(t)$ deve rimanere accessibile. Infine il layer di lettura, qui indicato con $\boldsymbol{F}$, mappa gli elementi di $\boldsymbol{x}(t)$ nell'output $\boldsymbol{y}(t)$:

$$\boldsymbol{F}(\boldsymbol{x}(t)) = \boldsymbol{y}(t).$$

Il layer di lettura è l'unico elemento che viene modificato durante la fase di addestramento. In particolare si definisce una funzione di perdita da minimizzare tipicamente dipendente dalla differenza fra $\boldsymbol{y}(t)$ e $\widetilde{\boldsymbol{y}}(t)$.
Uno dei modi più comunemente usati per realizzare un reservoir è usare una rete neurale ricorrente (RNN) caratterizzata dal fatto che qui l'informazione può anche tornare indietro. Questa caratteristica permette al sistema di possedere una memoria: se un reservoir è creato per processare un segnale $\boldsymbol{u}(t)$, per valutare $\boldsymbol{u}(t_2)$ può esser necessario utilizzare l'informazione di $\boldsymbol{u}(t_1)$ (con $t_1 < t_2$). Tale informazione può allora esser resa accessibile proprio propagando indietro il segnale. Da ciò è facile capire perché il RC è un paradigma particolarmente adatto per compiti di predizione in vari istanti temporali. È lecito ora chiedersi: come può un meccanismo di questo tipo esser utile anche per alcuni compiti indipendenti dal tempo? Ad un primo sguardo sembra essere più utile rendere anche i neuroni che formano la RNN addestrabili. La risposta sta nello sfruttare la grande dimensionalità del reservoir. L'informazione a cui siamo interessati, codificata all'interno del segnale di input, può esser mischiata (anche in modo non lineare) con del rumore che noi vorremmo poter distinguere e non considerare. La rappresentazione altodimensionale intermedia facilita proprio questo compito. In [56] è presentato il seguente esempio: supponiamo di voler classificare degli input, che sono delle semplici forme, in stelle e cerchi tuttavia, per come i dati sono distribuiti nel piano x e y, è impossibile trovare un iperpiano (in questo caso una linea) che permetta la separazione fra gli elementi. Se mappiamo i dati in uno spazio tridimensionale, è possibile allora che si crei la possibilità di separare le stelle e i cerchi tramite un piano (fig. 3.5). Si è quindi reso il problema di separazione molto più semplice sfruttando una rappresentazione altodimensionale praticamente ottenibile tramite un reservoir.

Com'è stato accennato sopra, una delle possibili realizzazioni di un reservoir è attraverso una RNN ma è stata esplorata negli anni anche la possibilità di utilizzare dei sistemi fisici come [57]: sistemi dinamici non lineari, circuiti elettronici con particolari componenti, sistemi fotonici, sistemi in cui viene sfruttata la spintronica, sistemi biologici (come cellule in vitro), ecc.

A seguito della pubblicazione dell'articolo [58] nel 2017 una nuova branca del RC è nata: il Quantum Reservoir Computing (QRC) in cui si fonde il quantum computing

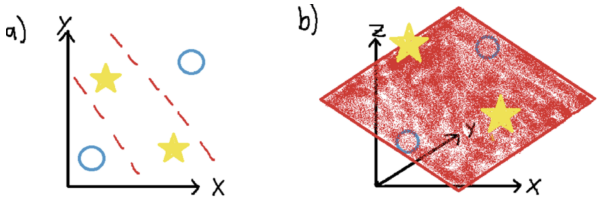


Figura 3.5: *In a la rappresentazione dei dati nel piano xy. In b la rappresentazione dei dati in uno spazio a dimensione più alta, cioè in xyz.* [56]

con il reservoir computing realizzato tramite sistemi fisici. Noi ci soffermeremo su come è possibile utilizzare un sistema composto da qubit come reservoir per compiti non dipendenti dal tempo (trascuriamo quindi le potenzialità offerte anche per la risoluzione di compiti dipendenti dal tempo), infatti il modello generativo proposto in questa tesi si applica proprio a questo tipo di task. La trattazione sotto proposta è tratta da [59].
Supponiamo di avere un sistema composto da n-qubit tutti inizializzati in $|0\rangle^{\otimes n}$, che faranno da reservoir, e una rete di lettura finale composta da un solo layer. Sia $\{x^{(j)}, \widetilde{y}^{(j)}\}$ un insieme di input e soluzioni, per semplicità unidimensionali, con $x \in [0,1]$ (utilizzo j per indicizzare i vari elementi del dataset).
Per inserire un dato $x^{(k)}$, che in seguito indichiamo solo con x, all'interno del sistema quantistico si utilizza un'operazione unitaria parametrizzata $V(x)$ che viene applicata allo stato iniziale:

$$V(x)\,|0\rangle^{\otimes n}\,.$$

Ad esempio ciò è realizzabile utilizzando una rotazione rispetto l'asse y di angolo $\theta = \arccos(\sqrt{x})$:

$$R_y(\theta)\,|0\rangle = e^{-i\theta Y}\,|0\rangle = \sqrt{x}\,|0\rangle + \sqrt{1-x}\,|1\rangle\,,$$

in cui Y è una delle tre matrici di Pauli X, Y, Z. Il valore di aspettazione di Z sarà

$$\langle Z\rangle = |\sqrt{x}|^2 - |\sqrt{1-x}|^2 = 2x - 1$$

e dipende linearmente dal dato x. Tuttavia se applichiamo questa operazione per ogni qubit, otteniamo uno stato

$$\begin{aligned} V(x)\,|0\rangle^{\otimes n} &= \left(\sqrt{x}\,|0\rangle + \sqrt{1-x}\,|1\rangle\right)^{\otimes n} \\ &= (1-x)^{n/2} \sum_{i_1,\ldots,i_n} \prod_k \left[\frac{x}{1-x}\right]^{\frac{i_k}{2}} |i_1,\ldots,i_n\rangle\,. \end{aligned}$$

Possiamo allora notare che il valore medio dell'operatore Z_k, cioè l'operatore Z applicato al k-esimo qubit, è di nuovo $2x-1$. Invece, valutando l'operatore $Z_1 Z_2$, otteniamo

$$\langle Z_1 Z_2\rangle = (2x-1)^2$$

che non dipende più linearmente dall'input x. Se misuriamo quindi l'output di un circuito quantistico dopo che è stato applicato un generico operatore unitario U,

$$UV(x)\left|0\right\rangle^{\otimes n},$$

si può ottenere un risultato che non dipende linearmente dall'input. U dovrebbe essere scelto in modo da ottenere un risultato abbastanza complesso ma considerando comunque la realizzabilità sperimentale. Indicando con z_i i risultati della misura dell'operatore Z_i per ogni qubit, questi saranno l'input per il layer di lettura finale che permetterà di ricavare

$$y = \sum_{i=1}^{n} w_i z_i.$$

Allora i pesi della rete w_i verranno scelti in modo tale da minimizzare la funzione

$$L = \sum_{j} ||y^{(j)} - \widetilde{y}^{(j)}||^2.$$

3.5 Quantum computing per la fisica delle alte energie

Negli ultimi anni la comunità della fisica delle alte energie (HEP) ha iniziato a guardare con crescente interesse al quantum computing, cioè alla possibilità di affrontare con hardware quantistico alcuni dei problemi computazionalmente più onerosi del settore. Tale interesse nasce da una doppia motivazione. Da un lato, sul piano teorico, esistono modelli che restano di fatto inaccessibili alle simulazioni Monte Carlo classiche a causa del cosiddetto sign problem, cioè il comportamento fortemente oscillante dell'integrale di cammino che rende il tempo di campionamento esponenziale nel numero di siti reticolari [60]. Dall'altro lato, sul piano sperimentale, esperimenti come ATLAS generano una mole di dati e richiedono una potenza di calcolo tali per cui bisogna necessariamente esplorare paradigmi computazionali alternativi a quelli classici.
Per queste ragioni, la comunità HEP si è organizzata attorno a iniziative dedicate, fra cui il QC4HEP Working Group, promosso da CERN, DESY e IBM, che riunisce numerosi ricercatori con l'obiettivo di tracciare una roadmap condivisa delle applicazioni più promettenti nel breve termine [60]. In modo analogo, la Quantum Technology Initiative del CERN (QTI), lanciata nel 2020, struttura la propria attività attorno a tre assi principali: il quantum computing ibrido (cioè l'utilizzo combinato di risorse classiche e quantistiche), il quantum sensing, rivolto principalmente allo sviluppo di nuove tecnologie per i rivelatori, e il quantum networking, per la comunicazione e sincronizzazione fra infrastrutture [61]. Nel caso del quantum computing ibrido, le applicazioni vengono

generalmente suddivise in due grandi categorie: da una parte i metodi teorici per la modellizzazione di problemi fisici, dall'altra i metodi numerici per l'interpretazione dei dati sperimentali [60]. Nei paragrafi seguenti ne diamo una breve panoramica.

Quantum computing per problemi teorici

Sul versante teorico, l'applicazione più naturale del quantum computing alla HEP riguarda la simulazione delle teorie di gauge reticolari (Lattice Gauge Theories, LGT), cioè la discretizzazione su reticolo delle teorie di campo che descrivono le interazioni fondamentali. L'approccio classico basato sull'integrale di cammino soffre del sign problem non appena si tenta di descrivere fenomeni dinamici in tempo reale e sistemi ad alta densità fermionica. Una via alternativa consiste nel riformulare la teoria nel formalismo hamiltoniano anziché lagrangiano, in questo modo il tempo resta una variabile continua e l'operatore di evoluzione temporale è ben definito per qualunque intervallo. Il prezzo da pagare è di dover immagazzinare l'intera funzione d'onda a molti corpi del sistema (un costo che cresce esponenzialmente con la dimensione del reticolo) ma questo è esattamente il tipo di problema per cui un simulatore quantistico risulterebbe vantaggioso [60].
Accanto ai simulatori quantistici veri e propri, in [60] viene sottolineato il ruolo delle Tensor Network (TN) come strategia numerica complementare, capace di descrivere efficacemente stati reticolari all'equilibrio e, in una certa misura, anche la dinamica fuori equilibrio. Diversi approcci ibridi propongono di combinare la rappresentazione tramite TN con circuiti quantistici, così da sfruttare i vantaggi di entrambi i paradigmi.
Un caso applicativo concreto riguarda le oscillazioni dei neutrini: tale sistema si presta naturalmente a essere descritto tramite un'Hamiltoniana di spin costruita a partire dalle matrici di Pauli, in cui un primo termine descrive le oscillazioni di vuoto fra autostati di sapore, un secondo l'interazione con il fondo di elettroni e un terzo, di natura collettiva, l'interazione neutrino-neutrino [60]. È un buon esempio di come problemi di fisica dei neutrini, difficili da trattare con i metodi Monte Carlo classici proprio a causa delle correlazioni collettive fra particelle, si prestino invece a essere naturalmente rimappati su un registro di qubit.

Machine learning quantistico per l'analisi dei dati

Sul versante sperimentale, l'ambito più esplorato resta quello del Quantum Machine Learning (QML) applicato ai dati di collisione, con l'obiettivo di migliorare le prestazioni degli algoritmi classici già impiegati per la classificazione, la ricostruzione di traiettorie e getti e l'individuazione di segnali rari.
Per la classificazione, un esempio interessante riguarda la misura della frazione di eventi

longitudinalmente polarizzati nello scattering di bosoni vettori (VBS), resa difficile dal fatto che tali eventi rappresentano solo una piccola frazione (circa il 7%) del totale previsto dal Modello Standard. In questo contesto, algoritmi variazionali quantistici (VQA) e Support Vector Machine quantistiche (QSVM) sono stati confrontati con le tecniche classiche tradizionalmente usate, mostrando risultati competitivi [60].
Un secondo filone riguarda la ricostruzione di traiettorie e getti: il tracking, in particolare, rappresenta oggi uno dei compiti computazionalmente più stressanti dal momento che la molteplicità dei possibili candidati traccia cresce in modo quadratico o cubico con il numero di hit nel rivelatore, motivo per cui la comunità sta valutando approcci quantistici sia globali (basati sul clustering dell'insieme dei punti) sia locali [60].
Bisogna infine menzionare l'anomaly detection, cioè la ricerca di eventi non compatibili con il Modello Standard senza assumere a priori un particolare modello di nuova fisica. È interessante notare come, in uno degli studi citati in [60], un Anomaly QGAN riesca a raggiungere la stessa accuratezza del corrispettivo classico utilizzando fino a dieci volte meno dati di addestramento. Tale risultato suggerisce come le proprietà dei modelli quantistici possano tradursi in un concreto vantaggio in termini di efficienza nell'uso dei dati di addestramento.
Va detto che tutti questi approcci condividono una difficoltà di fondo, cioè la codifica dei dati classici in un circuito quantistico: un problema tutt'altro che secondario, dato che proprio la scelta della codifica è riconosciuta come una delle cause principali dell'insorgere di Barren Plateau [60].

Machine learning generativo quantistico per la simulazione

Come visto nei capitoli precedenti, la simulazione Monte Carlo degli eventi rilevati dagli esperimenti rappresenta uno dei compiti più onerosi dal punto di vista computazionale in HEP. Accanto alle tecniche di simulazione veloce classiche, basate su approcci parametrici o su modelli generativi che imparano a riprodurre direttamente la risposta del rivelatore, si è iniziato a esplorare controparti quantistiche o ibride di questi stessi modelli [60]. A prescindere dal preciso modello utilizzato, le applicazioni esplorate si dividono in due categorie principali: la simulazione diretta della risposta del rivelatore, come la generazione degli shower calorimetrici, e la generazione di variabili di alto livello nello stato finale, come ad esempio i quadrimpulsi delle particelle prodotte.
Il limite principale degli approcci attuali è la dimensionalità: i modelli quantistici oggi disponibili producono simulazioni accurate solo per configurazioni molto piccole, dell'ordine della decina di variabili, tipicamente utilizzando un qubit per ogni sensore del rivelatore. Per aggirare tale limite si ricorre solitamente a tecniche di compressione dei dati come l'utilizzo di autoencoder classici: una rete riduce il dato originale a una rappresentazione latente di dimensione gestibile, che viene poi appresa dal modello generativo quantistico, mentre un secondo decoder classico riporta l'output sintetizzato alla dimen-

sione originale [60]. Un ulteriore ostacolo è legato alla discretizzazione: i qubit codificano naturalmente informazioni discrete, per cui la dimensione del sistema quantistico incide direttamente sulla risoluzione della simulazione ottenuta. Proprio per questo la generazione di variabili di alto livello, i cui gradi di libertà sono dell'ordine della decina anziché dei milioni di canali di un intero rivelatore, risulta oggi un obiettivo più abbordabile per i dispositivi quantistici attualmente disponibili.
Un traguardo concreto in questo senso è la cosiddetta IBM 100×100 challenge: l'obiettivo, annunciato da IBM, è quello di rendere disponibile un chip capace di calcolare osservabili, senza bias sistematico introdotto dal rumore, su circuiti da 100 qubit e 100 strati di porte logiche, in un tempo ragionevole, dell'ordine di un giorno [60]. Se raggiunto, un simile traguardo permetterebbe simulazioni di rivelatori sensibilmente più realistiche di quelle possibili oggi.
Per quanto riguarda l'addestramento, come già accennato in precedenza, anche in questo contesto ci si affida alla minimizzazione di una funzione costo che misuri la distanza fra la distribuzione generata e quella target. Le scelte più comuni includono la divergenza di Kullback-Leibler, quella di Jensen-Shannon, la divergenza di Rényi (nella sua versione quantistica) e la Maximum Mean Discrepancy (MMD) [60]. Non si tratta però di una scelta neutra: in [62] viene mostrato che, per le QCBM, l'utilizzo di funzioni costo esplicite come la divergenza di KL introduce una nuova forma di Barren Plateau, mentre la MMD, essendo una funzione costo implicita legata al valore d'aspettazione di un osservabile locale, risulta in generale più facilmente addestrabile. Tale risultato, unito al problema della codifica dei dati, suggerisce come l'insorgere del Barren Plateau abbia un impatto diretto e concreto sulle scelte pratiche di progettazione dei modelli generativi quantistici.
In sostanza, se da un lato i risultati ottenuti finora sono incoraggianti, dall'altro bisogna essere consapevoli dei limiti attuali della tecnologia. Non esistono ancora dimostrazioni definitive e sistematiche di un reale vantaggio quantistico applicato a casi di interesse per la HEP e l'approccio più promettente nel breve termine resta quello ibrido, in cui componenti classiche e quantistiche vengono combinate per sfruttare al meglio i rispettivi punti di forza [61].

Capitolo 4

ResQNN: un possibile approccio basato sul QRC per la simulazione veloce dei calorimetri

In questo capitolo viene presentato il modello generativo sviluppato in questa tesi: un modello ibrido classico-quantistico basato sul paradigma del Quantum Reservoir Computing, chiamato ResQNN.
Nella prima parte di questo capitolo vengono presentati il dataset su cui è stato sviluppato il modello, fornito dalla CaloChallenge, e la struttura generale del modello. Successivamente vengono descritte le due parti che lo compongono, l'energy model e lo shape model, e infine viene mostrato come queste vengono addestrate e utilizzate per generare nuovi shower. L'ultima sezione sarà invece dedicata a presentare e commentare i risultati ottenuti a seguito di alcuni test volti sia a comprendere alcune scelte architetturali del modello sia a valutare la capacità di ResQNN di generare shower calorimetrici confrontabili con quelli di Geant4.

4.1 Il dataset della CaloChallenge

La CaloChallenge [63] è una sfida, lanciata nel 2022, con l'obiettivo di stimolare lo sviluppo di modelli generativi per la simulazione veloce dei calorimetri e di confrontarli in modo omogeneo. A tale scopo sono stati messi a disposizione tre dataset di complessità crescente, costituiti da shower simulati tramite Geant4, e un insieme comune di metriche con cui valutare i modelli.
Il modello presentato in questa tesi è stato sviluppato sul primo di questi, il dataset 1, che deriva dalla simulazione del calorimetro di ATLAS utilizzata per l'addestramento dei modelli generativi di AtlFast3 [11]. Tale dataset è formato da due parti, una di fotoni e una di pioni carichi. Solo quest'ultima parte del dataset è stata usata durante questa

tesi. Il file utilizzato contiene 120800 eventi e ogni evento è costituito dall'energia della particella incidente E_{inc} e dalle energie depositate in 533 voxel. Le energie incidenti non sono distribuite in modo continuo ma assumono solo 15 valori discreti, le potenze di 2 da 2^8 MeV a 2^{22} MeV, cioè da 256 MeV a circa 4.2 TeV. Per brevità nel seguito queste energie verranno indicate con valori troncati, ad esempio 32 GeV per 2^{15} MeV = 32768 MeV.
In ciascun layer un voxel è individuato da un intervallo nella distanza radiale r dall'asse di propagazione della particella e da un intervallo nell'angolo α attorno ad esso. Per i pioni vengono utilizzati sette layer selezionati da quelli del calorimetro di ATLAS: secondo la numerazione di ATLAS, vengono utilizzati i layer 0, 1, 2, 3, 12, 13 e 14. Come si può vedere in table 4.1, la segmentazione varia molto da layer a layer: si passa da layer suddivisi solo radialmente, come i layer 0, 3 e 14, a layer con 10 suddivisioni angolari e fino a 16 radiali.

