

Corso di Laurea in Informatica

TESI DI LAUREA

Studio sperimentale sull'unione di strategie di esecuzione
distribuita e incrementale per sistemi quantistici NISQ

Relatore: Antonio Brogi

Candidato: Giovanni Del Bianco

Correlatore: Giuseppe Bisicchia,
Alessandro Bocci

ANNO ACCADEMICO 2025/2026

Indice

Introduzione	1
Il Contesto: Il Calcolo Quantistico nell’Era NISQ	1
Strategie per l’Esecuzione Efficiente	2
Distribuzione Shot-wise su più QPU	2
Esecuzione Incrementale Adattiva	4
Definizione del Problema e Obiettivi della Tesi	5
Struttura della Tesi	7
1 Stato dell’Arte	9
1.1 Strategie di Ottimizzazione Adattiva degli Shot	10
1.2 Framework Formali per la Minimizzazione degli Shot	11
1.3 Approcci basati su Intelligenza Artificiale	12
1.4 Posizionamento del Lavoro e Contributo della Tesi	13
2 Framework di Riferimento	15
2.1 Il Paradigma dell’Esecuzione Distribuita: Quantum Executor	15
2.1.1 Scopo e Filosofia del Framework	15
2.1.2 Concetti Chiave: il Sistema a Policy	16
2.2 Il Paradigma dell’Esecuzione Incrementale	17
2.2.1 Scopo e Filosofia del Framework	17
2.2.2 Concetti Chiave: il Ciclo di Convergenza e i Criteri	18
2.3 Simulazione Realistica: il Dataset qsimbench	19
2.3.1 Scopo e Filosofia del Framework	20
2.3.2 Il Vantaggio Cruciale: Dati Realistici, Riproducibili e su Larga Scala	20
2.4 Progettazione delle Architetture di Integrazione	21
3 Progettazione dell’Architettura e Analisi Preliminare	23
3.1 Analisi Esplorativa della Convergenza Distribuita	23

3.1.1	Metodologia dell’Analisi di Baseline	23
3.1.2	Risultati: Caratterizzazione della Convergenza	24
3.1.3	Motivazioni per un’Architettura Ibrida	26
3.2	Modelli di Integrazione e Gerarchia di Controllo	27
3.3	Architettura a Convergenza Globale	28
3.3.1	Architettura e Flusso Operativo	28
3.3.2	Proprietà Fondamentali e Vantaggi Ipotizzati	31
3.3.3	Compromessi Intrinseci e Strategie di Mitigazione	32
3.4	Architettura a Convergenza Locale	32
3.4.1	Architettura e Flusso Operativo	33
3.4.2	Criticità Fondamentali e Vizi Metodologici Ipotizzati	36
3.5	Analisi Comparativa e Scelta Architetturale	37
3.5.1	Confronto Diretto: Sintesi delle Proprietà Ipotizzate	37
3.5.2	Analisi di Robustezza: il Problema della “Falsa Convergenza”	38
3.5.3	Formulazione dell’Ipotesi Sperimentale	39
3.6	Implementazione del Prototipo	40
3.6.1	Panoramica del Software e Stack Tecnologico	40
3.6.2	Flusso Esecutivo	41
3.6.3	Architettura del Codice: la Classe Orchestrator	42
3.6.4	Il Sistema di Configurazione Data-Driven via JSON	43
4	Metodologia Sperimentale	48
4.1	Ambiente di Simulazione	48
4.2	Definizione del Benchmark: l’Oracolo	49
4.2.1	Strategia di Generazione dei Dati: i Team Virtuali	50
4.2.2	Metodologia di Analisi: l’Algoritmo a Finestra Inversa	51
4.3	Simulazione delle Architetture Dinamiche	53
4.3.1	Adattamento del Prototipo Orchestratore per la Simulazione	54
4.3.2	Metodologia di Simulazione per le Due Architetture	54
4.4	Metodologia di Valutazione Comparativa	56

4.4.1	Metrica di Performance Primaria: Differenza di Shot	56
4.4.2	Processo di Analisi Comparativa	57
4.5	Definizione della Configurazione di Test	58
5	Risultati Sperimentali e Analisi	61
5.1	Analisi per Algoritmo	61
5.1.1	Algoritmi Strutturati	62
5.1.2	Algoritmi Variazionali (VQA)	63
5.1.3	Circuiti Random e Ansatz Euristici	63
5.2	Analisi per Dimensione del Circuito	64
5.3	Discussione Critica dei Risultati	66
5.3.1	Revisione dell’Ipotesi Sperimentale	66
5.3.2	Interpretazione delle Dinamiche: Coerenza vs Indipendenza . .	67
5.3.3	Scalabilità e Complessità	67
Conclusioni		69
6	Conclusioni e Sviluppi Futuri	69
6.1	Sintesi del Lavoro Svolto	69
6.2	Riepilogo dei Risultati e Contributi	70
6.3	Sviluppi Futuri	71

Elenco delle figure

1	Illustrazione concettuale della Shot-wise Distribution. Un job quantistico viene suddiviso in batch di shot, che vengono eseguiti in parallelo su molteplici QPU. I risultati parziali vengono poi aggregati per formare la distribuzione finale.	3
2	Concetto di Esecuzione Incrementale. L'errore o la variazione della distribuzione diminuisce rapidamente all'aumentare degli shot, per poi stabilizzarsi. L'obiettivo è fermare l'esecuzione in questa regione di “diminishing returns” per ottimizzare le risorse.	5
3.1	Grafico aggregato dell'Accuratezza (TVD vs. Ground Truth) per tutte le esecuzioni simulate. L'asse X rappresenta il numero di shot cumulativi, mentre l'asse Y la distanza dalla distribuzione ideale.	25
3.2	Grafico aggregato della Stabilità (Delta-TVD tra iterazioni) per tutte le esecuzioni simulate. L'asse Y misura la variazione della distribuzione cumulativa rispetto al passo precedente.	26
3.3	Diagramma di flusso operativo per l'Architettura a Convergenza Globale. Un ciclo di controllo esterno (OrCHEstratore) gestisce lo stato dell'esperimento e delega, ad ogni iterazione, l'esecuzione di un batch di shot a un servizio di esecuzione distribuita.	30
3.4	Diagramma di flusso operativo dell'architettura a Convergenza Locale. Il controllo è distribuito, con cicli incrementali indipendenti eseguiti su ogni backend.	35
4.1	Analisi dell'Oracolo sull'accuratezza per l'algoritmo random (7 Qubit). Ogni curva rappresenta l'evoluzione della TVD rispetto alla Ground Truth per un diverso team di backend o backend singolo. I punti rossi indicano i punti di stabilità ottimali identificati dall'Oracolo.	52

4.2 Analisi della stabilità per l'algoritmo random (7 Qubit). Ogni curva mostra l'evoluzione della Delta-TVD (variazione tra iterazioni successive) per un diverso team di backend o beckend singolo, evidenziando il rapido decadimento della volatilità della stima.	53
5.1 Confronto della Differenza di Shot media (rispetto all'Oracolo) tra le due architetture, raggruppata per algoritmo. Valori più bassi indicano una maggiore efficienza.	62
5.2 Confronto della Differenza di Shot media (rispetto all'Oracolo) tra le due architetture, raggruppata per dimensione del circuito (numero di qubit).	64

Introduzione

Il Contesto: Il Calcolo Quantistico nell’Era NISQ

Il calcolo quantistico rispetto all’approccio classico rappresenta un cambio di paradigma, adoperando i principi della meccanica quantistica per affrontare problemi computazionali intrattabili per i supercomputer più potenti [15]. A differenza dei bit tradizionali, che possono trovarsi unicamente negli stati discreti 0 o 1, il *qubit*, l’unità fondamentale dell’informazione quantistica, può trovarsi in una *sovraposizione* lineare di entrambi gli stati. Questa proprietà, combinata con *entanglement*, consente a un sistema di n qubit di esplorare uno spazio computazionale di dimensione 2^n , garantendo un parallelismo potenziale di natura esponenziale [13].

Tuttavia, l’hardware quantistico contemporaneo opera in quella che John Preskill ha definito l’era *Noisy Intermediate-Scale Quantum* (NISQ) [17]. I processori di quest’era, pur superando la capacità di simulazione classica, sono caratterizzati da due limiti fondamentali: una scala intermedia (decine o poche migliaia di qubit) e un’elevata sensibilità al *rumore*. Il rumore, inteso come qualsiasi interazione indesiderata tra i qubit e il loro ambiente, degrada inevitabilmente l’informazione quantistica. Questo fenomeno, noto come *decoerenza*, si manifesta attraverso processi come il rilassamento energetico (descritto dal tempo T_1) e il defasamento (descritto dal tempo T_2), che distruggono rispettivamente l’ampiezza e la fase dello stato quantistico. A ciò si aggiungono le imperfezioni intrinseche delle operazioni (errori di gate) e del processo di misurazione (errori di readout) [1]. La combinazione di queste fonti di errore limita drasticamente la profondità (ovvero, la lunghezza) dei circuiti che possono essere eseguiti con successo, confinando l’hardware attuale a computazioni relativamente brevi.

Una delle caratteristiche più distinte del calcolo quantistico è la natura intrinsecamente probabilistica del suo output. L’esecuzione di un algoritmo quantistico prepara uno stato finale che codifica una complessa distribuzione di probabilità su tutti i 2^n possibili risultati classici. Tuttavia, l’atto della misurazione è un processo irreversibile che proietta lo

stato quantistico su uno solo di questi risultati, secondo le probabilità definite dalla regola di Born [15]. Una singola esecuzione, o *shot*, fornisce quindi un’informazione statistica molto limitata.

Per ricostruire in modo affidabile la distribuzione di probabilità finale, è necessario eseguire il medesimo circuito ripetutamente, tipicamente migliaia o milioni di volte, e aggregare i risultati. Questo processo di campionamento statistico è fondamentale ma rappresenta anche una risorsa computazionale critica e costosa, sia in termini di tempo di accesso all’hardware sia, in molti casi, di costi economici diretti [4, 8]. Il numero di shot influenza direttamente l’accuratezza della stima: un numero insufficiente introduce un significativo *shot noise* (errore di campionamento), mentre un numero eccessivo porta a uno spreco di risorse preziose. La gestione efficiente del budget di shot è, pertanto, una delle sfide centrali per rendere pratico ed economicamente sostenibile l’utilizzo dei computer quantistici nell’era NISQ.

Strategie per l’Esecuzione Efficiente

Per affrontare il dilemma dell’esecuzione efficiente la ricerca si è concentrata su due approcci ortogonali ma potenzialmente complementari. Da un lato, si cerca di ottimizzare l’uso dell’infrastruttura hardware disponibile; dall’altro, si mira a rendere più intelligente il processo di campionamento statistico. Questa tesi esplora l’integrazione di queste due filosofie.

Distribuzione Shot-wise su più QPU

La prima strategia, nota come *Shot-wise Distribution*, affronta direttamente i problemi legati all’eterogeneità dell’hardware e alla limitata disponibilità di singole QPU. Invece di vincolare un’intera computazione a un unico processore, l’idea è di distribuire il carico di lavoro (in particolare, il numero totale di shot richiesti) tra più QPU, che possono operare in parallelo (si veda Figura 1). Questa tecnica, esplorata in lavori come [4], offre diversi vantaggi strategici:

- **Parallelizzazione e Riduzione della Latenza:** Eseguendo batch di shot simultaneamente su più macchine, è possibile ridurre drasticamente il tempo totale di esecuzione (wall-clock time), superando i colli di bottiglia causati dalle code di attesa di un singolo dispositivo.
- **Mitigazione del Bias Hardware:** Ogni QPU fisica possiede un pattern di rumore sistematico e unico. Eseguendo la computazione su un singolo dispositivo, i risultati finali saranno inevitabilmente affetti da questo bias specifico. Aggregando i risultati provenienti da un insieme eterogeneo di QPU, si può ottenere un effetto di “media”, riducendo l’impatto del bias di una singola macchina e potenzialmente migliorando l’accuratezza complessiva del risultato finale [4].
- **Aumento della Resilienza:** La dipendenza da un’unica QPU introduce un *single point of failure*. Se il dispositivo fallisce o diventa temporaneamente non disponibile, l’intera computazione viene persa. Distribuendo gli shot, il fallimento di una QPU comporta solo la perdita di una frazione del lavoro, permettendo di continuare l’analisi con i dati rimanenti.

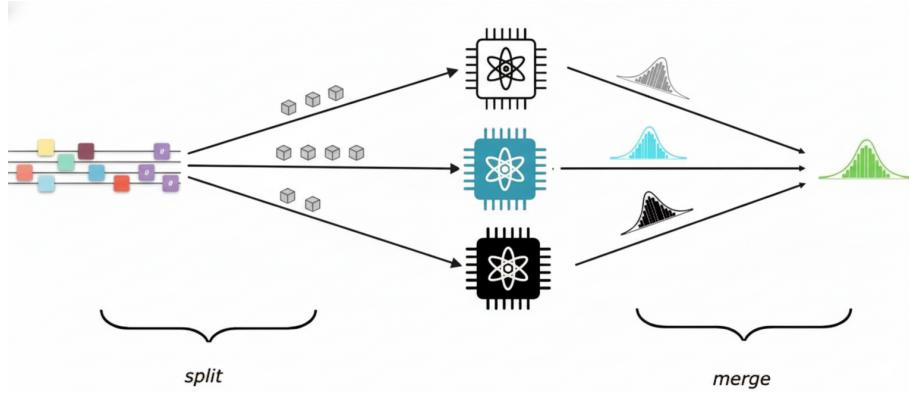


Figura 1: Illustrazione concettuale della Shot-wise Distribution. Un job quantistico viene suddiviso in batch di shot, che vengono eseguiti in parallelo su molteplici QPU. I risultati parziali vengono poi aggregati per formare la distribuzione finale.

Esecuzione Incrementale Adattiva

La seconda strategia, che chiameremo *Incremental Execution*, si concentra sull’ottimizzazione del processo di campionamento per minimizzare il numero totale di shot. L’approccio tradizionale prevede di definire a priori un budget di shot fisso (es. 10.000 shot), che è spesso difficile da stimare correttamente: potrebbe essere insufficiente per circuiti complessi o eccessivo per circuiti semplici, portando a inefficienza.

L’esecuzione incrementale, invece, adotta un processo iterativo e adattivo [3]:

1. Si inizia eseguendo un piccolo batch iniziale di shot.
2. Si calcola la distribuzione di probabilità cumulativa.
3. Ad ogni iterazione successiva, si esegue un altro batch di shot, si aggiorna la distribuzione cumulativa e si misura la “distanza” (es. tramite la Total Variation Distance) tra la distribuzione attuale e quella dell’iterazione precedente.
4. Il processo continua solo finché i nuovi shot alterano in modo significativo la distribuzione. Quando la distanza tra iterazioni consecutive scende al di sotto di una soglia di stabilità predefinita per un certo numero di volte, l’esecuzione termina.

Questo approccio (si veda Figura 2) permette di arrestare il campionamento non appena la distribuzione dei risultati si è stabilizzata, evitando di sprecare risorse su shot che non forniscono più informazioni utili. Ciò è particolarmente importante su hardware NISQ, dove l’obiettivo non è raggiungere la perfezione teorica (impossibile a causa del rumore), ma piuttosto convergere verso la distribuzione stabile e rumorosa caratteristica del dispositivo.

Le due strategie, quindi, affrontano il problema da angolazioni diverse: la distribuzione ottimizza l’uso dell’infrastruttura, mentre l’approccio incrementale ottimizza l’uso della risorsa “shot”. L’ipotesi centrale di questa tesi è che la loro integrazione sinergica possa portare a un sistema di esecuzione quantistica più robusto, veloce ed efficiente rispetto all’applicazione isolata di una sola delle due.

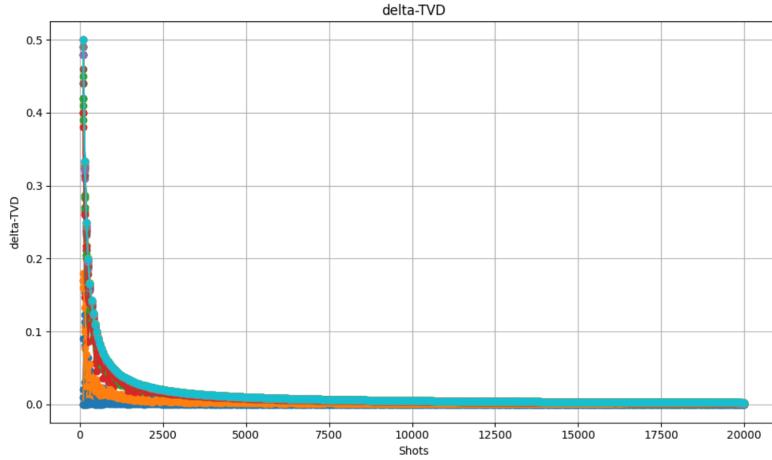


Figura 2: Concetto di Esecuzione Incrementale. L’errore o la variazione della distribuzione diminuisce rapidamente all’aumentare degli shot, per poi stabilizzarsi. L’obiettivo è fermare l’esecuzione in questa regione di “diminishing returns” per ottimizzare le risorse.

Definizione del Problema e Obiettivi della Tesi

L’esistenza di strategie complementari come la distribuzione shot-wise e l’esecuzione incrementale solleva una questione architettonale fondamentale, che costituisce il nucleo di questo lavoro di tesi. La domanda non è un semplice dettaglio implementativo, ma un problema concettuale che determina il comportamento, la correttezza e l’efficienza dell’intero sistema integrato. La domanda di ricerca centrale può essere formulata come segue:

Qual è l’architettura di controllo ottimale per integrare un framework di esecuzione distribuita e uno di esecuzione incrementale? In altre parole, chi orchestra chi?

Da questa sfida emergono due approcci architettonali logicamente opposti, che rappresentano le soluzioni candidate da analizzare [2]:

Architettura a Convergenza Globale: In questo modello, un singolo orchestratore implementa il ciclo di controllo incrementale a livello globale. A ogni iterazione, questo “cervello” strategico delega l’esecuzione di un batch di shot a un servizio di distribuzione specializzato (come il Quantum Executor), che gestisce la suddivisione tattica del lavoro tra le QPU della flotta. La decisione di arrestare la computazio-

ne viene presa centralmente, valutando la stabilità della distribuzione di probabilità *aggregata* di tutti i risultati ottenuti.

