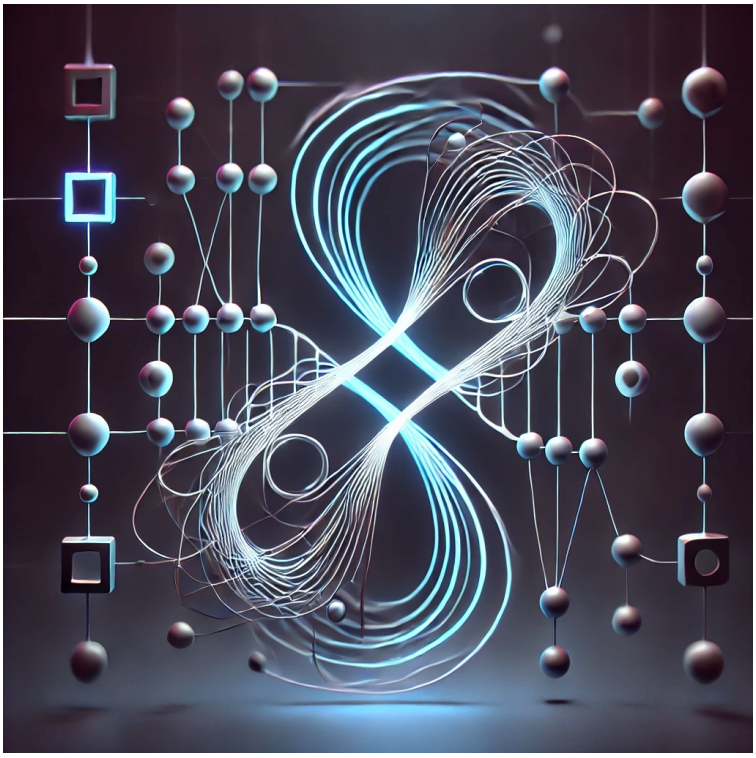
CHALMERS



Datadriven kvantfelskorrigering genom avkodning av repetitionskoden med grafneurala nätverk

Kandidatarbete inom civilingenjörsprogrammet Teknisk fysik

Simon Benstorp, Liam Bulut Falkenström, Simon Eriksson, Alexander Jonsson, Lovisa Petersson, Vidar Petersson

INSTITUTIONEN FÖR FYSIK

CHALMERS TEKNISKA HÖGSKOLA
Göteborg 2025
www.chalmers.se

Kandidatarbete 2025

Datadriven kvantfelskorrigering genom avkodning av repetitionskoden med grafneurala nätverk

Data-driven quantum error correction through decoding of the repetition code using graph neural networks

Simon Benstorp, Liam Bulut Falkenström,
Simon Eriksson, Alexander Jonsson,
Lovisa Petersson, Vidar Petersson

CHALMERS

Institutionen för fysik
Chalmers tekniska högskola
Göteborg 2025

Datadriven kvantfelskorrigering genom avkodning av repetitionskoden
med grafneurala nätverk
Simon Benstorp, Liam Bulut Falkenström, Simon Eriksson,
Alexander Jonsson, Lovisa Petersson, Vidar Petersson

Handledare: Mats Granath, Institutionen för fysik vid Göteborgs universitet
Examinator: Jan Swenson, Institutionen för fysik vid Chalmers tekniska högskola

Kandidatarbete 2025
Institutionen för fysik
Chalmers tekniska högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslagsbild: Abstrakt representation av sammanflätade kvantbitar i repetitionskoden [1].

Skriven i LaTeX
Göteborg 2025

Datadriven kvantfelskorrigering genom avkodning av repetitionskoden
med grafneurala nätverk
Simon Benstorp, Liam Bulut Falkenström, Simon Eriksson,
Alexander Jonsson, Lovisa Petersson, Vidar Petersson
Institutionen för fysik
Chalmers tekniska högskola

Sammandrag

Kvantfelskorrigering är en av de stora utmaningarna för att kunna realisera användbara kvantdatorsystem. Repetitionskoden är en enklare korrektionskod som detekterar bitfel *eller* fasfel. För att extrahera information från repetitionskoden krävs en lämplig avkodare. Arbetet syftar till att jämföra två olika avkodningsmetoder, en klassisk algoritm, Minimum-Weight Perfect Matching (MWPM) samt ett maskininlärningsbaserat grafneuralt nätverk (GNN). En modulär fasfelsdetekterande repetitionskod konstruerades och exekverades på en IBM kvantprocessor. De uppmätta syndromen användes som träningsunderlag för att träna GNN:et på Chalmers beräkningskluster *Alvis*. Det undersöktes även hur avkodningstiden för GNN skalar med ökande kodareor, jämfört med MWPM. Arbetet fann att GNN-avkodaren uppnådde likvärdig eller marginellt högre logisk noggrannhet, α_L, än MWPM, särskilt nära den kodarea där hyperparameteroptimeringen genomfördes. Vidare presterade GNN med en något högre eller motsvarande logisk noggrannhet jämfört med MWPM för kodavstånd och antal tidsrepetitioner d,$d_t \leq 9$. Detta då det för varje d,d_t användes syndrom från samma exekveringstillfälle på kvantdatorn. Däremot uppvisade GNN konsekvent högre logisk felsannolikhet, p_L, och presterar sämre än MWPM när den testades på andra syndrom, vilket kan försvåra praktisk tillämpning i miljöer med varierande felfördelningar.

Abstract

Quantum error correction is one challenge faced when implementing quantum computer systems. Repetition code is a simpler correcting decoder that detects bit- or phase-flip errors. To fully benefit from the repetition code, an appropriate decoder is required. Therefore, this report aims to compare two different decoding methods, a classical algorithm known as Minimum-Weight Perfect Matching, MWPM and a machine learning-based method using Graph Neural Networks, GNN. A modular phase-flip detecting repetition code is constructed and executed on an IBM Quantum processor. The measured syndromes are used as training data for GNN, which is trained on Chalmers high-performance computing cluster *Alvis*. The report also investigates how decoding time scales with increasing code area. Results showed that the GNN decoder achieved marginally higher logical accuracy α_L compared to MWPM, particularly near the code area where optimization of the hyperparameters was performed. Furthermore, GNN achieved slightly higher logical accuracy compared to MWPM for code distance and time repetitions d,$d_t \leq 9$. This is because the decoder was tested on syndromes generated during the same quantum computer execution as the training data. However, GNN also showed consistently higher logical error probability p_L and was less effective than MWPM when tested on different syndromes, which may hinder practical application in environments with varying error distributions.

Nyckelord: kvantfelskorrigering, repetitionskod, kvantfasfel, grafneurala nätverk GNN, syndrommätning, Qiskit, IBM Quantum, IBM Marrakesh.

Förord

Vi vill rikta ett stort tack till vår handledare Mats Granath för alla många och långa diskussioner. Mats har varit en ovärderlig del i arbetets utveckling och resultat. Vi vill även rikta ett tack till Moritz Lange, Isak Bengtsson och Jacob Olsson för deras stöd och att de delgivit sina tidigare erfarenheter med oss.

Beräkningarna möjliggjordes av resurser tillhandahållna av National Academic Infrastructure for Supercomputing in Sweden (NAISS) och Swedish National Infrastructure for Computing (SNIC) vid Chalmers Centre for Computational Science and Engineering (C3SE), delvis finansierade av Vetenskapsrådet genom bidragsavtal nr. 2022-06725.

Arbetets experimentella mätningar på IBMs kvantdatorer möjliggjordes av stöd från Wallenberg Centrum for Quantum Technology (WACQT) Testbed som drivs av Chalmers Next Labs och finansieras av Knut och Alice Wallenbergs stiftelse.

Simon B., Liam, Simon E., Alexander, Lovisa & Vidar
Göteborg, maj 2025

Beteckningar

Nedan följer nomenklaturen för akronymer, operatorer, parametrar och beteckningar som har använts genom hela detta arbete.

Akronymer

QPU	Kvantprocessor (eng. Quantum Processor Unit)
GNN	Grafneuralt nätverk (eng. Graph Neural Network)
MWPM	Minimum Weight Perfect Matching
MLD	Maximum Likelihood Decoder
QPU	Quantum Processing Unit
ReLU	Rectified Linear Unit (aktiveringsfunktion)
SGD	Stochastic Gradient Descent (optimeringsmetod)

Operatorer

X_i,Y_i,Z_i	Paulioperatorer verkande på kvantbit i
H	Hadamard-grinden
U_{CNOT}	Kontrollerade NOT-grinden

Parametrar

d	Kodavstånd
d_t	Tidsrepetitioner
dd_t	Kodarea
N	Antal samplingar av kvantkretsen i en körning.

Beteckningar

$\mathbf{s}^{(t)}$	Syndrommätning vid tidsrepetition t
S	Rumtidsmatris över upprepade syndrommätningar
p	Fysisk felfrekvens
p_{est}	Estimerad fysisk felfrekvens
α_L	Logisk noggrannhet
p_L	Logisk felfrekvens
f_L	Logisk felandel (innan avkodning)

Innehåll

1

Inledning

Kvantdatorer har potential att revolutionera beräkningsvetenskapen genom att utnyttja kvantmekanikens principer för att lösa vissa problem som är beräkningsmässigt olösbara för klassiska datorer [2], [3], [4], [5]. Till skillnad från klassiska bitar, som endast kan anta värdena noll eller ett, kan kvantbitar existera i ett superpositionstillstånd vilket möjliggör massiv parallellism i beräkningar [6], [7]. Trots denna fördel är kvantdatorer intrinsiskt brusiga och känsliga för störningar från sin omgivning på grund av dekoherens, mätfel och styrfel [8]. Detta begränsar deras praktiska användbarhet och gör kvantfelskorrigering nödvändig.