Layer di ATLAS	Bin radiali	Bin angolari	Voxel
0	8	1	8
1	10	10	100
2	10	10	100
3	5	1	5
12	15	10	150
13	16	10	160
14	10	1	10
Totale			533

Tabella 4.1: *Segmentazione in voxel dei sette layer utilizzati per i pioni nel dataset 1 della CaloChallenge.*

Vengono ora introdotte le notazioni utilizzate nel resto del capitolo. Si indica con $L = 7$ il numero di layer, con N_l il numero di voxel del layer l e con $\boldsymbol{x}_l \in \mathbb{R}^{N_l}$ il vettore delle energie depositate nei suoi voxel. L'energia depositata nel layer l è allora

$$E_l = \sum_{i=1}^{N_l} x_{l,i},$$

mentre il modo in cui tale energia si distribuisce all'interno del layer è descritto dalla forma normalizzata

$$\boldsymbol{p}_l = \frac{\boldsymbol{x}_l}{E_l}, \qquad \sum_{i=1}^{N_l} p_{l,i} = 1.$$

4.2 Struttura generale di ResQNN

L'obiettivo del modello ResQNN è generare la distribuzione congiunta dei 533 voxel condizionata all'energia incidente, cioè $p(\boldsymbol{x} \mid E_{\text{inc}})$. L'idea alla base di ResQNN è di sfruttare la struttura fisica dello shower, dividendo la generazione in due stadi: prima si generano le energie depositate in ciascun layer, poi si genera il modo in cui tale energia si distribuisce fra i voxel del layer. Formalmente, con le notazioni introdotte sopra,

$$p(\boldsymbol{x} \mid E_{\text{inc}}) = p(E_1, \ldots, E_L \mid E_{\text{inc}}) \prod_{l=1}^{L} p(\boldsymbol{p}_l \mid E_l, E_{\text{inc}}).$$

Il primo fattore è affidato all'energy model, il secondo allo shape model. Da notare che il prodotto sui layer rappresenta un'ipotesi del modello: fissate le energie dei layer, le forme dei diversi layer vengono generate indipendentemente l'una dall'altra. Una volta generati entrambi, lo shower viene ricostruito semplicemente come

$$\boldsymbol{x}_l = E_l \, \boldsymbol{p}_l, \qquad l = 1, \ldots, L.$$

La struttura complessiva del modello è schematizzata in fig. 4.1.
Entrambi gli stadi sono costruiti a partire dallo stesso elemento, che verrà chiamato blocco ibrido, formato da un encoder classico, da un reservoir quantistico e da un decoder classico, che verrà descritto in dettaglio nel seguito. Per ora è sufficiente sottolineare che un blocco ibrido riceve in input un vettore di condizionamento e un rumore latente e restituisce un vettore di numeri reali che viene poi interpretato in modo diverso nei due stadi (energy model e shape model). In totale ResQNN è formato da $L + 1 = 8$ blocchi ibridi, ciascuno con il proprio encoder, il proprio decoder e il proprio reservoir:

- energy model: è formato da un unico blocco ibrido, che riceve come condizionamento l'energia incidente e un rumore latente $\boldsymbol{\xi}_E$, e restituisce, dopo un'opportuna trasformazione del suo output, le energie totali di tutti i sette layer $E_1, \ldots, E_L$;
- shape model: è formato da $L = 7$ blocchi ibridi, uno per ciascun layer del calorimetro. Il blocco relativo al layer l riceve come condizionamento l'energia incidente, l'energia E_l depositata nel layer e un indice che identifica il layer, insieme a un proprio rumore latente $\boldsymbol{\xi}_l$, e restituisce la forma normalizzata $\boldsymbol{p}_l$ di quel solo layer, cioè un vettore di N_l elementi.

I rumori latenti $\boldsymbol{\xi}_E, \boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_L$ vengono estratti indipendentemente l'uno dall'altro per ogni shower generato e sono l'unica sorgente di casualità del modello addestrato: a parità di energia incidente, shower diversi differiscono solo per le diverse estrazioni dei rumori. Il fatto che i rumori dei blocchi dello shape model siano indipendenti, insieme al fatto che ciascun blocco veda solo l'energia del proprio layer, è proprio ciò che realizza l'ipotesi di fattorizzazione vista sopra: fissate le energie dei layer, la forma generata in

un layer non dipende in alcun modo da quelle generate negli altri.
Oltre che dall'energia incidente, comune a tutti i blocchi, i due stadi sono collegati unicamente dalle energie dei layer: le energie E_l prodotte dall'energy model entrano sia nel condizionamento dei blocchi dello shape model, sia nella ricostruzione dello shower. Infine i sette vettori $\boldsymbol{x}_l$ vengono concatenati nello stesso ordine del dataset, ottenendo lo shower completo di 533 voxel. Come si vedrà, questa struttura permette anche di addestrare i due stadi separatamente.

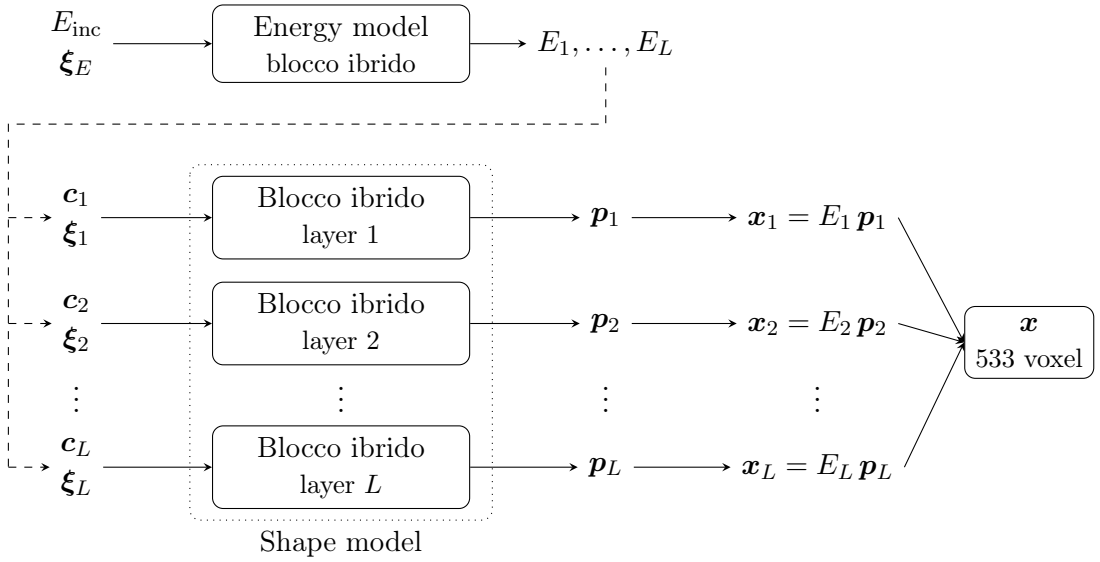


Figura 4.1: *Struttura generale di ResQNN. L'energy model è formato da un unico blocco ibrido che, a partire dall'energia incidente e da un rumore latente $\boldsymbol{\xi}_E$, genera le energie $E_1, \ldots, E_L$ dei sette layer. Lo shape model è formato da un blocco ibrido per ogni layer: il blocco del layer l riceve il condizionamento $\boldsymbol{c}_l$, che contiene l'energia incidente, l'indice del layer e l'energia E_l appena generata (linee tratteggiate), insieme a un proprio rumore latente $\boldsymbol{\xi}_l$, e genera la forma normalizzata $\boldsymbol{p}_l$. Lo shower di ogni layer è ricostruito come $\boldsymbol{x}_l = E_l\,\boldsymbol{p}_l$ e i sette layer vengono infine concatenati nello shower completo. A seconda della configurazione scelta, il rumore di ciascun blocco entra prima o dopo il reservoir.*

Il blocco ibrido

Sia l'energy model che lo shape model svolgono il proprio compito generativo tramite un elemento chiamato "blocco ibrido", formato da un encoder classico, da un reservoir quantistico e da un decoder classico. Il blocco realizza in sostanza lo schema del reservoir quantistico visto nel capitolo precedente, con due differenze: i dati non vengono caricati direttamente nel reservoir ma passano prima da un encoder addestrabile, e il layer di

lettura finale è sostituito da una rete neurale.
Precisamente, dato un vettore di condizionamento $\boldsymbol{c}$, l'encoder $g_{\boldsymbol{\phi}}$, una rete neurale feed forward con due strati nascosti e funzione di attivazione SiLU, lo trasforma in n angoli, uno per ciascun qubit del reservoir:

$$\boldsymbol{\theta} = \pi\, \sigma(g_{\boldsymbol{\phi}}(\boldsymbol{c})) \in [0, \pi]^n,$$

dove σ indica la funzione sigmoide, applicata a ogni componente. Limitare gli angoli all'intervallo $[0, \pi]$ ha un motivo preciso: dopo una rotazione $\mathrm{RX}(\theta)$ il valore d'aspettazione di Z è $\cos\theta$, che per la periodicità delle rotazioni assume lo stesso valore in angoli diversi (ad esempio in θ e $2\pi - \theta$). Sull'intervallo $[0, \pi]$ invece $\cos\theta$ è monotono, quindi due angoli diversi sul singolo qubit danno sempre misure di Z diverse. Gli angoli vengono caricati tramite rotation encoding, applicando a ogni qubit una rotazione attorno all'asse x:

$$V(\boldsymbol{\theta})\,|0\rangle^{\otimes n} = \bigotimes_{i=1}^{n} \mathrm{RX}(\theta_i)\,|0\rangle\,.$$

Segue il reservoir vero e proprio, un circuito U_{res} formato da D strati con la stessa struttura: in ciascuno, ad ogni qubit vengono applicate in sequenza le rotazioni RX, RZ e RX, seguite da un anello di porte CZ fra qubit adiacenti, che comprende anche la coppia formata dall'ultimo e dal primo qubit (fig. 4.2). Le rotazioni generano la dinamica del singolo qubit, mentre le porte CZ creano entanglement fra i qubit, così che ogni misura finale dipenda da tutti gli angoli in input.

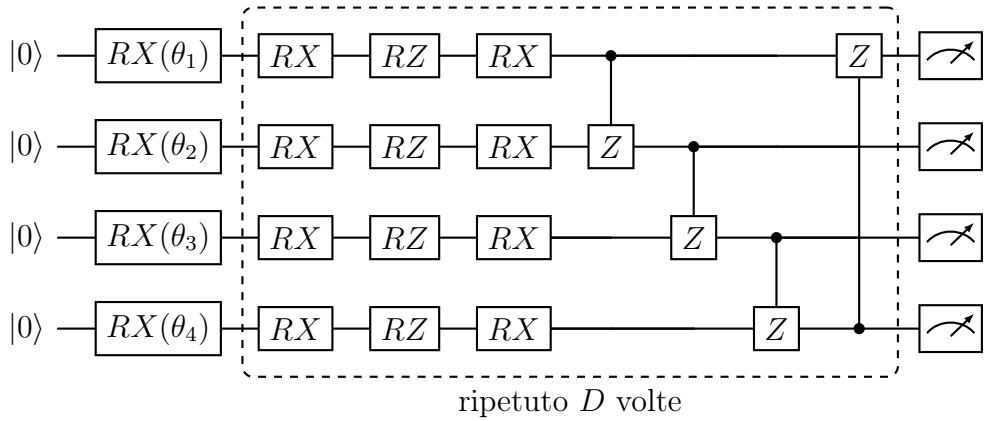


Figura 4.2: *Circuito del reservoir, qui con $n = 4$ qubit. Gli angoli θ_i prodotti dall'encoder vengono caricati tramite porte* RX. *Nel riquadro tratteggiato è mostrato uno strato del reservoir: le rotazioni hanno angoli fissi, estratti casualmente alla costruzione del modello, e sono seguite da un anello di porte* CZ. *Al termine si misura il valore d'aspettazione di Z su ogni qubit.*

L'elemento caratterizzante del reservoir è che gli angoli delle rotazioni, $3nD$ in totale, vengono estratti una sola volta in modo uniforme in $[0, 2\pi)$, al momento della costruzione del modello, e non vengono mai modificati durante l'addestramento. Lo stato finale del circuito è

$$|\psi(\boldsymbol{c})\rangle = U_{\text{res}}\, V(\boldsymbol{\theta})\, |0\rangle^{\otimes n}\,,$$

di cui viene misurato il valore d'aspettazione dell'operatore Z su ogni qubit:

$$z_i = \langle\psi(\boldsymbol{c})|\, Z_i\, |\psi(\boldsymbol{c})\rangle\,, \qquad i = 1, \ldots, n.$$

Il vettore $\boldsymbol{z} \in [-1, 1]^n$, che è la rappresentazione del condizionamento prodotta dal reservoir, viene infine elaborato dal decoder $h_{\boldsymbol{\omega}}$, anch'esso una rete neurale con due strati nascosti. Infine l'output del blocco ibrido è $\boldsymbol{y} = h_{\boldsymbol{\omega}}(\boldsymbol{z})$.
In sostanza quindi gli unici parametri addestrabili del blocco sono i pesi dell'encoder e del decoder, $\boldsymbol{\phi}$ e $\boldsymbol{\omega}$, mentre il circuito quantistico resta invariato. Da notare però che, affinché l'encoder possa essere addestrato, il gradiente della funzione costo deve comunque attraversare il reservoir: è necessario cioè calcolare la derivata dei valori d'aspettazione z_i rispetto agli angoli d'ingresso θ_i.
Il modello è stato implementato in PyTorch, con la parte quantistica simulata tramite PennyLane [64] su un simulatore a vettore di stato. In questo modo i valori d'aspettazione e i loro gradienti possono essere calcolati in modo esatto, senza il rumore statistico che deriverebbe da un numero finito di misure.

Il rumore latente

Il blocco così descritto è una funzione deterministica: a parità di condizionamento produce sempre lo stesso output. Per ottenere un modello generativo è necessario introdurre una sorgente di casualità, data qui da un vettore di rumore latente $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \sigma_\xi^2\, \mathbb{I}_d)$ estratto indipendentemente per ogni campione. Generare più eventi a parità di condizionamento significa quindi valutare il modello su diverse estrazioni di $\boldsymbol{\xi}$.
Sono state implementate due possibili scelte per il punto in cui il rumore entra nel blocco ibrido:

- pre: il rumore viene concatenato al condizionamento, $\boldsymbol{c} \to (\boldsymbol{c}, \boldsymbol{\xi})$, e attraversa quindi il reservoir;
- post: il rumore viene concatenato all'output del reservoir, cioè $\boldsymbol{y} = h_{\boldsymbol{\omega}}(\boldsymbol{z}, \boldsymbol{\xi})$, e il reservoir vede solo il condizionamento fisico.

La differenza non è solo formale. Nella configurazione post il circuito va valutato una sola volta per condizionamento: se si vogliono generare M campioni con lo stesso condizionamento, gli M rumori vengono combinati con lo stesso vettore $\boldsymbol{z}$ e il costo quantistico

non cresce con M. Nella configurazione pre invece ogni campione richiede una valutazione completa del circuito. Come si vedrà, tale vantaggio diventa determinante durante l'addestramento dell'energy model, la cui funzione costo richiede proprio di generare più campioni per ogni evento. Il prezzo da pagare è che nella configurazione post tutta la variabilità dei campioni generati viene prodotta dal decoder classico.

4.3 L'energy model

L'energy model ha il compito di generare, data l'energia incidente, le energie $E_1, \ldots, E_L$ depositate nei sette layer. È costituito da un unico blocco ibrido, il cui condizionamento è la sola energia incidente, opportunamente riscalata:

$$c = \frac{\log(1 + E_{\text{inc}})}{s_E}.$$

Il logaritmo comprime gli oltre quattro ordini di grandezza coperti dalle energie incidenti, mentre la divisione per la costante $s_E = 12$ porta il condizionamento a valori dell'ordine dell'unità, precisamente nell'intervallo [0.46, 1.27], più adatti come input di una rete neurale.
L'output del decoder è un vettore di $L + 1$ numeri reali $(a_0, a_1, \ldots, a_L)$, che non viene interpretato direttamente come un insieme di energie ma trasformato secondo una fattorizzazione suggerita dalla fisica dello shower. Il primo elemento determina la frazione visibile, cioè il rapporto fra l'energia complessivamente depositata e l'energia incidente:

$$v = v_{\max}\, \sigma(a_0),$$

mentre gli altri determinano come tale energia si ripartisce fra i layer, tramite la funzione softmax:

$$f_l = \frac{e^{a_l}}{\sum_{k=1}^{L} e^{a_k}}, \qquad l = 1, \ldots, L.$$

Le energie dei layer sono infine

$$E_l = E_{\text{inc}}\; v\; f_l.$$

Questa costruzione ha alcune proprietà che conviene sottolineare. Le energie generate sono sempre positive, come deve essere. Inoltre, poiché le frazioni f_l sommano a uno per costruzione, l'energia totale depositata è

$$E_{\text{dep}} = \sum_{l=1}^{L} E_l = E_{\text{inc}}\; v,$$

cioè dipende solamente da v: il modello separa nettamente la scala complessiva dello shower, controllata da un unico numero, dalla sua ripartizione longitudinale fra i layer.

Il parametro $v_{\max}$ rappresenta invece un tetto rigido e nessun evento generato può depositare più energia di $v_{\max} E_{\text{inc}}$. È lecito chiedersi perché non scegliere semplicemente $v_{\max} = 1$. Il motivo è che nel dataset il rapporto fra energia depositata ed energia incidente supera l'unità in una piccola ma non trascurabile frazione di eventi, soprattutto alle energie più basse. $v_{\max}$ è quindi un iperparametro del modello, da scegliere in modo che il tetto non renda irraggiungibili eventi reali.
Il rumore latente $\boldsymbol{\xi}_E$ entra nel blocco secondo una delle due modalità viste sopra. A parità di E_{inc}, estrazioni diverse di $\boldsymbol{\xi}_E$ producono valori diversi della frazione visibile e delle frazioni f_l, e quindi eventi diversi.

4.4 Lo shape model

Lo shape model genera, per ogni layer, la forma normalizzata $\boldsymbol{p}_l$, cioè la distribuzione dell'energia del layer fra i suoi N_l voxel. A differenza dell'energy model, qui si utilizza un blocco ibrido distinto per ciascun layer, ognuno con il proprio reservoir, i cui angoli sono estratti indipendentemente da quelli degli altri. Il condizionamento del blocco relativo al layer l è formato da tre elementi:

$$\boldsymbol{c}_l = \left(\frac{\log(1 + E_{\text{inc}})}{s_E},\ \frac{\log(1 + E_l)}{s_E},\ \frac{l-1}{L-1} \right),$$

cioè l'energia incidente e l'energia depositata nel layer, entrambe riscalate come nell'energy model, e un indice che identifica il layer, normalizzato in $[0, 1]$. Ogni blocco riceve inoltre un proprio rumore latente $\boldsymbol{\xi}_l$, estratto indipendentemente per ciascun layer.
Sono state sviluppate due versioni dello shape model che differiscono nel modo in cui l'output del blocco viene trasformato nella forma del layer.

Shape model flat

Nella versione più semplice il decoder del blocco produce direttamente un vettore $\boldsymbol{y}_l$ di N_l numeri reali, uno per ogni voxel del layer, trasformati in una distribuzione normalizzata tramite la funzione softmax:

$$p_{l,i} = \frac{e^{y_{l,i}}}{\sum_{j=1}^{N_l} e^{y_{l,j}}}, \qquad i = 1, \ldots, N_l.$$

La softmax garantisce per costruzione che la forma sia positiva e normalizzata a uno. Il limite di questo approccio è che il modello non ha alcuna nozione della posizione dei voxel nel calorimetro, infatti per il decoder i voxel sono semplicemente gli elementi di un vettore. Inoltre la dimensione dell'output, e quindi il numero di parametri dell'ultimo strato del decoder, cresce con il numero di voxel del layer, che come visto in table 4.1 varia da 5 a 160.

Shape model coordinate-aware

Per superare questo limite è stata sviluppata una seconda versione, in cui la geometria del calorimetro entra esplicitamente nel modello. Qui il blocco ibrido non produce più la forma del layer, ma un vettore $\boldsymbol{e}_l \in \mathbb{R}^n$ che descrive l'evento, della stessa dimensione del numero di qubit. La forma viene poi costruita da un secondo decoder $q_{\boldsymbol{\eta}}$, condiviso fra tutti i layer, che assegna un punteggio a ogni voxel a partire da $\boldsymbol{e}_l$ e dalle coordinate $\boldsymbol{u}_{l,i}$ del voxel stesso:

$$p_{l,i} = \frac{\exp\left(q_{\boldsymbol{\eta}}(\boldsymbol{e}_l, \boldsymbol{u}_{l,i})\right)}{\sum_{j=1}^{N_l} \exp\left(q_{\boldsymbol{\eta}}(\boldsymbol{e}_l, \boldsymbol{u}_{l,j})\right)}.$$

Le coordinate sono ricavate dal file di geometria ufficiale del dataset e sono formate, per ogni voxel, da seno e coseno dell'angolo α, dalla distanza radiale r, dalla profondità del layer e dal suo indice, gli ultimi tre normalizzati. L'uso di seno e coseno al posto dell'angolo stesso permette di rispettare la periodicità di α: due voxel adiacenti a cavallo di $\alpha = 0$ risultano vicini anche in input al decoder.
In questo modo il numero di parametri non dipende più dal numero di voxel, e il decoder, essendo condiviso, può sfruttare regolarità legate alla geometria comuni a tutti i layer.

4.5 Addestramento di ResQNN

Una conseguenza della fattorizzazione è che i due stadi possono essere addestrati separatamente. Precisamente, durante l'addestramento lo shape model riceve in input le energie dei layer vere, prese dal dataset, e non quelle generate dall'energy model (tecnica nota come teacher forcing). Ciò permette di addestrare lo shape model indipendentemente dalla qualità dell'energy model.