Architettura a Convergenza Locale: Questo modello inverte la gerarchia di controllo. Il sistema di distribuzione agisce come orchestratore principale, distribuendo “task di convergenza” indipendenti a ogni backend. Ciascun backend esegue il proprio ciclo incrementale in modo isolato, arrestandosi autonomamente una volta raggiunta la stabilità locale. I risultati, già dichiarati convergenti a livello individuale, vengono uniti solo al termine del processo.

La scelta tra queste due architetture non è banale e rappresenta il fondamento su cui si basano la logica di convergenza e l’efficienza delle risorse. Pertanto, l’obiettivo primario di questa tesi è **analizzare, implementare e confrontare sperimentalmente queste due architetture ibride**, al fine di determinarne le performance e i rispettivi compromessi.

Per raggiungere questo scopo, il lavoro si articola nei seguenti sotto-obiettivi specifici:

1. **Analisi Comparativa delle Architetture:** Condurre un’analisi concettuale delle due proposte, valutandone le proprietà, i vantaggi ipotizzati e i limiti intrinseci.
2. **Implementazione di un Prototipo Flessibile:** Sviluppare un prototipo software in grado di emulare il comportamento di entrambe le architetture per permetterne la validazione.
3. **Validazione Sperimentale:** Verificare le ipotesi attraverso una campagna di test, misurando e confrontando le performance delle due architetture al fine di quantificare i benefici e i trade-off di ciascun approccio.

In sintesi, questa tesi mira a fornire una risposta scientificamente fondata alla domanda sull’integrazione ottimale dei paradigmi di esecuzione quantistica, contribuendo con una metodologia chiara e una validazione empirica.

Struttura della Tesi

Il percorso argomentativo di questa tesi si articola nei seguenti capitoli, progettati per guidare il lettore dal contesto generale fino ai contributi specifici del lavoro.

- **Capitolo 1 - Stato dell'Arte:** Il primo capitolo posiziona il lavoro nel contesto della ricerca scientifica attuale. Viene presentata una rassegna delle principali metodologie per l'ottimizzazione degli shot in algoritmi quantistici, evidenziando come la letteratura si sia concentrata su architetture a singolo processore e identificando così la lacuna che questa tesi si propone di indirizzare.
- **Capitolo 2 - Framework di Riferimento:** Il secondo capitolo introduce le fondamenta tecniche del lavoro. Vengono descritti in dettaglio i framework computazionali per l'esecuzione distribuita e incrementale, e l'ambiente di simulazione utilizzato. Il capitolo si conclude definendo il problema architetturale dell'integrazione, che motiva la progettazione successiva.
- **Capitolo 3 - Progettazione dell'Architettura e Analisi Preliminare:** Il terzo capitolo affronta il nucleo progettuale della tesi. A partire da un'analisi esplorativa, vengono definite e confrontate a livello concettuale le due architetture candidate a Convergenza Globale e a Convergenza Locale. Il capitolo culmina nella formulazione di un'ipotesi sperimentale e nella descrizione del prototipo software implementato.
- **Capitolo 4 - Metodologia Sperimentale:** Il quarto capitolo definisce il “come” della validazione. Viene descritto in dettaglio il disegno degli esperimenti, presentando la metodologia per la costruzione del benchmark ottimale (l'Oracolo), le metriche di valutazione adottate e la configurazione di test finale.
- **Capitolo 5 - Risultati e Analisi Sperimentale:** Il quinto capitolo presenta i risultati quantitativi della campagna sperimentale. Le performance delle due architetture vengono confrontate in modo oggettivo sulla base delle metriche definite. Segue una discussione critica che interpreta i dati, ne analizza le cause e ne discute i limiti, al fine di validare l'ipotesi di ricerca.

- **Conclusioni:** Il capitolo finale sintetizza l'intero percorso di ricerca, riassumendo i contributi principali e rispondendo alla domanda di ricerca iniziale. Vengono infine discussi i limiti dello studio e proposte possibili direzioni per lavori futuri.

1. Stato dell’Arte

Come discusso nell’Introduzione, l’era del calcolo quantistico *Noisy Intermediate-Scale Quantum* (NISQ) è caratterizzata da un delicato equilibrio tra potenziale computazionale e limitazioni pratiche. L’elevata sensibilità al rumore e la profondità limitata dei circuiti eseguibili rendono molti degli algoritmi quantistici teoricamente potenti, come l’algoritmo di Shor, impraticabili sull’hardware attuale. In questo contesto, sono emerse classi di algoritmi specificamente progettate per operare entro i vincoli dell’hardware NISQ.

Tra queste, gli Algoritmi Quantistici Variazionali (VQA) rappresentano una delle direzioni di ricerca più promettenti e attive [10]. Sfruttando un’architettura ibrida, i VQA delegano il complesso compito di ottimizzazione a un processore classico, mentre utilizzano il computer quantistico solo per ciò che sa fare meglio: preparare stati quantistici complessi e stimare osservabili. Questa divisione del lavoro rende i VQA intrinsecamente più resistenti al rumore e adatti a circuiti di profondità ridotta, posizionandoli come candidati ideali per dimostrare un vantaggio quantistico a breve termine.

Tuttavia, anche i VQA si scontrano con il collo di bottiglia fondamentale dell’era NISQ: l’elevato costo in termini di misurazioni. A causa della natura probabilistica dell’output, ogni valutazione della funzione di costo (o del suo gradiente) richiede l’esecuzione ripetuta dello stesso circuito per un numero considerevole di volte (il numero di *shot*), al fine di ottenere una stima statisticamente affidabile. Questo processo di campionamento non solo è lento, ma rappresenta anche la principale fonte di costo nell’utilizzo delle piattaforme di calcolo quantistico commerciali.

Per rendere i VQA una tecnologia praticabile ed economicamente sostenibile, la ricerca scientifica si è quindi concentrata intensamente sullo sviluppo di strategie software per ridurre il numero totale di misurazioni. Questo campo di studi, noto come *shot optimization* o *adaptive shot allocation*, mira a sostituire l’approccio tradizionale a budget di shot fisso con metodologie più intelligenti, che adattano dinamicamente l’uso delle risorse di campionamento durante l’esecuzione dell’algoritmo.

Questo capitolo presenta una rassegna dello stato dell’arte in questo dominio. Verran-

no analizzate le principali classi di metodologie proposte in letteratura per affrontare la sfida dell’ottimizzazione degli shot, evidenziandone i contributi, le tecniche e i contesti di applicazione. Questa analisi critica permetterà di contestualizzare il presente lavoro di tesi, evidenziando una potenziale lacuna nella ricerca attuale e motivando il contributo specifico che si intende apportare.

1.1 Strategie di Ottimizzazione Adattiva degli Shot

L’approccio più diffuso in letteratura per l’ottimizzazione degli shot si basa sull’idea di *adaptive shot allocation*, ovvero l’allocazione dinamica del numero di misurazioni durante le diverse fasi di un algoritmo variazionale. La logica fondamentale di queste strategie è quella di essere “frugali”: si utilizzano poche risorse di campionamento (shot) nelle fasi iniziali dell’ottimizzazione, quando è sufficiente una stima approssimativa del gradiente per identificare una direzione di discesa, e si aumenta progressivamente il numero di shot man mano che ci si avvicina al minimo della funzione di costo, dove è richiesta una maggiore precisione per la convergenza fine.

Un lavoro pionieristico in questo campo è quello di Kübler et al. [12], che introducono un ottimizzatore per algoritmi variazionali denominato *individual Coupled Adaptive Number of Shots* (iCANS). Questo metodo, ispirato a tecniche del machine learning classico, calcola ad ogni passo dell’ottimizzazione il numero di shot “ottimale” da allocare per la stima di ciascuna componente del gradiente. L’algoritmo prioritizza le componenti che offrono il maggior “guadagno atteso” per shot, dimostrando numericamente di poter raggiungere la convergenza con un costo computazionale totale inferiore rispetto a strategie a budget fisso. Il lavoro di Gu et al. [9] rappresenta un’evoluzione di questo concetto. Il loro metodo, *global Coupled Adaptive Number of Shots* (gCANS), affina la regola di allocazione, basandola non più sul guadagno atteso di ogni singola componente, ma su una metrica di efficienza globale che considera l’intero vettore gradiente. Gli autori dimostrano che questo approccio globale porta a una convergenza geometrica più rapida in contesti convessi.

La validità di queste strategie non è limitata ai problemi di chimica quantistica, ma

si estende ad altri domini applicativi. Ad esempio, Phalak e Ghosh [16] hanno esplorato tecniche di allocazione adattiva per accelerare l’addestramento di architetture di *Quantum Machine Learning* (QML). Nel loro lavoro, il numero di shot viene ridotto dinamicamente durante le epoche di addestramento seguendo funzioni predefinite (lineari o a gradino), dimostrando che è possibile ottenere una significativa riduzione del costo computazionale con un impatto minimo sull’accuratezza finale del modello. Questi lavori, nel loro insieme, consolidano l’idea che l’allocazione adattiva degli shot sia una strategia fondamentale per rendere i VQA più efficienti.

1.2 Framework Formali per la Minimizzazione degli Shot

Oltre agli approcci adattivi basati su euristiche, un’altra direzione di ricerca si è concentrata sullo sviluppo di framework più formali, che cercano di legare in modo rigoroso il numero di shot a una metrica di errore o varianza. L’obiettivo di questi lavori è superare l’allocazione dinamica basata sul progresso dell’ottimizzazione, per definire invece un metodo che calcoli a priori il numero di misurazioni sufficienti per garantire che la stima di una grandezza rientri entro una soglia di errore prestabilita.

Un esempio di questo approccio è il framework generalizzato proposto da Kahani e Nobakhti [10]. La loro metodologia scomponete il problema in due parti distinte: un “problema di stima” e un “problema di ottimizzazione”. Invece di fornire all’ottimizzatore un numero fisso di shot, gli si fornisce un **budget di errore** desiderato (ad esempio, un Errore Quadratico Medio, MSE, massimo). L’“estimatore” ha quindi il compito di calcolare il numero di shot sufficienti per garantire che la stima della funzione di costo rispetti tale vincolo di errore. Questo approccio permette di controllare direttamente la precisione del processo di ottimizzazione, rendendolo più prevedibile e robusto. Gli autori dimostrano l’efficacia di questo concetto accoppiando un estimatore a media campionaria con diversi ottimizzatori, come il *simulated annealing* e la discesa del gradiente.

Sulla stessa linea, il lavoro di Seksaria e Prabhakar [18] affronta il problema di stimare il numero di shot necessari per raggiungere una **varianza desiderata** nei risultati di un circuito quantistico rumoroso. Partendo da un’analisi statistica delle principali fonti di ru-

more (come SPAM, smorzamento di ampiezza e di fase), gli autori sviluppano un modello sistematico che culmina in un'espressione in forma chiusa. Questa formula permette di calcolare la varianza attesa per un dato numero di shot, o, inversamente, di determinare il numero di shot necessari per ottenere una varianza non superiore a una certa soglia. Il loro approccio fornisce uno strumento analitico per la pianificazione delle risorse, consentendo di stimare il costo computazionale di un esperimento prima ancora di eseguirlo. Questi framework formali rappresentano un passo importante verso un controllo più rigoroso e predicibile delle risorse nel calcolo quantistico.

1.3 Approcci basati su Intelligenza Artificiale

Una delle direzioni di ricerca più recenti e promettenti per l'ottimizzazione degli shot consiste nell'utilizzare tecniche di Intelligenza Artificiale, in particolare l'Apprendimento per Rinforzo (*Reinforcement Learning*, RL). L'idea fondamentale è quella di trattare l'allocazione degli shot non come un problema da risolvere con euristiche predefinite o formule analitiche, ma come un **processo decisionale sequenziale** che può essere appreso automaticamente da un agente intelligente. In questo paradigma, un agente RL impara una “policy” ottimale osservando lo stato del processo di ottimizzazione (ad esempio, il progresso della funzione di costo) e decidendo, iterazione per iterazione, il numero di shot da allocare per massimizzare una ricompensa a lungo termine, che tipicamente bilancia il raggiungimento della convergenza con la minimizzazione del numero totale di shot.

Un lavoro esemplare in questo ambito è quello di Liang et al. [14], i quali propongono un approccio basato su RL per apprendere automaticamente le policy di assegnazione degli shot nel contesto del VQE. Il loro agente RL viene addestrato monitorando i progressi di molteplici esecuzioni di VQE e impara a prendere decisioni informate sul numero di shot da utilizzare in base allo stato attuale dell'ottimizzazione (ad esempio, se l'energia si sta stabilizzando o se è stato trovato un nuovo minimo). Un risultato particolarmente interessante del loro studio è la dimostrazione della **trasferibilità** della policy appresa: una policy addestrata su una molecola semplice come H_2 si dimostra efficace anche quando applicata a sistemi più complessi come HeH^+ , LiH e BeH_2 , suggerendo che l'agente impara

strategie di ottimizzazione generalizzabili. Altri lavori, come quello di Kim et al. [11], hanno ulteriormente esplorato strategie dinamiche che si adattano alle caratteristiche della distribuzione dei risultati per ottimizzare l’allocazione degli shot.

L’ottimizzazione dell’assegnazione degli shot (*shot assignment*) è cruciale anche quando si considerano Hamiltoniani complessi, composti da molti termini. In questi casi, il budget di shot per ogni iterazione deve essere distribuito non solo nel tempo, ma anche tra i diversi gruppi di termini misurabili simultaneamente. Anche in questo contesto, sono stati proposti approcci adattivi per ottimizzare tale distribuzione, come discusso, ad esempio, in un altro lavoro di Liang et al. [19]. Questi approcci basati su IA rappresentano la frontiera della ricerca sull’ottimizzazione delle risorse, mirando a creare strategie completamente autonome che superino le euristiche definite dall’uomo.

1.4 Posizionamento del Lavoro e Contributo della Tesi

Dall’analisi dello stato dell’arte condotta in questo capitolo, emerge un quadro di ricerca ricco e in rapida evoluzione, focalizzato quasi interamente sull’ottimizzazione delle risorse di campionamento per algoritmi eseguiti su un singolo processore quantistico. Come illustrato, la letteratura ha esplorato strategie adattive che regolano dinamicamente gli shot in base al progresso dell’ottimizzazione [12, 9, 16], ha sviluppato framework formali per legare il numero di shot a vincoli di errore o varianza [10, 18], e ha iniziato a impiegare tecniche di Intelligenza Artificiale per apprendere policy di allocazione autonome [14, 11].

Tuttavia, un elemento comune a quasi tutti questi lavori è l’assunzione di un’architettura di esecuzione centralizzata. Le metodologie proposte, pur essendo sofisticate, non considerano le complessità aggiuntive né le opportunità offerte da un’infrastruttura di calcolo distribuita, in cui lo stesso circuito viene eseguito in parallelo su molteplici QPU eterogenee. Manca quindi uno studio sistematico che analizzi come le strategie di ottimizzazione degli shot possano essere integrate in un paradigma di esecuzione distribuita di tipo *shot-wise*.

Qui risiede il **gap nella letteratura** che questa tesi si propone di indirizzare. Il **contributo principale di questo lavoro** è quello di investigare quest’area di ricerca meno

consolidata. Invece di proporre un ulteriore ottimizzatore di shot per un singolo QPU, questa tesi studia come le strategie di esecuzione adattiva possano essere integrate con un’architettura di esecuzione distribuita.

Per raggiungere tale scopo, questo studio si basa sui due framework di riferimento introdotti dal nostro gruppo di ricerca: il *Quantum Executor*, che implementa l’esecuzione *shot-wise*, e l’*Incremental Execution Framework*, che fornisce la logica di controllo adattivo. L’obiettivo è analizzare le implicazioni architettoniche di una loro integrazione, confrontando un modello a controllo centralizzato (l’*Architettura a Convergenza Globale*) con uno a controllo distribuito (l’*Architettura a Convergenza Locale*). Attraverso una validazione sperimentale rigorosa, si intende fornire una metodologia e una quantificazione delle performance per un’esecuzione quantistica che aspiri a essere contemporaneamente efficiente nelle risorse e scalabile sull’infrastruttura.

2. Framework di Riferimento

Dopo aver posizionato il lavoro nel contesto dello stato dell’arte, questo capitolo pone le fondamenta tecniche e concettuali della tesi. La prima parte introduce in dettaglio gli strumenti su cui si basa questo studio: i due **framework computazionali** di riferimento uno per l’esecuzione distribuita e uno per quella incrementale e l’**ambiente di simulazione** utilizzato per la validazione. Sebbene già menzionati a livello generale, la loro analisi tecnica è qui presentata in quanto propedeutica alla comprensione del progetto architetturale.

La seconda parte del capitolo affronta il nucleo progettuale del lavoro: a partire da un’analisi esplorativa, vengono definite e confrontate due architetture candidate per l’integrazione, culminando nella formulazione dell’ipotesi sperimentale che guiderà la validazione.

2.1 Il Paradigma dell’Esecuzione Distribuita: Quantum Executor

Il *Quantum Executor* è un framework software progettato per semplificare e unificare l’esecuzione di esperimenti quantistici su un’ampia gamma di piattaforme hardware e simulatori [6]. Nasce per risolvere il problema della frammentazione dell’ecosistema quantistico, dove l’interazione con provider diversi (come IBM Quantum, IonQ, o Amazon Braket) richiede spesso la gestione di SDK, paradigmi di esecuzione e oggetti specifici, ostacolando la portabilità del codice e la riproducibilità degli esperimenti.

2.1.1 Scopo e Filosofia del Framework

La filosofia alla base di Quantum Executor è quella dell’ **astrazione** e dell’ **unificazione**. L’obiettivo primario è fornire un’interfaccia di esecuzione *backend-agnostic*, che nasconde le complessità specifiche di ciascun provider dietro un unico livello di astrazione coerente. Grazie a questo approccio, un ricercatore può definire un esperimento una sola volta e

rieseguirlo su piattaforme diverse, o addirittura su più piattaforme in parallelo senza dover modificare il proprio codice sorgente.

Questo paradigma, descritto in [4], si fonda su alcuni principi di progettazione chiave:

- **Interoperabilità:** Il framework è progettato per essere agnostico non solo rispetto all'hardware (provider-independent), ma anche rispetto al linguaggio di programmazione quantistica (language-independent), supportando circuiti definiti in diversi formati come Qiskit, Cirq, PyQuil o OpenQASM.
- **Esecuzione Dichiarativa:** Gli esperimenti vengono definiti in modo dichiarativo, specificando *cosa* eseguire (i circuiti e il numero di shot) e *dove* (l'elenco dei backend candidati), separando nettamente la logica scientifica dalla gestione dell'infrastruttura.
- **Estensibilità:** L'architettura modulare, basata su un *VirtualProvider* che agisce come “agente di viaggio” per le piattaforme quantistiche, consente di integrare facilmente nuovi provider hardware o simulatori senza alterare il nucleo del sistema.