Eftersom kvantmekanikens icke-kloningsteorem omöjliggör direkt kopiering av kvantinformation behöver traditionella redundansbaserade felkorrigeringstekniker modifieras [9]. Istället används kvantfelskorrigeringskoder där kvantinformation sammanflätas över flera fysiska kvantbitar och pariteten mellan bitarna mäts i form av *syndrommätningar* [10]. Genom denna metod kan kvantfel upptäckas och korrigeras utan att själva kvanttillståndet direkt mäts. En fundamental utmaning är att tolka syndrommätningarna från dessa kodade kvanttillstånd och identifiera kvantfel i realtid, vilket kräver effektiva och noggranna *avkodare*. Traditionella avkodningsmetoder, som matchningsalgoritmer, förutsätter en explicit modell av felfördelningen, vilket kan vara begränsande när experimentella förhållanden varierar. Dessutom är de ofta beräkningsmässigt komplexa och svåra att implementera i realtid, särskilt vid skalning till större system med fler kvantbitar [11].

Ett lovande tillvägagångssätt är datadriven avkodning i form av *grafneurala nätverk* (GNN, eng. *Graph Neural Network*) för att förbättra prestandan hos kvantfelskorrigering [12]. Maskininlärningsbaserade metoder kan tränas på stora mängder experimentella eller simulerade syndrommätningar och identifiera komplexa mönster i dessa. När modellen väl är tränad kan den utföra inferens i realtid utan behov av omfattande beräkningar under körning, vilket potentiellt leder till snabbare och mer noggranna avkodningar än konventionella metoder.

Tidigare studier har dock enbart utforskat möjligheterna med datadriven avkodning utifrån helt simulerade eller begränsade experimentella syndrommätningar [12], [13]. I takt med ökad tillgänglighet till kvantdatorsystem, exempelvis IBM Quantum [14], ökar möjligheterna att generera stora mängder experimentella syndrommätningar och undersöka olika avkodningsmetoder på riktiga kvantdatorsystem. Experimentella mätningar kan belysa fördelarna med datadrivna avkodares förmåga att lära och anpassa sig till verkliga systemfel som inte modelleras väl i simuleringar.

1.1 Syfte och mål

Syftet med detta arbete är att undersöka hur grafneurala nätverk kan användas för datadriven avkodning inom kvantfelskorrigering, med särskilt fokus på syndrommätningar genererade med *repetitionskoden*, en enkel kvantfelskorrigerande kod, exekverad på en fysisk kvantprocessor hos IBM Quantum. Målet är att träna en GNN-avkodare på experimentell mätdata och jämföra dess prestanda med en traditionell matchningsalgoritm (Minimum-Weight Perfect Matching, MWPM), med avseende på noggrannhet, generaliserbarhet och tidskomplexitet samt hur de skalar med antalet kvantbitar.

Målet är att implementera repetitionskoden i `Qiskit`, samla in experimentell mätdata på en fysisk kvantprocessor, träna GNN-modellen på ett beräkningskluster och slutligen testa dess prestanda på nya mätserier. Arbetet avser inte att utveckla en fullskalig korrektionskod, utan att belysa fördelar och begränsningar hos maskininlärningsbaserad avkodning på verkliga systemfel som inte modelleras väl i simuleringar.

1.2 Avgränsningar

Endast repetitionskoden har använts som korrektionskod, vilken endast kan upptäcka och korrigera bitfel eller fasfel, men inte båda samtidigt. I detta arbete har fokus lagts på fasfel i grundtillståndet $|+\rangle$ i konjugatbasen, medan bitfel helt har uteslutits. Uteslutningen av den ena är nödvändig eftersom hantering av godtyckliga kvantfel (både bit- och fasfel) kräver mer avancerade koder, exempelvis topologiska ytkoder.

All experimentell data har samlats in på IBM Marrakesh, en 156-kvantbits *Heron r2*-processor. Genom att begränsa datainsamlingen till en enda maskin har variationer orsakade av olika hårdvarukonfigurationer eller kalibreringsskillnader undvikits, samtidigt som en av IBMs mest stabila enheter vid arbetets start har utnyttjats.

Hyperparameteroptimeringen av GNN-modellen har begränsats av tillgänglig beräkningstid på det använda beräkningsklustret. Endast parametrar för en enda repetitionskodsstorlek har optimerats. För övriga koder baseras parametrar på dessa optimerade parametrar, vilket kan påverka prestandajämförelser mot en fullständig optimering per konfiguration.

2

Teori

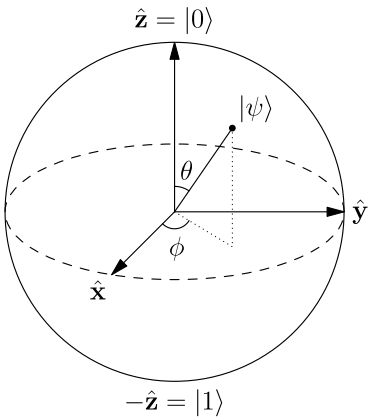
Nedan redogörs för relevant teori som ligger till grund för arbetet och dess resultat. Först introduceras grundläggande begrepp och principer inom kvantberäkning och kvantinformation. Därefter förklaras kvantfelskorrigering ingående och hur en korrektionskod tillsammans med en avkodare kan användas för att detektera kvantfel i logiska kvantbitar. Sedan beskrivs kortfattat utformningen av matchningsalgoritmen MWPM och dess funktion som avkodare. Slutligen redovisas grundläggande teori inom artificiella neurala nätverk och applikationen av grafneurala nätverk som ett verktyg inom kvantfelskorrigering.

2.1 Kvantdatorer och kvantberäkning

En *kvantbit* (eng. *qubit*) är den grundläggande enheten för kvantdatorberäkning, analogt med biten i klassisk datoranvändning. Till skillnad från en klassisk bit, vilken måste vara antingen 0 eller 1, kan en kvantbit existera i en superposition av dessa tillstånd. Tillståndet kan representeras med Dirac-notation som en linjärkombination av de två ortogonala basvektorerna $|0\rangle = \begin{bmatrix}1 & 0\end{bmatrix}^T$ och $|1\rangle = \begin{bmatrix}0 & 1\end{bmatrix}^T$, ofta kallade *beräknings-* eller z-basen, enligt $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, där $\alpha, \beta \in \mathbb{C}$ som uppfyller $|\alpha|^2 + |\beta|^2 = 1$ [15, ss. 13-14]. Eftersom en godtycklig global fas $e^{i\gamma}$ inte påverkar mätbara storheter kan tillståndet parametriseras med två reella vinklar $\theta \in [0,\pi]$ och $\phi \in [0,2\pi]$ så att

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle.$$

Detta gör att kvanttillståndet kan representeras i ett två-nivåsystem som en punkt på enhetssfären i $\mathbb{R}^3$, kallad *Bloch-sfären*, se figur 2.1 [15, s. 15].



Figur 2.1: Bloch-sfären, en geometrisk representation i $\mathbb{R}^3$ av kvanttillstånd för ett två-nivåsystem. Hämtad från [16], CC-BY-SA.

Alla tillstånd som kan beskrivas med en vektor $|\psi\rangle$ i Hilbertrummet kallas *rena tillstånd*. Dessa är idealiserade kvanttillstånd där all information om systemet är känd. I praktiken är det dock ofta fallet att kvantsystem interagerar med en omgivning eller att tillståndsberedningen är statistisk, vilket leder till att systemet inte kan beskrivas av en enda vektor. I dessa fall används istället en *densitetsoperator* ρ, som är en hermitesk operator med spår 1, för att representera systemets tillstånd

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|, \tag{2.1}$$

där varje $|\psi_i\rangle$ är ett rent tillstånd och p_i är en klassisk sannolikhet med $\sum_i p_i = 1$ [15, s. 99]. Ett tillstånd ρ är alltså ett rent tillstånd om och endast om det kan skrivas som $\rho = |\psi\rangle\langle\psi|$, vilket också innebär att $\rho^2 = \rho$ och $\mathrm{tr}(\rho^2) = 1$ [15, s. 101].

2.1.1 System med flera kvantbitar och sammanflätning

Ett kvantsystem bestående av flera kvantbitar beskrivs av ett sammansatt Hilbertrum, vilket erhålls genom tensorprodukten av de individuella kvantbitarnas Hilbertrum. Om varje kvantbit beskrivs av ett tvådimensionellt komplext vektorrum, $\mathcal{H}_i = \mathbb{C}^2$, så är det sammansatta rummet för n kvantbitar $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \cdots \otimes \mathcal{H}_n$, vilket har dimension 2^n. Detta innebär att tillståndet för hela systemet kan beskrivas som en vektor i ett 2^n-dimensionellt vektorrum [10]. För ett system med exempelvis två kvantbitar spänns tillståndsrymden upp av de fyra ortogonala basvektorerna

$$|00\rangle, \quad |01\rangle, \quad |10\rangle, \quad |11\rangle.$$

Ett generellt tillstånd av två kvantbitar skrivs som en linjärkombination

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle,$$

där amplituderna $\alpha_{ij} \in \mathbb{C}$ uppfyller $\sum_{i,j} |\alpha_{ij}|^2 = 1$ [15, s. 16].