Addestramento dell'energy model

Per l'addestramento dell'energy model sono state implementate tre diverse funzioni costo: quella che fornisce dei risultati migliori fa uso dell'energy score, mentre le altre due sono di tipo regressivo.
Partendo dalla prima, l'energy score [65] è una regola di punteggio strettamente propria, cioè una funzione il cui valore atteso viene minimizzato se e solo se la distribuzione generata coincide con quella vera. Dato un evento con energie vere $\boldsymbol{y}$ e indicando con $\boldsymbol{X}$ e $\boldsymbol{X}'$ due campioni indipendenti generati dal modello con lo stesso condizionamento, l'energy score è

$$\mathrm{ES} = \mathbb{E}\,\|\boldsymbol{X} - \boldsymbol{y}\|^{\beta} - \frac{1}{2}\,\mathbb{E}\,\|\boldsymbol{X} - \boldsymbol{X}'\|^{\beta}, \qquad \beta \in (0, 2).$$

Il primo termine premia i campioni vicini al valore vero, mentre il secondo, con segno negativo, premia la dispersione fra i campioni. Da notare che l'energy score non viene

calcolato direttamente sulle energie dei layer ma sui loro logaritmi. Precisamente, indicando con $E_{l,i}$ l'energia del layer l nell'i-esimo evento generato, le componenti dei vettori sono

$$X_{l,i} = \log\left(1 + E_{l,i}\right), \qquad y_l = \log\left(1 + E_l\right), \qquad l = 1, \ldots, L,$$

con le energie espresse in MeV. Il motivo è che le energie dei layer coprono molti ordini di grandezza, sia al variare di E_{inc} sia da layer a layer: in scala lineare la norma euclidea sarebbe dominata dagli eventi più energetici e, all'interno di ciascun evento, dai layer in cui si deposita più energia, mentre gli altri contribuirebbero in modo trascurabile alla funzione costo. Precisamente vengono generati M eventi sintetici, tutti a stessa E_{inc} (ogni evento è costituito da un array di L componenti o energie, una per ogni layer chiaramente), per il confronto, che può esser sia fra due eventi generati sia fra campione generato ed evento vero, viene calcolata la differenza fra le componenti. Da tale differenza si ottiene un vettore di L elementi di cui infine si va a calcolare la norma e la si eleva all'iperparametro β. Dato che si dispone di M eventi sintetici, si va a calcolare sia la media delle $||\boldsymbol{X}_i - \boldsymbol{y}||^\beta$ (la media di M termini, uno per ogni differenza fra evento generato $\boldsymbol{X}_i$ ed evento vero $\boldsymbol{y}$) sia la media dei termini $||\boldsymbol{X}_i - \boldsymbol{X}_j||^\beta$ (in cui $\boldsymbol{X}_i$ e $\boldsymbol{X}_j$ sono due eventi generati diversi). Formalmente:

$$ES = \frac{1}{M}\sum_i ||\boldsymbol{X}_i - \boldsymbol{y}||^\beta - \frac{1}{2}\frac{1}{M(M-1)}\sum_{i \neq j} ||\boldsymbol{X}_i - \boldsymbol{X}_j||^\beta.$$

(i e j variano da 1 a M e si ricorda che in questo caso generale $\boldsymbol{X}_{i/j}$ e $\boldsymbol{y}$ sono degli array con L elementi, cioè il numero di layer) Introdotto quindi l'energy score, è possibile ora scrivere la funzione costo utilizzata per addestrare l'energy model:

$$\mathcal{L} = ES + \lambda_{\text{tot}} ES_{\text{tot}},$$

in cui ES è l'energy score appena definito, ES_{tot} invece è un energy score in cui si ragiona sulle energie depositate totali e non sulle energie depositate nei specifici layer. Cioè sia degli M eventi generati sia dell'evento vero si va prima a sommare tutte le energie di ogni layer per ottenere X_{tot_i} e y_{tot} e poi si applica ES a questi. In questo secondo termine della funzione costo, la differenza fra X_{tot_i} e y_{tot} è una semplice differenza fra scalari. Così all'interno della funzione costo c'è sia un energy score ES che quantifica la differenza fra l'energia generata e quella attesa depositata nei layer, sia un energy score ES_{tot} che quantifica la differenza fra l'energia totale generata e attesa depositata. La funzione costo con energy score quindi dipende da 3 iperparametri: β, λ_{tot} e il numero di campioni M.
Oltre all'energy score sono state implementate altre due funzioni costo per l'energy model, di tipo regressivo, che sono state le prime a essere utilizzate durante lo sviluppo del modello. Vale la pena descriverle perché il loro comportamento mostra bene come, per un modello generativo, la scelta della funzione costo non sia un dettaglio.

A differenza dell'energy score, in questo caso per ogni evento vero viene generato un solo evento sintetico, con un'unica estrazione del rumore latente. Indicando con $\hat{E}_l$ le energie dei layer così generate, con E_l quelle vere e con $\hat{E}_{\text{tot}}$ e E_{tot} le corrispondenti energie totali, la prima funzione costo, che verrà chiamata "regressione", è

$$\mathcal{L}_{\text{reg}} = \frac{1}{L}\sum_{l=1}^{L} \ell\left(\log(1+\hat{E}_l) - \log(1+E_l)\right) + \ell\left(\log(1+\hat{E}_{\text{tot}}) - \log(1+E_{\text{tot}})\right) + \frac{\lambda_{\text{frac}}}{L}\sum_{l=1}^{L}\left(\frac{\hat{E}_l}{\hat{E}_{\text{tot}}} - \frac{E_l}{E_{\text{tot}}}\right)^2,$$

mediata sugli eventi del batch, dove ℓ è la funzione smooth L1:

$$\ell(u) = \begin{cases} \dfrac{u^2}{2} & \text{se } |u| < 1, \\ |u| - \dfrac{1}{2} & \text{se } |u| \geq 1. \end{cases}$$

Il primo termine confronta le energie dei singoli layer, il secondo l'energia totale e il terzo la ripartizione dell'energia fra i layer. Come per l'energy score, i primi due termini lavorano sui logaritmi delle energie, mentre la funzione smooth L1 si comporta come un errore quadratico per errori piccoli e come un errore assoluto per errori grandi, così da limitare il peso degli eventi molto lontani da quelli previsti. In questa versione della funzione costo il peso λ_{frac} è fissato a 0.2.
Il limite di questa funzione costo è strutturale. Essa è infatti una somma di termini che dipendono ciascuno da un solo evento generato e non contiene alcun termine che, come il secondo termine dell'energy score, confronti fra loro più campioni generati con lo stesso condizionamento. In sostanza la funzione costo viene minimizzata da un modello che ignora il rumore latente e che restituisce, per ogni energia incidente, sempre le stesse energie dei layer, prossime a un valore centrale della distribuzione vera. Un modello addestrato in questo modo tende quindi a perdere la variabilità a parità di condizionamento.
Per contrastarlo è stata sviluppata una seconda versione, che verrà chiamata "regressione con λ", in cui il peso λ_{frac} diventa un iperparametro libero e viene aggiunto un termine sulla varianza:

$$\mathcal{L}_{\text{reg},\lambda} = \mathcal{L}_{\text{reg}} + \lambda_{\text{var}}\ \max\left(0,\ \text{Var}\left[\log(1+E_l)\right] - \text{Var}\left[\log(1+\hat{E}_l)\right]\right).$$

Le due varianze vengono calcolate sull'insieme di tutti i valori $\log(1+E_l)$ del batch, cioè su tutti gli eventi e su tutti i layer insieme. Il termine è unilaterale: penalizza il modello solo quando la distribuzione generata è più stretta di quella vera, e non quando è più larga (questo perché il problema da risolvere, emerso sperimentalmente, era la scarsa

varianza dei campioni generati con la funzione costo precedente). Con $\lambda_{\text{frac}} = 0.2$ e $\lambda_{\text{var}} = 0$ si ritrova esattamente la precedente funzione costo.

Addestramento dello shape model

Per lo shape model la situazione è diversa, poiché l'output è esso stesso una distribuzione normalizzata sui voxel del layer. La funzione costo principale è quindi la cross entropy fra la forma vera $\boldsymbol{p}_l$ e quella generata $\hat{\boldsymbol{p}}_l$, mediata sui layer:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{L} \sum_{l=1}^{L} \sum_{i=1}^{N_l} p_{l,i} \log \hat{p}_{l,i},$$

a cui viene aggiunto, con peso λ_{vox}, l'errore quadratico medio fra i logaritmi delle energie dei voxel dello shower ricostruito e di quello vero:

$$\mathcal{L}_S = \mathcal{L}_{\text{CE}} + \frac{\lambda_{\text{vox}}}{533} \sum_{l=1}^{L} \sum_{i=1}^{N_l} \left[\log\left(1 + E_l\, \hat{p}_{l,i}\right) - \log\left(1 + x_{l,i}\right)\right]^2,$$

in cui si utilizza di nuovo l'energia vera del layer E_l. Entrambi i termini sono poi mediati sugli eventi del batch. Il primo confronta le forme indipendentemente dall'energia del layer, mentre il secondo pesa gli errori tenendo conto della scala di energia effettivamente depositata in ciascun voxel.

4.6 Generazione degli shower con ResQNN

Una volta addestrati, i due stadi vengono concatenati per generare nuovi shower. A differenza di quanto accade durante l'addestramento, lo shape model non riceve più le energie dei layer vere ma quelle generate dall'energy model. Precisamente, la generazione di un nuovo shower con energia incidente E_{inc} procede come segue:

1. condizionamento: si calcola il condizionamento dell'energy model, $c = \log(1 + E_{\text{inc}})/s_E$. Per confrontare gli shower generati con quelli di Geant4 si utilizzano le stesse energie incidenti degli eventi di riferimento, cioè uno dei 15 valori presenti nel dataset;

2. energie dei layer: si estrae il rumore latente $\boldsymbol{\xi}_E \sim \mathcal{N}(\mathbf{0}, \sigma_\xi^2\, \mathbb{I}_d)$ e si valuta l'energy model, ottenendo l'output $(a_0, \ldots, a_L)$ del decoder e da questo la frazione visibile v, le frazioni f_l e le energie dei layer $E_l = E_{\text{inc}}\, v\, f_l$;

3. jitter (opzionale): le energie generate vengono moltiplicate per un fattore casuale log-normale, estratto indipendentemente per ogni layer,

$$E_l \to E_l \, e^{\sigma_s \epsilon_l}, \qquad \epsilon_l \sim \mathcal{N}(0,1);$$

4. forme: per ogni layer si estrae un rumore latente $\boldsymbol{\xi}_l$, indipendente da quello degli altri layer e da $\boldsymbol{\xi}_E$, e si valuta il blocco dello shape model relativo al layer, con condizionamento

$$\boldsymbol{c}_l = \left(\frac{\log(1+E_{\text{inc}})}{s_E}, \; \frac{\log(1+E_l)}{s_E}, \; \frac{l-1}{L-1} \right),$$

   in cui E_l è l'energia appena generata. Si ottiene così la forma normalizzata $\boldsymbol{p}_l$;

5. temperatura (opzionale): la forma viene modificata tramite una temperatura T,

$$p_{l,i} \to \frac{p_{l,i}^{1/T}}{\sum_{j=1}^{N_l} p_{l,j}^{1/T}};$$

6. ricostruzione: si ricostruisce lo shower come $\boldsymbol{x}_l = E_l \, \boldsymbol{p}_l$ e si concatenano i vettori dei sette layer nello stesso ordine del dataset, ottenendo un vettore di 533 energie.

I passi 3 e 5 non fanno parte del modello addestrato: sono due correzioni che agiscono solo al momento della generazione e che, con $\sigma_s = 0$ e $T = 1$, lasciano l'output invariato. Il jitter è stato introdotto per i modelli addestrati con le funzioni costo di tipo regressivo: poiché questi tendono a ignorare il rumore latente, la variabilità mancante può essere reintrodotta a posteriori allargando artificialmente la distribuzione delle energie generate. Si tratta però di una correzione che non viene appresa dai dati e che ha due effetti collaterali. Essendo estratto indipendentemente per ogni layer, il fattore casuale altera le correlazioni fra le energie dei layer. Inoltre rompe il vincolo $\sum_l E_l = E_{\text{inc}} \, v$, per cui l'energia totale non è più controllata dalla sola frazione visibile. Per i modelli addestrati con l'energy score, che apprendono direttamente la dispersione, si utilizza quindi $\sigma_s = 0$. La temperatura agisce invece sulle forme: con $T < 1$ l'energia del layer si concentra nei voxel più probabili, mentre con $T > 1$ si distribuisce in modo più uniforme fra i voxel. In entrambi i casi la forma resta normalizzata.
Lo shower così generato eredita per costruzione alcune proprietà. In assenza di jitter l'energia totale depositata è

$$\sum_{l=1}^{L} \sum_{i=1}^{N_l} x_{l,i} = \sum_{l=1}^{L} E_l = E_{\text{inc}} \, v,$$

ed è quindi determinata interamente dall'energy model, lo shape model infatti distribuisce solamente l'energia, già determinata, fra i voxel. Inoltre, poiché la softmax è

strettamente positiva, tutti i voxel di tutti i layer ricevono un'energia maggiore di zero, mentre negli shower di Geant4 una frazione consistente dei voxel, e in alcuni eventi interi layer, sono esattamente vuoti. Il modello può avvicinarsi ad un voxel vuoto rendendone molto piccola l'energia, ma non può riprodurlo esattamente. Per questo motivo nel confronto con i dati è utile applicare una soglia azzerando i voxel con energia inferiore a un valore fissato.
Per come è stato pensato ResQNN, gli addestramenti dell'energy model e dello shape model sono fra loro indipendenti. Tuttavia durante la fase di generazione lo shape model riceve energie che non provengono dal dataset ma dall'energy model, quindi i possibili errori nella generazione delle energie si propagano anche sulla generazione delle forme. Tale disaccoppiamento dei due stadi ha però un vantaggio: i due stadi possono anche essere valutati separatamente, infatti fornendo allo shape model le energie dei layer di Geant4 si ottengono dei risultati indipendenti dalle prestazioni dell'energy model.
Gli shower generati vengono infine salvati nello stesso formato HDF5 dei file della CaloChallenge, così da poter essere analizzati con gli stessi strumenti utilizzati per gli shower di Geant4.
Per completezza, in appendice A sono riportati tutti gli iperparametri di ResQNN.

4.7 Test e validazione del modello

Una volta appresa l'architettura di ResQNN, i principali parametri che caratterizzano il modello ed in che modo vengono generati gli shower calorimetrici, si possono presentare i risultati ottenuti a seguito dei vari test effettuati. Data la struttura sequenziale del modello, come accennato sopra, è possibile addestrare separatamente energy model e shape model in modo da potersi concentrare sui problemi che caratterizzano ciascuno. In questa tesi ho effettuato esclusivamente test sull'energy model.
In particolare verrà presentato il motivo empirico per cui è stato necessario implementare tre funzioni costo e quale fra queste si presta meglio allo scopo dell'energy model. Tramite un confronto diretto fra le due versioni, si capirà perché è stato utile implementare la possibilità di inserire il rumore prima o dopo il reservoir all'interno del blocco ibrido. Sono inoltre stati effettuati dei test per capire meglio la dipendenza della capacità generativa del modello al variare del numero di neuroni della rete classica e del numero M di campioni usati per valutare l'energy score. Infine viene indagato cosa accade quando si rimuove il reservoir, cioè il circuito quantistico, dal blocco ibrido.

4.7.1 Confronto fra le funzioni costo

Nel suo stato embrionale ResQNN prevedeva per l'energy model un'unica funzione costo, quella chiamata "regressione". Come anticipato nella sezione dedicata all'addestramento, tale funzione costo viene minimizzata da un modello che ignora il rumore latente, ed è esattamente ciò che accade in pratica: il modello così addestrato genera, a parità di energia incidente, eventi quasi identici fra loro. In sostanza il modello ha imparato un valore tipico delle energie dei layer, ma non la loro distribuzione.
Per contrastare questo collasso è stata introdotta la funzione costo "regressione con λ", che aggiunge il termine sulla varianza visto in precedenza. Anche in questo caso però il modello genera eventi quasi deterministici, con un leggero miglioramento della JSD rispetto al caso precedente.
Per entrambi i modelli si è quindi provato a reintrodurre a posteriori la variabilità mancante tramite il jitter descritto al passo 3 della generazione. Come si vedrà, questa correzione non risolve il problema. Il jitter infatti allarga la distribuzione degli eventi generati sempre della stessa quantità relativa, qualunque sia l'energia incidente, mentre negli shower di Geant4 la dispersione relativa dell'energia depositata diminuisce di un ordine di grandezza passando da 256 MeV a qualche TeV. Inoltre il jitter non viene appreso dai dati, per cui non può riprodurre né la forma delle distribuzioni né le correlazioni fra i layer. È per questi motivi che è stata introdotta la terza funzione costo, basata sull'energy score, che contiene esplicitamente un termine che premia la dispersione fra i campioni generati con lo stesso condizionamento.

Configurazione dei modelli. I tre modelli confrontati differiscono solamente per la funzione costo: architettura, eventi e parametri di addestramento sono identici e sono riportati in table 4.2. Si tratta di modelli volutamente piccoli, con un reservoir di 8 qubit e 4 strati, su 6000 eventi, così da poter essere addestrati in pochi minuti (questi modelli infatti non servono a valutare in modo assoluto le capacità generative ma solo a confrontare fra loro le varie funzioni costo). Tutti utilizzano la configurazione post del rumore che, come mostrato nella sottosezione successiva, rende molto più veloce l'addestramento con l'energy score.

Valutazione. Per il confronto vengono utilizzati 11123 eventi di Geant4 mai visti durante l'addestramento[1]: per ciascuno di essi ogni modello genera un solo evento sintetico con la stessa energia incidente, utilizzando per tutti e tre i modelli le stesse estrazioni del rumore latente, così che le differenze osservate siano dovute solamente ai modelli. Per misurare quanto è larga la distribuzione generata rispetto a quella vera si utilizza il

[1]Precisamente si tratta degli eventi dello split di test dei primi 80000 eventi del file, esclusi quelli che cadono fra i primi 6000 e che sono quindi stati utilizzati per addestrare questi modelli. In questo modo gli eventi di riferimento sono, a meno di quelli esclusi, gli stessi utilizzati per valutare i modelli addestrati su 80000 eventi dei confronti successivi.

	regressione	**regressione con λ**	**energy score**
Qubit n e strati D del reservoir	$n = 8$, $D = 4$		
Neuroni dei due strati nascosti di encoder e decoder	64		
Rumore latente: dimensione d, deviazione standard σ_ξ, configurazione	$d = 8$, $\sigma_\xi = 1$, post		
Tetto della frazione visibile $v_{\max}$, scala del condizionamento s_E	$v_{\max} = 2.5$, $s_E = 12$		
Eventi (addestramento / validazione / test)	i primi 6000 del file (70% / 15% / 15%)		
Epoche, batch size, learning rate (Adam)	40, 64, 10^{-3}		
Seed	42		
λ_{frac}	0.2	0.1	–
λ_{var}	–	0.01	–
M, β, λ_{tot}	–	–	5, 1, 1
Jitter in generazione σ_s (un fattore per layer)	0.30	0.30	0

Tabella 4.2: *Configurazione dei tre modelli del confronto fra funzioni costo. Le righe nella parte superiore sono comuni a tutti i modelli, che condividono quindi architettura, eventi di addestramento e seed, e differiscono solo per la funzione costo. Nella parte inferiore sono riportati gli iperparametri specifici di ciascuna funzione costo e il jitter σ_s applicato in fase di generazione. I pesi della regressione con λ e il jitter dei due modelli regressivi sono stati scelti tramite lo sweep descritto nel testo, mentre per la regressione il valore $\lambda_{frac} = 0.2$ fa parte della definizione stessa della funzione costo.*

rapporto di spread

$$s(E_{\text{inc}}) = \frac{\sigma_{\text{gen}}(E_{\text{inc}})}{\sigma_{\text{vero}}(E_{\text{inc}})},$$

dove σ_{gen} e σ_{vero} sono le deviazioni standard dell'energia, totale o di un singolo layer, degli eventi generati e di quelli veri con la stessa energia incidente. Un modello ben calibrato ha $s = 1$, mentre valori minori o maggiori di 1 indicano distribuzioni generate rispettivamente troppo strette o troppo larghe. Per riassumere il comportamento su tutte le energie incidenti si riportano la mediana di s sulle 15 energie incidenti e la media di $|\log s|$, che vale 0 solo se il modello è calibrato a tutte le energie. La somiglianza fra le forme delle distribuzioni vere e generate è misurata invece dalla distanza di Jensen-Shannon (JSD) fra i rispettivi istogrammi, che vale 0 per distribuzioni identiche e 1 per distribuzioni che non si sovrappongono. Nei grafici le energie sono rappresentate tramite la funzione $\text{log1p}(x) = \log(1 + x)$, che comprime i diversi ordini di grandezza coperti dalle energie e assegna alle energie nulle il valore $\text{log1p}(0) = 0$.
Per valutare la struttura fra i layer si introduce infine, a energia incidente fissata, il

rapporto

$$R = \frac{\mathrm{Var}\left(\sum_l E_l\right)}{\sum_l \mathrm{Var}\left(E_l\right)},$$

che vale 1 se le energie dei layer fluttuano indipendentemente l'una dall'altra, è maggiore di 1 se tendono a crescere e diminuire insieme ed è minore di 1 se sono anticorrelate. In tabella si riporta la mediana sulle energie incidenti del rapporto $R_{\text{gen}}/R_{\text{vero}}$, che vale 1 se il modello riproduce le correlazioni degli shower veri.