2.1.2 Concetti Chiave: il Sistema a Policy

La potenza e la flessibilità di Quantum Executor risiedono nel suo sistema a *policy* (Policies), che permette all'utente di personalizzare in modo programmatico la logica di distribuzione e aggregazione dei risultati. Le policy sono funzioni Python che incapsulano la strategia dell'esperimento e agiscono su due livelli distinti.

Split Policy (Politica di Suddivisione) La *Split Policy* è il “cervello strategico” della distribuzione. Il suo compito è tradurre una richiesta di esperimento di alto livello (es. “esegui questo circuito per 10.000 shot totali su questi tre backend”) in un piano di esecuzione dettagliato e concreto. Questo piano, chiamato Dispatch, è una collezione di Job atomici, dove ogni Job specifica in modo univoco il circuito da eseguire, il numero di shot e la coppia provider/backend di destinazione.

La Split Policy determina:

1. **Quali backend** verranno effettivamente utilizzati tra quelli disponibili.

2. **Come suddividere il numero totale di shot** tra i backend selezionati. Ad esempio, una policy `uniform` li distribuisce equamente, mentre una policy `multiplier` invia l’intero budget di shot a ciascun backend, utile per esperimenti di benchmarking.
3. **Se modificare i circuiti** per ciascun job, permettendo scenari avanzati.

Questo meccanismo consente di implementare logiche di distribuzione complesse, come lo scheduling basato sul rumore, sui tempi di attesa o sui costi, delegando la decisione a una funzione personalizzabile.

Merge Policy (Politica di Unione) Se la Split Policy si occupa di “dividere” il lavoro, la *Merge Policy* si occupa di “ricomporre” i risultati. Poiché un’esecuzione distribuita produce risultati parziali e grezzi da ogni Job, è necessario un meccanismo per aggregarli in un unico output significativo per l’intero esperimento. La Merge Policy è una funzione che riceve la collezione di tutti i risultati grezzi (raccolti in un oggetto `ResultCollector`) e applica una logica per combinarli. L’esempio più comune è la policy `simple_aggregate`, che somma i conteggi (counts) di ogni stringa di bit misurata su tutti i job, producendo una distribuzione di frequenza globale.

2.2 Il Paradigma dell’Esecuzione Incrementale

Mentre il Quantum Executor si concentra sull’ottimizzazione dell’infrastruttura, il paradigma dell’Esecuzione Incrementale affronta il problema complementare: l’ottimizzazione della risorsa *shot*. L’approccio tradizionale a budget fisso risulta spesso inefficiente, poiché difficile da stimare a priori: rischia di essere eccessivo per compiti semplici o insufficiente per quelli complessi. Il framework di Esecuzione Incrementale [3] propone una soluzione adattiva.

2.2.1 Scopo e Filosofia del Framework

La filosofia del framework è la separazione netta tra la **logica di controllo** (criteri di arresto e pianificazione degli shot) e il **lavoro effettivo** (esecuzione). Il sistema agisce come un

motore agnostico che orchestra un compito iterativo fino al raggiungimento della stabilità, ignorando i dettagli specifici del compito stesso.

Questo design modulare rende il sistema flessibile e applicabile non solo al calcolo quantistico, ma a qualsiasi processo convergente (es. simulazioni Monte Carlo). Nel nostro contesto, il “lavoro effettivo” è delegato a un `runner` che esegue il circuito e restituisce i conteggi.

L’obiettivo strategico è arrestare il campionamento in presenza di rendimenti decrescenti, ovvero quando ulteriori shot non alterano significativamente la stima della distribuzione. Su hardware NISQ questo è cruciale: non si cerca la convergenza a una distribuzione ideale, ma al *noise floor* caratteristico del dispositivo, minimizzando lo spreco di risorse.

2.2.2 Concetti Chiave: il Ciclo di Convergenza e i Criteri

Il comportamento del framework è governato da un ciclo iterativo e da policy configurabili.

Il Ciclo di Convergenza Il cuore del framework è un ciclo di esecuzione che opera come segue:

1. **Inizializzazione:** Avvio con un batch iniziale di shot.
2. **Esecuzione:** Invocazione del `runner` per l’acquisizione dei risultati del batch.
3. **Aggiornamento:** Aggregazione dei nuovi dati alla distribuzione cumulativa storica.
4. **Valutazione:** Applicazione dei criteri per verificare la stabilità della stima.
5. **Decisione:** Terminazione in caso di stabilità confermata, oppure calcolo del prossimo batch e reiterazione.

Fondamenti Matematici: la Total Variation Distance (TVD) Alla base di tutte le valutazioni di convergenza e accuratezza di questa tesi vi è la *Total Variation Distance*. Si tratta di una metrica statistica che quantifica la distanza tra due distribuzioni di probabilità

P e Q :

$$\text{TVD}(P, Q) = \frac{1}{2} \sum_{x \in \{0,1\}^n} |P(x) - Q(x)| \quad (2.1)$$

Questa metrica viene utilizzata in due contesti distinti:

- **Stabilità (Delta Distance):** Si pone $P = C_t$ (distribuzione cumulativa corrente) e $Q = C_{t-1}$ (iterazione precedente) per misurare quanto la stima sta cambiando nel tempo.
- **Accuratezza:** Si pone $P = C_t$ e $Q = \text{GT}$ (Ground Truth ideale) per valutare la qualità assoluta del risultato rispetto al teorico (utilizzato in fase di analisi).

Criterio di Arresto: Delta Distance Per governare il ciclo di esecuzione, il sistema implementa il criterio della **Delta Distance**, che applica la TVD alla storia evolutiva della stima. Formalmente, sia C_t la distribuzione normalizzata al passo t . La variazione $\Delta(t)$ è definita come:

$$\Delta(t) = \text{TVD}(C_t, C_{t-1})$$

L'iterazione t è considerata stabile se:

$$\Delta(t) < \text{soglia}$$

Per garantire la robustezza contro fluttuazioni statistiche temporanee (shot noise), l'arresto definitivo è governato dallo **Stability Criterion**: la convergenza viene dichiarata solo quando la condizione $\Delta(t) < \text{soglia}$ è soddisfatta per k iterazioni consecutive (identificato dal parametro `consecutive_iterations`). Infine, la **Next Shots Strategy** determina la dimensione del batch successivo qualora la stabilità non sia stata raggiunta (es. mantenendo un passo costante o adattandolo alla velocità di convergenza).

2.3 Simulazione Realistica: il Dataset `qsimbench`

La validazione sperimentale di nuove architetture e algoritmi quantistici presenta una sfida pratica significativa. L'esecuzione di esperimenti su hardware quantistico reale è un processo costoso, lento a causa delle code di accesso, e spesso difficile da riprodurre a causa

della deriva temporale delle calibrazioni dei dispositivi (calibration drift). D'altra parte, l'uso di simulatori ideali ignora l'impatto del rumore, che è invece il fattore dominante nelle performance dei sistemi NISQ. Per superare questi ostacoli, la metodologia di questa tesi si avvale di *qsimbench*, una libreria Python e un dataset di risultati di esperimenti quantistici pre-eseguiti [5].

2.3.1 Scopo e Filosofia del Framework

qsimbench è concepito come una “biblioteca di esperimenti quantistici” pre-calcolati. Invece di eseguire un circuito da zero ogni volta, il framework permette di accedere istantaneamente a “tracce di esecuzione” realistiche, registrate in precedenza. Ogni traccia contiene non solo i risultati delle misurazioni (i conteggi), ma anche un “quaderno di laboratorio” digitale con tutti i metadati contestuali: il codice OpenQASM del circuito, la configurazione precisa del backend simulato e i dettagli del modello di rumore applicato.

Lo scopo principale è abilitare test e benchmark **rapidi, efficienti e perfettamente riproducibili** per l'ingegneria del software quantistico. Permette di simulare l'interazione con hardware rumoroso senza subire i costi e le latenze del mondo reale.

2.3.2 Il Vantaggio Cruciale: Dati Realistici, Riproducibili e su Larga Scala

La scelta di utilizzare *qsimbench* come fondamento per la validazione sperimentale in questa tesi è motivata da un vantaggio cruciale: esso disaccoppia la logica di orchestrazione (il cuore del nostro lavoro) dalla generazione fisica dei dati. Questo permette di concentrarsi sull'analisi delle performance delle architetture in un ambiente controllato.

Il vantaggio più significativo è la possibilità di condurre test su larga scala, impossibili da realizzare direttamente su hardware fisico per vincoli di tempo e costi. Il dataset di *qsimbench* contiene i risultati di decine di algoritmi diversi, eseguiti per un'ampia gamma di dimensioni (numero di qubit) e su molteplici backend simulati, ognuno con un modello di rumore basato sulle caratteristiche di un dispositivo IBM reale.

Un concetto fondamentale è quello del **rumore “congelato”**. A differenza di un QPU fisico, i cui parametri di rumore fluttuano nel tempo, i modelli di rumore in `qsimbench` sono fissi e stabili. Questo garantisce che ogni richiesta per una data configurazione (algoritmo, dimensione, backend) produca sempre risultati provenienti da una distribuzione statisticamente identica, assicurando una perfetta riproducibilità. L’accesso ai dati è gestito in modo efficiente tramite un sistema di caching locale: dopo il primo download, le richieste successive per gli stessi dati vengono soddisfatte istantaneamente dal disco locale, accelerando drasticamente il ciclo di sviluppo e test.

In sintesi, `qsimbench` funge da “oracolo” che, interrogato con una specifica configurazione, fornisce i risultati come se fossero stati ottenuti da un’esecuzione reale, permettendo di condurre la validazione sperimentale in modo rigoroso, efficiente e scalabile.

2.4 Progettazione delle Architetture di Integrazione

Dall’analisi dei framework presentati nelle sezioni precedenti, emerge un quadro chiaro. Da un lato, il *Quantum Executor* [4, 6] offre una soluzione robusta per l’orchestrazione di computazioni su un’infrastruttura hardware eterogenea. Dall’altro, il framework di *Esecuzione Incrementale* [3] fornisce una metodologia intelligente per l’ottimizzazione della risorsa *shot*.

Entrambi gli strumenti sono efficaci nei rispettivi domini, ma operano in modo largamente indipendente. Il *Quantum Executor* opera con un budget di shot predefinito, mentre il framework incrementale è stato concepito principalmente per un contesto a singolo backend. La semplice combinazione “ingenua” di questi due approcci non è sufficiente e solleva una questione architetturale fondamentale che costituisce il nucleo di questo lavoro: **qual è la gerarchia di controllo ottimale per integrare queste due strategie in modo sinergico?**

Questa domanda non è un mero dettaglio implementativo, ma un problema concettuale che definisce il comportamento, l’efficienza e la correttezza dell’intero sistema ibrido. Le sezioni seguenti affronteranno questa sfida attraverso:

1. Un'analisi sperimentale preliminare per caratterizzare il comportamento di base del sistema distribuito.
2. La progettazione formale di due architetture di integrazione logicamente opposte, che verranno denominate Architettura a Convergenza Globale e Architettura a Convergenza Locale.
3. Un'analisi comparativa teorica per selezionare l'architettura candidata più promettente per la validazione sperimentale.

Questo percorso progettuale mira a sviluppare una metodologia coerente per un'esecuzione quantistica che sia contemporaneamente distribuita e adattiva, massimizzando così l'efficienza complessiva nell'utilizzo delle preziose e limitate risorse dell'era NISQ.

3. Progettazione dell’Architettura e Analisi Preliminare

Dopo aver definito nel capitolo precedente i paradigmi di esecuzione distribuita e incrementale, questo capitolo affronta il nucleo concettuale della tesi. Si inizia con un’analisi sperimentale di base, volta a caratterizzare il comportamento di un sistema puramente distribuito e a motivare la necessità di un’integrazione. Successivamente, si passa alla progettazione formale di due architetture candidate, che realizzano tale integrazione secondo logiche di controllo opposte. L’analisi comparativa di queste proposte condurrà alla selezione dell’architettura ritenuta teoricamente superiore, la cui validazione empirica sarà oggetto dei capitoli successivi.

3.1 Analisi Esplorativa della Convergenza Distribuita

Prima di progettare architetture di integrazione complesse, è stata condotta un’analisi sperimentale di base per stabilire una *baseline* qualitativa. L’obiettivo di questa fase è stato osservare e caratterizzare il comportamento di convergenza di un sistema puramente distribuito, basato esclusivamente sul framework *Quantum Executor*, in assenza di qualsiasi meccanismo di arresto adattivo. Questa indagine preliminare è fondamentale per verificare l’ipotesi che esista un “margine di manovra” per l’ottimizzazione, giustificando così la successiva integrazione con l’approccio incrementale.

3.1.1 Metodologia dell’Analisi di Baseline

La metodologia simula un processo di raccolta dati esaustivo, sfruttando il dataset realistico fornito da `qsimbench`. Per ogni esperimento disponibile (definito da una coppia algoritmo-dimensione), lo studio itera su ogni possibile “team” di backend (con dimensione ≥ 2), emulando un’esecuzione distribuita con un budget fisso di 20.000 shot. Il processo è articolato come segue:

1. **Setup della Simulazione:** Per ogni team, viene simulata un'esecuzione incrementale suddivisa in 400 iterazioni, ciascuna composta da un batch di 50 shot, permettendo un'analisi a grana fine della dinamica di convergenza.
2. **Distribuzione e Aggregazione (Emulazione del Quantum Executor):** Ad ogni iterazione, il budget di 50 shot viene distribuito uniformemente tra i membri del team (politica *uniform split*). I risultati parziali, ottenuti tramite chiamate on-demand a qsimbench, vengono poi aggregati sommando i conteggi per ogni stato di output (politica *simple aggregate*).
3. **Calcolo delle Metriche:** Sulla distribuzione cumulativa aggiornata ad ogni passo, vengono calcolate due metriche basate sulla Total Variation Distance (TVD):
 - **Accuratezza:** La TVD tra la distribuzione cumulativa corrente e la *ground truth* ideale (ottenuta dal simulatore ideale aer_simulator).
 - **Stabilità:** La TVD tra la distribuzione cumulativa corrente e quella dell'iterazione precedente, per misurare la volatilità della stima.

3.1.2 Risultati: Caratterizzazione della Convergenza

I risultati di questa analisi su vasta scala, che comprende migliaia di esecuzioni simulate, sono sintetizzati nei grafici “aggregati” mostrati in Figura 3.1 e Figura 3.2. Ciascun grafico sovrappone, con un'elevata trasparenza, le traiettorie di convergenza di tutti gli esperimenti e di tutti i team analizzati.

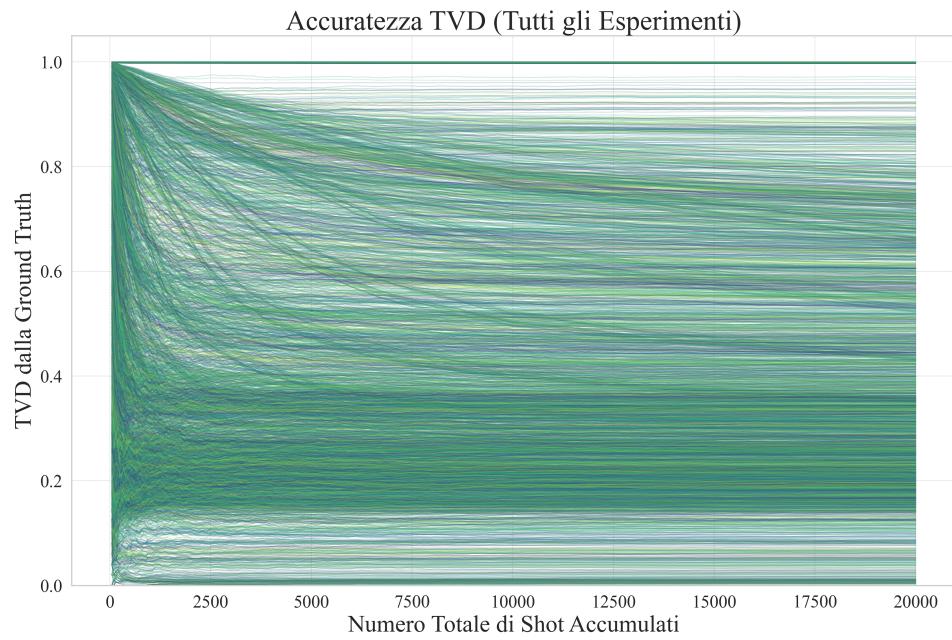


Figura 3.1: Grafico aggregato dell’Accuratezza (TVD vs. Ground Truth) per tutte le esecuzioni simulate. L’asse X rappresenta il numero di shot cumulativi, mentre l’asse Y la distanza dalla distribuzione ideale.

La Figura 3.1 mostra l’evoluzione dell’accuratezza all’aumentare degli shot. Nonostante l’eterogeneità degli esperimenti e dei team di backend analizzati, emerge una tendenza qualitativa comune. Si osserva, nella maggior parte dei casi, una rapida diminuzione iniziale della TVD, che indica un miglioramento dell’accuratezza, seguita da una progressiva stabilizzazione in un *plateau* orizzontale. Questo comportamento è consistente con il concetto di “noise floor”: un limite pratico all’accuratezza, imposto dal rumore sistematico dell’hardware, oltre il quale ulteriori campionamenti non producono miglioramenti significativi.