En fundamental egenskap hos system med flera kvantbitar är förekomsten av *sammanflätade tillstånd*. Ett tillstånd sägs vara sammanflätat om det inte kan skrivas som en produkt av tillstånd för de enskilda kvantbitarna, det vill säga

$$|\psi\rangle \neq |\phi_1\rangle \otimes |\phi_2\rangle, \tag{2.2}$$

för alla möjliga tillstånd $|\phi_1\rangle, |\phi_2\rangle$ i respektive delsystem [10]. Sammanflätning innebär att kvantbitarnas tillstånd inte längre kan betraktas oberoende av varandra; mätning av en kvantbit påverkar sannolikhetsfördelningen för mätutfallet av den andra oavsett fysisk separation.

2.1.2 Kvantlogiska grindar

En fundamental aspekt av kvantberäkning är manipulationen av kvanttillstånd genom kvantlogiska grindar. Eftersom kvanttillstånd representeras som vektorer i ett Hilbertrum, beskrivs grindar verkande på n kvantbitar som operationer av unitära matriser $U \in \mathbb{C}^{2^n \times 2^n}$, vilket innebär att $UU^\dagger = I$. De vanligaste enbitsgrindarna är de tre Paulimatriserna $X = \sigma_x$,

$Y = \sigma_y$ och $Z = \sigma_z$. De kan ses som enhetsrotationer på Bloch-sfären för ett två-nivåsystem $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ enligt

$$X|\psi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} |\psi\rangle = \alpha|1\rangle + \beta|0\rangle$$
$$Y|\psi\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} |\psi\rangle = i\alpha|1\rangle - i\beta|0\rangle$$
$$Z|\psi\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} |\psi\rangle = \alpha|0\rangle - \beta|1\rangle,$$

där X-grinden är kvantanalog till en klassisk NOT-grind (bitflipp), Z-grinden inverterar den relativa fasen (fasflipp) och Y-grinden kan även skrivas som X och Z enligt $Y = iXZ$ [10].

Ytterligare en grundläggande grind är Hadamard-grinden som definieras utifrån relationen $X = HZH$ enligt

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Hadamard-grinden möjliggör skapande av superpositionstillstånd och en alternativ representation i *konjugat-* eller *x-basen* $\{|+\rangle, |-\rangle\}$ [10] genom att transformera basvektorerna i beräkningsbasen $\{|0\rangle, |1\rangle\}$ enligt

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle\,, \quad H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle\,.$$

Den kontrollerade NOT-grinden (CNOT) är en två-kvantbitsgrind som används för att skapa sammanflätade tillstånd enligt ekvation (2.2) [15, s. 21]. På den andra kvantbiten (målet) appliceras en X-grind om och endast om den första kvantbiten (kontrollen) är $|1\rangle$ och kan uttryckas på matrisform enligt

$$U_{\text{CNOT}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

2.2 Kvantfelskorrigering

Till skillnad från klassisk felkorrigering, där informationen kan replikeras och redundans kan användas för att identifiera och korrigera fel, ställs kvantfelskorrigering inför flera fundamentala utmaningar [15, s. 427]. Dessa inkluderar icke-kloningsteoremet [9], det faktum att kvantfel är kontinuerliga snarare än diskreta och att mätningar i regel kollapsar kvanttillståndet.

2.2.1 Kvantfel och diskretisering

I praktiken är kvantbitar mycket känsliga för sin omgivning. I en ideal kvantdator förutsätts det att kvantbitar endast påverkas av avsiktliga operationer. I verkligheten är däremot kvantbitar ständigt utsatta för störningar och brus. Detta leder till att deras tillstånd förändras på ett okontrollerat sätt över tid, även kallat *dekoherens*.

För att beskriva generella fel modelleras störningar av en *felkanal* $\mathcal{E}(\rho)$, en transformation av densitetsoperatorn ρ, enligt

$$\rho' = \mathcal{E}(\rho) = \sum_i E_i \rho E_i^\dagger$$

där E_i är feloperatorer som uppfyller $\sum_k E_k^\dagger E_k = I$ och i allmänhet utgör ett kontinuerligt och oändligt antal möjliga störningsutfall [15, ss. 425–434].

För att kvantfelskorrigering ska vara praktiskt möjlig krävs att dessa kontinuerliga fel kan projiceras ner på ett ändligt antal underrum. Det görs genom att införa en *projektiv mätning* P [15, ss. 435–438] så att $PE_i^\dagger E_j P = \alpha_{ij} P \quad \forall i,j$. P är exempelvis en syndrom-mätning i en felkorrigerande kod (se nästa avsnitt 2.2.2) som projekterar tillståndet in i ortogonala delrum, varje delrum motsvarande en specifik diskret feloperator E_i. Efter mätningen kollapsar alltså den ursprungliga blandningen av kontinuerliga störningar till en sannolikhetsfördelning över ett ändligt antal feletiketter [15, ss. 438–440]. Ett vanligt exempel är den *Pauli-kanalen*, som antar att kvanttillståndet i form av ρ från ekvation (2.1) med sannolikheterna p_X, p_Y, p_Z drabbas av Pauli-fel

$$\mathcal{E}(\rho) = \sum_{i=0}^{3} E_i \rho E_i = (1 - p_X - p_Y - p_Z)\rho + p_X X\rho X + p_Y Y\rho Y + p_Z Z\rho Z,$$

där $E_0 = \sqrt{1-p}I$, $E_1 = \sqrt{p}X$, $E_2 = \sqrt{p}Y$, $E_3 = \sqrt{p}Z$. Denna uppdelning gör det möjligt att konstruera kvantfelskorrigerande koder som specifikt riktar in sig på bitflipp X, fasflipp Z och deras kombination Y, vilket i praktiken räcker för att hantera godtyckliga fel [12]. Denna egenskap, även kallat diskretisering av kvantfel, är en nyckelfaktor för att kunna detektera och applicera korrekta återställningsoperationer.

2.2.2 Repetitionskoden

Trebitars-repetitionskoden är den enklaste kvantfelskorrigerande koden som i teorin entydigt kan detektera och korrigera ett enstaka Pauli X-fel *eller* Pauli Z-fel [15, s. 427]. För i z-basen bygger den på att inledningsvis koda en godtycklig datakvantbit $|\psi\rangle_1 = \alpha|0\rangle + \beta|1\rangle$ till en logisk kvantbit $|\psi\rangle_L = \alpha|000\rangle + \beta|111\rangle$ genom sammanflätning.

För att därefter detektera eventuella fel utan att direkt mäta kvantbitarna, vilket skulle kollapsa kvanttillståndet, används *stabilisatorer*. En stabilisator är en Hermitesk operator A som lämnar ett givet kvanttillstånd $|\psi\rangle$ oförändrat enligt $A|\psi\rangle = |\psi\rangle$. Det vill säga, $|\psi\rangle$ är en egenvektor till A med egenvärde $+1$ [15, ss. 454–462]. Den generella logiska kvantbiten $|\psi\rangle_L = \alpha|000\rangle + \beta|111\rangle$ stabiliseras exempelvis av operatorerna Z_1Z_2 och Z_2Z_3, vilket kan uttryckas som

$$Z_1Z_2|\psi\rangle_L = \alpha(+1)(+1)|000\rangle + \beta(-1)(-1)|111\rangle = (+1)|\psi\rangle_L,$$

och analogt $Z_2Z_3|\psi\rangle_L = |\psi\rangle_L$. Dessa stabilisatorer mäter paritet mellan kvantbitar utan att avslöja information om amplituderna α och β. Ett utfall på $+1$ innebär att de två kvantbitarna som stabilisatorerna verkar på har samma tillstånd. Det kan ses utifrån att operatorerna Z_i och X_i antikommuterar ($X_iZ_i = -Z_iX_i$), medan X_i och Z_j kommuterar om $i \neq j$. Om exempelvis ett bitflippfel (X) påverkar den första kvantbiten erhålls $X_1|\psi\rangle_L = \alpha|100\rangle + \beta|011\rangle$ och stabilisatormätningen ger

$$Z_1Z_2(X_1|\psi\rangle_L) = \alpha(-1)(+1)|100\rangle + \beta(+1)(-1)|011\rangle = (-1)X_1|\psi\rangle_L$$
$$Z_2Z_3(X_1|\psi\rangle_L) = \alpha(+1)(+1)|100\rangle + \beta(-1)(-1)|011\rangle = (+1)X_1|\psi\rangle_L$$

Det negativa egenvärdet för Z_1Z_2 och positiva för Z_2Z_3 signalerar att ett bitflippfel inträffat på första kvantbit i $|\psi\rangle_L$.