Scelta dei pesi della funzione costo e del jitter. I pesi λ_{frac} e λ_{var} della regressione con λ e il valore σ_s del jitter dei due modelli regressivi sono stati scelti tramite uno sweep. Per la regressione con λ ho addestrato 25 modelli, uno per ogni combinazione di

$$\lambda_{\text{frac}} \in \{0.1,\ 0.2,\ 0.3,\ 0.5,\ 1\}, \qquad \lambda_{\text{var}} \in \{0.01,\ 0.03,\ 0.1,\ 0.3,\ 1\},$$

con tutti gli altri parametri fissati ai valori di table 4.2, mentre per la regressione, in cui $\lambda_{\text{frac}} = 0.2$ e il termine sulla varianza è assente per definizione, è stato utilizzato un solo modello. A ciascun modello è stato poi applicato il jitter, con

$$\sigma_s \in \{0,\ 0.15,\ 0.30,\ 0.45,\ 0.60,\ 0.75,\ 0.90\}.$$

Per ogni combinazione è stato generato un evento sintetico per ciascuno di 6000 eventi di Geant4, diversi sia da quelli di addestramento sia dagli 11123 eventi utilizzati per il confronto, ed è stata scelta la combinazione che minimizza la JSD dell'energia totale a energia incidente fissata, presa come mediana sulle 15 energie incidenti. Scegliere i parametri su eventi diversi da quelli del confronto evita di ottenere metriche migliori di quelle che il modello avrebbe su eventi nuovi.
Il risultato dello sweep è netto per σ_s e per λ_{var}. Per tutti i modelli regressivi il jitter migliore è $\sigma_s = 0.30$. Il termine sulla varianza migliora invece il modello solo se ha un peso molto piccolo: la JSD passa da 0.386 della regressione, in cui il termine è assente, a circa 0.35 con $\lambda_{\text{var}} = 0.01$, per poi peggiorare al crescere del peso. Invece λ_{frac} ha un effetto trascurabile: a parità di $\lambda_{\text{var}} = 0.01$ la JSD varia fra 0.348 e 0.350, una differenza compatibile con le fluttuazioni statistiche della stima. Per la regressione con λ è stata quindi scelta la combinazione $\lambda_{\text{frac}} = 0.1$, $\lambda_{\text{var}} = 0.01$, quella con la JSD minore.
Da notare che anche con i pesi migliori il modello resta collassato: senza jitter la deviazione standard dell'energia totale generata a energia incidente fissata è appena il 9% di quella degli eventi di Geant4, esattamente come per la regressione (table 4.3). Il piccolo miglioramento portato dal termine sulla varianza non riguarda quindi la dispersione a condizionamento fissato, che resta interamente affidata al jitter. Lo stesso sweep applicato al modello addestrato con l'energy score indica infine $\sigma_s = 0$, infatti già con $\sigma_s = 0.15$ la JSD peggiora da 0.21 a 0.25, coerentemente con il fatto che questo modello apprende la dispersione direttamente durante l'addestramento.

Metrica	**regressione**	**regressione con λ**	**energy score**	**ideale**
Spread dell'energia totale senza jitter ($\sigma_s = 0$)	0.09	0.09	1.43	1
Spread dell'energia totale	1.30	1.33	1.43	1
$\langle \lvert \log s \rvert \rangle$, energia totale	0.61	0.62	0.30	0
$\langle \lvert \log s \rvert \rangle$, energie dei layer	3.11	3.22	0.64	0
JSD dell'energia totale a E_{inc} fissata	0.33	0.29	0.22	0
$R_{\text{gen}}/R_{\text{vero}}$	12.3	12.2	1.74	1

Tabella 4.3: *Metriche del confronto fra le tre funzioni costo. Lo spread è la mediana sulle 15 energie incidenti del rapporto s fra la deviazione standard degli eventi generati e quella degli eventi veri,* $\langle \lvert \log s \rvert \rangle$ *la media di* $\lvert \log s \rvert$ *sulle energie incidenti (e, per le energie dei layer, anche sui sette layer), la JSD la mediana sulle energie incidenti della distanza di Jensen-Shannon fra gli istogrammi di* $\log(1 + E_{dep})$ *veri e generati,* R_{gen}/R_{vero} *il rapporto fra le correlazioni fra layer generate e vere. Tali metriche sono tutte calcolate sugli 11123 eventi di test. La prima riga mostra la dispersione dei modelli senza jitter, tutte le altre si riferiscono ai modelli con il jitter riportato in table 4.2. Nell'ultima colonna il valore che la metrica corrispondente avrebbe per un modello perfetto.*

Energia totale depositata. Come mostra table 4.3, il modello addestrato con l'energy score è migliore dei due modelli regressivi su tutte le metriche tranne la mediana del rapporto di spread dell'energia totale, che per i modelli regressivi risulta più vicina a 1. Come mostrato di seguito, si tratta però di un risultato fuorviante.
Per capire da dove nascono le differenze conviene partire dall'energia totale depositata, che, in assenza di jitter, è controllata dalla sola frazione visibile v. La differenza principale si vede nella risoluzione (fig. 4.3, a destra): quella di Geant4 diminuisce all'aumentare dell'energia incidente, mentre quella dei modelli regressivi resta pressoché costante, perché la loro dispersione è introdotta unicamente dal jitter, che non dipende dall'energia incidente. Di conseguenza i modelli regressivi generano distribuzioni troppo strette alle energie più basse e troppo larghe a quelle più alte (fig. 4.4, a sinistra). La mediana sulle 15 energie incidenti mescola quindi valori troppo piccoli e troppo grandi, ed è per questo che risulta vicina a 1, mentre la media di $\lvert \log s \rvert$, doppia rispetto a quella dell'energy score, mostra che la dispersione dei modelli regressivi è sbagliata quasi a tutte le energie. Anche la loro risposta è distorta, soprattutto alle energie più basse: a 256 MeV, ad esempio, vale circa 0.34 contro 0.49 di Geant4, mentre alle energie più alte l'errore si riduce.
Il modello addestrato con l'energy score genera invece una dispersione che dipende dall'energia incidente: la sua risoluzione segue quella di Geant4 fino a circa 2 GeV e continua a seguirne l'andamento decrescente anche alle energie più alte, pur sovrastimandola, mentre la risposta è riprodotta entro pochi punti percentuali a tutte le energie.

Le distribuzioni dell'energia totale a energia incidente fissata (fig. 4.5 per tre delle

energie più rappresentate nel campione di test e fig. 4.6 per le tre meno rappresentate) confermano questo quadro. Gli eventi di Geant4 formano un picco stretto, con una coda verso le energie più basse: il modello addestrato con l'energy score ne riproduce posizione, larghezza e coda, e a tutte e sei le energie mostrate ha la JSD più bassa. I modelli regressivi generano invece un picco spostato verso energie più basse, solo leggermente più largo di quello vero a 32 e 131 GeV, troppo stretto e privo di coda a 1 GeV e troppo largo alle energie del TeV, dove l'energy score resta più concentrato attorno al picco di Geant4. Con poche centinaia di eventi, però, a queste energie gli istogrammi sono più rumorosi.

Lo stesso si vede nella distribuzione dell'energia totale su tutti gli eventi (fig. 4.7), formata da una serie di picchi, uno per ognuna delle 15 energie incidenti, ben visibili soprattutto in scala logaritmica. Il modello addestrato con l'energy score riproduce i picchi e la coda di eventi con un'energia depositata molto piccola. I modelli regressivi riproducono solo i picchi delle energie incidenti più alte: spostano verso il basso quelli delle energie più basse e non generano praticamente alcun evento sotto i 50 MeV circa, dove si trova la coda degli eventi di Geant4.

Energie dei layer. Sulle energie dei singoli layer la situazione dei modelli regressivi è ancora peggiore: in tutti i layer la loro dispersione resta molto minore di quella vera, nonostante il jitter (fig. 4.4, a destra). Negli shower veri, infatti, l'energia di un layer fluttua moltissimo da un evento all'altro anche a parità di energia incidente (la frazione di energia nel layer 2, ad esempio, varia fra l'1% e il 66% fra il 16-esimo e l'84-esimo percentile, fig. 4.8), mentre i modelli regressivi, avendo ignorato il rumore latente, generano quasi sempre la stessa ripartizione fra i layer, con la sola variabilità del jitter, circa il 30%, trascurabile rispetto a fluttuazioni di questo tipo.

Anche le correlazioni fra i layer non vengono riprodotte. Negli shower di Geant4 il rapporto R diminuisce all'aumentare dell'energia incidente, da circa 1 alle energie più basse fino a valori inferiori a 0.1 sopra i 65 GeV: alle energie più alte la ripartizione dell'energia fra i layer fluttua molto, mentre l'energia totale resta relativamente stabile. Nei modelli regressivi R resta invece circa uguale a 1 a tutte le energie, come per layer che fluttuano indipendentemente l'uno dall'altro per effetto del jitter, e il rapporto $R_{\text{gen}}/R_{\text{vero}}$ supera 12. Il modello addestrato con l'energy score segue invece l'andamento di Geant4, pur restando, alle energie più alte, fino a circa due volte sopra il valore vero.
Le distribuzioni delle energie dei layer a 32 GeV (fig. 4.9) mostrano che i modelli regressivi generano in ogni layer un picco stretto attorno a un valore tipico, mentre il modello addestrato con l'energy score genera distribuzioni larghe che coprono l'intervallo di quelle vere, come conferma anche il suo χ^2 ridotto, molto più basso. Nessuno dei tre modelli riproduce però i picchi stretti dei layer da 0 a 3 di Geant4, che si trovano nella stessa posizione anche ad altre energie incidenti. Un discorso a parte merita il layer 6, il più

esterno: in molti eventi le particelle dello shower si fermano nei layer precedenti e non vi depositano alcuna energia, per cui a 32 GeV il 39% degli eventi di Geant4 ha il layer 6 esattamente vuoto. Poiché $\log1p(0) = 0$, questi eventi formano in fig. 4.9 un picco in $\log1p(E) = 0$. Il modello addestrato con l'energy score riproduce questo picco solo in parte, collocando circa il 22% degli eventi sotto 1 MeV, mentre i modelli regressivi non lo riproducono affatto e generano invece un picco stretto attorno a qualche MeV. Poiché le frazioni f_l sono ottenute con una softmax, strettamente positiva, l'energy model non può comunque generare un layer esattamente vuoto.
Infine, nella ripartizione dell'energia fra i layer su tutti gli eventi (fig. 4.8), i modelli regressivi sovrastimano la frazione di energia dei layer 2 e 4 e nel layer 2 generano frazioni spostate verso valori più alti, mentre il modello addestrato con l'energy score riproduce abbastanza bene l'ampiezza delle fluttuazioni e in parte anche le mediane, con l'eccezione del layer 2, la cui frazione mediana è sottostimata.

Conclusioni. L'introduzione dell'energy score risolve il problema principale delle funzioni costo regressive: il modello non ignora più il rumore latente e apprende dai dati una dispersione che dipende dall'energia incidente, senza bisogno di alcuna correzione in fase di generazione. Ne seguono una risposta corretta entro pochi punti percentuali a tutte le energie, una risoluzione che segue l'andamento di quella di Geant4, distribuzioni dell'energia totale molto più simili a quelle vere e correlazioni fra i layer che seguono, almeno qualitativamente, quelle degli shower veri. Tutte le metriche di table 4.3 sono migliori per questo modello, con l'unica eccezione, spiegata sopra, della mediana dello spread.
Restano però alcuni problemi aperti. In primo luogo il modello resta troppo disperso: sopra i 16 GeV circa genera distribuzioni dell'energia totale nettamente più larghe di quelle vere e alle energie più alte sovrastima la risoluzione. In secondo luogo, e soprattutto, non riproduce correttamente il modo in cui l'energia si distribuisce nei layer: mancano i picchi stretti che caratterizzano i primi quattro layer, nel layer 6 vengono generati ancora pochi eventi con energia bassissima, la dispersione dei singoli layer va da circa tre quarti di quella vera nel layer 5 a più del doppio nel layer 3, le anticorrelazioni fra i layer sono più deboli di quelle vere e la frazione mediana del layer 2 è sottostimata. Come si vedrà, il problema della distribuzione dell'energia nei layer si presenterà anche in tutti i confronti successivi.
Per questo motivo, in tutti i confronti che seguono, l'energy model è addestrato con l'energy score.

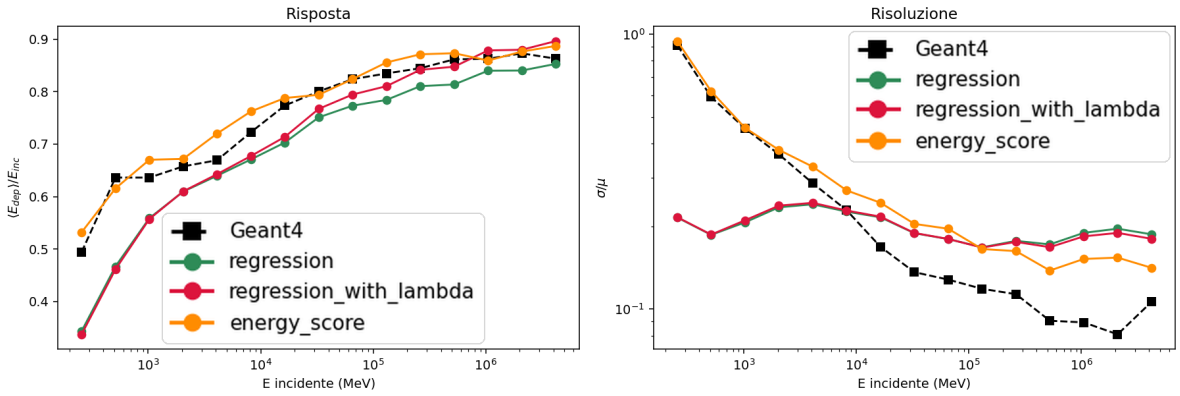


Figura 4.3: *Risposta (a sinistra), cioè il rapporto fra l'energia depositata media e l'energia incidente, e risoluzione (a destra), cioè il rapporto fra la deviazione standard e la media dell'energia depositata, in funzione dell'energia incidente, per Geant4 (quadrati neri) e per i tre modelli. Entrambe sono calcolate separatamente per ognuna delle 15 energie incidenti sui 11123 eventi di test.*

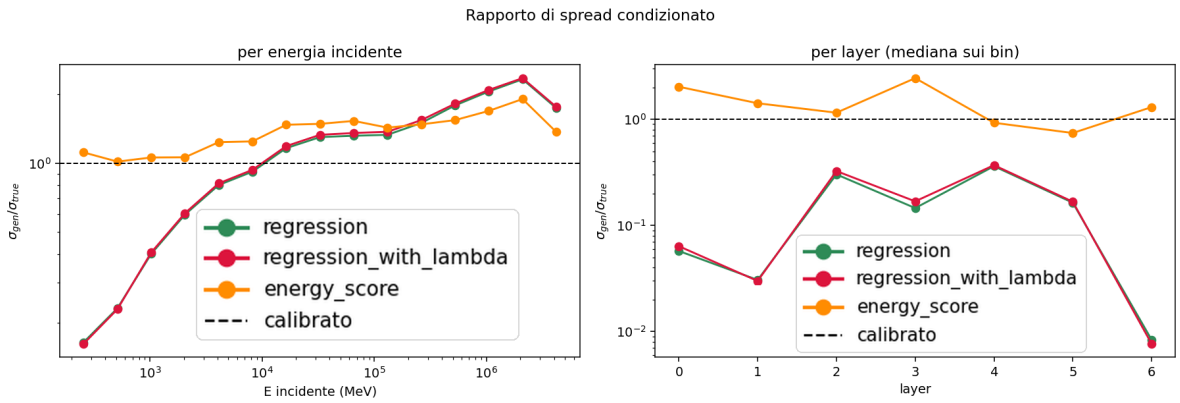


Figura 4.4: *Rapporto di spread $s = \sigma_{gen}/\sigma_{vero}$ fra la deviazione standard delle energie generate e quella delle energie di Geant4. A sinistra in funzione dell'energia incidente, a destra in funzione di ciascun layer. La linea tratteggiata corrisponde a una dispersione corretta: valori maggiori di 1 indicano distribuzioni generate troppo larghe, valori minori troppo strette.*

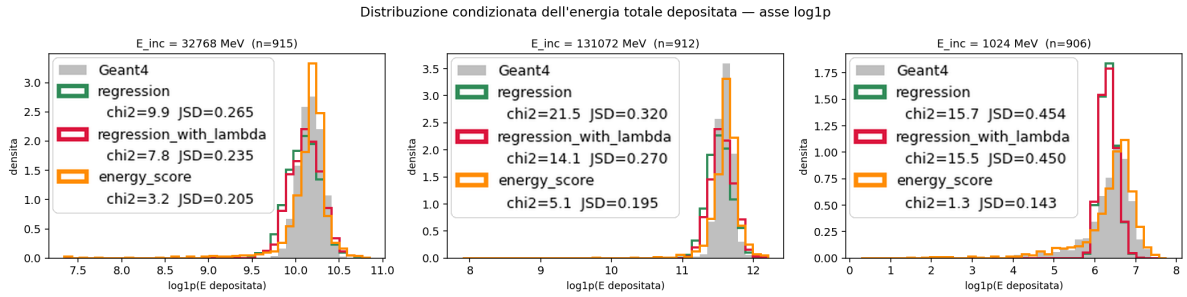


Figura 4.5: *Distribuzione dell'energia totale depositata, in scala* $\log 1\mathrm{p}(E_{dep})$*, a tre energie incidenti fissate fra le più rappresentate nel campione di test: 32 GeV, 131 GeV e 1 GeV (n indica il numero di eventi veri). L'istogramma grigio rappresenta gli eventi di Geant4, le linee colorate i tre modelli. In legenda sono riportati, per ciascun modello, il* χ^2 *ridotto e la distanza di Jensen-Shannon rispetto a Geant4, calcolati sugli stessi bin del grafico.*

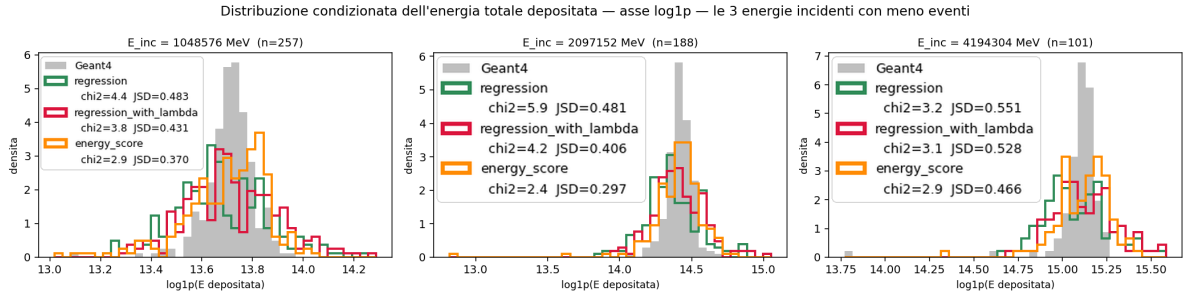


Figura 4.6: *Come in fig. 4.5, ma per le tre energie incidenti meno rappresentate nel campione di test, cioè 1, 2 e 4 TeV, con rispettivamente 257, 188 e 101 eventi veri. Il minor numero di eventi rende gli istogrammi più rumorosi, ma il comportamento dei modelli è lo stesso delle energie più rappresentate.*