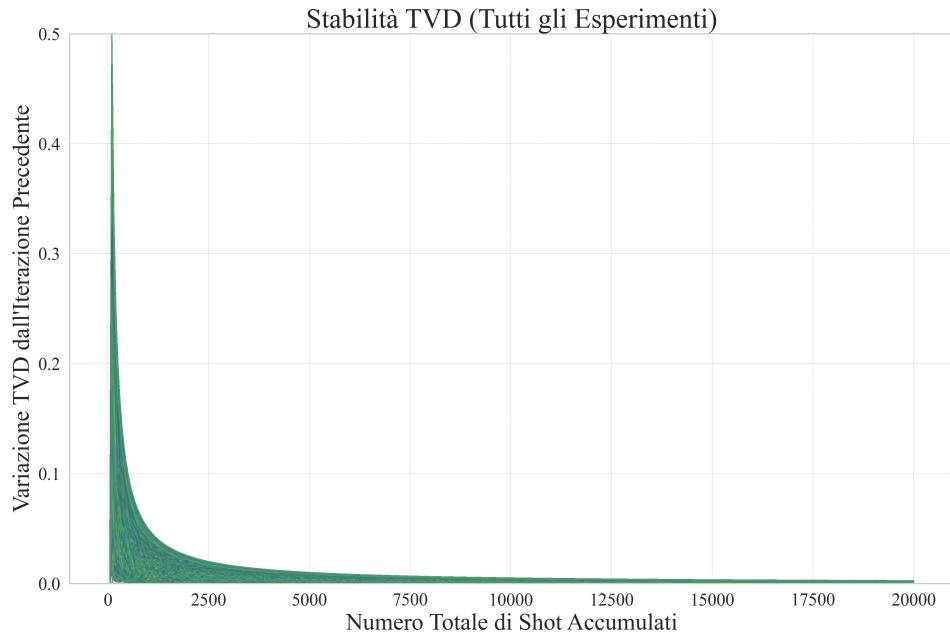


Figura 3.2: Grafico aggregato della Stabilità (Delta-TVD tra iterazioni) per tutte le esecuzioni simulate. L'asse Y misura la variazione della distribuzione cumulativa rispetto al passo precedente.

La Figura 3.2, che mostra l'andamento della stabilità, conferma questa osservazione. Si nota un picco iniziale molto elevato, che corrisponde alla fase in cui ogni nuovo batch di shot altera significativamente la forma della distribuzione. Questo picco è seguito da un decadimento esponenziale molto rapido, che indica come, dopo un certo numero di shot, i nuovi dati contribuiscano sempre meno a modificare la stima cumulativa, la quale ha ormai raggiunto una forma stabile.

3.1.3 Motivazioni per un'Architettura Ibrida

L'analisi di baseline ha rivelato un comportamento ricorrente e fondamentale: il processo di campionamento statistico in un ambiente distribuito e rumoroso è caratterizzato da **rendimenti decrescenti**. Dopo una fase iniziale di rapida convergenza, si raggiunge un punto oltre il quale l'esecuzione di ulteriori migliaia di shot non produce un miglioramento significativo né dell'accuratezza (Figura 3.1) né della stabilità della stima (Figura 3.2).

Questa osservazione non è un limite, ma un' **opportunità**. Essa dimostra empiricamente che esiste un vasto margine per l'ottimizzazione: se fosse possibile rilevare in tempo reale il momento in cui la distribuzione si stabilizza sul suo *plateau*, si potrebbe arrestare l'esecuzione prematuramente, ottenendo un risultato di qualità quasi identica ma con un notevole risparmio di risorse computazionali (shot).

È proprio questa opportunità a motivare il nucleo di questa tesi: l'integrazione del framework di esecuzione distribuita, *Quantum Executor*, con un meccanismo di controllo adattivo, l'*Incremental Execution Framework*. La sfida, tuttavia, non è banale e solleva una questione architettonale fondamentale, che verrà esplorata nella sezione successiva.

3.2 Modelli di Integrazione e Gerarchia di Controllo

L'integrazione di due framework complessi come il Quantum Executor e l'Incremental Execution Framework non è un semplice esercizio di composizione software. La sfida fondamentale risiede nella definizione della **gerarchia di controllo** tra i due paradigmi. La questione centrale, come formulato in [2], è concettuale e determina il comportamento intrinseco, l'efficienza e la correttezza metodologica dell'intero sistema ibrido.

Il problema può essere formalizzato in due scenari opposti:

1. Il ciclo di controllo incrementale, responsabile della logica di convergenza, deve orchestrare l'esecuzione distribuita, trattando il Quantum Executor come un servizio atomico specializzato nell'eseguire un batch di shot?
2. Oppure, al contrario, è il sistema di distribuzione, il Quantum Executor, che deve agire da orchestratore principale, distribuendo cicli di controllo incrementali indipendenti ai singoli backend, trattandoli come task atomici da eseguire in parallelo?

La risposta a questa domanda è il fondamento su cui si basano la logica di convergenza, l'efficienza nell'uso delle risorse e, soprattutto, la validità statistica del risultato finale. Da questa dicotomia emergono due architetture candidate, che rappresentano approcci logicamente opposti e che verranno analizzate criticamente nelle sezioni seguenti, al fine di formulare un'ipotesi sulla loro relativa efficacia.

3.3 Architettura a Convergenza Globale

La prima architettura candidata, denominata *Architettura a Convergenza Globale*, si fonda sul principio della *separazione delle responsabilità* (*separation of concerns*). In questo modello, il controllo strategico della convergenza è nettamente disaccoppiato dall'esecuzione tattica distribuita. Si ipotizza che questa centralizzazione della logica di convergenza porti a un comportamento più robusto e metodologicamente corretto.

3.3.1 Architettura e Flusso Operativo

Come illustrato nel diagramma di flusso in Figura 3.3, il modello a Convergenza Globale prevede due ruoli nettamente distinti:

- **OrCHEstratore (Ciclo Incrementale):** Agisce come il “cervello” *stateful* del sistema. È sua responsabilità esclusiva mantenere lo stato della stima (la distribuzione di probabilità cumulativa e la cronologia delle esecuzioni), implementare la logica di convergenza (valutando a ogni iterazione se la stima aggregata ha raggiunto la stabilità) e calcolare il numero di shot per il batch successivo.
- **Quantum Executor (Servizio di Esecuzione Distribuita):** Agisce come un motore di astrazione *stateless* e agnostico. La sua interfaccia espone un unico compito: “esegui N shot di questo circuito”. Internamente, gestisce tutta la complessità della distribuzione, applicando una *SplitPolicy* per suddividere il task, inviando i job ai backend e aggregando i risultati tramite una *MergePolicy* prima di restituire un'unica distribuzione aggregata del batch all'OrCHEstratore.

Il flusso operativo ipotizzato è un ciclo iterativo sincrono:

1. L'OrCHEstratore determina il numero di shot N da eseguire nel batch corrente.
2. Delega l'esecuzione di questi N shot al Quantum Executor.
3. L'OrCHEstratore attende la ricezione della distribuzione aggregata, risultato del completamento di *tutti* i job del batch.

4. Aggiorna il proprio stato interno (la distribuzione globale cumulativa) con i nuovi dati.
5. Valuta i criteri di convergenza sulla stima globale.
6. Se non è stata raggiunta la convergenza e se non sono stati esauriti gli shot disponibili, il ciclo si ripete. Altrimenti, termina.

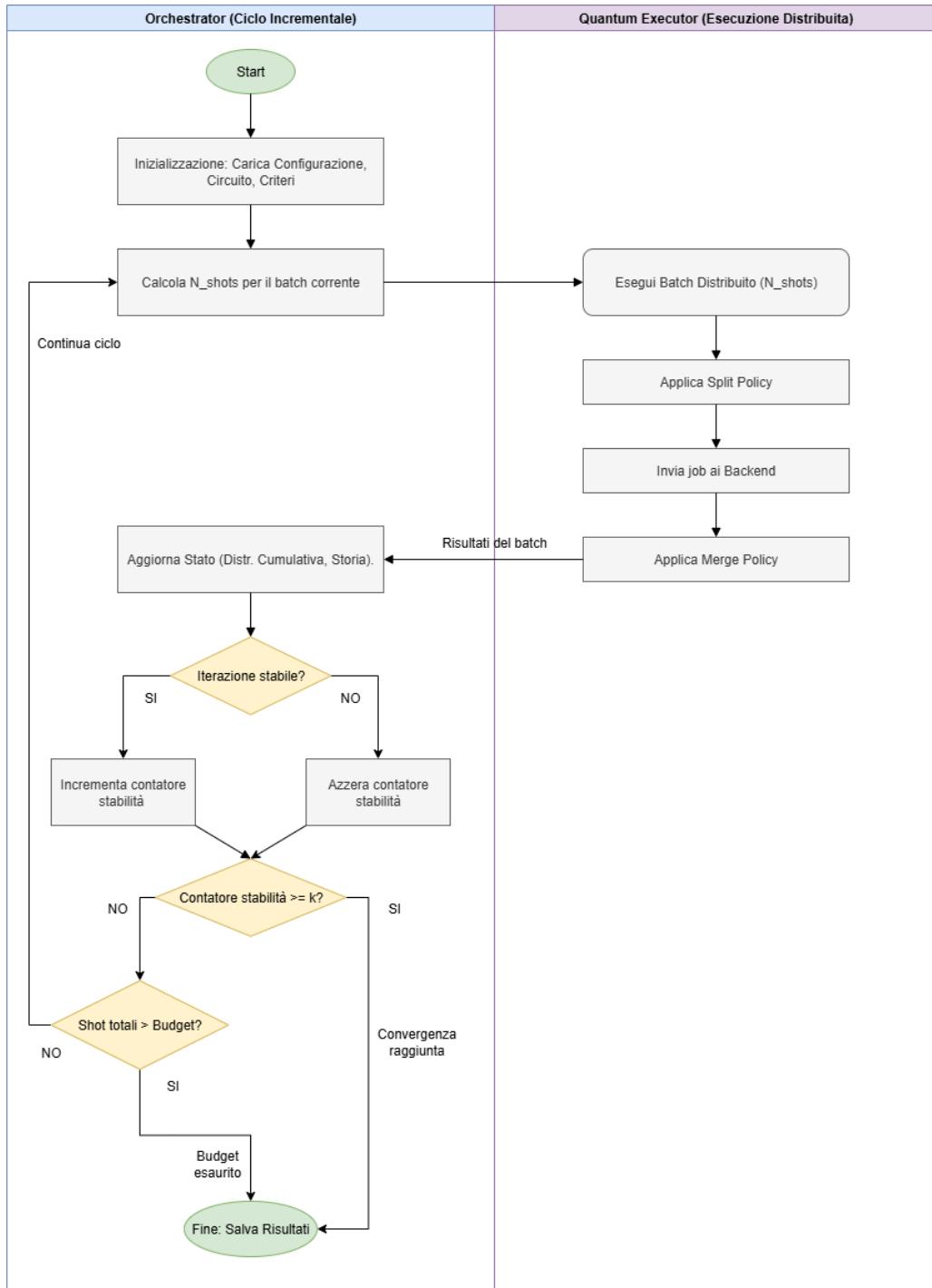


Figura 3.3: Diagramma di flusso operativo per l’Architettura a Convergenza Globale. Un ciclo di controllo esterno (Orchestratore) gestisce lo stato dell’esperimento e delega, ad ogni iterazione, l’esecuzione di un batch di shot a un servizio di esecuzione distribuita.

3.3.2 Proprietà Fondamentali e Vantaggi Ipotizzati

Si postula che questa architettura presenti proprietà chiave che ne supportano l’ipotesi di validità metodologica e pratica [2].

Correttezza Concettuale. L’ipotesi fondamentale è che questo sistema sia progettato per ottimizzare la stabilità della grandezza di reale interesse per l’utente: la **stima aggregata finale**. Poiché la decisione di arresto si basa sull’informazione combinata dell’intera flotta di QPU, l’obiettivo è garantire che il risultato finale sia stabile nel suo complesso, allineandosi direttamente con l’obiettivo scientifico.

Efficienza Informativa. Si ipotizza che, sfruttando un unico pool di informazione condiviso, il sistema possa raggiungere la convergenza in modo più efficiente. Backend con performance eterogenee (ad esempio, con bias sistematici diversi) contribuirebbero a formare una stima media che, grazie all’aggregazione, diventa progressivamente più robusta. Il sistema si arresterebbe non appena questa stima media si stabilizza, evitando di sprecare risorse su backend che, se considerati isolatamente, potrebbero richiedere più shot per convergere.

Modularità e Flessibilità. L’architettura rispetta la filosofia *black-box* dei singoli strumenti. Il Quantum Executor viene utilizzato per lo scopo per cui è stato progettato (astrarre l’esecuzione distribuita), senza richiedere modifiche interne. Questo disaccoppiamento dovrebbe semplificare l’implementazione, favorire la manutenibilità e permettere una sperimentazione agile, consentendo di testare diverse SplitPolicy e MergePolicy modificando esclusivamente file di configurazione.

Convergenza Onesta. Si ipotizza che questo approccio sia intrinsecamente resistente al rischio di “false convergenze”. Poiché la decisione di arresto è basata su una stima globale che media le performance dell’intera flotta, il sistema non persegue l’illusione di una “distribuzione ideale”, ma è progettato per convergere robustamente verso la migliore stima stabile della **distribuzione media rumorosa** che la flotta può offrire.

3.3.3 Compromessi Intrinseci e Strategie di Mitigazione

L’ipotesi della superiorità della *Architettura a Convergenza Globale* non implica l’assenza di compromessi. Essi rappresentano scelte ingegneristiche consapevoli, fatte in favore della correttezza e della robustezza.

Latenza di Sincronizzazione Iterativa. L’architettura è intrinsecamente sincrona a livello di batch. L’Orchestratore deve attendere il completamento di *tutti* i job del batch corrente prima di poter avviare l’iterazione successiva. Questo implica che un singolo QPU lento o con una lunga coda di attesa può diventare un collo di bottiglia, rallentando l’intero processo. Il trade-off scelto è quello di privilegiare la correttezza della stima globale (che richiede tutti i dati) rispetto al tempo di esecuzione totale (wall-clock time).

- *Possibile mitigazione:* Si potrebbero implementare timeout aggressivi sui singoli job e politiche di riallocazione dinamica per escludere temporaneamente i backend non performanti.

Rischio di Allocazione Inefficiente degli Shot. Una SplitPolicy “ingenua” (es. distribuzione uniforme) non considera le performance eterogenee dei backend. Ciò potrebbe portare ad allocare un numero eccessivo di shot a un backend molto rumoroso o un numero insufficiente a un backend di alta qualità.

- *Possibile mitigazione:* Si ipotizza che lo sviluppo di SplitPolicy avanzate e “pesate” (noise-aware), che allocano gli shot in modo proporzionale alla qualità o alla velocità storica dei backend, possa ottimizzare l’uso delle risorse senza sacrificare la convergenza globale.

3.4 Architettura a Convergenza Locale

La seconda architettura candidata, d’ora in poi denominata *Architettura a Convergenza Locale*, rappresenta l’approccio diametralmente opposto alla Convergenza Globale. Questa architettura inverte la gerarchia di controllo, promuovendo il Quantum Executor a

orchestratore principale e relegando il processo di convergenza a un task esecutivo isolato su ogni backend. Il principio guida teorico di questo modello è la massimizzazione dell'autonomia e del parallelismo asincrono.

3.4.1 Architettura e Flusso Operativo

Come illustrato nel diagramma di flusso in Figura 3.4, in questa configurazione i ruoli dei componenti sono invertiti, seguendo un modello a orchestrazione distribuita di processi *stateful*:

- **Quantum Executor (Orchestratore Principale):** Agisce come coordinatore di alto livello. La sua responsabilità passa dalla gestione tattica dei batch all'allocazione strategica delle risorse. Tramite una `SplitPolicy` di partizionamento, fraziona il budget globale di shot tra le M QPU disponibili (assegnando a ciascuna una quota $B_{loc} \approx B_{tot}/M$). Assegna quindi a ogni backend un macro-task autonomo: “esegui il ciclo incrementale gestendo la tua convergenza locale, entro il limite della tua quota di budget”.
- **Ciclo Incrementale (Task Esecutivo Locale):** Un’istanza indipendente dell’Incremental Execution Framework viene eseguita per ogni QPU. Ciascun processo locale è *stateful*: gestisce la propria cronologia, la propria distribuzione cumulativa e i propri criteri di arresto, operando in totale autonomia dagli altri.

Il flusso operativo ipotizzato è radicalmente diverso e asincrono:

1. Il Quantum Executor dispaccia M processi di convergenza indipendenti, uno per ciascuno degli M backend disponibili.
2. Ogni processo locale esegue il proprio ciclo incrementale, richiedendo batch di shot al proprio backend fino a quando non raggiunge la stabilità secondo i propri criteri locali o fino a quando non esaurisce gli shot a disposizione.
3. L’Orchestratore principale attende il completamento di *tutti* gli M processi.

4. Solo al termine di tutte le esecuzioni locali, una MergePolicy finale aggrega le M distribuzioni, già dichiarate stabili individualmente, per produrre il risultato finale.

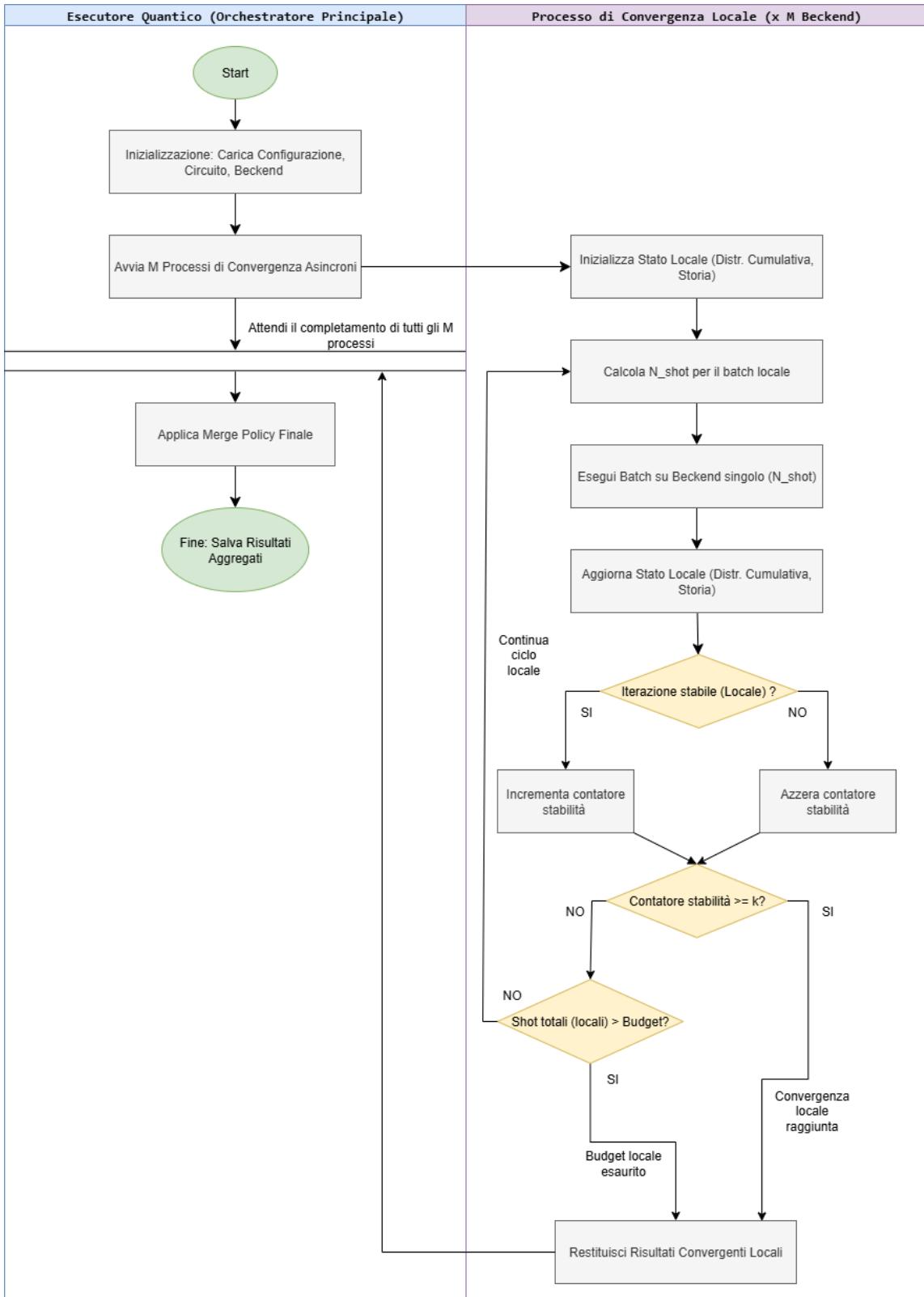


Figura 3.4: Diagramma di flusso operativo dell'architettura a Convergenza Locale. Il controllo è distribuito, con cicli incrementali indipendenti eseguiti su ogni backend.