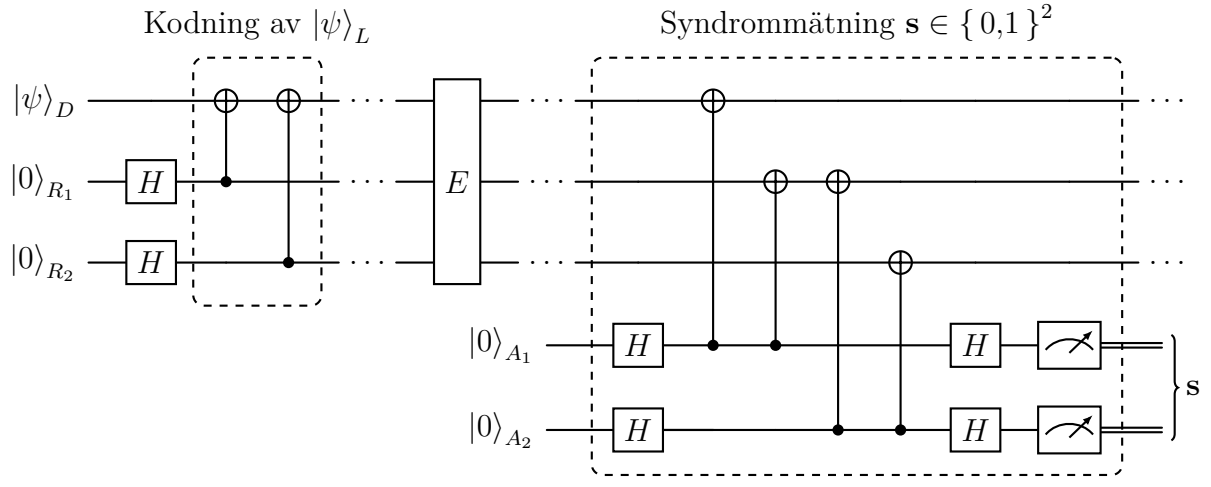
För att mäta utfallet av stabilisatorerna Z_2Z_2 och Z_2Z_3 utan att kollapsa tillståndet direkt, används två *ancillakvantbitar* $|0\rangle_{A_1}, |0\rangle_{A_2}$. Dessa kopplas till de fysiska kvantbitarna genom $CNOT$-grindar och mäts därefter i den vanliga $\{|0\rangle, |1\rangle\}$-basen. Resultatet av dessa mätningar bildar ett *syndrom* $\mathbf{s} \in \{\,0,1\,\}^2$, en binär vektor som indikerar var i systemet ett enstaka fel har inträffat, se tabell 2.1.

2.2.2.1 Repetitionskoden för fasflippfel Z

Trebitars-repetitionskoden kan även användas för att detektera enstaka fasflippfel Z. Den logiska kvantbiten kodas i detta fall i konjugatbasen som $|\psi\rangle_L = \alpha|{+}{+}{+}\rangle + \beta|{-}{-}{-}\rangle$ som stabiliseras av operatorerna X_1X_2 och X_2X_3. Om ett fasflippfel, motsvarande Z-operatorn, uppstår på en av kvantbitarna, förändras den relativa fasen i superpositionstillståndet. Detta påverkar i sin tur utfallet av paritetsmätningen. Till exempel ger ett fasflippfel på den första kvantbiten ($Z_1 |\psi\rangle_L = \alpha\,|{-}{+}{+}\rangle + \beta\,|{+}{-}{-}\rangle$) stabilisatormätningarna

$$X_1X_2(Z_1\,|\psi\rangle_L) = (-1)Z_1\,|\psi\rangle_L\,, \quad X_2X_3(Z_1\,|\psi\rangle_L) = (+1)Z_1\,|\psi\rangle_L\,.$$

För att byta från kodning och mätning z-basen till x-basen kan en modifierad version av repetitionskoden användas, se figur 2.2. Grundidén är att notera likheten $Z = HXH$. Genom att applicera Hadamardgrindar på samtliga kvantbitar kan detektion av fasflipp omvandlas och betraktas som en bitflipp [15, ss. 430–431]. På så sätt kan den vanliga repetitionskoden, som egentligen bara detekterar bitflippfel, återanvändas för att istället detektera fasflippfel.



Figur 2.2: Kvantkrets av en trebitars-repetitionskod för detektion av fasflippfel Z. Inledningsvis kodas en godtycklig datakvantbit $|\psi\rangle_D = \alpha\,|+\rangle + \beta\,|-\rangle$ med redundanskvantbitarna $|+\rangle_{R_1}, |+\rangle_{R_2}$ till en logisk kvantbit $|\psi\rangle_L = \alpha\,|{+}{+}{+}\rangle + \beta\,|{-}{-}{-}\rangle$ i x-basen. Notera att CNOT-grindarna är inverterade jämfört med kodning i z-basen. Därefter utförs stabilisatormätningarna X_1X_2 och X_2X_3 i form av flätade CNOT-grindar på fel-tillståndet $E\,|\psi\rangle_L$. Mätning av stabilisatorerna X_1X_2 och X_2X_3 sker via interaktion med två ancillakvantbitar som initieras och mäts i x-basen med H-grindar. Mätningen av ancillakvantbitarna ger en syndromvektor $\mathbf{s}$ med information om ett eventuellt fasflippfel.

2.2.2.2 Flera kvantfel och degenererade syndrommätningar

Hantering av flera samtidiga fel kräver i regel fler kvantbitar [10]. En repetitionskod som ska korrigera upp till n_f flippade kvantbitar entydigt behöver minst $2n_f+1$ kvantbitar, exklusive ancillakvantbitar, vilket motsvarar ett *kodavstånd* $d = 2n_f+1$. Om antalet fel kan överskrida $(d-1)/2$ blir syndromet *degenererat*, men en *avkodare* kan dock fortfarande lokalisera det mest sannolika felet genom att prediktera tillståndet på första kvantbiten i den logiska biten. Anta att den logiska kvantbiten är kodad i Hadamard-basen som $|\psi\rangle_L = |+++\rangle$, att ett eller två kvantfel har skett samt att stabilisatormätningarna X_1X_2 och X_2X_3 ger syndromet $\begin{bmatrix}1 & 0\end{bmatrix}^T$. Avkodaren konfronteras i detta fall med två kompatibla felmönster, Z_1 eller Z_2Z_3. Genom att använda information om den underliggande felfördelningen (till exempel att enstaka fel är mer sannolika än dubbelfel) predikterar avkodaren vilket tillstånd första kvantbiten skulle ha haft om inget fel inträffat ($|+\rangle$) respektive vid ett enskilt fel på bit 1 ($|-\rangle$). Utifrån denna prediktion appliceras korrektionen: om avkodaren predikterar $|+\rangle$, antas felet vara Z_2Z_3, om avkodaren predikterar $|-\rangle$, antas felet vara Z_1. På så sätt kan avkodaren, utan någon ytterligare mätning, entydigt identifiera och korrigera felet. Detta illustrerar principen i en generisk felavkodningsalgoritm, baserat på syndrom och felstatistik predikteras den mest sannolika felkandidaten och korrigeras.

Tabell 2.1: Syndrommätning **s** från en fasfelsdetekterande trebitars-repetitionskod. Notera att om antalet fel kan vara $n_f > 1$ går det ej att entydigt inferera vilken kvantbit som har drabbats av fasflippsfel.

Antal fel n_f	$n_f = 0$	$n_f = 1$			$n_f = 2$		
Kvantfel E	$I_1I_2I_3$	$Z_1I_2I_3$	$I_1Z_2I_3$	$I_1I_2Z_3$	$Z_1Z_2I_3$	$I_1Z_2Z_3$	$Z_1I_2Z_3$
Syndrom s	$\begin{bmatrix}0\\0\end{bmatrix}$	$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$	$\begin{bmatrix}0\\1\end{bmatrix}$	$\begin{bmatrix}0\\1\end{bmatrix}$	$\begin{bmatrix}1\\0\end{bmatrix}$	$\begin{bmatrix}1\\1\end{bmatrix}$

2.2.2.3 Tröskelsatsen för fasfel i repetitionskoden

Tröskelsatsen inom kvantfelskorrigering säger att om den *fysiska felfrekvensen* p per kvantbit och operation är lägre än ett visst *tröskelvärde* p_{tr}, kan godtyckligt låga *logiska felfrekvenser* ε_d uppnås genom att öka kodavståndet d [13]. Approximativt kan ε_d skrivas proportionellt till p enligt

$$\varepsilon_d \propto \left(\frac{p}{p_{\text{tr}}}\right)^{(d+1)/2}, \tag{2.3}$$