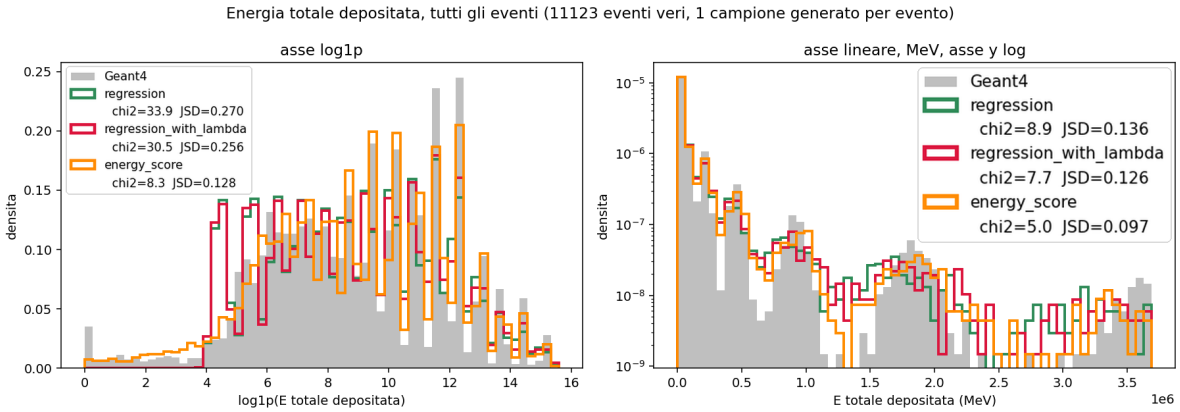


Figura 4.7: *Distribuzione di* $\log 1p(E_{dep})$ *(a sinistra) e distribuzione di* E_{dep} *in scala logaritmica (a destra) su tutti gli 11123 eventi di test, cioè unendo le 15 energie incidenti. Da notare che in un grafico di questo tipo la dispersione fra energie incidenti diverse nasconde in parte quella a energia incidente fissata, che va valutata con i grafici precedenti.*

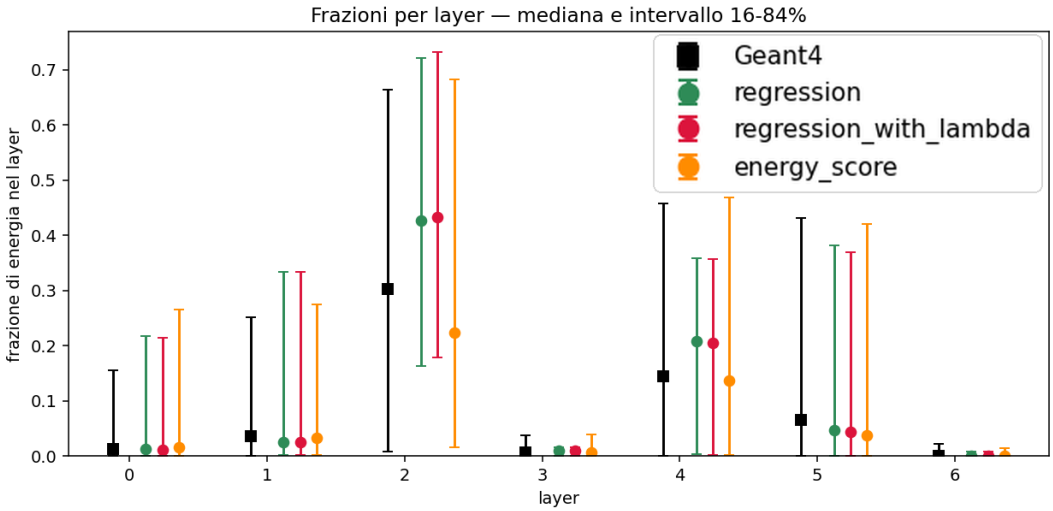


Figura 4.8: *Frazione* E_l/E_{dep} *dell'energia totale depositata nel layer* l*, calcolata su tutti gli 11123 eventi di test e quindi su tutte le energie incidenti. Il punto indica la mediana e la barra l'intervallo fra il 16-esimo e l'84-esimo percentile, che per una distribuzione gaussiana corrisponderebbe a* $\pm 1\sigma$*. Per ogni layer sono mostrati, da sinistra, Geant4 e i tre modelli.*

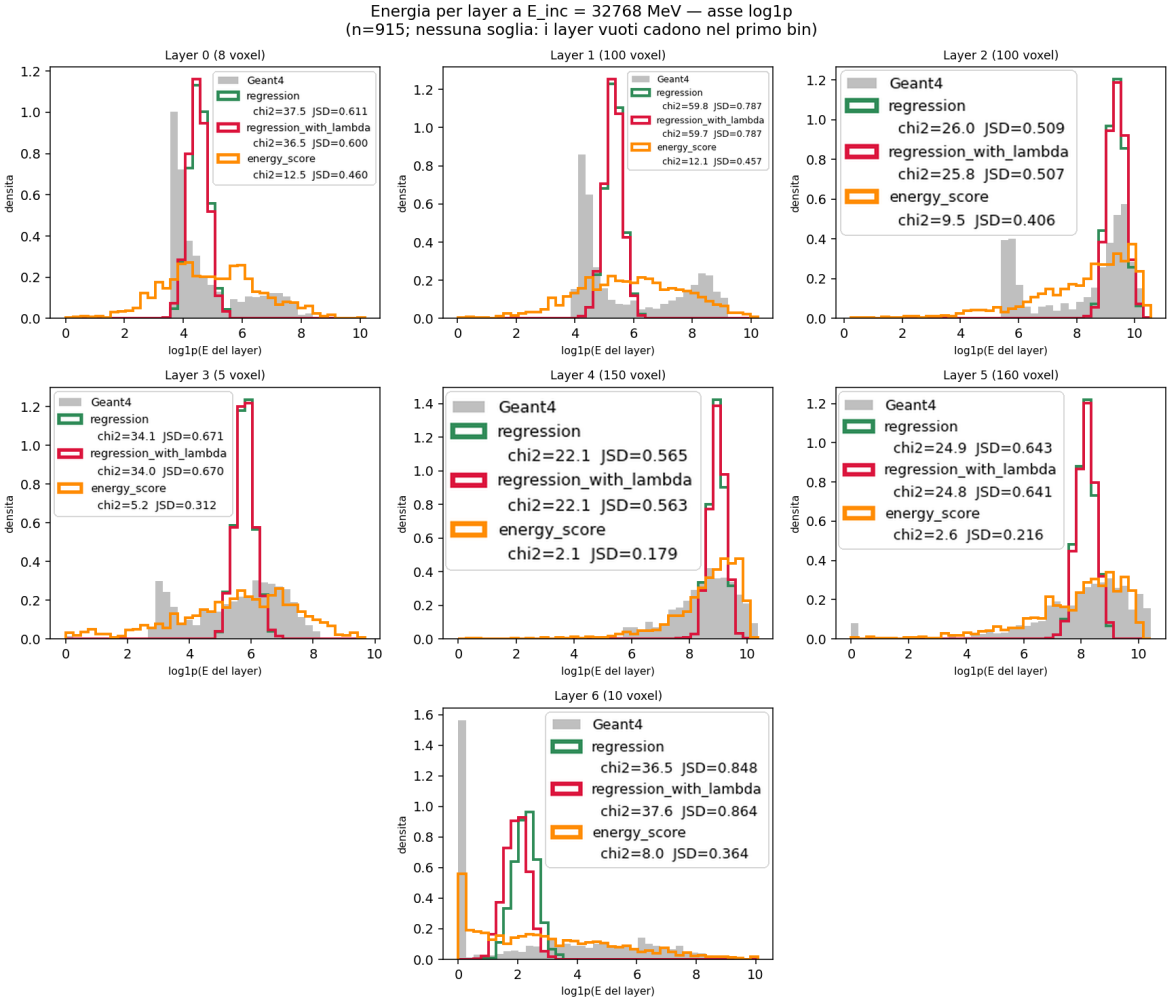


Figura 4.9: *Distribuzione dell'energia depositata in ciascuno dei sette layer, in scala* $\log1p(E_l)$*, per i 915 eventi di test con energia incidente di 32 GeV.*

4.7.2 Confronto fra le configurazioni pre e post del rumore

Come visto nella descrizione del blocco ibrido, il rumore latente può entrare nel blocco in due punti diversi: prima del reservoir, concatenato al condizionamento (configurazione pre), oppure dopo il reservoir, concatenato al vettore $\boldsymbol{z}$ dei valori d'aspettazione (configurazione post). In questa sottosezione ho confrontato le due configurazioni sia dal punto di vista del costo computazionale sia da quello della qualità degli eventi generati. È lecito chiedersi, infatti, se il vantaggio computazionale della configurazione post venga pagato con una perdita di qualità, visto che in questo caso tutta la variabilità degli eventi generati è prodotta dal decoder classico.

Configurazione dei modelli. Rispetto alla sottosezione precedente i due modelli sono più grandi e sono addestrati su un numero di eventi oltre 13 volte maggiore, più vicino quindi a un utilizzo realistico del modello. Entrambi sono addestrati con l'energy score e differiscono solamente per la configurazione del rumore; i parametri sono riportati in table 4.4.

Qubit n e strati D del reservoir	$n = 12$, $D = 6$
Neuroni dei due strati nascosti di encoder e decoder	128
Rumore latente: dimensione d, deviazione standard σ_ξ	$d = 16$, $\sigma_\xi = 1$
Tetto della frazione visibile $v_{\max}$, scala del condizionamento s_E	$v_{\max} = 2.5$, $s_E = 12$
Eventi (addestramento / validazione / test)	i primi 80000 del file (70% / 15% / 15%)
Epoche, batch size, learning rate (Adam)	80, 64, 10^{-3}
Funzione costo	energy score, $M = 5$, $\beta = 1$, $\lambda_{\text{tot}} = 1$
Seed	42
Configurazione del rumore	pre oppure post

Tabella 4.4: *Configurazione dei due modelli del confronto fra le configurazioni pre e post del rumore latente. Tutti i parametri sono comuni ai due modelli, che differiscono solamente per il punto in cui il rumore entra nel blocco ibrido.*

Costo computazionale. Durante l'addestramento l'energy score richiede di generare M campioni per ogni evento. Nella configurazione post il circuito viene valutato una sola volta per evento e gli M rumori vengono combinati con lo stesso vettore $\boldsymbol{z}$ dal solo decoder. Nella configurazione pre, invece, ogni campione richiede una valutazione completa del circuito, insieme al calcolo dei gradienti rispetto agli angoli d'ingresso, per cui il costo di ogni passo di addestramento, dovuto alla presenza del circuito quantistico, è M volte maggiore.
Per verificarlo ho misurato i tempi delle due configurazioni sullo stesso computer (Apple

M5), con l'architettura di table 4.4, il simulatore a vettore di stato di PennyLane, che calcola i valori d'aspettazione in modo esatto, e il calcolo limitato a 4 thread. Per passo di addestramento si intende l'intero aggiornamento dei pesi su un batch di 64 eventi: generazione degli $M = 5$ campioni per ogni evento, calcolo dell'energy score e dei gradienti e aggiornamento con Adam. Il tempo riportato è la media su 5 passi, escluso il primo passo. Ho ottenuto 1.4 s per la configurazione post contro 7.1 s per quella pre, cioè un fattore 5, esattamente pari a M, segno che il costo dell'addestramento è dominato dalla simulazione del circuito.
Gli addestramenti completi, di 80 epoche, sono stati invece eseguiti al CERN tramite HTCondor, ciascuno su 4 core di CPU e senza GPU, e sono durati, dall'inizio alla fine del job, 33.3 ore per post e 70.3 ore per pre. Il rapporto fra le due durate, circa 2.1, è minore di quello misurato perché i due job sono stati eseguiti su nodi diversi del cluster, e va quindi preso solo come indicativo.
Durante la generazione la situazione è diversa. I tempi di generazione sono misurati nelle stesse condizioni del passo di addestramento, ma senza il calcolo dei gradienti e con batch di 64 condizionamenti. Se per ogni condizionamento viene generato un solo evento, entrambe le configurazioni richiedono una valutazione del circuito per evento e il costo è praticamente lo stesso: per generare 640 eventi con condizionamenti tutti diversi, cioè 10 batch da 64, sono necessari 7.3 s in post e 7.9 s in pre. Se invece si generano più eventi con lo stesso condizionamento, la configurazione post valuta il circuito una sola volta: generando 10 eventi per 64 condizionamenti diversi, quindi sempre 640 eventi in totale, si passa da 7.9 s in pre a 0.75 s in post, cioè un fattore circa 10. Da notare che questo è proprio il caso di interesse per la simulazione veloce infatti le energie incidenti del dataset della CaloChallenge assumono solamente 15 valori, per cui nella configurazione post basterebbe valutare il circuito dell'energy model 15 volte per generare un numero qualsiasi di shower.

Valutazione. Le metriche e i grafici sono gli stessi della sottosezione precedente, calcolati sui 12000 eventi di test, mai visti durante l'addestramento, con un evento generato per ciascuno. Gli eventi veri sono gli stessi per i due modelli, mentre, a differenza della sottosezione precedente, le estrazioni del rumore latente non coincidono, poiché gli eventi generati provengono da due valutazioni separate.

Risultati. Le metriche di table 4.5 mostrano che la configurazione post non penalizza la capacità del modello di riprodurre la dispersione dell'energia totale: lo spread mediano è lo stesso, 1.05, e la media di $|\log s|$ è molto simile, 0.11 contro 0.12. Sull'energia totale la configurazione post è anzi leggermente migliore, con una JSD di 0.12 contro 0.18 e un rapporto $R_{\text{gen}}/R_{\text{vero}}$ più vicino a 1 (1.13 contro 1.41). La configurazione pre è invece leggermente migliore sulla dispersione dei singoli layer, con una media di $|\log s|$ pari a 0.26 contro 0.33.

Metrica	**post**	**pre**	**ideale**
Durata dell'addestramento completo (80 epoche, al CERN su nodi diversi)	33.3 h	70.3 h	–
Passo di addestramento (batch di 64 eventi, $M = 5$)	1.4 s	7.1 s	–
Generazione di 640 eventi, uno per condizionamento	7.3 s	7.9 s	–
Generazione di 640 eventi, 10 per condizionamento	0.75 s	7.9 s	–
Spread dell'energia totale	1.05	1.05	1
$\langle \lvert \log s \rvert \rangle$, energia totale	0.11	0.12	0
$\langle \lvert \log s \rvert \rangle$, energie dei layer	0.33	0.26	0
JSD dell'energia totale a E_{inc} fissata	0.12	0.18	0
$R_{\text{gen}}/R_{\text{vero}}$	1.13	1.41	1

Tabella 4.5: *Costo computazionale (in alto) e metriche (in basso) delle configurazioni post e pre. I tempi del passo di addestramento e della generazione sono misurati sullo stesso computer (Apple M5, 4 thread), con l'architettura di table 4.4 e pesi casuali, mentre la durata dell'addestramento completo è quella dei job eseguiti al CERN. Le metriche, definite come in table 4.3, sono calcolate sui 12000 eventi di test con un evento generato per ciascuno.*

In fig. 4.10 si vede che entrambi i modelli riproducono la risposta entro circa il 5% a tutte le energie, con l'unica eccezione della configurazione post a 256 MeV (0.55 contro 0.49 di Geant4), e seguono l'andamento della risoluzione di Geant4. La configurazione pre sovrastima la risoluzione fra 16 e 65 GeV (fino al 33% in più), la configurazione post sopra i 500 GeV (fino al 35% in più). Come mostra fig. 4.11, il rapporto di spread dell'energia totale resta fra 0.84 e 1.33 a tutte le energie per entrambi i modelli. Da notare che si tratta di un netto miglioramento rispetto al modello della sottosezione precedente, il cui rapporto di spread arrivava a 1.9: un modello più grande, addestrato su più eventi e per più epoche, riproduce meglio la dispersione degli shower. Lo stesso si vede nelle distribuzioni a energia incidente fissata (fig. 4.12), riprodotte bene da entrambi i modelli, con una JSD fra 0.11 e 0.16 per post e fra 0.18 e 0.21 per pre. Anche la distribuzione dell'energia totale su tutti gli eventi (fig. 4.13) è riprodotta bene: entrambi i modelli generano i picchi corrispondenti alle 15 energie incidenti e la coda di eventi con un'energia depositata molto piccola, con una JSD di 0.07 per post e di 0.10 per pre.
Anche la ripartizione dell'energia fra i layer migliora rispetto alla sottosezione precedente: le frazioni mediane dei layer sono riprodotte da entrambi i modelli, ad esempio 0.32 per post e 0.30 per pre nel layer 2 contro 0.30 di Geant4, e il rapporto R segue quello di Geant4. Nel layer 6 la quota di eventi con un'energia inferiore a 1 MeV è del 41% per post e del 37% per pre, contro il 40% di Geant4. Restano però i problemi visti in precedenza (fig. 4.14): nessuno dei due modelli riproduce i picchi stretti dei layer da 0 a 3, e la

dispersione dei singoli layer è sbagliata fino a un fattore 1.6, ad esempio nel layer 3 per post (1.50) e nel layer 1 per pre (1.58).

Conclusioni. Le due configurazioni generano eventi di qualità molto simile: la configurazione post è leggermente migliore sull'energia totale e sulle correlazioni fra i layer, quella pre sulla dispersione dei singoli layer, e le differenze sono piccole rispetto a quelle viste fra le funzioni costo. Il vantaggio computazionale della configurazione post è invece netto: l'addestramento con l'energy score è circa M volte più veloce per ogni passo e la generazione di più eventi con lo stesso condizionamento richiede una sola valutazione del circuito. Per questi motivi, nel seguito la configurazione post è quella di riferimento.

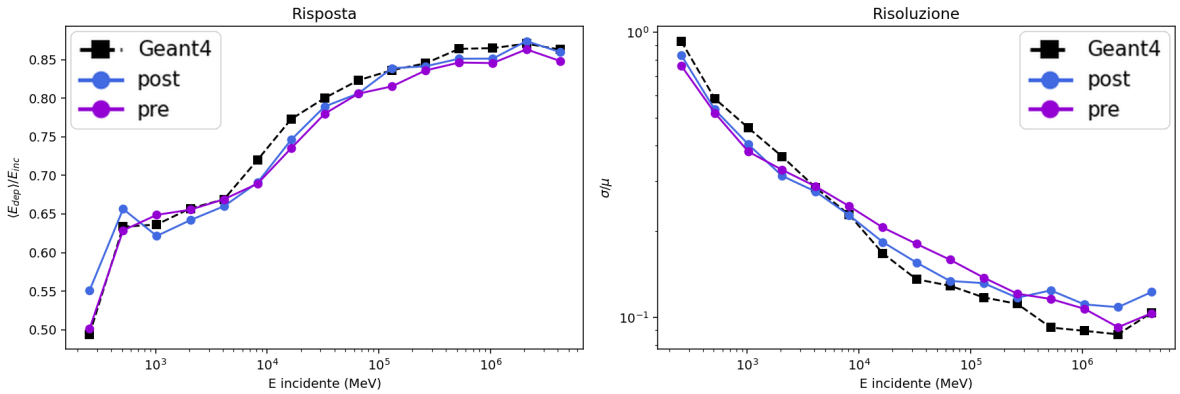


Figura 4.10: *Risposta (a sinistra) e risoluzione (a destra) in funzione dell'energia incidente per Geant4 e per i modelli con rumore pre e post, calcolate sui 12000 eventi di test con un evento generato per ciascuno.*

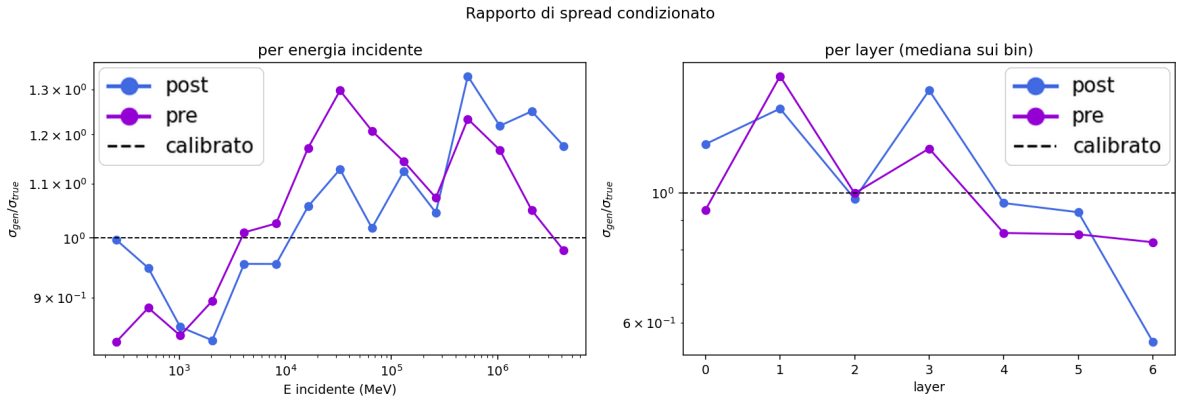


Figura 4.11: *Rapporto di spread $s = \sigma_{gen}/\sigma_{vero}$, in scala doppio logaritmica, per l'energia totale in funzione dell'energia incidente (a sinistra) e per l'energia di ciascun layer, come mediana sulle 15 energie incidenti in scala logaritmica (a destra). La linea tratteggiata corrisponde a una dispersione corretta.*