3.4.2 Criticità Fondamentali e Vizi Metodologici Ipotizzati

Nonostante il potenziale vantaggio in termini di latenza, si ipotizza che i benefici teorici di questa architettura siano messi in ombra da criticità metodologiche e strutturali che ne comprometterebbero la validità e l'efficienza [2].

Ipotesi di Disallineamento Concettuale tra Ottimizzazione e Obiettivo. La criticità principale risiede nella definizione del target di ottimizzazione. L'architettura punta alla stabilità delle singole distribuzioni parziali, mentre l'interesse scientifico dell'utente è rivolto alla stabilità del risultato aggregato finale. Ottimizzare M processi indipendenti non garantisce automaticamente l'efficienza o la stabilità della loro combinazione, rischiando di disperdere risorse per soddisfare criteri locali (la convergenza del singolo backend) che potrebbero non essere rilevanti per l'obiettivo globale.

Ipotesi di Inefficienza Strutturale delle Risorse (Spreco di Shot). L'autonomia locale potrebbe portare a un significativo spreco di shot. La stabilità di una stima statistica dipende primariamente dalla soppressione della varianza, legata al numero di campioni. In uno scenario con più QPU, è probabile che la stima globale (se fosse calcolata) si stabilizzi molto prima che il backend più “lento a convergere” (perché più rumoroso o statisticamente sfortunato) completi il suo task locale. Il sistema sarebbe quindi costretto a eseguire un numero eccessivo di shot su quel backend, il cui contributo marginale alla stabilità globale sarebbe ormai nullo.

Ipotesi Sbilanciamento del Peso nell'Aggregazione Finale. La fase di merge della proposta locale, realizzata tramite la somma diretta dei conteggi (pooling), introduce una criticità specifica in ambienti eterogenei. Sebbene statisticamente corretta per aggregare campioni indipendenti, questa procedura assegna implicitamente un peso maggiore al backend che ha eseguito il maggior numero di shot (N_{shot}).

Questo crea un paradosso operativo: se un backend è particolarmente rumoroso e richiede un numero elevato di shot per raggiungere la condizione di convergenza, il suo contributo finirà per dominare la distribuzione aggregata finale. Al contrario, un backend

più “pulito” e performante, che converge rapidamente con pochi shot, avrà un impatto minoritario sul risultato complessivo.

Di conseguenza, la strategia a convergenza locale rischia di soffrire di un effetto di “diluizione della qualità”: l’accuratezza del risultato finale potrebbe essere compromessa proprio dai membri del team che hanno richiesto più risorse per convergere, oscurando il contributo dei backend più efficienti.

Complessità Implementativa e Violazione dei Principi di Design. Infine, si ipotizza che questa architettura sia “invasiva”. Richiederebbe modifiche sostanziali al nucleo di Quantum Executor per trasformarlo da un esecutore *stateless* a un orchestratore di processi *stateful*, contraddicendo la sua filosofia di design come motore semplice e agnostico.

3.5 Analisi Comparativa e Scelta Architetturale

Dopo aver esaminato le due architetture singolarmente, questa sezione conduce un confronto diretto per motivare la selezione di una delle due come candidata per l’implementazione e la validazione sperimentale. La scelta si basa sulle ipotesi formulate riguardo la loro superiorità metodologica, efficienza e robustezza.

3.5.1 Confronto Diretto: Sintesi delle Proprietà Ipotizzate

La Tabella 3.1 riassume le differenze fondamentali ipotizzate tra le due proposte, evidenziando la superiorità metodologica ipotizzata dell’Architettura a *Convergenza Globale* su tutti i criteri chiave, ad eccezione della potenziale latenza temporale.

Tabella 3.1: Confronto delle proprietà ipotizzate per le architetture candidate.

Caratteristica	Convergenza Globale	Convergenza Locale
Target di Ottimizzazione	Corretto: Stabilità della distribuzione aggregata , coerente con l'obiettivo scientifico.	Disallineato: Stabilità di M distribuzioni locali isolate , ignorando l'effetto combinato.
Efficienza Risorse	Alta: L'arresto avviene sulla base dell'informazione condivisa, evitando sprechi su singoli backend.	Bassa: Spreco di shot su backend lenti/rumorosi che hanno un contributo marginale nullo sul risultato globale.
Qualità dell'Aggregazione	Bilanciata: Converge alla media della flotta, mediando i bias dei singoli dispositivi in modo equo.	Sbilanciata: Paradosso del peso: i backend peggiori (che richiedono più shot) finiscono per dominare il risultato finale, diluendo la qualità.
Implementazione	Modulare: Utilizza i componenti esistenti <i>as-is</i> , garantendo manutenibilità.	Invasiva: Richiede modifiche profonde al nucleo delle librerie <i>stateful</i> .
Flessibilità	Alta: Logica di convergenza centralizzata e facilmente configurabile.	Bassa: Logica di convergenza “cablata” all'interno dei task esecutivi distribuiti.

3.5.2 Analisi di Robustezza: il Problema della “Falsa Convergenza”

Una delle critiche più pertinenti all'approccio a convergenza globale è il rischio di “falsa convergenza”. Questo scenario si verificherebbe se due o più QPU con bias sistematici (rumore hardware) perfettamente opposti si annullassero a vicenda nella media, inducendo il sistema a credere che la distribuzione aggregata sia stabile e portando a un arresto prematuro. Sebbene fisicamente improbabile, è fondamentale analizzare la robustezza intrinseca dell'architettura a questo fenomeno.

L'analisi si basa sulla distinzione cruciale tra due fonti di errore:

- **Rumore Statistico (Shot Noise):** È l'incertezza casuale dovuta al campionamento finito. **Questo è il target primario del framework incrementale**, e diminuisce all'aumentare del numero di shot.
- **Bias Sistematico (Rumore Hardware):** È un errore deterministico che sposta la distribuzione misurata. **Non diminuisce con più shot.**

Si ipotizza che il rumore statistico stesso agisca come un meccanismo di sicurezza intrinseco contro la falsa convergenza. Nelle prime fasi di un'esecuzione, quando il numero di shot è basso, lo shot noise domina sul bias sistematico. Anche nell'improbabile scenario di bias hardware perfettamente opposti, le distribuzioni misurate in piccoli batch sarebbero campioni statistici altamente variabili. La loro aggregazione produrrebbe una distribuzione globale che fluttua ampiamente di iterazione in iterazione. Il criterio di stop, basato sulla distanza tra distribuzioni cumulative successive (es. TVD), rileverebbe correttamente questa alta varianza come un segno di instabilità, impedendo un arresto prematuro.

Quando il numero di shot aumenta, il rumore statistico viene soppresso e le distribuzioni locali convergono stabilmente alle loro rispettive “valli rumorose”, determinate dal bias unico di ciascun dispositivo. La distribuzione globale, di conseguenza, converge alla **media aritmetica** di queste valli. Il sistema si arresta quando questa media è stabile, fornendo non un risultato ideale, ma una **convergenza onesta** alla migliore stima della performance media della flotta, che è esattamente l'obiettivo scientifico desiderato [2].

3.5.3 Formulazione dell'Ipotesi Sperimentale

L'analisi comparativa condotta in questo capitolo ha messo in luce le differenze concettuali e i compromessi intrinseci delle due architetture candidate. Sulla base di questa analisi di design, è ora possibile formulare un'ipotesi chiara, che guiderà la validazione sperimentale descritta nei capitoli successivi.

L'*Architettura a Convergenza Globale* appare, su basi progettuali, la candidata più promettente. Il suo allineamento con l'obiettivo scientifico primario – la stabilità della distribuzione aggregata finale – e le sue proprietà di efficienza informativa e modulari-

tà, la rendono un’ipotesi di soluzione robusta. I suoi compromessi, come la latenza di sincronizzazione, sembrano essere di natura ingegneristica e potenzialmente mitigabili.

Al contrario, l’*Architettura a Convergenza Locale* solleva perplessità di natura metodologica. L’ipotesi è che il suo focus sulla stabilità locale possa portare a un’inefficienza strutturale nell’uso delle risorse (spreco di shot) e a un’ambiguità statistica nella fase di aggregazione finale. Questi potenziali difetti, essendo di natura concettuale, potrebbero limitarne l’efficacia pratica.

Pertanto, l’ipotesi centrale che verrà testata in questo lavoro è la seguente:

Si ipotizza che l’Architettura a Convergenza Globale dimostrerà un’efficienza superiore (in termini di costo computazionale in shot) rispetto all’Architettura a Convergenza Locale, a parità di accuratezza del risultato finale.

Sulla base di questa ipotesi, l’*Architettura a Convergenza Globale* viene selezionata come approccio primario per lo sviluppo del prototipo e per la validazione. Il confronto diretto delle sue performance con quelle dell’architettura a convergenza locale, descritto nel capitolo sulla validazione sperimentale, avrà lo scopo di confermare o confutare questa ipotesi in modo empirico.

3.6 Implementazione del Prototipo

A seguito della selezione dell’architettura a Convergenza Globale, è stato sviluppato un prototipo software funzionante, denominato *OrCHEstratore*, per tradurre i concetti teorici in un sistema tangibile. Il codice e la documentazione di questo prototipo sono disponibili pubblicamente [7]. Questa sezione ne descrive in dettaglio la struttura, le scelte tecnologiche e i meccanismi implementativi, illustrando come il design dell’*Architettura a Convergenza Globale* sia stato concretamente realizzato.

3.6.1 Panoramica del Software e Stack Tecnologico

Il prototipo è stato sviluppato interamente in linguaggio **Python** (versione 3.12+), scelto per il suo vasto ecosistema di librerie scientifiche e per il supporto di prima classe of-

ferto dai principali framework di calcolo quantistico. Le dipendenze chiave del progetto includono:

- **Qiskit:** Utilizzato come libreria di riferimento per la definizione e la manipolazione dei circuiti quantistici.
- **Quantum Executor:** Il motore per l'esecuzione distribuita, utilizzato come libreria esterna per gestire tutte le interazioni con i backend quantistici.
- **Incremental Execution Framework:** La libreria contenente le funzioni per la logica di convergenza, integrata come modulo locale.

3.6.2 Flusso Esecutivo

Il prototipo è stato progettato come uno strumento da linea di comando il cui comportamento è interamente guidato da un file di configurazione in formato JSON. Una volta avviata l'esecuzione, il sistema segue un flusso di lavoro ben definito:

1. **Inizializzazione:** La classe `Orchestrator` viene istanziata. Il costruttore legge il file di configurazione, inizializza il `Quantum Executor`, scopre i backend disponibili e prepara le funzioni di criterio.
2. **Ciclo Incrementale:** Viene avviato il ciclo di esecuzione. A ogni iterazione, l'`Orchestratore` calcola il numero di shot necessari e delega l'esecuzione di un batch distribuito al `Quantum Executor`. L'output a console fornisce un resoconto in tempo reale del progresso, mostrando il numero di shot eseguiti, la distribuzione cumulativa e le metriche di stabilità calcolate.
3. **Terminazione:** Il ciclo si arresta al raggiungimento della convergenza o del budget massimo di shot.
4. **Output Finale:** Al termine, viene generato un singolo file JSON nella cartella `risultati_esperimenti`. Questo file contiene un report completo e auto-consistente dell'esperimento, includendo la configurazione utilizzata, le statistiche finali (shot

totali, tempo, etc.), la storia dettagliata della convergenza e la distribuzione di probabilità finale, garantendo la piena riproducibilità.

3.6.3 Architettura del Codice: la Classe Orchestrator

Il cuore del prototipo è la classe Orchestrator (`orchestratore.py`), che incarna il ruolo del controllore a ciclo esterno. Le sue responsabilità sono nettamente separate e gestite da metodi specifici.

Il Costruttore (`__init__`): Setup e Configurazione Il costruttore agisce come il “regista” che prepara il set prima dell’inizio delle riprese. Le sue operazioni principali sono:

- `_carica_configurazione`: Legge e valida il file JSON fornito.
- `_inizializza_quantum_executor`: Crea l’istanza di Quantum Executor, gestendo dinamicamente le credenziali dei provider cloud leggendole da variabili d’ambiente per motivi di sicurezza.
- `_filtra_backend_attivi`: Interroga il Quantum Executor per scoprire quali dei backend richiesti sono effettivamente online e disponibili, rendendo il sistema robusto a provider temporaneamente offline.
- `_carica_circuito`: Carica il circuito quantistico da un file QASM.
- `_costruisci_criteri`: Questo è un metodo chiave che implementa un design a *factory*. Legge i nomi e i parametri dei criteri dal file di configurazione (es. “`delta_distance`” con “`soglia`”: 0.01) e li usa per istanziare le funzioni Python corrispondenti dalla libreria `inc_exc`. Questo permette di cambiare la logica di convergenza senza toccare il codice dell’Orchestratore.

Il Motore Esecutivo (`run_experiment`): il Ciclo di Controllo Questo metodo implementa il ciclo `while` principale. Ad ogni iterazione, orchestra la sequenza di operazioni:

- **Decisione:** Invoca la funzione `next_shots_fn` per determinare il numero di shot del batch.
- **Delega:** Chiama il metodo `_esegui_batch_distribuito`, che funge da wrapper per `QuantumExecutor.run_experiment`. Questo metodo astrae completamente l'esecuzione distribuita: l'Orchestratore non sa come gli shot vengono divisi o uniti, riceve solo il risultato aggregato del batch.
- **Aggiornamento dello Stato:** Aggiorna la distribuzione cumulativa e le cronologie con i nuovi dati.
- **Valutazione:** Invoca le funzioni `stop_crit` e `stab_crit` per verificare le condizioni di terminazione.

Al termine del ciclo, invoca `_salva_risultati` per generare il report finale.

3.6.4 Il Sistema di Configurazione Data-Driven via JSON

Una scelta di design fondamentale è l'adozione di un approccio *data-driven*. L'intero comportamento dell'esperimento è controllato da un singolo file di configurazione JSON. Di seguito un esempio commentato basato su `config_orchestratore.json`.

```

1  {
2      "nome_esperimento": "Configurazione_Orchestratore",
3      "percorso_circuito": "circuits/bell_state.qasm",
4
5      "parametri_incrementali": {
6          "://" : "
=====
7          " // " : " IMPOSTAZIONI GENERALI DEL CICLO INCREMENTALE",
8          "://" : "
=====
9          " ,
10         "shot_iniziali": 50,

```

```

11     /**: "Numero di shot da eseguire nella primissima iterazione.",  

12  

13     "max_shot_totali": 20000,  

14     /**: "Budget massimo di shot. L'esecuzione si ferma se viene  

superato.",  

15  

16  

17     /**: "  

=====  

",  

18     /**: " CRITERIO DI STOP: Decide se una singola iterazione è  

stabile.",  

19     /**: "  

=====  

",  

20     "criterio_stop": {  

21  

22         "tipo": "delta_distance",  

23  

24         "parametri": {  

25             "soglia": 0.001,  

26             /**: "La soglia di distanza (TVD) sotto cui l'iterazione è  

stabile.",  

27  

28             "offset": 1  

29         }  

30     },  

31  

32     /**: "  

=====  

",  

33     /**: " CRITERIO DI STABILITA': Decide quando l'INTERO  

esperimento è finito.",  

34     /**: "  

=====  

",

```

```

35     "criterio_stabilita": {
36         "tipo": "consecutivo",
37         "parametri": {
38             "k": 30,
39             "///": "Il numero di iterazioni stabili CONSECUTIVE richieste
40             per terminare."
41         }
42     },
43     "///": "
=====
",
44     "///": " STRATEGIA NEXT SHOTS: Decide quanti shot eseguire nel
45     prossimo batch.",
46     "///": "
=====
",
47     "strategia_next_shots": {
48         "tipo": "costante",
49         "parametri": {
50             "shot_per_batch": 50
51         }
52     },
53     "///": "
=====
",
54     "///": " PARAMETRI MINIMALI PER QUANTUM EXECUTOR",
55     "///": "
=====
",
56     "parametri_quantum_executor": {
57         "inizializzazione": {
58             "providers": {
59                 "local_aer": null,

```

```

61     "ibmq": {
62         "token": "leggi_da_env:IBM_QUANTUM_TOKEN"
63     }
64 }
65 },
66
67 /**
68 * SEZIONE DI ESECUZIONE PER OGNI BATCH
69 */
70
71 "esecuzione_batch": {
72     "provider_e_backend": {
73         "local_aer": [ "aer_simulator", "fake_torino", "fake_kolkata" ],
74         "ibmq": [ "ibmq_qasm_simulator" ]
75     },
76     "split_policy": "uniform",
77     "merge_policy": "simple_aggregate",
78     "opzioni_esecuzione": {
79         "multiprocess": false,
80         "wait": false
81     }
82 }
83 }

```

Code 3.1: Esempio di file di configurazione per l'Orchestratore, in formato JSON.