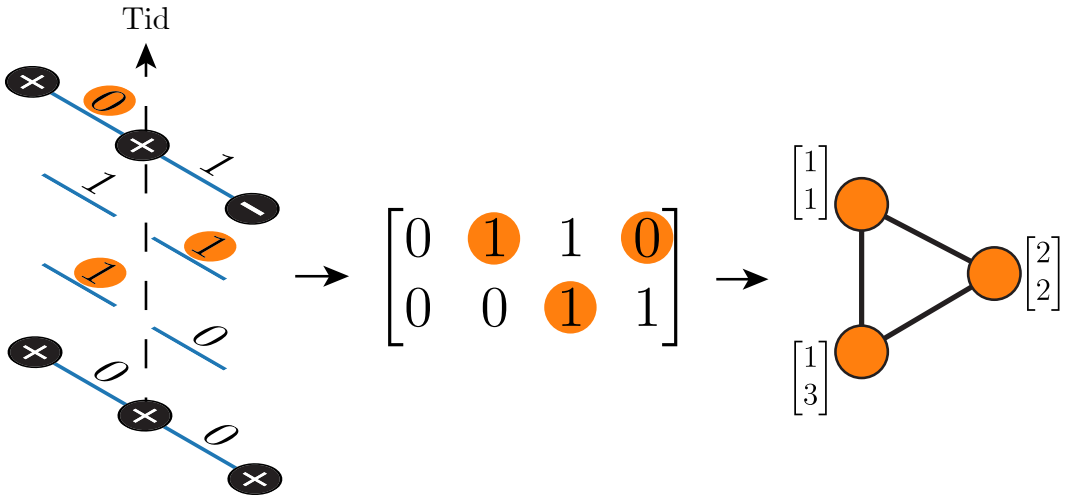
för topologiska ytkoder [13, ekv. 1]. För $p < p_{\text{tr}}$ sjunker alltså ε_d exponentiellt med d, vilket innebär att man teoretiskt sett kan åstadkomma ett godtyckligt lågt logiskt fel genom att öka antalet fysiska kvantbitar per logisk kvantbit. Om däremot $p > p_{\text{tr}}$ växer det logiska felet med ökande kodavstånd, vilket istället gör felkorrigeringen kontraproduktiv. Tröskelvärdet beror på flera faktorer, inklusive vilken kod som används, vilken typ av felmodell som antas samt vilken avkodningsmetod som implementeras [10]. Tröskelsatsen garanterar alltså att felkorrigerande protokoll blir praktiskt användbara endast när de underliggande fysiska operationerna håller en felnivå under en viss tröskel.

2.2.3 Syndrommätningar i rumtid och deras grafrepresentation

Stabilisatormätningar är i praktiken inte perfekta utan är själva föremål för störningar. För att mitigera dessa mätfel är det fördelaktigt att upprepa stabilisatormätningarna under flera tidssteg, kallat *tidsrepetitioner* d_t, för att erhålla en tidsserie av syndrommätningar [12]. Antag att d_t upprepade syndrommätningar utförs på en repetitionskod med kodavstånd d. Vid varje tidssteg $t \in \{1, \ldots, d_t\}$ erhålls en binär syndromvektor $\mathbf{s}^{(t)} \in \{0,1\}^{d-1}$, där varje element $s_i^{(t)}$ för $i \in \{1, \ldots, d-1\}$ motsvarar utfallet av stabilisatormätningen $X_i X_{i+1}$ vid tidssteg t. Eftersom kvantbitarnas initialisering är känd och att mätning av deras verkliga tillstånd kan göras i slutet så kan två *perfekta syndrommätningar* $\mathbf{s}^{(0)}$ och $\mathbf{s}^{(d_t+1)}$ infereras. Mätningarna kan representeras som en binär rumtidsmatris

$$S = \begin{bmatrix} | & | & & | \\ \mathbf{s}^{(0)} & \mathbf{s}^{(1)} & \cdots & \mathbf{s}^{(d_t+1)} \\ | & | & & | \end{bmatrix} \in \{0,1\}^{(d-1)\times(d_t+2)}.$$

Denna tidsupplösta information möjliggör bland annat en differentiering mellan kvantfel och mätfel, där temporärt isolerade syndromavvikelser indikerar sannolikt mätfel, medan konsekventa avvikelser över flera tidssteg tyder på kvantfel. Det gör det även möjligt att spåra felens utveckling över tid, vilket ger information om det underliggande felförloppet.



Figur 2.3: Representation av hur detektionshändelser (orange cirklar) uppstår mellan tidssteg som följd av flera fasflippfel eller mätfel i repetitionskoden. Till vänster ses en repetitionskod med $d = 3$, $d_t = 2$ där syndrommätningarna underst och överst motsvarar perfekta syndrommätningar av initial- och sluttillstånd och de två mittersta motsvarar stabilisatormätningar. Utifrån syndrommätningarna (till vänster) konstrueras en syndrommatris (i mitten) som sedan kan transformeras till en graf med detektionshändelserna (till höger).

Ett tillvägagångssätt att strukturera dessa syndrommätningar i rumtid är att omvandla matrisen S till en graf G [12]. En *detektionshändelse* inträffar då syndromet förändras mellan två på varandra följande tidssteg, se figur 2.3. Detta definieras som att om $s_i^{(t)} \oplus s_i^{(t-1)} = 1$, så uppstår en detektionshändelse som representeras av vektorn $\mathbf{v}_{i,t} := \begin{bmatrix} i & t \end{bmatrix}^T$. Detta innebär att ett fel upptäckts mellan tid $t-1$ och t vid stabilisator i. Därefter definieras en oriktad graf $G = (V, E)$, där varje nod $\mathbf{v}_{i,t} \in V$ representerar en detektionshändelse och varje kant $e = \{\mathbf{v}_{i,t}, \mathbf{v}_{i',t'}\} \in E$ kopplar samman två sådana händelser. För att möjliggöra

avkodning som är oberoende av det absoluta antalet stabilisatorer och tidssteg, normaliseras koordinaterna så att varje detektionshändelse $\mathbf{v}_j = \begin{bmatrix} i_j & t_j \end{bmatrix}^T$ uppfyller $i_j, t_j \in [0,1]$, där $j \in \mathcal{D}$ och $\mathcal{D}$ betecknar mängden av alla detektionshändelser.

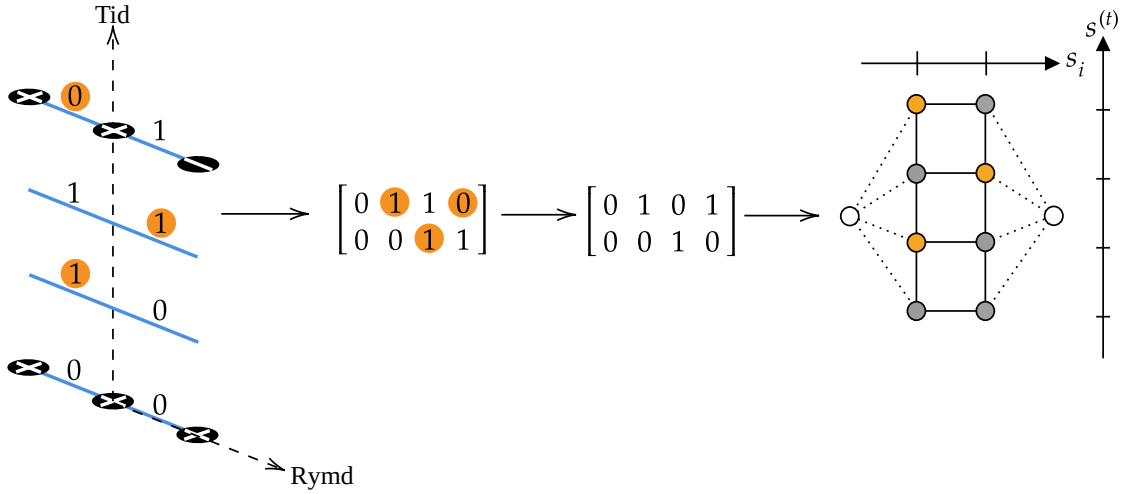
Syftet med att konstruera denna graf är att översätta felets lokalisering till ett grafproblem. Antas felen oberoende och sällsynta, är den mest sannolika felkonfigurationen den som minimerar summan av vikter mellan kopplade detektionspunkter. Eftersom detektorerna (förutom de på randen) kommer i par, leder detta till ett *perfect matching*-problem som löses med hjälp av MWPM. Alternativt kan flera grafer användas som träningsdata till ett grafneuralt nätverk vilket möjliggör inlärning av felmönster.

2.3 Minimum Weight Perfect Matching (MWPM)

Matchningsalgoritmer har historiskt använts som en effektiv metod för att avkoda syndrom från olika kvantfelskorrigeringskoder. En matchningsalgoritm är en metod för att para ihop data, exempelvis noder i en graf, på ett sätt som minimerar den totala felkostnaden. För avkodning av syndrommätningar från repetitionskoden tillämpas matchningsalgoritmen Minimum Weight Perfect Matching (MWPM). Den parar ihop ett givet syndrom på ett sätt som ger det mest sannolika felet. Till skillnad från GNN, där vikter definieras mellan alla par av noder, begränsas vikterna i MWPM-algoritmen till parvis koppling mellan närliggande detektionshändelser i rumtiden. Eftersom risken för bit- och fasfel skiljer sig åt beroende på vilket kvantdatorchip som används är det omöjligt att på förhand känna till sannolikheten för en viss felkedja, vilket begränsar matchningsalgoritmen.

Den mest utbredda implementeringen av MWPM är PyMatching, ett Python-bibliotek som använder algoritmen Blossom V för att lösa matchningsproblemet[11]. PyMatching fungerar som en avkodare genom att använda MWPM för att para ihop detektionshändelser, likt figur 2.4. I MWPM-algoritmen byggs detektorgrafen upp från kanter mellan närmsta grannar i både tid- och rumsdimension, varpå MWPM beräknar sannolikheten för olika fel genom att skapa felförbindelser med vikter mellan dessa noder.