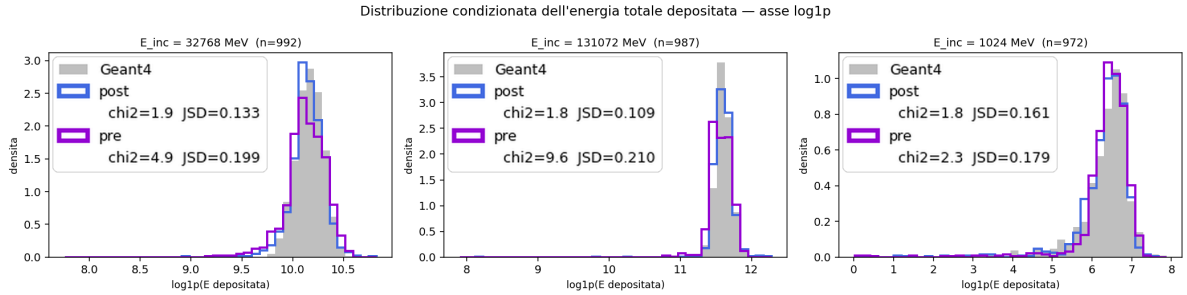


Figura 4.12: *Distribuzione dell'energia totale depositata, in scala* $\log1p(E_{dep})$*, a 32 GeV, 131 GeV e 1 GeV di energia incidente (n indica il numero di eventi veri). In legenda il* χ^2 *ridotto e la distanza di Jensen-Shannon di ciascun modello rispetto a Geant4.*

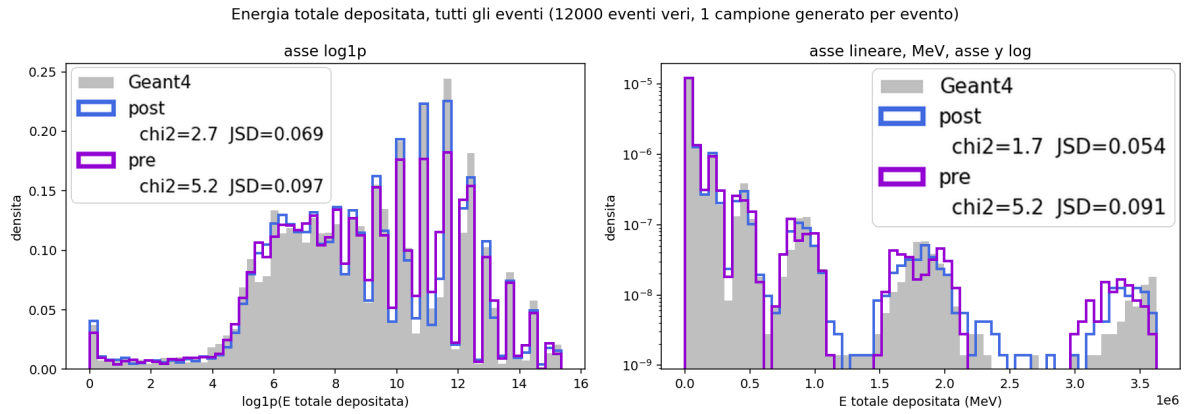


Figura 4.13: *Distribuzione di* $\log1p(E_{dep})$ *(a sinistra) e distribuzione di* E_{dep} *in scala logaritmica (a destra) su tutti gli 12000 eventi di test, per Geant4 e per i modelli con rumore pre e post.*

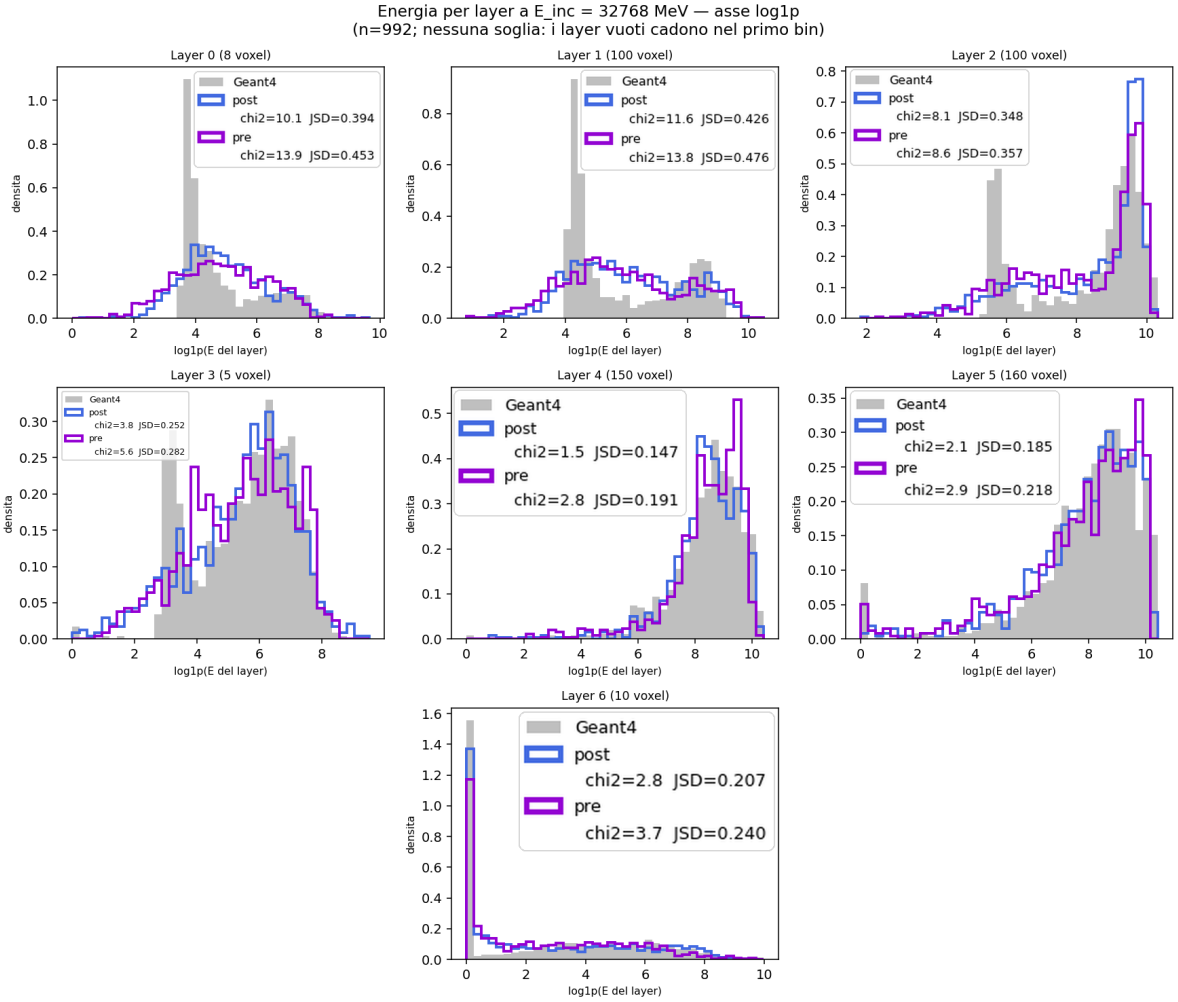


Figura 4.14: *Distribuzione dell'energia depositata in ciascuno dei sette layer, in scala* $\log1p(E_l)$*, per i 992 eventi di test con energia incidente di 32 GeV.*

4.7.3 Ottimizzazione dell'energy model

Il modello addestrato con l'energy score nella configurazione post riproduce bene l'energia totale, ma presenta ancora alcuni problemi: la dispersione dei singoli layer è in disaccordo fino a un fattore 1.6, i picchi stretti dei layer da 0 a 3 non vengono riprodotti e i layer non sono mai esattamente vuoti. In questa sottosezione ho verificato se questi problemi possono essere risolti ottimizzando il modello senza cambiarne la struttura, in due modi diversi:

- aumentando il numero M di campioni con cui viene stimato l'energy score, da 5 a 20: la stima della funzione costo, e quindi del suo gradiente, diventa meno rumorosa, in particolare per il termine che premia la dispersione fra i campioni. Nella configurazione post questo non aumenta il numero di valutazioni del circuito;
- aumentando la capacità del decoder classico, da 2 strati nascosti da 128 neuroni a 4 strati da 512, per capire se i limiti del modello dipendano dalla capacità del decoder di trasformare le misure del reservoir e il rumore nella distribuzione delle energie.

Configurazione dei modelli. I tre modelli hanno la stessa architettura dei modelli precedenti (table 4.4) nella configurazione post e sono addestrati per 30 epoche. Il modello di riferimento, con $M = 5$ e decoder 2×128, differisce da ciascuno degli altri due per un solo parametro (table 4.6). La valutazione è la stessa della sottosezione precedente: 12000 eventi di test con un evento generato per ciascuno.

	riferimento	$M = 20$	**decoder** 4×512	**ideale**
Campioni M dell'energy score	5	20	5	–
Strati nascosti del decoder	2×128	2×128	4×512	–
Spread dell'energia totale	0.94	0.95	1.25	1
$\langle \lvert \log s \rvert \rangle$, energia totale	0.088	0.068	0.230	0
$\langle \lvert \log s \rvert \rangle$, energie dei layer	0.34	0.26	0.55	0
JSD dell'energia totale a E_{inc} fissata	0.142	0.116	0.135	0
$R_{\text{gen}}/R_{\text{vero}}$	0.91	0.92	1.47	1

Tabella 4.6: *Configurazione (in alto) e metriche (in basso) del modello di riferimento e dei due modelli potenziati, tutti addestrati per 30 epoche con l'architettura di table 4.4. Le metriche sono calcolate sui 12000 eventi di test con un evento generato per ciascuno.*

Numero di campioni. Con $M = 20$ il modello è migliore su quasi tutte le metriche di table 4.6: la media di $|\log s|$ scende da 0.088 a 0.068 per l'energia totale, da 0.34 a 0.26 per i layer e la JSD dell'energia totale da 0.142 a 0.116. Lo spread mediano e il rapporto $R_{\text{gen}}/R_{\text{vero}}$ restano invece praticamente uguali. Come mostra fig. 4.15, il miglioramento

riguarda soprattutto le energie incidenti fra 1 e 10 GeV, dove il modello con $M = 5$ genera distribuzioni troppo strette (rapporto di spread fra 0.82 e 0.87) e quello con $M = 20$ resta più vicino a 1 (fra 0.92 e 0.99), e i layer 1, 3 e 6, la cui dispersione passa rispettivamente da 1.49, 1.26 e 0.73 volte quella vera a 1.39, 1.10 e 1.00.

Dimensione del decoder. Il decoder più grande, con circa 38 volte i parametri di quello di riferimento, non migliora il modello. Come mostrano table 4.6 e fig. 4.16, genera distribuzioni troppo larghe quasi a tutte le energie incidenti, con uno spread mediano dell'energia totale di 1.25, e nei layer 0, 1 e 3 una dispersione fra 1.8 e 2.7 volte quella vera. Inoltre i layer generati tendono a variare troppo insieme ($R_{\text{gen}}/R_{\text{vero}} = 1.47$). L'unica metrica leggermente migliore è la JSD dell'energia totale (0.135 contro 0.142).

Discussione. Nessuno dei due potenziamenti risolve i problemi legati alla ripartizione dell'energia fra i layer. Come si vede in fig. 4.17, né il modello con $M = 20$ né quello con il decoder più grande riproducono i picchi stretti dei layer da 0 a 3. Il decoder grande, a questa energia incidente, è anzi il più lontano da Geant4 in sei layer su sette (ad esempio nel layer 3 la sua JSD vale 0.30, contro 0.21 per $M = 20$). Anche con $M = 20$ la dispersione dei layer 0 e 1 resta circa 1.4–1.5 volte quella vera e i layer non sono mai esattamente vuoti: nel layer 6 a 32 GeV solo il 14% degli eventi generati con $M = 20$ ha un'energia inferiore a 1 keV, contro il 39% degli eventi di Geant4 (con il decoder grande l'8%). La persistenza di questi difetti in entrambe le configurazioni suggerisce che né l'aumento del numero di campioni utilizzati nella stima dell'energy score né l'incremento della capacità del decoder siano sufficienti a risolverli. I risultati indicano pertanto la necessità di indagare altri possibili limiti dell'architettura.

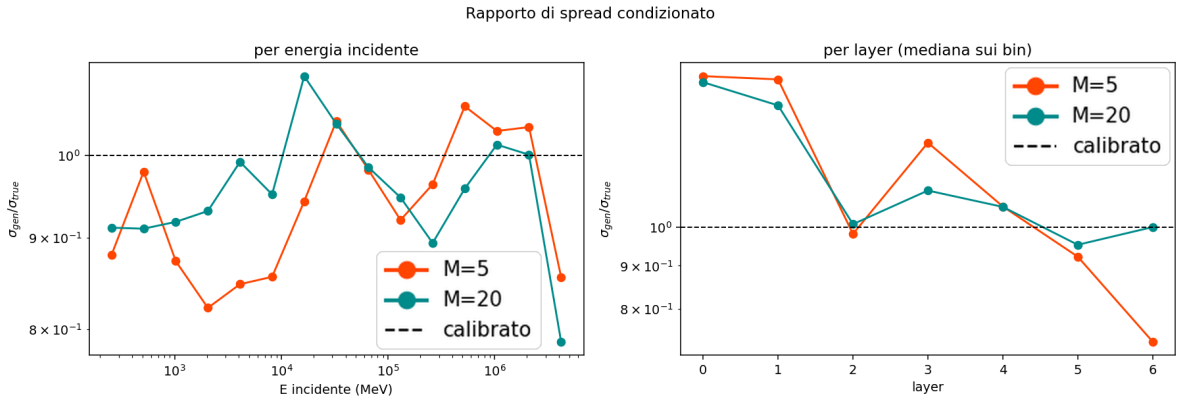


Figura 4.15: *Rapporto di spread $s = \sigma_{gen}/\sigma_{vero}$, in scala logaritmica, per i modelli con $M = 5$ e $M = 20$: a sinistra per l'energia totale in funzione dell'energia incidente, a destra per l'energia di ciascun layer, come mediana sulle 15 energie incidenti.*

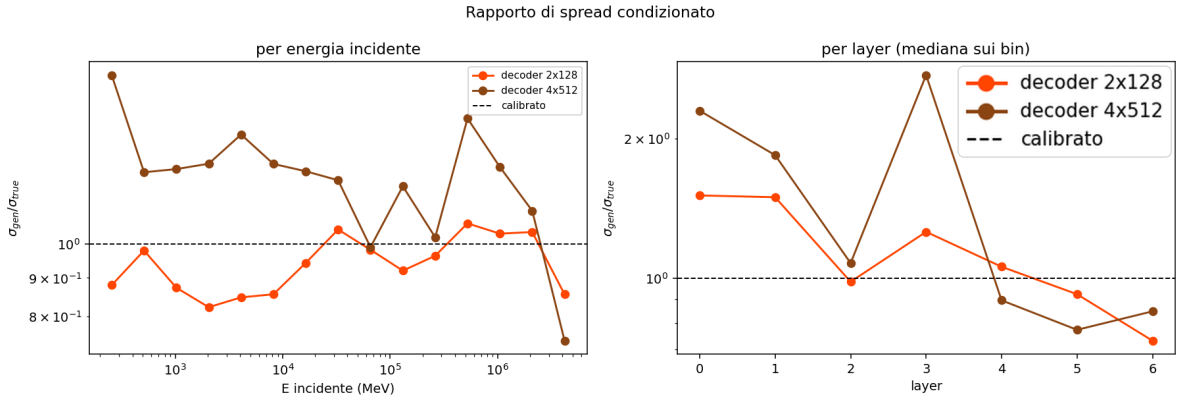


Figura 4.16: *Rapporto di spread $s = \sigma_{gen}/\sigma_{vero}$, in scala logaritmica, per i modelli con decoder 2×128 (riferimento) e 4×512: a sinistra per l'energia totale in funzione dell'energia incidente, a destra per l'energia di ciascun layer.*

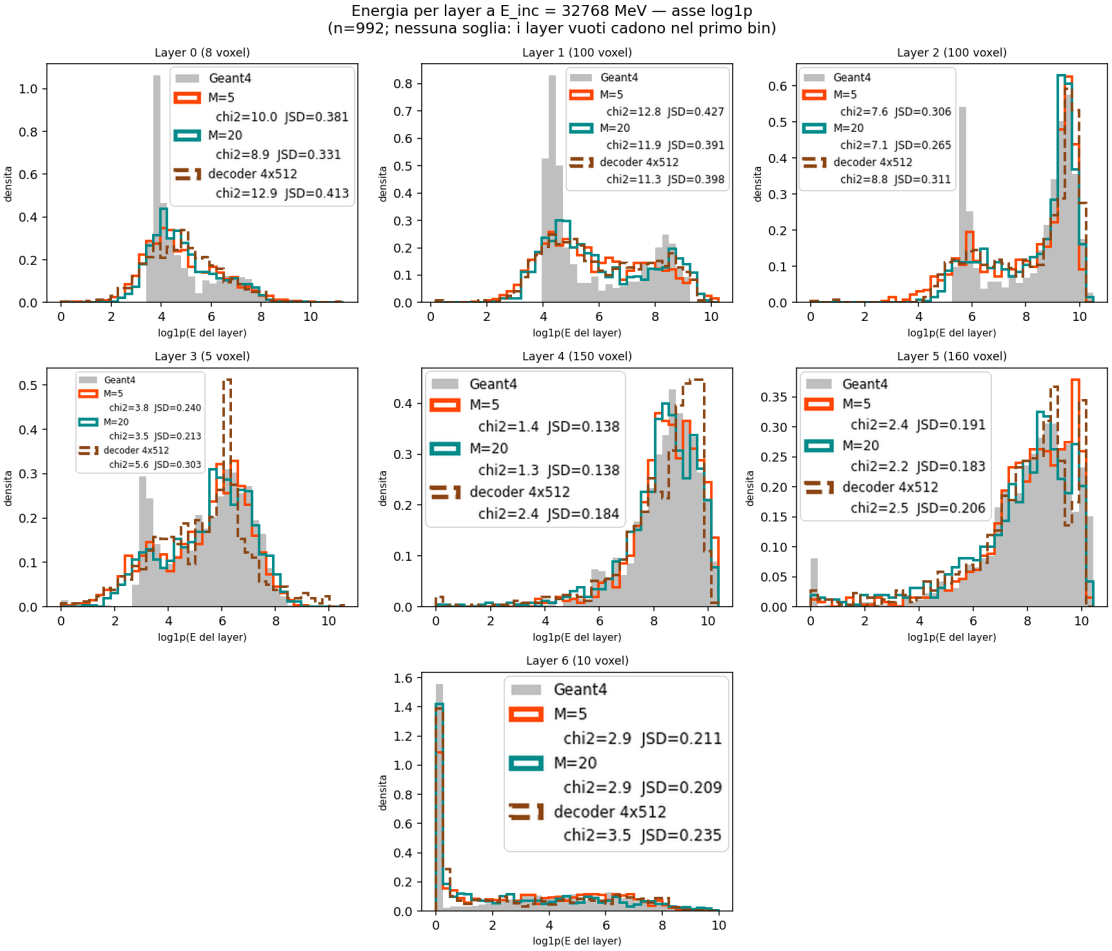


Figura 4.17: *Distribuzione dell'energia depositata in ciascuno dei sette layer, in scala* $\log1p(E_l)$*, per i 992 eventi di test con energia incidente di 32 GeV, per Geant4, per i modelli con* $M = 5$ *e* $M = 20$ *e per quello con il decoder* 4×512 *(linea tratteggiata).*

4.7.4 Confronto fra modello quantistico e classico

I confronti visti finora riguardano scelte interne a ResQNN, ma resta ancora una questione da capire: il reservoir quantistico porta effettivamente un vantaggio rispetto a un modello puramente classico? Per rispondere, ho confrontato ogni modello con un gemello classico, identico in tutto tranne che per l'assenza del circuito.
L'idea alla base del reservoir computing è che il reservoir, pur non essendo addestrato, trasformi il dato d'ingresso in una rappresentazione più ricca, dalla quale il decoder possa estrarre più facilmente l'informazione necessaria. Ci si aspetta quindi che il contributo del reservoir sia tanto più rilevante quanto minore è la capacità delle reti classiche, infatti se encoder e decoder sono abbastanza grandi possono costruire da soli una buona rappresentazione del condizionamento, mentre con pochi neuroni la rappresentazione intermedia fornita dal circuito potrebbe fare la differenza. Per questo il confronto è stato ripetuto riducendo progressivamente il numero di neuroni degli strati nascosti di encoder e decoder.