Il file è diviso in tre blocchi principali:

1. **Impostazioni Generali:** Contiene metadati come nome_esperimento e percorso_circuito.
2. **Parametri Incrementali:** Controlla il “cervello” strategico.

- `shot_iniziali` e `max_shot_totali`: Definiscono il budget di shot.
- `criterio_stop`: Specifica la logica per considerare stabile una singola iterazione. Il “`tipo`” scelto è il “`delta_distance`” e i “`parametri`” (es. “`soglia`”) ne configurano il comportamento.
- `criterio_stabilita`: Definisce quando terminare l’intero esperimento. Il tipo “`consecutivo`” richiede un numero ‘`k`’ di iterazioni stabili di fila.
- `strategia_next_shots`: Definisce quanti shot aggiungere. Il tipo “`costante`” usa un valore fisso, mentre “`dinamica`” adatta il numero di shot alla velocità di convergenza osservata.

3. Parametri di Quantum Executor:

Controlla il “braccio” operativo.

- `inizializzazione.providers`: Specifica quali provider attivare. Il valore “`leggi_da_env:VAR`” permette di caricare token API in modo sicuro.
- `esecuzione_batch.provider_e_backend`: Mappa i provider ai backend su cui distribuire gli shot.
- `esecuzione_batch.split_policy`: Definisce la strategia di suddivisione (es. “`uniform`”).
- `esecuzione_batch.merge_policy`: Definisce la strategia di aggregazione (es. “`simple_aggregate`”).

Questo sistema di configurazione gerarchico trasforma l’Orchestratore in un motore di esecuzione generico, garantendo la massima versatilità e riproducibilità.

4. Metodologia Sperimentale

Dopo aver definito nel capitolo precedente le due architetture candidate, questo capitolo descrive in dettaglio la metodologia sperimentale progettata per la loro validazione rigorosa. L’obiettivo è trascendere l’analisi teorica, raccogliendo dati quantitativi su larga scala per misurare le performance, caratterizzare i compromessi e rispondere in modo empirico alle ipotesi formulate.

Lo scopo di questo studio sperimentale è duplice. In primo luogo, si intende validare l’efficacia intrinseca di un’architettura ibrida dotata di criteri di arresto dinamici. In secondo luogo, si mira a confrontare in modo equo e quantitativo la filosofia a controllo globale con quella a controllo locale, al fine di caratterizzare il trade-off fondamentale tra le due. Per raggiungere tale scopo, la campagna di test è stata progettata per investigare l’efficienza delle strategie di arresto, quantificare il compromesso tra le due proposte e analizzare l’impatto dei parametri di configurazione sulla performance.

Per rispondere a queste domande in modo sistematico, è stato sviluppato un framework di analisi riproducibile, i cui componenti metodologici verranno descritti in dettaglio nelle sezioni successive.

4.1 Ambiente di Simulazione

Per condurre una validazione su larga scala in modo rigoroso e riproducibile, l’intera metodologia sperimentale si fonda sull’utilizzo del dataset `qsimbench` [5]. Questa scelta strategica permette di superare i limiti pratici associati all’accesso diretto a hardware quantistico reale quali costi elevati, lunghi tempi di attesa e l’irriplicabilità intrinseca dovuta al *drift* temporale delle caratteristiche dei dispositivi, pur mantenendo un elevato grado di realismo.

`qsimbench` fornisce un vasto archivio di risultati pre-registrati, ottenuti da decine di algoritmi eseguiti per diverse dimensioni di qubit e su molteplici backend che emulano il

comportamento rumoroso di dispositivi IBM reali. Questo approccio garantisce che ogni esperimento condotto in questa tesi sia:

- **Realistico:** I dati non provengono da simulatori ideali, ma da modelli di rumore empirici che riflettono le imperfezioni e i bias sistematici di hardware specifico.
- **Riproducibile:** Poiché i dati sono pre-calcolati, ogni esecuzione della pipeline di analisi produce risultati identici, una condizione essenziale per la validità scientifica.
- **Scalabile:** Il dataset permette di testare le architetture proposte su migliaia di scenari sperimentali, un’impresa irrealizzabile su hardware fisico a causa dei vincoli di tempo e di costo.

L’interazione con il dataset è gestita tramite un’architettura di simulazione “on-demand”. A differenza di un approccio che richiederebbe il pre-caricamento di un’enorme mole di dati, il sistema è progettato per interrogare dinamicamente la libreria `qsimbench` solo quando necessario, simulando fedelmente il ciclo di vita di un’esecuzione distribuita reale. Questo adattamento è cruciale: permette di testare la **logica di controllo** del sistema incrementale in modo isolato, utilizzando `qsimbench` come un oracolo deterministico che fornisce dati realistici su richiesta. In questo modo, le performance misurate dipendono esclusivamente dall’efficacia dell’architettura e dei criteri di convergenza, e non da fattori esterni e incontrollabili come la latenza di rete o la disponibilità delle QPU.

4.2 Definizione del Benchmark: l’Oracolo

Per valutare l’efficienza di una strategia di arresto dinamico è indispensabile definire un benchmark oggettivo. Il confronto con la *ground truth* teorica risulta insufficiente, poiché l’obiettivo in un ambiente NISQ non è eliminare il rumore, ma gestirlo efficientemente. È necessario, quindi, identificare il **punto di convergenza ottimo**: il numero minimo di shot necessari per raggiungere una stabilità finale e duratura.

A tale scopo è stato implementato un algoritmo “Oracolo” che analizza a posteriori l’intera traiettoria di convergenza. Tuttavia, poiché le due architetture proposte (Globale e

Locale) gestiscono le risorse in modo diverso, l’Oracolo deve adattarsi ai vincoli specifici di ogni configurazione anziché calcolare un singolo punto ottimo universale.

4.2.1 Strategia di Generazione dei Dati: i Team Virtuali

Un requisito fondamentale della validazione è la comparabilità diretta tra i dati dell’Oracolo e quelli sperimentali. La differenza strutturale tra le due architetture impone una gestione differenziata del budget di shot:

- **Scenario Globale (Traiettoria Unica):** Nell’Architettura Globale, i backend collaborano per costruire un’unica distribuzione di probabilità. Il budget di 20.000 shot è condiviso e alimenta una sola curva di convergenza. L’Oracolo, in questo caso, analizza questa singola traiettoria aggregata per identificare il punto di stabilità globale.
- **Scenario Locale (Traiettorie Parallelle):** Nell’Architettura Locale, il budget viene frammentato. Se un team è composto da N backend, ciascuno opera indipendentemente con un budget massimo di $20.000/N$ shot. Questo produce N traiettorie di convergenza distinte.

Per gestire questa dualità e garantire un confronto equo, l’Oracolo adotta una strategia di generazione basata su **Team Virtuali**. Per ogni coppia algoritmo-dimensione, vengono generate e analizzate molteplici istanze di esecuzione simulate, variando il limite massimo di shot:

- **Scenario Base (Globale):** Viene calcolato il punto ottimo su una traiettoria completa di 20.000 shot. Questo dato costituisce il benchmark per l’Architettura Globale.
- **Scenari Frazionati (Locali):** Viene simulata l’esecuzione dello stesso backend ipotizzando la sua appartenenza a team di dimensione variabile ($N = 2, 3, 4, 5$). Per ciascun caso, la traiettoria viene troncata al limite corrispondente (es. 5.000 shot per $N = 4$) e viene ricalcolato il punto di stabilità locale entro quel vincolo.

Questo approccio crea un archivio di benchmark condizionali. In fase di analisi (Capitolo 5), per valutare un team di N backend, il sistema recupera per ciascuno il punto ottimo calcolato con il vincolo corretto ($20.000/N$), garantendo che la somma dei benchmark locali sia matematicamente coerente con il budget totale dell’architettura globale.

4.2.2 Metodologia di Analisi: l’Algoritmo a Finestra Inversa

Una volta generate le traiettorie (estese o troncate) secondo la logica dei team virtuali, su ciascuna di esse viene applicato un algoritmo di analisi specializzato, denominato a “Finestra Inversa”. Questa tecnica è progettata per identificare il punto di arresto ideale in modo robusto contro le “false convergenze” iniziali.

Il funzionamento dell’algoritmo è il seguente:

1. **Definizione del Tunnel di Stabilità:** Si considera il valore finale della serie, $\text{TVD}_{\text{finale}}$, che rappresenta la massima accuratezza raggiunta al termine dei 20.000 shot. Attorno a questo valore viene definito un “tunnel di stabilità” verticale, dato dall’intervallo $[\text{TVD}_{\text{finale}} - H, \text{TVD}_{\text{finale}} + H]$, dove H è un iperparametro di tolleranza, impostato a 0.003 in questo studio.
2. **Scansione Inversa:** L’algoritmo scorre la serie temporale *all’indietro*, partendo dal penultimo punto verso l’inizio.
3. **Identificazione del Punto di Rottura:** La scansione si arresta non appena incontra un punto il cui valore di TVD si trova *al di fuori* del tunnel di stabilità. Questo punto è interpretato come la fine del più lungo periodo di stabilità finale.
4. **Determinazione del Punto di Convergenza:** Il punto di convergenza ottimale è definito come l’iterazione immediatamente successiva al punto di rottura. Se l’intera serie temporale rientra nel tunnel, la convergenza viene fatta risalire alla prima iterazione utile.

Le Figure 4.1 e 4.2 forniscono una rappresentazione visiva dettagliata di questo processo, applicato a titolo esemplificativo all’algoritmo `random` a 7 qubit. Ogni curva colorata

in entrambi i grafici rappresenta la traiettoria di una diversa configurazione hardware (un singolo backend o un team).

La Figura 4.1 si concentra sull’analisi dell’accuratezza. Le diverse traiettorie mostrano come team differenti convergano a “noise floor” eterogenei: alcune curve si stabilizzano su valori di TVD più bassi (maggiore accuratezza), mentre altre su valori più alti. I punti rossi, calcolati dall’Oracolo, indicano per ciascuna curva il momento esatto in cui si entra nella fase di stabilità. L’area semitrasparente, il “tunnel di stabilità”, visualizza la regione di tolleranza utilizzata dall’algoritmo per prendere questa decisione, proiettata all’indietro a partire dal valore finale di ogni curva.

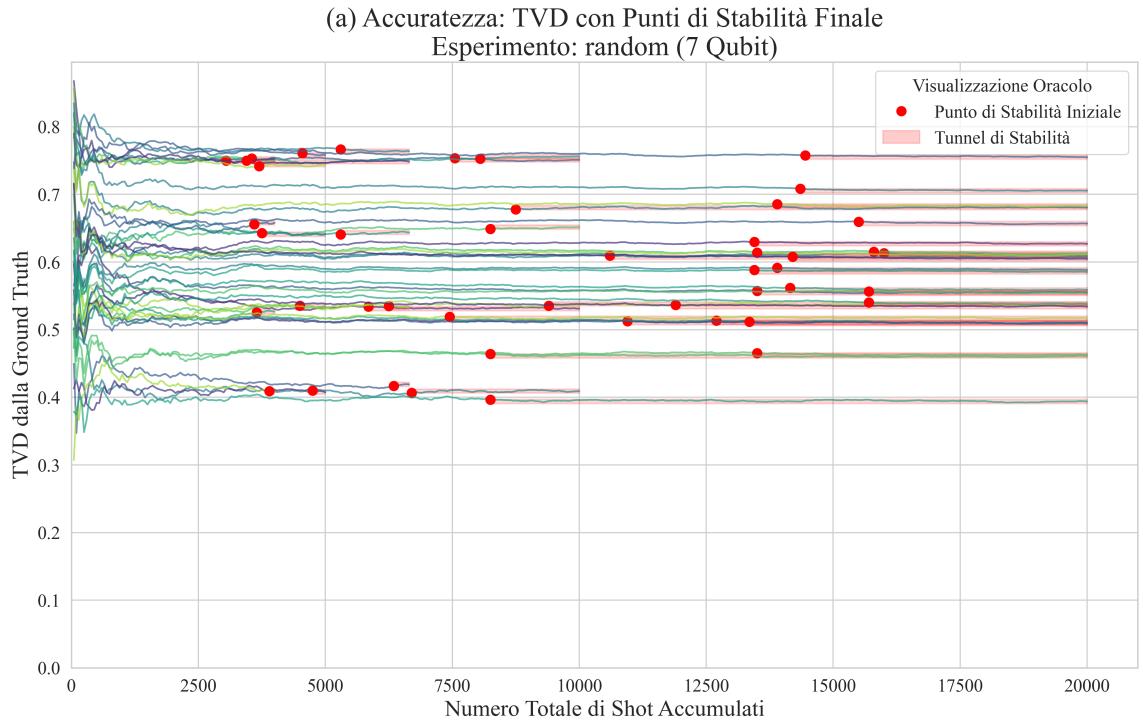


Figura 4.1: Analisi dell’Oracolo sull’accuratezza per l’algoritmo `random` (7 Qubit). Ogni curva rappresenta l’evoluzione della TVD rispetto alla Ground Truth per un diverso team di backend o backend singolo. I punti rossi indicano i punti di stabilità ottimali identificati dall’Oracolo.

La Figura 4.2, invece, mostra l’evoluzione della stabilità interna della stima. Le curve tracciano la Delta-TVD, ovvero la variazione della distribuzione cumulativa tra un’iterazione e la successiva. Si osserva chiaramente il comportamento atteso: una variazione

iniziale molto alta, che decade rapidamente in modo esponenziale man mano che vengono accumulati più shot. Questo grafico conferma visivamente l'esistenza di un punto di “rendimenti decrescenti”, in cui l'aggiunta di nuovi shot contribuisce in modo sempre meno significativo a modificare la stima, giustificando la ricerca di un punto di arresto ottimale.

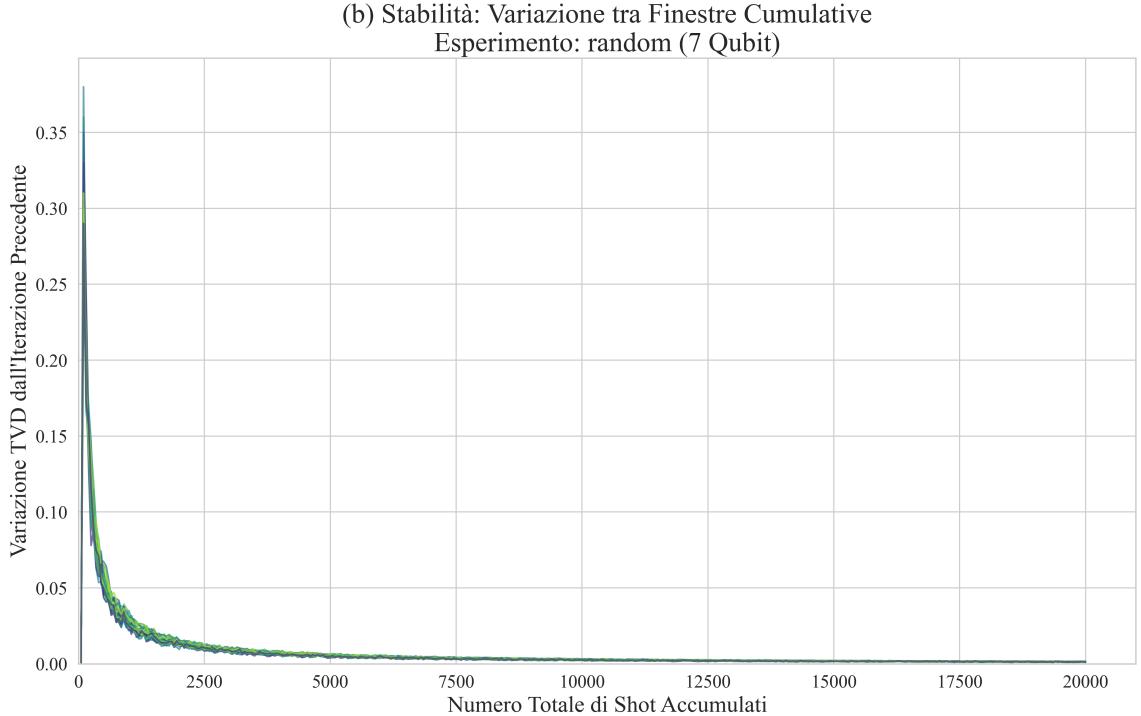


Figura 4.2: Analisi della stabilità per l'algoritmo `random` (7 Qubit). Ogni curva mostra l'evoluzione della Delta-TVD (variazione tra iterazioni successive) per un diverso team di backend o beckend singolo, evidenziando il rapido decadimento della volatilità della stima.

4.3 Simulazione delle Architetture Dinamiche

Una volta definito il benchmark ottimale tramite l'Oracolo, il passo successivo è simulare l'esecuzione delle architetture dinamiche per raccogliere i dati di performance. Questa fase costituisce il cuore sperimentale del progetto. A differenza degli approcci a budget fisso delle fasi precedenti, qui viene messo in atto anche il processo di *Incremental Execution*:

l'esecuzione si arresta autonomamente non appena un criterio di stabilità statistica viene soddisfatto.

4.3.1 Adattamento del Prototipo Orchestratore per la Simulazione

Lo script di simulazione è costruito attorno alla classe `Orchestrator`, derivata direttamente dal prototipo software descritto nel Capitolo 2. Tuttavia, mentre il prototipo è stato progettato per interagire con hardware reale tramite il *Quantum Executor*, questa versione è stata adattata per operare in un ambiente di simulazione basato su `qsimbench`.

La modifica chiave risiede nel metodo `_esegui_batch_simulato`, che sostituisce la chiamata al *Quantum Executor*. Questo metodo emula fedelmente il paradigma *shot-wise*: riceve in input il numero di shot richiesti dal ciclo di controllo, li distribuisce tra i backend secondo una `SplitPolicy` e richiede i dati on-demand a `qsimbench`, per poi aggregarli tramite una `MergePolicy`. Questo adattamento permette di testare la logica di controllo delle architetture in modo isolato, garantendo la riproducibilità e la scalabilità dell'analisi.

Il comportamento dell'Orchestratore è interamente guidato da un file di configurazione in formato JSON, che permette di specificare tutti i parametri del ciclo incrementale (criteri di stop, stabilità, budget di shot, etc.), rendendo il framework estremamente flessibile.