När en enskild kvantbit drabbas av ett Z-fel uppstår en detektionshändelse. Detta mönster kan representeras som en graf G, där varje nod motsvarar en detektionshändelse, och varje kant representerar ett möjligt Z-fel som kopplar ihop två detektionshändelser. Detta bildar grunden för en *matchningsgraf*, se figur 2.4. Till skillnad från grafen som används som input till GNN är denna graf inte fullt ihopkopplad, utan noderna kopplas istället ihop i par av två. För att varje nod alltid ska kunna ingå i ett par introduceras *virtuella noder* som kan kopplas till kantnoderna om de inte har någon närliggande granne att koppla till. Varje fel $E \in \{I, Z\}^{d-1}$, det vill säga varje kombination av antingen felfri eller ett Z-fel på varje kvantbit, motsvarar en *1-kedja*, vilket är en uppsättning av kanter i grafen. Syndromet motsvarar de noder där en fasflipp har skett, alltså detektionshändelserna, och som sedan avläses av MWPM-algoritmen.



Figur 2.4: Figuren visar hur samma exempel som visas i figur 2.3 först översätts till en matris som innehåller samtliga syndrom. Därefter reduceras matrisen till att enbart visa detektorhändelser, varefter MWPM-algoritmen konstruerar den graf som illustreras i figuren. De heldragna kanterna mellan noderna representerar kantvikter. De helvita noderna är de virtuella noderna, alltså noder som kan paras ihop med randnoderna i grafen för att säkerställa att alla noder ingår i ett par.

Om varje kvantbit i har en individuell sannolikhet p_i för att drabbas av ett Z-fel, tilldelas varje kant en vikt enligt

$$w_i = \log\left(\frac{1-p_i}{p_i}\right).$$

Denna vikt motsvarar ett mått på hur sannolikt ett fel är. Ju mindre sannolikt att ett fel uppstår, desto större vikt. Målet med MWPM-avkodning är att hitta den uppsättning kanter, det vill säga det felmönster, som kopplar ihop detektionshändelserna med minsta totala vikt. Detta motsvarar det mest sannolika felet givet syndromet. Sannolikheten för ett visst fel E kan skrivas som

$$p(E) = \prod_i (1-p_i)^{1-g[i]} \cdot p_i^{g[i]} = \prod_i (1-p_i) \cdot \left(\frac{p_i}{1-p_i}\right)^{g[i]},$$

där $g[i] = 1$ om det finns ett Z-fel på kvantbit i, och 0 annars. Detta leder till

$$\log(p(E)) = \sum_i \log(1-p_i) - \sum_i w_i g[i].$$

Mer sannolika fel har alltså lägre total vikt. Eftersom MWPM-algoritmen parar ihop noderna på ett sådant sätt som minimerar vikterna kommer därmed algoritmen uppge det högst sannolika felet. Om sannolikheterna p_i inte är kända, antas i stället en konstant vikt för alla kanter. Detta gör att MWPM-algoritmen optimerar efter det mest sannolika felet under antagandet om att alla kvantbitar har samma felsannolikhet.

2.4 Artificiella neurala nätverk

Ett artificiellt neuralt nätverk är en modell inspirerad av hur den mänskliga hjärnan hanterar och processar information. Nätverket består av neuroner, vilka har en varierande

vikt och position i nätverksstrukturen. Dessa kan med hjälp av sammanlänkningar mellan neuroner kommunicera i nätverket. Sammantaget har det neurala nätverket, likt hjärnan, möjlighet att med simpla men varierande operationer syntetisera och processa komplexa samband.

2.4.1 Grunderna i neurala nätverk

I den mänskliga hjärnan samlar neuroner insignaler som, för att generera en utsignal, måste överstiga en aktiveringströskel. Om inte förblir neuronen inaktiv. Den artificiella neuronen efterliknar detta koncept och har inspirerat uttrycket

$$y = f\left(\sum_{j=1}^{n}(w_j x_j) + b\right), \tag{2.4}$$

där utsignalen, y, bestäms då en aktiveringsfunktion, f, appliceras på kommande insignaler, x_j, med en tillhörande vikt, w_j, från n noder samt en bias, b.

De i arbetet använda aktiveringsfunktionerna är ReLU, $f(x) = \max(0, x)$, samt sigmoid-funktionen, $\sigma(x) = \frac{1}{1+e^{-x}}$. ReLU tillför icke-linjäritet till nätverket och sigmoidfunktionen avbildar en reell insignal bijektivt på intervallet (0,1), vilket gör den väl lämpad för användning i binära klassificeringsproblem.

För att skapa ett neuralt nätverk ordnas neuroner i lager. De flesta artificiella neurala nätverken utgörs av ett inlager, vilket tar in insignalen, ett utlager och flera dolda lager som processar signalen mellan in- och utlager. Med matrismultiplikation kopplas noderna samman och viktas med varandra. Aktiveringen för l dolda lager kan uttryckas som

$$\mathbf{y}^l = f(\mathbf{W}^l f(\mathbf{W}^{l-1}...) + \mathbf{b}^l) = f\left(\mathbf{W}^l \mathbf{y}^{l-1} + \mathbf{b}^l\right), \tag{2.5}$$

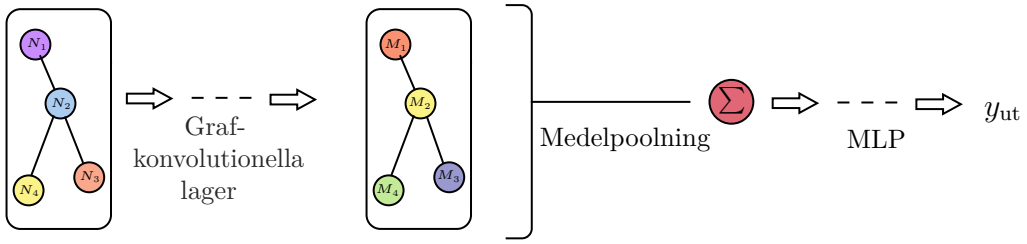
vilket är ett rekursivt beräknat uttryck av aktiveringsfunktioner där utsignalen är en funktion av parametrarna $\mathbf{W}$ och $\mathbf{b}$. $\mathbf{y}^{l-1}$ uttrycker utgången från föregående lager, $\mathbf{W}^l$ syftar till själva viktmatrisen för nuvarande lager, varpå $\mathbf{y}^{l-1}\mathbf{W}^l$ beräknar en ny insignal till nästa lager noder genom matrismultiplikation av föregående lager. Notera särskilt att vektorn $\mathbf{y}^0$ representerar utsignalen från inlagrets aktiveringsfunktion. $\mathbf{b}^l$ är bias-vektorn, f är aktiveringsfunktionen som appliceras elementvis på varje nods utsignal [17].

2.4.2 Grafneurala nätverk (GNN)

Grafneurala nätverk (GNN) är en typ av neurala nätverk som är särskilt anpassade för att hantera data som är strukturerad i form av grafer. I detta arbete ligger fokus främst på GNN med *grafkonvolutionella lager* (eng. Graph Convolutional Layer). GNN försöker lära sig, precis som traditionella neurala nätverk, att reproducera korrekta resultat från givna indata. Genom att propagera egenskaper i graferna kan mönster tas hänsyn till både lokalt och globalt i grafen.

Händelseförloppet där grafen processas av nätverket inleds med att en graf skickas in i nätverket, som propagerar mellan nätverkslager med ekvationen

$$\mathbf{X}^{l+1} = f(\mathbf{W}_1^l \mathbf{X}^l + \mathbf{W}_2^l \mathbf{E}\mathbf{X}^l), \tag{2.6}$$



Figur 2.5: Figuren visar övergripande struktur över hur information propageras i ett grafneuralt nätverk. Nätverket tar in en graf G med detektionshändelser, N_i, som propagerar genom grafkonvolutionella lager. Sedan används medelvärdespoolning där nodvektorerna medelvärdesbildas för att reducera den propagerande informationen till en vektor. Vektorn propagerar vidare genom fullt anslutna lager, i bilden benämnda MLP, vars utlager ger en binär klassifikation y_{ut} som är nätverkets prediktion.

där $\mathbf{X}^l$ är aktiveringsmatrisen för lager l, $\mathbf{E}$ är viktmatrisen för tillhörande graf vilken beskriver relationen mellan noderna enligt ekvation (2.7), $\mathbf{W}_1^l$ och $\mathbf{W}_2^l$ är viktmatriserna för nätverkslager l och f är aktiveringsfunktionen [18]. Sammantaget vägs egenskaperna i nätverkslager samman, både för noden självt med $\mathbf{W}_1^l\mathbf{X}^l$-termen samt från grannar med $\mathbf{W}_2^l\mathbf{E}\mathbf{X}^l$-termen. Genom *medelvärdespoolning* (eng. *global mean pool*), det vill säga att medelvärdesbilda för varje nod i det sista grafkonvolutionella lagret, återförs graferna till en enda vektor som sedan propageras genom ett traditionellt neuralt nätverk med fullt anslutna linjära lager (MLP), se figur 2.5.