Configurazione dei modelli. Nel modello classico l'uscita dell'encoder, un vettore di n numeri reali, entra direttamente nel decoder al posto dei valori d'aspettazione $\boldsymbol{z}$:

$$\boldsymbol{y} = h_{\boldsymbol{\omega}}\left(g_{\boldsymbol{\phi}}(c),\, \boldsymbol{\xi}\right),$$

senza la sigmoide che limita gli angoli e senza alcun circuito. Encoder e decoder hanno la stessa architettura nelle coppie di modelli che si confrontano, per cui il numero di parametri addestrabili è identico. L'unica differenza è che il modello quantistico contiene in più le $3nD$ rotazioni del reservoir che sono però fisse. I due modelli di ogni coppia condividono anche il seed, gli eventi di addestramento e la funzione costo.
Tutti i modelli sono energy model addestrati con l'energy score nella configurazione post, con gli stessi parametri del modello energy score di table 4.2 (6000 eventi, 40 epoche, $d = 8$, $M = 5$, $\beta = 1$, $\lambda_{\text{tot}} = 1$), tranne il reservoir, che ha qui 10 qubit e 8 strati. Ho ripetuto il confronto per sei larghezze W degli strati nascosti di encoder e decoder, da 64 a 2 neuroni, e ogni coppia è stata addestrata con due seed diversi, così da poter, almeno in prima approssimazione, distinguere le differenze dovute al circuito da quelle dovute alla casualità dell'addestramento (table 4.7).

Valutazione. La valutazione è la stessa della sottosezione dedicata alle funzioni costo: 11123 eventi di test, un evento generato per ciascuno e le stesse estrazioni del rumore latente per tutti i modelli. Ogni modello quantistico è confrontato con il gemello classico con la stessa larghezza e lo stesso seed. Con due soli seed non è possibile un vero test di significatività. Si considera, almeno in prima approssimazione, affidabile una differenza fra i due modelli solo se ha lo stesso segno in entrambi i seed ed è maggiore della variazione che si osserva, per uno stesso modello, passando da un seed all'altro.

Qubit n e strati D del reservoir	$n = 10$, $D = 8$
Neuroni per strato nascosto W	64, 32, 16, 8, 4, 2
Parametri addestrabili	da 10834 ($W = 64$) a 108 ($W = 2$)
Seed per ogni coppia	2
Tempo per epoca di addestramento, quantistico / classico	circa 30 s / 0.1 s

Tabella 4.7: *Configurazione del confronto fra modelli quantistici e gemelli classici. Tutti gli altri parametri sono quelli del modello energy score di table 4.2. I tempi per epoca sono stati misurati sullo stesso computer (Apple M5), eseguendo un addestramento alla volta e limitando il calcolo a 4 thread per entrambi i modelli; si riferiscono al solo ciclo di addestramento di un'epoca, esclusa la prima, e variano poco con W.*

Risultati. Il risultato principale è riassunto in fig. 4.18. Con reti classiche larghe, $W = 64$ e $W = 32$, il modello classico è migliore di quello quantistico in entrambi i seed e su tutte le metriche. Ad esempio, con $W = 64$ la JSD dell'energia totale vale fra 0.14 e 0.17 per il classico contro valori fra 0.18 e 0.25 per il quantistico, e la media di $|\log s|$ per l'energia totale vale fra 0.10 e 0.13 contro valori fra 0.24 e 0.33.
Riducendo la larghezza delle reti tutte le metriche peggiorano per entrambi i modelli, con la JSD che sale fino a circa 0.47 per $W = 2$, e la differenza fra modello puramente classico e modello ibrido classico-quantistico si riduce. Già con $W = 16$ i risultati dipendono molto dal seed: la JSD del modello classico passa da 0.18 a 0.29 cambiando solo il seed. Con $W = 8$ il classico è ancora migliore in entrambi i seed e su tutte e tre le metriche. Con $W = 4$ la JSD del quantistico è leggermente minore in entrambi i seed (0.299 contro 0.302 e 0.373 contro 0.397), ma la media di $|\log s|$ per l'energia totale favorisce il classico in entrambi i seed. Bisogna però notare che tali differenze fra i due modelli sono più piccole della variazione fra un seed e l'altro per uno stesso modello. Con $W = 2$, infine, i due modelli sono praticamente equivalenti: la JSD vale 0.465 contro 0.467 e 0.471 contro 0.485, e come si vede in fig. 4.19 le distribuzioni generate sono quasi sovrapposte.
Inoltre, a parità di parametri addestrabili, il modello quantistico è molto più costoso. Misurando i due modelli nelle stesse condizioni, cioè sullo stesso computer, un addestramento alla volta e con lo stesso numero di thread, un'epoca di addestramento richiede circa 30 s per il modello quantistico contro circa 0.1 s per il gemello classico, cioè circa 300 volte di più, quasi indipendentemente da W (table 4.7). Sulle 40 epoche dell'addestramento si tratta di circa 20 minuti contro pochi secondi.

Discussione. Nei test che ho effettuato il reservoir quantistico non ha portato a un miglioramento significativo. Con reti classiche larghe ($W \geq 8$) il modello quantistico è peggiore del gemello classico, invece riducendo la capacità delle reti le differenze diventano piccole e confrontabili con quelle dovute alla casualità dell'addestramento. Il vantaggio atteso nel caso di reti piccole, quindi, non si osserva.

Questo risultato va però letto entro i limiti dei test effettuati: riguarda il solo energy model, il cui condizionamento è molto semplice, un unico numero che assume solamente 15 valori; riguarda un solo reservoir, con una struttura fissata, e due soli seed per ogni configurazione. Sono attualmente in corso dei test in cui encoder e decoder hanno un solo strato nascosto, così da ridurre ulteriormente la capacità della parte classica. Inoltre è possibile che il reservoir risulti più utile per lo shape model, il cui condizionamento è più complesso, dato che comprende l'energia incidente, l'energia del layer, che varia in modo continuo, e l'indice del layer. In un compito di questo tipo una rappresentazione intermedia ricca potrebbe avere un ruolo maggiore.

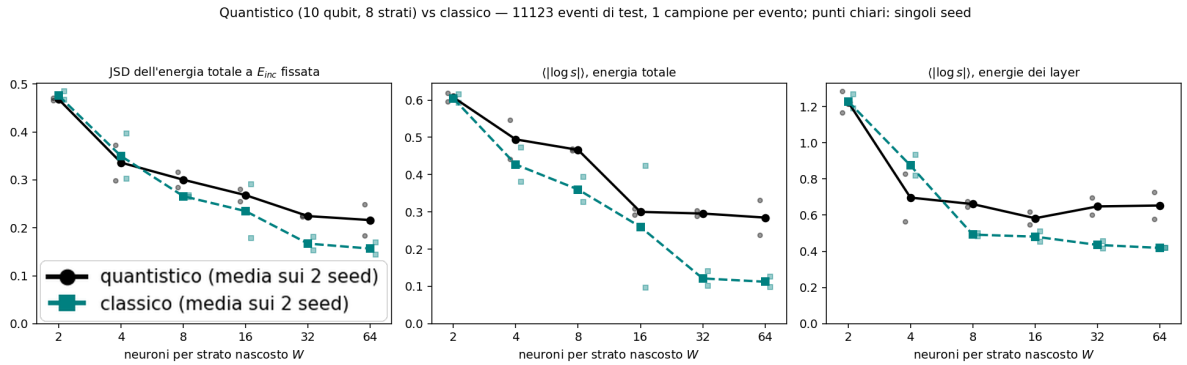


Figura 4.18: *JSD dell'energia totale a energia incidente fissata (a sinistra), media di* $|\log s|$ *per l'energia totale (al centro) e per le energie dei layer (a destra), in funzione del numero* W *di neuroni degli strati nascosti. Le linee indicano la media sui due seed, i punti chiari i singoli seed.*

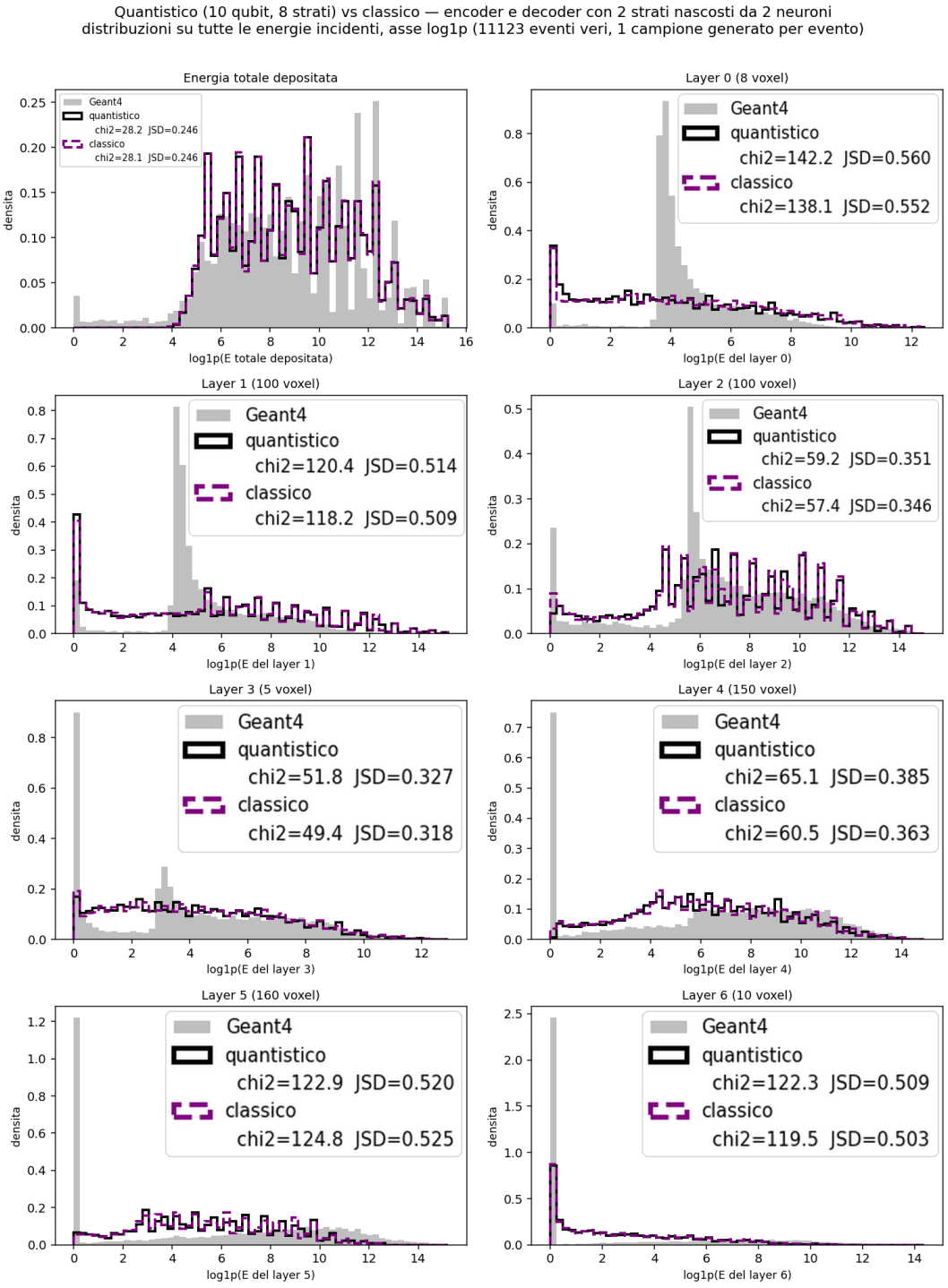


Figura 4.19: *Distribuzione dell'energia totale e delle energie dei sette layer su tutti gli 11123 eventi di test, in scala* $\log 1\mathrm{p}(E_l)$*, per Geant4, per il modello quantistico e per il suo gemello classico, entrambi con due soli neuroni per strato nascosto (seed 42).*

Conclusioni

In questa tesi è stato presentato e valutato ResQNN: un modello ibrido classico-quantistico per la generazione degli shower calorimetrici basato sul paradigma del Quantum Reservoir Computing.
In particolare lo studio è stato riservato all'energy model, lo stadio del modello avente il compito di generare, data l'energia incidente, le energie depositate nei layer del calorimetro. L'analisi è stata svolta su un dataset di soli pioni incidenti fornito dalla CaloChallenge.
Il primo risultato riguarda la funzione costo. Le due funzioni costo regressive, con cui ResQNN è stato inizialmente sviluppato, portano l'energy model a ignorare il rumore latente: il modello impara un valore tipico delle energie dei layer, ma non la loro distribuzione. Il jitter, utilizzato per reintrodurre a posteriori la variabilità mancante, non risolve il problema, poiché allarga la distribuzione degli eventi generati sempre della stessa quantità relativa. L'energy score, che premia esplicitamente la dispersione fra i campioni generati con lo stesso condizionamento, risolve invece questo problema, infatti il modello apprende direttamente dai dati una dispersione che dipende dall'energia incidente.
Il secondo risultato riguarda il punto in cui il rumore latente entra nel blocco ibrido. Le configurazioni pre e post generano eventi di qualità molto simile, ma nella configurazione post il circuito viene valutato una sola volta per condizionamento, per cui ogni passo di addestramento con l'energy score è circa M volte più veloce e generare più eventi con lo stesso condizionamento richiede una sola valutazione del circuito.
Il principale problema ancora aperto riguarda invece la ripartizione dell'energia fra i layer: nessun modello riproduce i picchi stretti delle distribuzioni delle energie dei primi quattro layer, la dispersione dei singoli layer è sbagliata fino a un fattore 1.6 e i layer non sono mai esattamente vuoti. Questi difetti non scompaiono potenziando il modello: aumentare da 5 a 20 il numero di campioni dell'energy score migliora quasi tutte le metriche, mentre un decoder con circa 38 volte i parametri del modello base non migliora particolarmente la propria capacità generativa. Ciò suggerisce che tali limiti dipendano da altri aspetti dell'architettura.
Infine, in un test effettuato, il reservoir quantistico non ha portato a un miglioramento significativo rispetto a un gemello classico, identico in tutto tranne che per l'assenza del

circuito. Usando reti classiche formate da 2 strati con un numero di neuroni $W \geq 8$ il modello classico è migliore, mentre riducendo la capacità di encoder e decoder le differenze diventano confrontabili con quelle dovute alla casualità dell'addestramento. Inoltre, con il circuito simulato su CPU, l'addestramento del modello quantistico richiede circa 300 volte il tempo di quello classico.
Questi risultati vanno però letti entro i limiti dello studio svolto. I test riguardano il solo energy model, il cui condizionamento è un unico numero che assume solamente 15 valori, mentre lo shape model non è stato valutato. Ogni confronto si basa inoltre su pochi addestramenti, in alcuni casi su modelli piccoli addestrati su 6000 eventi, e su un'unica struttura del reservoir, simulata in modo esatto, cioè senza il rumore dovuto a un numero finito di misure né quello di un dispositivo quantistico reale.
Da qui seguono i possibili sviluppi futuri. Per il problema della ripartizione fra i layer in primis è necessario effettuare uno sweep degli iperparametri dell'energy score per capire la vera entità della questione. Se il problema persiste, si potrebbe provare a introdurre nella funzione costo un nuovo termine che favorisca la somiglianza delle distribuzioni energetiche nei layer oppure provare nuove funzioni costo. Per rendere più affidabili i confronti sarebbe necessario ripetere ogni test con più seed. Per capire meglio il ruolo del reservoir si potrebbe ridurre ulteriormente la capacità delle reti classiche, ad esempio con un solo strato nascosto, variare la struttura del circuito e confrontare il reservoir quantistico con uno classico, ad esempio una RNN con parametri fissati. Resta poi da valutare lo shape model, il cui condizionamento più complesso potrebbe rendere il reservoir più utile, e lo shower completo con le metriche della CaloChallenge, eventualmente estendendo il modello ai dataset 2 e 3. Infine sarebbe interessante studiare l'effetto di un numero finito di misure e l'esecuzione su un computer quantistico reale, per la quale la configurazione post è particolarmente adatta, dato che riduce al minimo il numero di valutazioni del circuito.
In sostanza ResQNN, nella sua forma attuale, riproduce bene la scala complessiva degli shower ma non ancora il modo in cui l'energia si distribuisce fra i layer, e nei test effettuati il reservoir quantistico non ha mostrato un vantaggio rispetto a un modello classico. I risultati ottenuti indicano però quali parti del modello migliorare e quali test effettuare per capire se, e in quali condizioni, un reservoir quantistico possa contribuire alla simulazione veloce dei calorimetri.

Appendice A

Riepilogo degli iperparametri

Di seguito sono riportati gli iperparametri di ResQNN, cioè quelle quantità che, a differenza dei pesi di encoder e decoder, non vengono apprese durante l'addestramento ma vanno fissate a priori. Per ciascuno riportiamo il nome dell'opzione con cui viene impostato nel codice, i valori che accetta, il simbolo con cui compare nel capitolo 4, se presente, e una breve descrizione del suo ruolo. Gli iperparametri sono mostrati, divisi per tipologia, nelle tabelle delle pagine seguenti. Da notare che quelli del blocco ibrido e del rumore latente sono comuni all'energy model e allo shape model. Nelle tabelle non sono riportate alcune opzioni del codice che non fanno parte del modello vero e proprio, ma riguardano la gestione dei file, la verifica del funzionamento del codice o le sole metriche calcolate durante l'addestramento. Precisamente:

- `--data-path` e `--output-dir` indicano rispettivamente il file HDF5 del dataset da cui vengono letti gli eventi e la cartella in cui vengono salvati checkpoint, metriche e grafici;
- `--geometry-dir`, `--geometry-xml` e `--particle` permettono di indicare dove si trova il file di geometria della CaloChallenge, oppure di utilizzarne uno diverso da quello associato al dataset scelto, mentre `--dump-geometry` salva la geometria letta così da poterla controllare. Tutte e quattro vengono utilizzate solo dallo shape model coordinate-aware, l'unico che necessita della geometria del calorimetro;
- `--demo` sostituisce il dataset con degli shower sintetici, utili solamente a verificare che il codice funzioni correttamente;
- `--ibmq-backend` e `--ibmq-token` indicano il dispositivo IBM e la chiave di accesso necessari per un'esecuzione su un computer quantistico reale;
- `--eval-samples`, `--active-threshold-rel` e `--active-threshold-abs` intervengono solo nelle metriche calcolate durante e al termine dell'addestramento. La prima fissa il numero di eventi utilizzati per la validazione ad ogni epoca e per il test

finale, mentre le altre due definiscono la soglia, rispettivamente relativa all'energia media dei voxel dell'evento e assoluta, oltre la quale un voxel viene considerato attivo.

Opzione	**Valori accettati**	**Simbolo**	**Descrizione**
`--n-qubits`	intero, al più 20	n	Numero di qubit del reservoir. Il costo della simulazione cresce come 2^n.
`--n-layers-reservoir`	intero, al più 15	D	Numero di strati del reservoir.
`--hidden-dim`	intero positivo	W	Neuroni di ciascuno dei due strati nascosti di encoder e decoder. Controlla la capacità della parte classica del blocco.
`--seed`	intero	–	Seme dei generatori casuali: fissa gli angoli del reservoir, l'inizializzazione delle reti, la divisione degli eventi e l'ordine dei batch, rendendo l'addestramento riproducibile.
`--backend`	`simulation`, `lightning`, `ibmq`	–	Strumento con cui viene valutato il circuito: `simulation` è il simulatore a vettore di stato di PennyLane, `lightning` una sua versione ottimizzata. L'hardware IBM (`ibmq`) è pensato per una possibile esecuzione su un circuito quantistico in futuro.
`--n-shots`	intero positivo, oppure non impostato	–	Numero di misure con cui stimare i valori d'aspettazione. Se non impostato questi vengono calcolati in modo esatto.

Tabella A.1: *Iperparametri del blocco ibrido, comuni all'energy model e allo shape model.*

Opzione	**Valori accettati**	**Simbolo**	**Descrizione**
`--latent-dim`	intero positivo	d	Numero di componenti del rumore latente di ciascun blocco ibrido.
`--noise-mode`	`pre`, `post`	–	Punto in cui il rumore entra nel blocco: insieme al condizionamento (`pre`) o dopo il reservoir (`post`). Con `post` il circuito viene valutato una sola volta per tutti i campioni generati con lo stesso condizionamento.
`--energy-noise-scale`	reale ≥ 0	σ_ξ	Deviazione standard del rumore latente dell'energy model, la stessa in addestramento e in generazione. Con 0 il modello diventa deterministico.
`--shape-noise-scale`	reale ≥ 0	σ_ξ	Deviazione standard del rumore latente dello shape model.