4.3.2 Metodologia di Simulazione per le Due Architetture

Una delle caratteristiche metodologiche più importanti di questo script è la sua capacità di raccogliere i dati per la validazione di entrambe le Proposte, sia a convergenza globale sia a convergenza locale, attraverso un unico processo di esecuzione sistematica. Ciò è reso possibile da un'astuta progettazione dell'analisi.

Lo script itera su ogni possibile scenario sperimentale, definito da una coppia (algoritmo, dimensione) e da ogni possibile sottoinsieme non vuoto di backend disponibili, includendo sia i team (dimensione ≥ 2) sia i backend singoli (dimensione = 1).

Simulazione Diretta dell'Architettura a Convergenza Globale Le esecuzioni condotte su **team di backend** (sottoinsiemi di dimensione ≥ 2) simulano direttamente il comportamento dell'Architettura a Convergenza Globale.

1. **Avvio del Ciclo di Controllo:** L'Orchestratore avvia un singolo ciclo di controllo incrementale per l'intero team.
2. **Distribuzione del Batch** (SplitPolicy): Ad ogni iterazione, un batch di shot viene distribuito tra i membri del team. Questo passaggio emula una SplitPolicy di tipo “uniform”. Il numero di shot del batch viene diviso equamente tra i backend del team, e l’eventuale resto della divisione viene assegnato sequenzialmente ai primi backend della lista. Questo garantisce che il numero totale di shot eseguiti nel batch sia sempre esatto.
3. **Esecuzione e Aggregazione** (MergePolicy): Per ogni backend, viene invocata la libreria qsimbench per ottenere il numero preciso di risultati corrispondenti agli shot assegnati. I risultati parziali (istogrammi di conteggio) restituiti da ogni backend vengono quindi combinati emulando una MergePolicy di tipo *simple aggregate*: i conteggi per ogni stato di output vengono sommati per formare un unico istogramma complessivo per il batch.
4. **Valutazione e Terminazione:** L’istogramma del batch viene sommato a quello cumulativo delle iterazioni precedenti. La stabilità viene valutata su questa **distribuzione cumulativa globale**. Il processo termina quando la stima aggregata soddisfa i criteri di stabilità definiti nella configurazione.

Il costo computazionale (in termini di shot) e la qualità del risultato (TVD finale) registrati al termine di queste esecuzioni rappresentano le performance dirette della proposta ad Architettura a Convergenza Globale.

Simulazione Indiretta dell’Architettura a Convergenza Locale Per riprodurre fedelmente il comportamento dell’Architettura Locale, in cui le risorse sono partizionate staticamente, la simulazione adotta un approccio a **budget frazionato**, speculare alla strategia dei Team Virtuali dell’Oracolo.

Il processo non si limita a eseguire ogni backend singolarmente, ma itera su tutte le possibili dimensioni del team N (da 2 a 5). Per ogni configurazione:

1. **Partizionamento del Budget:** Il budget totale B_{tot} (es. 20.000 shot) viene diviso per la dimensione del team N . A ogni backend viene assegnata una quota massima $B_{\text{locale}} \approx B_{\text{tot}}/N$ (gestendo i resti della divisione intera per garantire che la somma sia esatta).
2. **Esecuzione Vincolata:** L'Orchestratore avvia il ciclo incrementale per il singolo backend, imponendo B_{locale} come limite massimo invalicabile (`max_shot_totali`).
3. **Terminazione:** Il processo per il singolo backend termina quando raggiunge la propria stabilità locale oppure quando esaurisce la sua quota di budget frazionata.

Questo metodo genera un set di dati granulare che permette di costruire a posteriori le performance di qualsiasi team. Ad esempio, per valutare un team $\{B_1, B_2\}$, si prelevano i risultati delle esecuzioni di B_1 e B_2 effettuate con il vincolo $N = 2$ e si sommano i costi effettivi. Ciò garantisce che il confronto con l'Architettura Globale avvenga a parità rigorosa di budget massimo disponibile.

Al termine di questo processo, viene prodotto un file di dati dettagliato in formato CSV, che registra le metriche chiave per ogni esecuzione e fornisce l'input grezzo per l'analisi comparativa.

4.4 Metodologia di Valutazione Comparativa

Dopo aver descritto come vengono simulati i comportamenti delle due architetture candidate, questa sezione definisce il framework metodologico utilizzato per confrontarle in modo rigoroso e quantitativo. Verranno presentati: la configurazione di test selezionata, le metriche di valutazione e il processo di analisi comparativa che garantisce un confronto equo.

4.4.1 Metrica di Performance Primaria: Differenza di Shot

Per valutare l'efficienza computazionale delle strategie di arresto dinamico, viene adottata un'unica metrica di performance principale: la **differenza di shot** rispetto al benchmark ottimale definito dall'Oracolo.

- **Definizione:** Per una data esecuzione, siano S_{usati} il numero di shot totali consumati dalla strategia dinamica e S_{ottimali} il numero di shot identificati dall’Oracolo come punto di stabilità. La metrica è definita come:

$$\text{Differenza di Shot} = S_{\text{usati}} - S_{\text{ottimali}}$$

- **Interpretazione e Obiettivo:** L’obiettivo fondamentale di una strategia di arresto robusta non è minimizzare gli shot a ogni costo, ma garantire di arrestarsi solo dopo aver raggiunto una stabilità consolidata. Pertanto, l’obiettivo di questo studio è identificare configurazioni che producano una **differenza di shot sempre positiva e il più vicina possibile allo zero.**
 - Un valore **positivo** è il risultato desiderato. Indica che la strategia dinamica ha correttamente superato il punto di stabilità minimo definito dall’Oracolo prima di arrestarsi. Un valore positivo ma contenuto (“spreco” di shot) rappresenta un compromesso accettabile, e spesso necessario, per garantire la robustezza del meccanismo di arresto contro fluttuazioni statistiche.
 - Un valore **negativo** è considerato un fallimento della configurazione. Indica un “risparmio” di shot ottenuto tramite un arresto prematuro, avvenuto prima di raggiungere la zona di stabilità identificata dall’Oracolo. Questo comportamento, sebbene apparentemente efficiente, è inaccettabile in quanto espone a un rischio elevato di compromettere l’accuratezza finale del risultato.

4.4.2 Processo di Analisi Comparativa

Per garantire un confronto equo e significativo tra l’Architettura a Convergenza Globale e quella a Convergenza Locale, il framework di analisi implementa un processo di analisi dati strutturato, volto a identificare e confrontare le configurazioni più performanti di entrambe le architetture per ciascun esperimento.

1. **Caricamento e Integrazione dei Dati:** In primo luogo, vengono caricati e uniti i dati di performance dettagliati (generati dalla simulazione) e i punti di convergenza

ottimali (calcolati dall’Oracolo). A ogni esecuzione dinamica viene così associato il suo corrispettivo benchmark ideale per il calcolo delle metriche relative.

2. **Identificazione delle Configurazioni Ottimali per Esperimento:** Poiché l’obiettivo è confrontare la migliore incarnazione di ciascuna architettura, per ogni esperimento (coppia algoritmo-dimensione) viene eseguita una ricerca per identificare le configurazioni più efficienti.

- **Migliore Configurazione a Convergenza Globale:** Viene identificato il *team* di backend (con dimensione $N \geq 2$) che ha raggiunto la convergenza con il minor numero di shot totali. Sia N_{ottimale} la dimensione di questo team.
- **Migliore Configurazione a Convergenza Locale:** Per garantire un confronto a parità di risorse, viene cercato il miglior team per l’architettura locale della stessa dimensione N_{ottimale} . Vengono considerate tutte le possibili combinazioni di N_{ottimale} backend singoli, e viene selezionata quella la cui somma dei costi individuali in shot è minima.

3. **Aggregazione dei Risultati per la Convergenza Locale:** Una volta identificato il team ottimale per l’Architettura a Convergenza Locale, se ne simula il comportamento aggregato *simple aggregate*. I costi in shot dei singoli membri vengono sommati per ottenere il costo totale. Analogamente, le distribuzioni di conteggio finali vengono unite per calcolare la TVD finale complessiva dell’architettura, un passo cruciale per poter confrontare l’accuratezza tra le due proposte.

Questo processo assicura che il confronto finale, che sarà presentato nel prossimo capitolo, avvenga tra le configurazioni più efficienti di entrambe le architetture, a parità di numero di risorse computazionali (backend) impiegate.

4.5 Definizione della Configurazione di Test

Le performance di un’architettura di esecuzione incrementale dipendono in modo critico dai parametri che ne governano il comportamento. Una configurazione troppo “aggressiva”

va” potrebbe portare ad arresti prematuri, compromettendo l’accuratezza, mentre una troppo “conservativa” rischierebbe di annullare i benefici dell’approccio adattivo, sprecando shot.

La configurazione finale selezionata per la campagna sperimentale è il risultato di un compromesso progettuale tra efficienza e robustezza, ed è stata applicata in modo uniforme a tutte le esecuzioni per garantire la confrontabilità dei risultati.

Parametri Generali del Ciclo. I parametri che definiscono i limiti e la granularità del processo sono stati mantenuti fissi per coerenza con le analisi precedenti:

- `shot_iniziali=50`: Un piccolo batch iniziale per avviare il processo di stima.
- `shot_per_batch=50`: Batch di dimensioni ridotte per consentire un monitoraggio a grana fine della dinamica di convergenza.
- `max_shot_totali=20000`: Un budget massimo identico a quello utilizzato per la generazione delle curve dell’Oracolo.

Criteri di Arresto e Stabilità. I parametri che governano la logica di arresto sono stati scelti per favorire un comportamento conservativo, in linea con l’obiettivo di minimizzare il rischio di arresti prematuri.

- **Criterio di Stop:** È stato scelto il criterio `delta_distance`, che valuta la stabilità confrontando la TVD tra la distribuzione cumulativa attuale e quella dell’iterazione precedente.
- **soglia = 0.001:** La scelta di un valore di soglia molto basso è una decisione di design mirata a garantire la robustezza del meccanismo di arresto. Questo valore è inferiore al livello di *shot noise* tipicamente osservato nelle fasi iniziali del campionamento. Di conseguenza, la condizione di stabilità può essere soddisfatta solo quando la variazione della distribuzione è genuinamente minima, e non per effetto di fluttuazioni statistiche casuali.
- **Criterio di Stabilità:** È stato utilizzato il criterio `consecutivo`, che richiede un certo numero di iterazioni stabili di fila per terminare l’esperimento.

- **k = 30:** Il parametro k definisce la “memoria” del criterio di stabilità. È stato scelto un valore elevato per immunizzare il sistema contro arresti prematuri causati da fluttuazioni casuali. Richiedere una stabilità prolungata per 30 iterazioni consecutive (corrispondenti a un periodo di osservazione di 1500 shot) assicura che l’arresto avvenga solo dopo che il sistema ha raggiunto un *plateau* di convergenza consolidato e non una stabilità temporanea.

Questa configurazione, sebbene conservativa, è stata progettata per rispondere all’obiettivo primario di questo studio: confrontare le due architetture in uno scenario in cui l’affidabilità del meccanismo di arresto è privilegiata rispetto alla pura minimizzazione degli shot.

5. Risultati Sperimentali e Analisi

Questo capitolo presenta i risultati quantitativi della validazione sperimentale, mettendo a confronto diretto le performance dell'*Architettura a Convergenza Globale* e dell'*Architettura a Convergenza Locale*. L'analisi si concentra sulla metrica di efficienza primaria, la **Differenza di Shot** rispetto al benchmark ottimale (l'Oracolo), al fine di verificare empiricamente le ipotesi di design formulate nel capitolo precedente.

A differenza delle previsioni teoriche, che ipotizzavano una superiorità sistematica dell'approccio centralizzato, i dati raccolti mostrano un quadro più sfaccettato e dipendente dal contesto. I risultati verranno prima presentati per tipologia di algoritmo e per dimensione del circuito, seguiti da una discussione critica volta a interpretare le dinamiche osservate.

5.1 Analisi per Algoritmo

La Figura 5.1 illustra la Differenza di Shot media calcolata per ciascun algoritmo presente nel dataset di test. Il grafico confronta le performance della migliore configurazione a Convergenza Globale (in blu) con quelle della migliore configurazione a Convergenza Locale (in arancione).

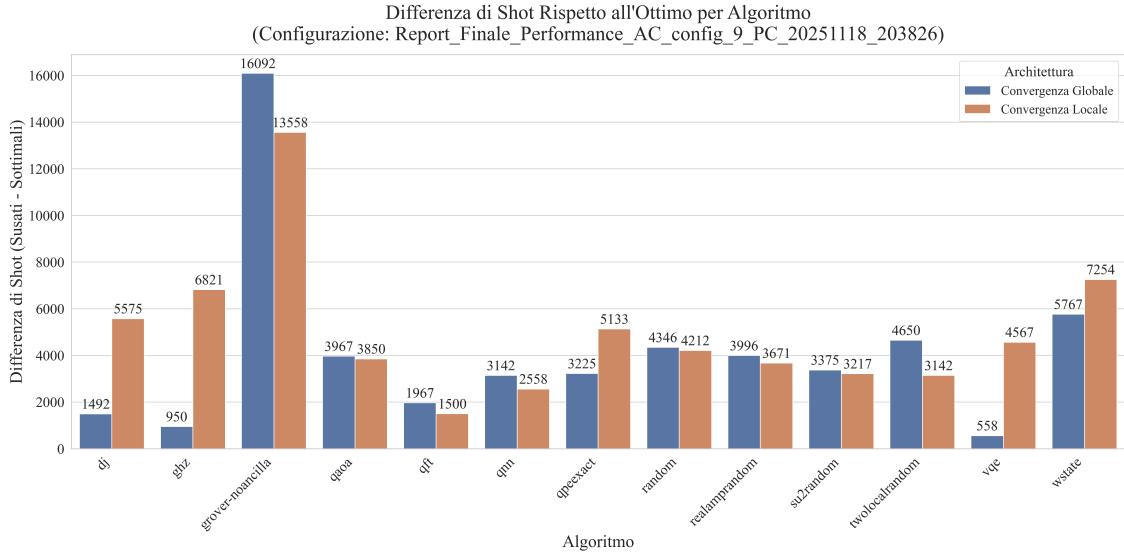


Figura 5.1: Confronto della Differenza di Shot media (rispetto all'Oracolo) tra le due architetture, raggruppata per algoritmo. Valori più bassi indicano una maggiore efficienza.

Per facilitare l'interpretazione dei risultati, l'analisi è stata suddivisa in base alle tre categorie principali di algoritmi quantistici presenti nel dataset: Algoritmi Strutturati, Algoritmi Variazionali (VQA) e Circuiti Randomici/Euristici.

5.1.1 Algoritmi Strutturati

Questa categoria include algoritmi con una struttura circuitale ben definita, progettati per sfruttare l'interferenza quantistica per ottenere risultati specifici. Nel dataset analizzato rientrano: dj (Deutsch-Jozsa), ghz, qpeexact, wstate, qft e grover-noancilla.

I dati mostrano una tendenza prevalente, sebbene non assoluta, a favore dell'**Architettura a Convergenza Globale**:

- Nel caso di dj, la Convergenza Globale registra un'efficienza notevole, con uno scarto dall'ottimo di 1.492 shot contro i 5.575 della Locale.
- Risultati simili si osservano per ghz (950 vs 6.821 shot), qpeexact (3.225 vs 5.133 shot) e wstate (5.767 vs 7.254 shot).

Tuttavia, esistono eccezioni significative in cui l'**Architettura a Convergenza Locale** risulta più performante:

- Per l'algoritmo `qft`, l'approccio Locale ottiene una Differenza di Shot inferiore (1.500) rispetto al Globale (1.967).
- Il caso più rilevante è rappresentato da `grover-noancilla`, che costituisce l'algoritmo più oneroso dell'intero dataset. Qui, l'architettura Locale dimostra una migliore gestione delle risorse, con 13.558 shot rispetto ai 16.092 dell'architettura Globale.

5.1.2 Algoritmi Variazionali (VQA)

Questa classe comprende algoritmi ibridi classico-quantistici che utilizzano circuiti parametrici ottimizzati iterativamente. Il dataset include: `vqe`, `qaoa` e `qnn`.

In questo ambito, i risultati sono eterogenei e non indicano una prevalenza definita:

- L'algoritmo `vqe` mostra una chiara preferenza per l'**Architettura Globale**, che ottiene il risultato migliore in assoluto tra i VQA con soli 558 shot di differenza dall'ottimo, contro i 4.567 della Locale.
- Al contrario, `qnn` favorisce l'**Architettura Locale** (2.558 shot contro 3.142).
- L'algoritmo `qaoa` presenta una sostanziale parità tra le due strategie, con una differenza minima tra Globale (3.967 shot) e Locale (3.850 shot).

5.1.3 Circuiti Random e Ansatz Euristici

L'ultima categoria raggruppa circuiti basati su template standard inizializzati con parametri casuali, spesso usati per benchmarking. Include: `random`, `realamprandom`, `su2random` e `twolocalrandom`.

In questo gruppo si osserva una tendenza verso una maggiore efficienza dell'**Architettura a Convergenza Locale**, o una situazione di parità:

- L'algoritmo `twolocalrandom` evidenzia il divario maggiore a favore della Locale, con 3.142 shot di differenza contro i 4.650 della Globale.

- Anche `realamprandom` e `su2random` registrano prestazioni migliori con l'approccio Locale, sebbene con margini più ridotti.
- Il circuito `random` generico mostra una sostanziale comparabilità tra le due architetture (4.346 vs 4.212 shot).

In sintesi, l'analisi per algoritmo dimostra che nessuna delle due architetture domina universalmente sull'altra. Mentre l'approccio Globale eccelle in molti algoritmi strutturati e nel VQE, l'approccio Locale si dimostra competitivo e spesso superiore nella gestione di circuiti euristici e random.

5.2 Analisi per Dimensione del Circuito

Un'analisi complementare consiste nel raggruppare i risultati in base alla dimensione del circuito, ovvero al numero di qubit, per investigare come la scalabilità del problema influenzi le performance delle due architetture. La Figura 5.2 mostra la Differenza di Shot media per circuiti da 4 a 15 qubit.