2.4.3 Definition av kantvikter mellan noder

För att skapa graferna till det grafneurala nätverket definieras kanterna $e_{ij} \in E$ mellan noder $\mathbf{v}_i$ och $\mathbf{v}_j$ i detta arbetet heuristiskt som möjliga felkedjor (sekvens av Pauli-Z-fel). Viktfunktionen som används är det euklidiska avståndet i rumtid

$$w(e_{ij}) = \min\left((x_i - x_j)^{-2}, (t_i - t_j)^{-2}\right), \tag{2.7}$$

vilket återspeglar antagandet att felkedjor med kortare avstånd är mer sannolika [12].

I det fall där alla grafer är både fullt ihopkopplade och att samtliga kantvikter i graferna är lika, $\mathbf{E} = \mathbf{1} - \mathbf{I}$, kan ekvation (2.6) skrivas om enligt appendix B.1 till

$$\mathbf{X}^{l+1} = f(\left(\mathbf{W}_1^l - \mathbf{W}_2^l\right)\mathbf{X}^l + \mathbf{W}_2\mathbf{1}\mathbf{s}), \tag{2.8}$$

där $\mathbf{s} = \mathbf{1}^T\mathbf{X}^l$. I ekvationen absorberas termen som tar hänsyn till nodens grannar till termen som tar hänsyn till noden självt. Detta innebär att det grafkonvolutionella lagret reduceras till två kompakta matrismultiplikationer $\left(\mathbf{W}_1^l - \mathbf{W}_2^l\right)\mathbf{X}^l$ och $\mathbf{W}_2^l\mathbf{1}\mathbf{s}$, samt en addition. Detta innebär att det kvadratiska tidsberoendet i förhållande till grafens storlek, N^2, från ekvation (2.6) reduceras till N.

2.4.4 Träning av neurala nätverk

Träning av ett neuralt nätverk innebär att vikter iterativt justeras för att minska fel i prediktionen. Detta sker med hjälp av bakåtpropagation samt en optimeringsalgoritm. Ett vidare centralt begrepp som även kommer att användas för att presentera resultaten är *hyperparametrar*, som till exempel kan ange hur mycket vikterna uppdateras under bakåtpropagation. Dessa är parametrar vars begynnelsevärde väljs av användaren före träning och påverkar modellens beteende och prestanda.

För att möjliggöra träning införs en kostnadsfunktion, $C(\boldsymbol{\theta})$, där $\boldsymbol{\theta}$ motsvarar nätverkets modellparametrar och används för att mäta prestandan på modellen som tränas och där vikterna, w_{ij}, uppdateras för att minimera denna funktion. För att straffa osäkra prediktioner från nätverket kan en binär korsentropifunktion användas och beskrivs,

$$C(\hat{x}, \sigma(y_{\text{ut}})) = -[\hat{x} ln(\sigma(y_{\text{ut}})) + (1 - \hat{x}) ln(1 - \sigma(y_{\text{ut}}))],$$

där y_{ut} är nätverksprediktionen, $\hat{x}$ är de sanna etiketterna och σ är en sigmoidfunktion, vilken transformerar nätverksprediktionen till intervallet (0,1) [19].

Under träning uppdateras vikterna $\mathbf{W}^l$ genom bakåtpropagation och optimeras med gradientstegsoptimering. I arbetet används optimeringsfunktionen *Adam* [20]. Parameteruppdateringen från bakåtpropagationen vid tidpunkt t, $\boldsymbol{\theta}_t$, erhålls genom att iterativt beräkna derivatan av kostnadsfunktionen och ger uttrycket

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \nu_t J(\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}_{t-1}), \boldsymbol{\beta}),$$

där ν_t är *inlärningstakt*, vilken bestämmer hur stora steg nätverket tar. $\boldsymbol{\beta}$ är nätverkets övriga hyperparametrar och J är den för optimeringsfunktionen Adam specifika funktionen för gradientstegsuppdatering [20]. Gradientstegsuppdateringen sker sedan iterativt utefter antalet *epoker* och *partistorlek* (eng. *batch size*), det vill säga antalet gånger alla datapunkter används under träning av nätverket samt antalet datapunkter i epoken som används samtidigt för att göra en uppdatering av vikterna.

2.4.5 Att välja hyperparametrar i neurala nätverk

Beroende på vilka hyperparametrar som används kan neurala nätverk erhålla olika egenskaper och inlärningsförmåga. Det är därför viktigt att undersöka vilken kombination av parametrar som ger det bäst presterande nätverket. Nedan presenteras två olika metoder som kan användas för att försöka finna optimala designparametrar.

2.4.5.1 Rutnätsförsök

I ett rutnätsförsök varieras varje parameter, x_i, i alla möjliga kombinationer i ett fördefinierat intervall, uppdelat i L_i olika och ofta jämt fördelade nivåer per parameter. Responsen för försöket beskrivs med en utvärderingsfunktion, $y(\mathbf{x})$, där $\mathbf{x} = \{x_0,...,x_n\}$ och n är antalet parametrar. Optimal parameterkombination väljs sedan sådan att utvärderingsfunktionen maximeras. Antalet försök för modellen skrivs som $\prod_{i=1}^{n} L_i$ och ökar exponentiellt med antalet faktorer och nivåer i försöket och är därför beräkningsmässigt kostsam [21].

2.4.5.2 2^n-faktorförsök

I ett 2^n-faktorförsök antas att varje faktor, x_i, definieras i hög och låg nivå så att $x_i \in \{-1, +1\}$. Responsen, y, beskrivs som

$$y(\mathbf{x}) = \beta_0 + \sum_{i=1}^{n} \beta_i x_i + \sum_{i<j} \beta_{ij} x_i x_j + ... + \beta_1 x_1 x_2 ... x_n, \tag{2.9}$$

där β_i är faktorkoefficienter.

För att bygga upp faktorförsöket används en designmatris $\mathbf{X}$ som byggs upp med en interceptkolumn, det vill säga en $\mathbf{1}$-kolonn, samt kolumner bestående av huvud- samt samverkanstermer x_i,$x_i x_j$,...,$x_1 ... x_n$ där $0 < i \leq j \leq n$. Således uppfyller designmatrisen villkoret $\mathbf{X}^T\mathbf{X} = 2^n\mathbf{I}$. Effekterna, $E_i(\mathbf{x},\mathbf{y})$, för varje faktor kan sedan skattas som

$$E_i(\mathbf{x},\mathbf{y}) = 2\hat{\beta}_i = 2^{-(n-1)} \sum_{k=1}^{2^n} \left(\prod_{l \in S} x_l(k) \right) y(k),$$

där $S \subseteq \{1, \ldots, n\}$ är precis den uppsättning faktorer som ingår i termen, exempelvis $S = \{i,j\}$ för en samverkansterm.

För att pröva signifikansen i de skattade termerna används ett t-test som beräknas för varje skattad $\hat{\beta}_i$ med hjälp av minstakvadratmetoden [21]. I det fall där endast samverkanstermer är statistiskt signifikanta inkluderas även tillhörande huvudeffekter, också kallat *hierarkiprincipen* [22, s. 89].

3

Metod

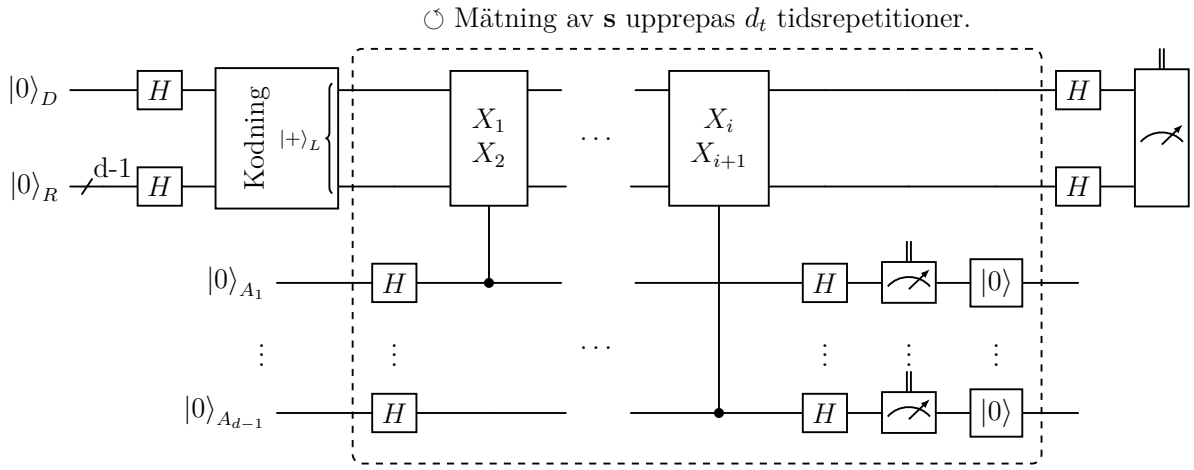
Nedan redogörs för tillvägagångssättet för insamlingen av de experimentella mätningarna och hur det grafneurala nätverket konstruerades och tränades. Inledningsvis presenteras konstruktionen och exekveringen av en modulär repetitionskod på en av IBMs kvantdatorer. Därefter beskrivs olika designval i konstruktionen av GNN-avkodaren och hur den tränades utifrån de experimentella syndrommätningarna på ett beräkningskluster.