Tabella A.2: *Iperparametri del rumore latente.*

Opzione	**Valori accettati**	**Simbolo**	**Descrizione**
`--energy-scale`	reale > 0	s_E	Divisore di $\log(1+E)$ nel condizionamento. Utilizzato anche dallo shape model.
`--max-visible-fraction`	reale > 0	$v_{\max}$	Tetto della frazione visibile, cioè il massimo rapporto fra energia depositata ed energia incidente che il modello può generare.
`--loss`	`regression`, `regression-with-lambda`, `energy-score`	–	Funzione costo dell'energy model: una delle due versioni regressive oppure l'energy score.
`--energy-score-samples`	intero ≥ 2	M	Campioni generati per ogni evento vero nella stima dell'energy score.
`--energy-score-beta`	reale in $(0, 2)$	β	Esponente delle distanze nell'energy score. Fuori da questo intervallo la regola non è più strettamente propria.
`--lambda-total-score`	reale ≥ 0	λ_{tot}	Peso del termine ES_{tot} nella loss con energy score.
`--lambda-fraction`	reale ≥ 0	λ_{frac}	Peso del termine sulle frazioni dei layer. Usato solo con `regression-with-lambda`.
`--lambda-variance`	reale ≥ 0	λ_{var}	Peso del termine unilaterale sulla varianza. Usato solo con `regression-with-lambda`.

Tabella A.3: *Iperparametri dell'energy model e della sua funzione costo.*

Opzione	Valori accettati	Simbolo	Descrizione
`--shape-model`	`flat`, `coordinate`	–	Versione dello shape model: un'uscita per ogni voxel (`flat`) oppure con un ulteriore decoder condiviso fra i layer che costruisce la forma (`coordinate`).
`--lambda-voxel`	reale ≥ 0	λ_{vox}	Peso dell'errore quadratico sui logaritmi delle energie dei voxel.
`--geometry-coordinates`	`alpharz`, `etaphi_rz`	–	Coordinate dei voxel fornite al decoder dello shape model coordinate-aware: `alpharz` è la rappresentazione descritta in questa tesi. Usato solo con `coordinate`.
`--layer-z-file`	percorso di un file JSON, oppure non impostato	–	Posizioni dei layer lungo la direzione dello shower, usate come coordinata di profondità. Se non impostato, la profondità coincide con l'indice normalizzato del layer.

Tabella A.4: *Iperparametri dello shape model.*

Opzione	Valori accettati	Simbolo	Descrizione
`--dataset`	`dataset_1_photons`, `dataset_1_pions`, `dataset_2`, `dataset_3`	–	Dataset della CaloChallenge di cui viene letta la geometria. In questa tesi `dataset_1_pions`.
`--layer-sizes`	nome di un dataset, oppure lista di interi separati da virgole	–	Numero di voxel di ciascun layer, utilizzato per dividere lo shower nei layer. Con lo shape model coordinate-aware viene ricavato direttamente dalla geometria.
`--nEv`	intero positivo	–	Numero di eventi caricati dal file, divisi in training, validazione e test nelle proporzioni 70%, 15% e 15%.
`--stage`	`both`, `energy`, `shape`	–	Stadi da addestrare: entrambi, solo l'energy model o solo lo shape model.
`--epochs`	intero positivo	–	Numero di epoche dell'addestramento.
`--batch-size`	intero positivo	–	Numero di eventi utilizzati per ogni aggiornamento dei pesi.
`--lr`	reale > 0	–	Learning rate dell'ottimizzatore Adam.

Tabella A.5: *Iperparametri relativi ai dati e all'addestramento.*

Opzione	Valori accettati	Simbolo	Descrizione
`--energy-sampling-sigma`	reale ≥ 0	σ_s	Ampiezza del jitter log-normale applicato alle energie generate.
`--shape-temperature`	reale > 0	T	Temperatura applicata alle forme generate.

Tabella A.6: *Iperparametri che agiscono solo al momento della generazione.*

Bibliografia

[1] Florian Rehm. «Deep learning and quantum generative models for high energy physics calorimeter simulations». Tesi di dott. Rheinisch Westfaelische Tech Hoch DE, apr. 2023.

[2] Jakub M. Tomczak. «Flow-Based Models». In: *Deep Generative Modeling*. Cham: Springer International Publishing, 2022, pp. 27–56. ISBN: 978-3-030-93158-2. DOI: 10.1007/978-3-030-93158-2_3. URL: https://doi.org/10.1007/978-3-030-93158-2_3.

[3] Ivan Kobyzev, Simon J.D. Prince e Marcus A. Brubaker. «Normalizing Flows: An Introduction and Review of Current Methods». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.11 (nov. 2021), pp. 3964–3979. ISSN: 1939-3539. DOI: 10.1109/tpami.2020.2992934. URL: http://dx.doi.org/10.1109/TPAMI.2020.2992934.

[4] Dor Bank, Noam Koenigstein e Raja Giryes. *Autoencoders*. 2021. arXiv: 2003.05991 [cs.LG]. URL: https://arxiv.org/abs/2003.05991.

[5] Ankush Ganguly e Samuel W. F. Earp. *An Introduction to Variational Inference*. 2021. arXiv: 2108.13083 [cs.LG]. URL: https://arxiv.org/abs/2108.13083.

[6] Ian J. Goodfellow et al. «Generative Adversarial Nets». In: *Advances in Neural Information Processing Systems*. A cura di Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/f033ed80deb0234979a61f95710dbe25-Paper.pdf.

[7] Moez Krichen. «Generative Adversarial Networks». In: *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. 2023, pp. 1–7. DOI: 10.1109/ICCCNT56998.2023.10306417.

[8] Ling Yang et al. *Diffusion Models: A Comprehensive Survey of Methods and Applications*. 2025. arXiv: 2209.00796 [cs.LG]. URL: https://arxiv.org/abs/2209.00796.

[9] Yang Song et al. *Score-Based Generative Modeling through Stochastic Differential Equations*. 2021. arXiv: 2011.13456 [cs.LG]. URL: https://arxiv.org/abs/2011.13456.

[10] Minshuo Chen et al. *An Overview of Diffusion Models: Applications, Guided Generation, Statistical Rates and Optimization*. 2024. arXiv: 2404.07771 [cs.LG]. URL: https://arxiv.org/abs/2404.07771.

[11] Georges Aad et al. «AtlFast3: The Next Generation of Fast Simulation in ATLAS». In: *Comput. Softw. Big Sci.* 6.1 (2022), p. 7. DOI: 10.1007/s41781-021-00079-7. arXiv: 2109.02551 [hep-ex].

[12] Simone Campana et al. *HEP computing collaborations for the challenges of the next decade*. 2022. arXiv: 2203.07237 [physics.comp-ph]. URL: https://arxiv.org/abs/2203.07237.

[13] Anja Butter et al. «Machine learning and LHC event generation». In: *SciPost Physics* 14 (apr. 2023). DOI: 10.21468/SciPostPhys.14.4.079.

[14] Christina Gao et al. «Event generation with normalizing flows». In: *Phys. Rev. D* 101 (7 apr. 2020), p. 076002. DOI: 10.1103/PhysRevD.101.076002. URL: https://link.aps.org/doi/10.1103/PhysRevD.101.076002.

[15] S. Agostinelli et al. «GEANT4 - A Simulation Toolkit». In: *Nucl. Instrum. Meth. A* 506 (2003), pp. 250–303. DOI: 10.1016/S0168-9002(03)01368-8.

[16] Christian Bierlich et al. «A comprehensive guide to the physics and usage of PYTHIA 8.3». In: *SciPost Phys. Codebases* (2022), p. 8. DOI: 10.21468/SciPostPhysCodeb.8. URL: https://scipost.org/10.21468/SciPostPhysCodeb.8.

[17] Enrico Bothmann et al. *Efficient many-jet event generation with Flow Matching*. 2025. arXiv: 2506.18987 [hep-ph]. URL: https://arxiv.org/abs/2506.18987.

[18] Florencia Canelli et al. «A practical guide to unbinned unfolding». In: *Eur. Phys. J. C* 86.2 (2026), p. 106. DOI: 10.1140/epjc/s10052-025-15265-9. arXiv: 2507.09582 [hep-ph].

[19] Benjamin Nachman e David Shih. «Anomaly detection with density estimation». In: *Phys. Rev. D* 101 (7 apr. 2020), p. 075042. DOI: 10.1103/PhysRevD.101.075042. URL: https://link.aps.org/doi/10.1103/PhysRevD.101.075042.

[20] Anja Butter et al. «GANplifying event samples». In: *SciPost Phys.* 10 (2021), p. 139. DOI: 10.21468/SciPostPhys.10.6.139. URL: https://scipost.org/10.21468/SciPostPhys.10.6.139.

[21] I. Dawson, cur. *Radiation effects in the LHC experiments: Impact on detector performance and operation*. Vol. 1/2021. CERN Yellow Reports: Monographs. Geneva: CERN, 2021. ISBN: 978-92-9083-590-5, 978-92-9083-591-2. DOI: 10.23731/CYRM-2021-001.

[22] Ewa Lopienska. «The CERN accelerator complex, layout in 2022». In: (2022). General Photo. URL: https://cds.cern.ch/record/2800984.

[23] Lyndon Evans e Philip Bryant. «LHC Machine». In: *Journal of Instrumentation* 3.08 (ago. 2008), S08001. DOI: 10.1088/1748-0221/3/08/S08001. URL: https://doi.org/10.1088/1748-0221/3/08/S08001.

[24] Georges Aad et al. «Luminosity determination in pp collisions at $\sqrt{s} = 13$ TeV using the ATLAS detector at the LHC». In: *Eur. Phys. J. C* 83.10 (2023), p. 982. DOI: 10.1140/epjc/s10052-023-11747-w. arXiv: 2212.09379 [hep-ex].

[25] «Preliminary luminosity calibration of the ATLAS 13.6 TeV data recorded in 2024 and combination with the 2022 and 2023 measurements». In: (2025).

[26] Georges Aad et al. «The ATLAS experiment at the CERN Large Hadron Collider: a description of the detector configuration for Run 3». In: *JINST* 19.05 (2024), P05063. DOI: 10.1088/1748-0221/19/05/P05063. arXiv: 2305.16623 [physics.ins-det].

[27] CERN. *The Large Hadron Collider*. CERN. URL: https://home.cern/science/accelerators/large-hadron-collider/ (visitato il giorno 10/09/2026).

[28] ATLAS Collaboration. «The ATLAS Experiment at the CERN Large Hadron Collider». In: *Journal of Instrumentation* 3.08 (ago. 2008), S08003. DOI: 10.1088/1748-0221/3/08/S08003. URL: https://doi.org/10.1088/1748-0221/3/08/S08003.

[29] Peilian Liu. «Expected performance of the upgrade ATLAS experiment for HL-LHC». In: (2018). URL: https://cds.cern.ch/record/2621410.

[30] ATLAS Collaboration. *Keeping the ATLAS Inner Detector in perfect alignment*. URL: https://atlas.cern/updates/experiment-briefing/inner-detector-alignment (visitato il giorno 2020).

[31] Federico Andrea Guillaume Corchia et al. «AtlFast3: Fast Simulation in ATLAS for LHC Run 3 and Beyond». In: *EPJ Web Conf.* 337 (2025), p. 01355. DOI: 10.1051/epjconf/202533701355.

[32] G. Aad et al. «Software and computing for Run 3 of the ATLAS experiment at the LHC». In: *Eur. Phys. J. C* 85.3 (2025). [Erratum: Eur.Phys.J.C 85, 907 (2025)], p. 234. DOI: 10.1140/epjc/s10052-024-13701-w. arXiv: 2404.06335 [hep-ex].

[33] Jacob Biamonte et al. «Quantum machine learning». In: *Nature* 549.7671 (set. 2017), pp. 195–202. ISSN: 1476-4687. DOI: 10.1038/nature23474. URL: http://dx.doi.org/10.1038/nature23474.

[34] Kwok Ho Wan et al. «Quantum generalisation of feedforward neural networks». In: *npj Quantum Information* 3.1 (set. 2017). ISSN: 2056-6387. DOI: 10.1038/s41534-017-0032-4. URL: http://dx.doi.org/10.1038/s41534-017-0032-4.

[35] Alessandro Giovagnoli, Yunpu Ma e Volker Tresp. *QNEAT: Natural Evolution of Variational Quantum Circuit Architecture*. 2023. arXiv: 2304.06981 [quant-ph]. URL: https://arxiv.org/abs/2304.06981.

[36] Ivana Miháliková, Michal Krejčí e Martin Friák. «The impact of quantum circuit architecture and hyperparameters on variational quantum algorithms exemplified in the electronic structure of the GaAs crystal». In: *Scientific Reports* 15 (mag. 2025). DOI: 10.1038/s41598-025-00151-x.

[37] M. Cerezo et al. «Variational quantum algorithms». In: *Nature Reviews Physics* 3.9 (ago. 2021), pp. 625–644. ISSN: 2522-5820. DOI: 10.1038/s42254-021-00348-9. URL: http://dx.doi.org/10.1038/s42254-021-00348-9.

[38] Darya Martyniuk, Johannes Jung e Adrian Paschke. «Quantum Architecture Search: A Survey». In: *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 01. 2024, pp. 1695–1706. DOI: 10.1109/QCE60285.2024.00198.

[39] Haozhen Situ et al. «AutoML-driven optimization of variational quantum circuit». In: *Information Sciences* 717 (2025), p. 122272. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2025.122272. URL: https://www.sciencedirect.com/science/article/pii/S0020025525004049.

[40] Ben Poole et al. «Exponential expressivity in deep neural networks through transient chaos». In: *Advances in Neural Information Processing Systems*. A cura di D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper_files/paper/2016/file/148510031349642de5ca0c544f31b2ef-Paper.pdf.

[41] Maithra Raghu et al. *Survey of Expressivity in Deep Neural Networks*. 2016. arXiv: 1611.08083 [stat.ML]. URL: https://arxiv.org/abs/1611.08083.

[42] Sukin Sim, Peter D. Johnson e Alán Aspuru-Guzik. «Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms». In: *Advanced Quantum Technologies* 2.12 (ott. 2019). ISSN: 2511-9044. DOI: 10.1002/qute.201900070. URL: http://dx.doi.org/10.1002/qute.201900070.

[43] Jarrod R. McClean et al. «Barren plateaus in quantum neural network training landscapes». In: *Nature Communications* 9.1 (nov. 2018). ISSN: 2041-1723. DOI: 10.1038/s41467-018-07090-4. URL: http://dx.doi.org/10.1038/s41467-018-07090-4.

[44] M. Cerezo et al. «Cost function dependent barren plateaus in shallow parametrized quantum circuits». In: *Nature Communications* 12.1 (mar. 2021). ISSN: 2041-1723. DOI: 10.1038/s41467-021-21728-w. URL: http://dx.doi.org/10.1038/s41467-021-21728-w.

[45] Samson Wang et al. «Noise-induced barren plateaus in variational quantum algorithms». In: *Nature Communications* 12.1 (nov. 2021). ISSN: 2041-1723. DOI: 10.1038/s41467-021-27045-6. URL: http://dx.doi.org/10.1038/s41467-021-27045-6.

[46] Carlos Ortiz Marrero, Mária Kieferová e Nathan Wiebe. *Entanglement Induced Barren Plateaus*. 2021. arXiv: 2010.15968 [quant-ph]. URL: https://arxiv.org/abs/2010.15968.

[47] Brian Coyle et al. «The Born supremacy: quantum advantage and training of an Ising Born machine». In: *npj Quantum Information* 6.1 (lug. 2020). ISSN: 2056-6387. DOI: 10.1038/s41534-020-00288-9. URL: http://dx.doi.org/10.1038/s41534-020-00288-9.

[48] Ieva Čepaitė, Brian Coyle e Elham Kashefi. «A continuous variable Born machine». In: *Quantum Machine Intelligence* 4.1 (mar. 2022). ISSN: 2524-4914. DOI: 10.1007/s42484-022-00063-3. URL: http://dx.doi.org/10.1007/s42484-022-00063-3.

[49] Jin-Guo Liu e Lei Wang. «Differentiable learning of quantum circuit Born machines». In: *Phys. Rev. A* 98 (6 dic. 2018), p. 062324. DOI: 10.1103/PhysRevA.98.062324. URL: https://link.aps.org/doi/10.1103/PhysRevA.98.062324.

[50] Seth Lloyd e Christian Weedbrook. «Quantum Generative Adversarial Learning». In: *Physical Review Letters* 121.4 (lug. 2018). ISSN: 1079-7114. DOI: 10.1103/physrevlett.121.040502. URL: http://dx.doi.org/10.1103/PhysRevLett.121.040502.

[51] Pierre-Luc Dallaire-Demers e Nathan Killoran. «Quantum generative adversarial networks». In: *Physical Review A* 98.1 (lug. 2018). ISSN: 2469-9934. DOI: 10.1103/physreva.98.012324. URL: http://dx.doi.org/10.1103/PhysRevA.98.012324.

[52] Nidhi Munikote. *Comparing Quantum Encoding Techniques*. 2024. arXiv: 2410.09121 [quant-ph]. URL: https://arxiv.org/abs/2410.09121.

[53] Florian Rehm et al. *A Full Quantum Generative Adversarial Network Model for High Energy Physics Simulations*. 2024. arXiv: 2305.07284 [quant-ph]. URL: https://arxiv.org/abs/2305.07284.

[54] Antonio Tudisco et al. «Evaluating Angle and Amplitude Encoding Strategies for Variational Quantum Machine Learning: Their Impact on Model's Accuracy». In: *Advanced Quantum Technologies* 9.4 (apr. 2026). ISSN: 2511-9044. DOI: 10.1002/qute.202500611. URL: http://dx.doi.org/10.1002/qute.202500611.

[55] Armand Rousselot e Michael Spannowsky. «Generative invertible quantum neural networks». In: *SciPost Physics* 16 (giu. 2024). DOI: 10.21468/SciPostPhys.16.6.146.

[56] Michael te Vrugt. «An Introduction to Reservoir Computing». In: *Artificial Intelligence and Intelligent Matter*. Springer Nature Switzerland, 2026, pp. 53–66. ISBN: 9783032041296. DOI: 10.1007/978-3-032-04129-6_4. URL: http://dx.doi.org/10.1007/978-3-032-04129-6_4.

[57] Gouhei Tanaka et al. «Recent advances in physical reservoir computing: A review». In: *Neural Networks* 115 (2019), pp. 100–123. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2019.03.005. URL: https://www.sciencedirect.com/science/article/pii/S0893608019300784.

[58] Keisuke Fujii e Kohei Nakajima. «Harnessing Disordered-Ensemble Quantum Dynamics for Machine Learning». In: *Phys. Rev. Appl.* 8 (2 ago. 2017), p. 024030. DOI: 10.1103/PhysRevApplied.8.024030. URL: https://link.aps.org/doi/10.1103/PhysRevApplied.8.024030.

[59] Keisuke Fujii e Kohei Nakajima. «Quantum Reservoir Computing: A Reservoir Approach Toward Quantum Machine Learning on Near-Term Quantum Devices». In: *Reservoir Computing: Theory, Physical Implementations, and Applications*. A cura di Kohei Nakajima e Ingo Fischer. Singapore: Springer Singapore, 2021, pp. 423–450. ISBN: 978-981-13-1687-6. DOI: 10.1007/978-981-13-1687-6_18. URL: https://doi.org/10.1007/978-981-13-1687-6_18.

[60] Alberto Di Meglio et al. «Quantum Computing for High-Energy Physics: State of the Art and Challenges». In: *PRX Quantum* 5 (3 ago. 2024), p. 037001. DOI: 10.1103/PRXQuantum.5.037001. URL: https://link.aps.org/doi/10.1103/PRXQuantum.5.037001.

[61] Sergio Calatroni et al. *Quantum Technologies in High Energy Physics: The CERN Quantum Technology Initiative input to the European Strategy for Particle Physics*. 2025. URL: https://indico.cern.ch/event/1439855/contributions/6461631/.

[62] Manuel S. Rudolph et al. «Trainability barriers and opportunities in quantum generative modeling». In: *npj Quantum Inf.* 10.1 (2024), p. 116. DOI: 10.1038/s41534-024-00902-0. arXiv: 2305.02881 [quant-ph].

[63] Claudius Krause et al. «CaloChallenge 2022: a community challenge for fast calorimeter simulation». In: *Reports on Progress in Physics* 88.11 (nov. 2025), p. 116201. DOI: 10.1088/1361-6633/ae1304. URL: https://doi.org/10.1088/1361-6633/ae1304.

[64] Ville Bergholm et al. «PennyLane: Automatic differentiation of hybrid quantum-classical computations». In: (nov. 2018). arXiv: 1811.04968 [quant-ph].

[65] Kartik Waghmare e Johanna Ziegel. «Proper scoring rules for estimation and forecast evaluation». In: *Annual Review of Statistics and Its Application* 13 (2025).