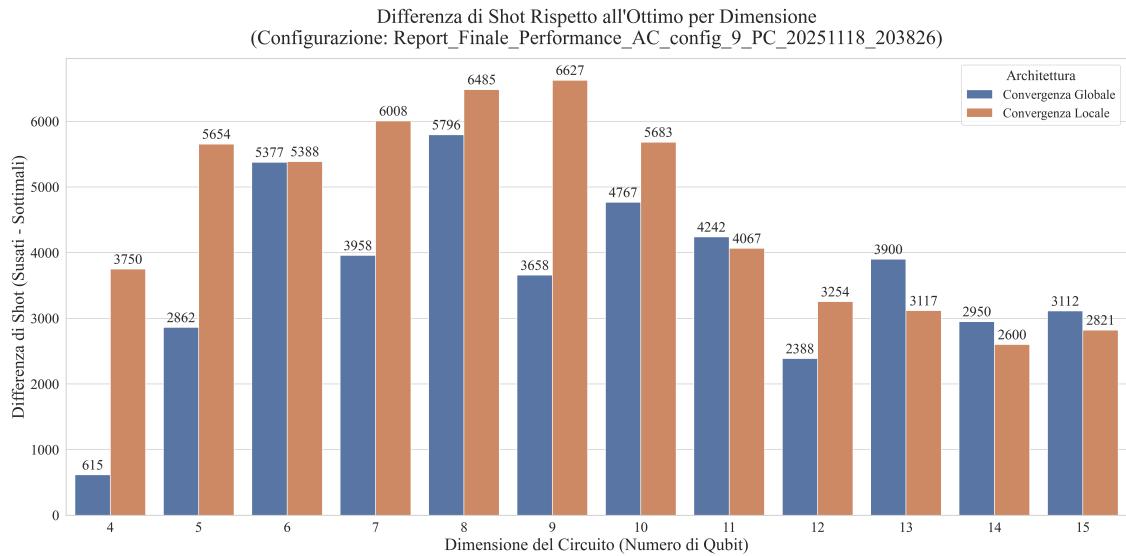


Figura 5.2: Confronto della Differenza di Shot media (rispetto all'Oracolo) tra le due architetture, raggruppata per dimensione del circuito (numero di qubit).

L'osservazione dell'andamento per dimensione rivela due fasi distinte nel rapporto di efficienza tra le due architetture:

- **Dominio Medio-Basso (4-10 Qubit):** Nelle dimensioni inferiori e medie, l'**Architettura a Convergenza Globale** tende a mantenere una maggiore efficienza rispetto alla controparte locale. Il vantaggio è particolarmente marcato agli estremi di questo intervallo:

- A **4 qubit**, l'approccio Globale registra il valore più basso in assoluto (615 shot di differenza), contro i 3.750 della Locale.
- A **5 qubit**, il divario si amplia ulteriormente (2.862 vs 5.654 shot).
- A **9 qubit**, si osserva il picco massimo di inefficienza per l'Architettura Locale (6.627 shot), mentre la Globale riesce a contenere un eccesso di 3.658 shot rispetto all'oracolo, quasi la metà.

Tuttavia, anche in questa fascia non mancano eccezioni: a 6 qubit si registra una sostanziale parità (5.377 vs 5.388 shot), indicando che la superiorità globale non è uniforme.

- **Inversione di Tendenza (11-15 Qubit):** Superata la soglia dei 10 qubit, la dinamica subisce un cambiamento. L'**Architettura a Convergenza Locale** inizia a recuperare competitività, fino a superare l'efficienza della Globale nelle dimensioni maggiori.

- Già a **11 qubit**, l'approccio Locale ottiene un leggero vantaggio (4.067 vs 4.242 shot).
- Dopo un ritorno della Globale a 12 qubit, negli ultimi tre step analizzati (**13, 14 e 15 qubit**) l'Architettura Locale si dimostra costantemente più efficiente. Ad esempio, a 13 qubit registra 3.117 shot di differenza contro i 3.900 della Globale, mantenendo un vantaggio simile anche a 14 e 15 qubit.

Complessivamente, i dati suggeriscono che l'Architettura a Convergenza Globale sia preferibile per circuiti di dimensioni ridotte, dove il coordinamento centralizzato riesce

a minimizzare efficacemente gli shot. Al contrario, all'aumentare della complessità del sistema (e quindi dello spazio di Hilbert), il sovraccarico o la dinamica dell'aggregazione globale sembrano perdere efficacia, rendendo l'autonomia dei singoli backend dell'Architettura Locale una strategia vantaggiosa (o più efficiente) per i circuiti più grandi.

5.3 Discussione Critica dei Risultati

La presentazione oggettiva dei dati nelle sezioni precedenti ha delineato uno scenario complesso, in cui le performance relative delle due architetture si invertono in base al contesto applicativo. Questa sezione si propone di interpretare tali risultati, collegandoli alle ipotesi architettoniche formulate nel Capitolo 3 e fornendo una nuova chiave di lettura basata sulla natura degli algoritmi eseguiti.

5.3.1 Revisione dell’Ipotesi Sperimentale

L’ipotesi centrale di questo lavoro postulava che l’*Architettura a Convergenza Globale* avrebbe dimostrato un’efficienza superiore in modo trasversale, grazie alla sua capacità di aggregare l’informazione e mitigare il rumore prima di valutare la stabilità. I risultati sperimentali, tuttavia, **non validano pienamente questa ipotesi nella sua formulazione assoluta.**

Sebbene l’approccio Globale si confermi superiore per una specifica classe di problemi (algoritmi strutturati e di piccole dimensioni), l’analisi ha rivelato che l’*Architettura a Convergenza Locale* non soffre dell’inefficienza strutturale inizialmente ipotizzata. Al contrario, essa si dimostra spesso la strategia ottimale per circuiti euristici e di grandi dimensioni. Questo risultato suggerisce il superamento di una visione gerarchica (una architettura migliore dell’altra) in favore di una visione **complementare**, dove la scelta dell’architettura dipende dalle caratteristiche intrinseche del carico di lavoro quantistico.

5.3.2 Interpretazione delle Dinamiche: Coerenza vs Indipendenza

La dicotomia osservata tra algoritmi strutturati (favorevoli alla Globale) e algoritmi variazionali/random (favorevoli alla Locale) può essere interpretata analizzando come il rumore interagisce con la struttura dell'output.

- **Il vantaggio Globale nella Struttura:** Gli algoritmi strutturati (es. dj, ghz) tendono a produrre output con picchi di probabilità netti. In questo contesto, l'aggregazione globale agisce come un filtro efficace: il segnale “vero” viene rinforzato costruttivamente dai vari backend, mentre il rumore scorrelato tende a mediarsi. Questo permette all'orchestratore globale di rilevare la stabilità del segnale utile più rapidamente di quanto un singolo backend possa stabilizzare il proprio rumore locale.
- **Il vantaggio Locale nell'Entropia:** Nei circuiti variazionali o randomici, la distribuzione di probabilità in uscita è caratterizzata da un'alta entropia e da un supporto esteso su un ampio sottoinsieme dello spazio di Hilbert, privo di picchi di probabilità netti. Inoltre, ogni backend fisico possiede un'impronta di rumore (bias) unica, che può deformare questa distribuzione complessa in modi diversi. Tentare di forzare una convergenza su una media globale di queste distribuzioni deformate in modo eterogeneo può risultare controproducente: l'orchestratore vede fluttuazioni statistiche maggiori derivanti dal disallineamento tra i backend. In questo scenario, lasciare che ogni backend converga indipendentemente verso la *propria* rappresentazione stazionaria (Convergenza Locale) risulta computazionalmente più economico.

5.3.3 Scalabilità e Complessità

L'analisi dell'efficienza in relazione alla dimensione del circuito evidenzia una transizione nel comportamento delle due architetture, individuando due regimi di funzionamento distinti.

Nei circuiti di dimensioni medio-basse (fino a 10 qubit), l'**Architettura a Convergenza Globale** risulta generalmente più efficiente. In questi scenari, lo spazio degli stati

è ancora relativamente contenuto e la sovrapposizione dei risultati provenienti da backend diversi permette di raggiungere la stabilità statistica sfruttando l'aggregazione come filtro per le fluttuazioni.

Al crescere della dimensione (11-15 qubit), si osserva invece una prevalenza dell'**Architettura a Convergenza Locale**. Questo fenomeno può essere ricondotto alla maggiore complessità dello spazio di Hilbert e alla conseguente divergenza dei profili di rumore dei dispositivi. Su grandi dimensioni, i backend tendono a manifestare bias sistematici sempre più specifici e differenti tra loro. In questo contesto, l'approccio Locale risulta vantaggioso perché disaccoppia il problema della convergenza: ogni backend deve stabilizzarsi solo rispetto alla propria distribuzione rumorosa (internamente coerente), evitando il costo computazionale aggiuntivo che l'architettura Globale deve sostenere per stabilizzare una media aggregata di segnali altamente eterogenei.

6. Conclusioni e Sviluppi Futuri

Questa tesi ha affrontato la sfida dell'esecuzione efficiente di algoritmi quantistici in un contesto computazionale distribuito e rumoroso (NISQ). Il lavoro ha investigato le implicazioni architettonali derivanti dall'integrazione di due paradigmi complementari: l'esecuzione distribuita di tipo *shot-wise*, per lo sfruttamento parallelo di infrastrutture hardware eterogenee, e l'esecuzione incrementale, per l'ottimizzazione adattiva delle risorse di campionamento.

L'obiettivo primario è stato determinare quale gerarchia di controllo, centralizzata (Globale) o distribuita (Locale), offrisse il miglior bilanciamento tra l'accuratezza del risultato finale e il costo computazionale necessario per ottenerlo.

6.1 Sintesi del Lavoro Svolto

Il percorso di ricerca si è articolato in diverse fasi metodologiche. Partendo dall'analisi dello stato dell'arte, è emersa la mancanza di studi sull'applicazione di strategie di *adaptive shot allocation* in contesti multi-backend. Per colmare questa lacuna, sono state formalizzate due architetture candidate:

- **L'Architettura a Convergenza Globale**, che centralizza la logica di arresto valutando la stabilità sulla distribuzione aggregata di tutti i backend.
- **L'Architettura a Convergenza Locale**, che delega la decisione di arresto ai singoli backend, unendo i risultati solo al termine dei processi autonomi.

Per garantire un confronto rigoroso e quantitativo, è stato sviluppato un ambiente di simulazione basato sul dataset qsimbench. Inoltre, è stato definito un benchmark di performance ideale (l'Oracolo) tramite un algoritmo di analisi a posteriori, permettendo di misurare l'efficienza delle strategie dinamiche in termini di “Differenza di Shot” rispetto all'ottimo teorico.

6.2 Riepilogo dei Risultati e Contributi

La validazione sperimentale ha condotto a risultati che ridisegnano le aspettative teoriche iniziali. L'ipotesi che l'Architettura a Convergenza Globale fosse sistematicamente superiore è stata parzialmente confutata dai dati, lasciando spazio a una visione più articolata basata sulla **complementarietà** dei due approcci.

I contributi principali emersi dall'analisi possono essere riassunti come segue:

1. **Identificazione di Regimi di Efficienza Distinti:** Lo studio ha dimostrato che non esiste un'architettura ottimale in ogni scenario analizzato. L'approccio Globale eccelle nella gestione di **algoritmi strutturati** (es. Deutsch-Jozsa, GHZ) e circuiti di piccole dimensioni, dove l'aggregazione funge da filtro efficace per il rumore. Al contrario, l'approccio Locale si rivela superiore per **algoritmi variazionali, euristici** e circuiti di grandi dimensioni (> 10 qubit), dove la gestione autonoma dei singoli profili di rumore risulta computazionalmente meno onerosa.
2. **Caratterizzazione del Trade-off Scalabilità/Coerenza:** È stato osservato un punto di inversione (crossover) nelle performance all'aumentare della dimensione del circuito. Questo risultato suggerisce che, in spazi di Hilbert vasti, il costo di stabilizzare una media globale eterogenea supera il costo di stabilizzare localmente i singoli backend.
3. **Validazione della Robustezza:** Entrambe le architetture, configurate con parametri conservativi, hanno garantito il raggiungimento della stabilità statistica corretta (Differenza di Shot positiva). Questo conferma che l'integrazione tra distribuzione shot-wise ed esecuzione incrementale è tecnicamente solida e capace di produrre risultati scientificamente validi riducendo, in molti casi, l'uso delle risorse rispetto a un approccio statico a budget fisso.

In conclusione, questa tesi contribuisce al campo dell'ingegneria del software quantistico fornendo non solo una metodologia di valutazione, ma anche linee guida pratiche per la scelta dell'architettura di orchestrazione in base alla natura del carico di lavoro: centra-

lizzata per problemi strutturati e “compatti”, distribuita per problemi euristici e su larga scala.

6.3 Sviluppi Futuri

Il lavoro svolto apre diverse prospettive di ricerca per raffinare ulteriormente l’efficienza dell’esecuzione quantistica ibrida.

- **Validazione su Hardware Quantistico Reale:** La simulazione ha isolato l’efficienza in termini di shot, ma ha astratto dai tempi di latenza di rete e di accodamento (*queue time*). Dato che l’Architettura a Convergenza Locale ha dimostrato un’efficienza inaspettata sugli shot ed è intrinsecamente asincrona, una validazione su hardware reale potrebbe rivelare un vantaggio ancora più netto in termini di *Wall-Clock Time*, non dovendo attendere il backend più lento ad ogni iterazione.
- **Orchestrazione Adattiva (Meta-Scheduling):** Alla luce dei risultati ottenuti, un’evoluzione naturale del framework sarebbe l’implementazione di un meta-orchestratore potenzialmente capace di selezionare l’architettura (Globale o Locale) analizzando le caratteristiche del circuito in input (es. struttura, profondità, numero di qubit) prima dell’esecuzione.
- **Politiche di Distribuzione Noise-Aware:** Attualmente, la distribuzione degli shot avviene in modo uniforme. L’integrazione di politiche che allocano più shot ai backend più performanti (o che convergono più rapidamente) potrebbe migliorare ulteriormente l’efficienza di entrambe le architetture, specialmente nel caso della Convergenza Globale, riducendo l’impatto dei dispositivi più rumorosi sulla media aggregata.
- **Integrazione con Tecniche di Mitigazione degli Errori:** L’arresto dinamico riduce l’errore statistico (shot noise). Un passo successivo cruciale è l’integrazione con tecniche come la *Zero-Noise Extrapolation* (ZNE) [1] per mitigare anche l’errore sistematico (bias hardware). Sarebbe interessante indagare se l’estrapolazione debba

avvenire a livello locale (su ogni backend) o globale (sulla media), parallelamente alla logica di convergenza qui studiata.

- **Estensione ad Altri Criteri di Arresto Adattivi.** La validazione ha utilizzato una configurazione di test basata sul criterio della `delta_distance`. Sarebbe interessante investigare le performance dell'architettura distribuita utilizzando criteri di arresto più sofisticati, come quelli basati su machine learning o Reinforcement Learning discussi nello Stato dell'Arte, per esplorare ulteriori possibilità di ottimizzazione e automazione.

Bibliografia

- [1] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S. Kottmann, Tim Menke, Wai-Keong Mok, Soon-Joo Sim, Leong-Chuan Kwek, and Alán Aspuru-Guzik. Noisy intermediate-scale quantum algorithms. *Reviews of Modern Physics*, 94(1):015004, 2022.
- [2] Giovanni Del Bianco. Analisi comparativa di architetture per l'integrazione di esecuzione quantistica distribuita e incrementale, 2024. Documento di analisi interna per tesi di laurea. Università di Pisa.
- [3] Giuseppe Bisicchia. Incremental execution for reducing quantum circuit execution shots. Internal report, University of Pisa, 2023. Unpublished.
- [4] Giuseppe Bisicchia, Jose García-Alonso, Juan M. Murillo, and Antonio Brogi. Distributing quantum computations, by shots. In *Service-Oriented Computing – ICSOC 2023*, pages 363–377. Springer Nature Switzerland, 2023.
- [5] Giuseppe Bisicchia and the qsimbench contributors. qsimbench: A library and dataset of pre-computed quantum experiments. <https://github.com/GBisi/qsimbench>, 2024.
- [6] Giuseppe Bisicchia and the Quantum Executor contributors. Quantum executor: A unified interface for quantum computing. <https://github.com/GBisi/quantum-executor>, 2024.
- [7] Giovanni Del Bianco. Hybrid Quantum Orchestrator: Distributed and Incremental Execution, 2025.
- [8] J. Garcia-Alonso, D. B-C, J. Garcia-Rodriguez, J. M. Murillo, J. Criado, and A. Brogi. Quantum software as a service through a quantum api gateway. *IEEE Internet Computing*, 26(1):34–41, 2022.

- [9] Andi Gu, Angus Lowe, Pavel A. Dub, Patrick J. Coles, and Andrew Arrasmith. Adaptive shot allocation for fast convergence in variational quantum algorithms. *arXiv preprint arXiv:2108.10434*, 2021.
- [10] Seyed Sajad Kahani and Amin Nobakhti. A novel framework for shot number minimization in quantum variational algorithms. *arXiv preprint arXiv:2307.04035*, 2023.
- [11] Youngmin Kim, Enhyeok Jang, Hyungseok Kim, Seungwoo Choi, Changhun Lee, Donghwi Kim, Woomin Kyoung, Kyujin Shin, and Won Woo Ro. Distribution-adaptive dynamic shot optimization for variational quantum algorithms. *arXiv preprint arXiv:2412.17485*, 2024.
- [12] Jonas M. Kübler, Andrew Arrasmith, Lukasz Cincio, and Patrick J. Coles. An adaptive optimizer for measurement-frugal variational algorithms. *Quantum*, 4:263, 2020.
- [13] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O’Brien. Quantum computers. *Nature*, 464(7285):45–53, 2010.
- [14] Senwei Liang, Linghua Zhu, Xiaolin Liu, Chao Yang, and Xiaosong Li. Artificial-intelligence-driven shot reduction in quantum measurement. *Chemical Physics Reviews*, 5(4), 2024.
- [15] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, Cambridge, 10th edition, 2010.
- [16] Koustubh Phalak and Swaroop Ghosh. Shot optimization in quantum machine learning architectures to accelerate training. *IEEE Access*, 11:41514–41523, 2023.
- [17] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.
- [18] Manav Seksaria and Anil Prabhakar. Shots and variance on noisy quantum circuits. *arXiv preprint arXiv:2501.03194*, 2025.

- [19] Linghua Zhu, Senwei Liang, Chao Yang, and Xiaosong Li. Optimizing shot assignment in variational quantum eigensolver measurement. *Journal of Chemical Theory and Computation*, 20(6):2390–2403, 2024.