3.1 Insamling av experimentella syndrommätningar

Experimentell insamling av syndrommätningar för träning och validering av avkodare gjordes med `Qiskit`, ett öppet källkodsramverk från IBM för programmering av kvantdatorer. `Qiskit` möjliggjorde skapande och exekvering av kvantalgoritmer på både simulerade och fysiska kvantdatorsystem från IBM. Inom detta ramverk konstruerades en modulär repetitionskod baserad på teorin presenterad i avsnitt 2.2.2. Källkoden för implementationen återfinns som `repetition_code.py` i [23].

3.1.1 Konstruktionen av den modulära repetitionskoden

Genom att generalisera kretsen i figur 2.2 till en med valbart kodavstånd d och antal tidsrepetitioner d_t fås en modulär repetitionskod, se figur 3.1. Kretsen inleddes med att d kvantbitar, en datakvantbit $|\psi\rangle_D$ och $d-1$ redundansbitar $|\psi\rangle_R = |\psi\rangle^{\otimes(d-1)}$, initialiserades som $|0\rangle$ och transformerades till x-basen enligt $H|0\rangle = |+\rangle$. Därefter sammanflätades datakvantbiten $|+\rangle_D$ med redundanskvantbitarna $|+\rangle^{(\otimes(d-1)}$ till en logisk kvantbit $|+\rangle_L = |+++\ldots\rangle$. Samtidigt allokerades $d + (d-1)d_t$ klassiska register för lagring av $\mathbf{s}^{(t)}$, $t \in \{\, 1, \ldots, d-1 \,\}$ och den slutgiltiga mätningen av $|\psi\rangle_L$. Om fler än en tidsmässig repetition användes, det vill säga om $d_t > 1$, nollställdes ancillakvantbitarna till $|0\rangle^{\otimes(d-1)}$ efter mätningen, varefter processen med baskonvertering och stabilisatormätning upprepades d_t gånger. Slutligen applicerades H-grindar och mätoperationer på de individuella datakvantbitarna i den logiska kvantbiten. Utifrån kunskapen om initialiseringen och $|\psi\rangle_L$ kunde de två perfekta syndrommätningarna infereras, vilket tillsammans med syndrommätningarna $\mathbf{s}^{(t)}$ skapade S.



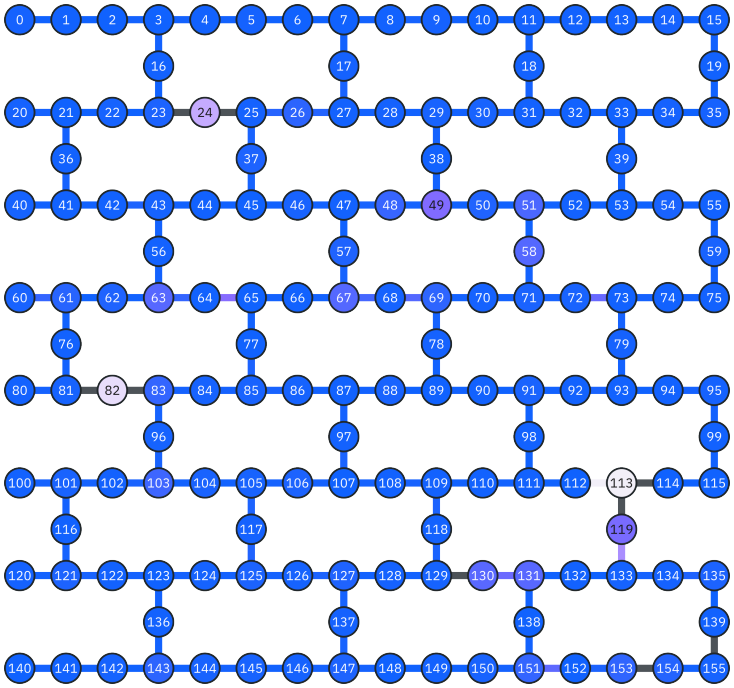
Figur 3.1: Kvantkretsschema över den modulära fasfelsdetekterande repetitionskoden med godtyckligt kodavstånd d och antal tidsrepetitioner d_t. Datakvantbiten $|0\rangle_D$ och redundanskvantbitarna $|0\rangle_R = |0\rangle^{\otimes(d-1)}$ initialiseras som $|0\rangle$, transformeras till x-basen och kodas till $|+\rangle_L$. Syndrommätningen består av transformation av ancillakvantbitarna $|0\rangle_A$ till x-basen, stabilisatormätning med X_iX_{i+1} $i \in \{1, \ldots, d-1\}$, transformation av ancillabitarna tillbaka till z-basen, mätning och sedan nollställning av ancillabitarna till $|\psi\rangle_A = |0\rangle^{\otimes(d-1)}$. Detta upprepas d_t antal gånger och bildar $\mathbf{s}^{(t)}$, $t \in \{1, \ldots, d_t\}$ innan $|\psi\rangle_L$ mäts och bildar den perfekta syndrommätningen $\mathbf{s}^{(d_t+1)}$. Notera att inget fel E är inlagt då syftet med kretsen är att mäta experimentellt uppkomna fasflipp- och mätfel.

3.1.2 Exekvering av repetitionskoden på IBM Quantum

För att experimentellt implementera repetitionskoden beskriven i avsnitt 3.1.1 och figur 3.1 exekverades kvantkretsarna på en *kvantprocessor* (QPU, eng. *Quantum processing unit*) benämnd IBM Marrakesh genom molnplattformen IBM Quantum. IBM Marrakesh valdes på grund av hög tillgänglighet och lågt fysiskt fel vid tvåkvantbitsoperationer. Det är inget entydigt mått på den underliggande fysiska felsannolikheten p men det var en förutsättning för eventuell avkodning under tröskelvärdet p_{tr}.

För att transformera den konstruerade kretsen i figur 3.1 till något som kunde exekveras på den aktuella hårdvaran användes *enhetsmappning* (eng. *layout transpilation*). Detta är en inbyggd metod i `Qiskit` för att optimera kretsens placering på QPU:ns kvantbitar och minska antalet nödvändiga tvåkvantbitsoperationer. För att öka reproducerbarheten av mätningarna användes transpileringsinställningarna `optimization_level=2`, `seed_transpiler=42`. Genom att även specificera `initial layout` kunde den exakta mappningen av kvantbitarna från kretsen till hårdvaran, se figur 3.2, specificeras.

Den konstruerade kretsen exekverades N gånger, även kallat *samplingar* (eng. *shots*), för att säkerställa tillförlitliga mätresultat och samla in tillräcklig statistik över utfallen. Efter exekveringen extraherades och sparades resultatet från en körning i form av en `.json`-fil. Filen användes senare för träning av GNN-avkodaren och i ett senare skede validering av avkodarna. Exekveringarna hade i regel $N = 2 \cdot 10^6$ samplingar för körningar som användes för träning. I de fallen nätverken förtränades med ett tidigare nätverk innehöll exekveringen i regel $N = 4 \cdot 10^5$ samplingar. Utöver detta gjordes flera mindre exekveringar med $N = 2 \cdot 10^4$ för olika kombinationer av parametrarna och på olika kvantindex för att användas som valideringsdata.



Figur 3.2: Kopplingskarta för kvantprocessorn *IBM Marrakesh*, som visar alla fysiska kvantbitar och de respektive tvåkvantbitskopplingar som stöds. Färgskalan för noderna visualiserar avläsningsfelet för varje kvantbit och färgen på kopplingarna representerar felet för CZ-grindar mellan kvantbitar. Mörkare färger indikerar lägre fel och därmed bättre prestanda och grå färg motsvarar ingen tillgänglig data. Notera att detta endast är ett sätt att mäta fidelitet och ej entydigt ger den fysiska felsannolikheten p. Hämtad från [24].

3.1.3 Felbenägenhet hos repetitionskoden

Två mått på kodens felbenägenhet, *logisk felandel* f_L och *fysisk felfrekvens* p definieras för att analysera datan från den experimentellt implementerade repetitionskoden. Logisk felandel definieras som andelen av gånger för ett visst dataset där den logiska kvantbiten genomgått en logisk fasflipp. Logisk fasflipp definieras ha inträffat när majoriteten av de uppmätta bitarna i sluttillståndsmätningen har genomgått en fasflipp. Den fysiska felfrekvensen estimeras genom att anta en förenklad felmodell och betrakta den *första* syndrommätningen, $\mathbf{s}^{(1)}$. Endast första mätningen används för att undvika att behöva modellera för ackumulerade fel. För ett syndrom $\mathbf{s}^{(1)} \in \{0,1\}^{d-1}$ finns två möjliga felmönster som är varandras komplement, en med första bit 0 och en med 1. Detta innebär att sannolikheten för att erhålla syndromet kan skrivas som

$$P(\mathbf{s}^{(1)}) = p^m(p-1)^{d-m} + p^{d-m}(p-1)^m \tag{3.1}$$

där m är antalet bitar som genomgått en fasflipp och erhålls från syndromet. Observera att den här förenklade modellen antar en perfekt syndrommätning utan mätfel, samt att sannolikheten att en bit genomgår en fasflipp är oberoende av de andra bitarna. Genom att räkna hur många gånger varje syndrom mäts och tillämpa modellen kan en likelihood-funktion beroende av p konstrueras för datasetet. Därefter erhålls den estimerade fysiska felfrekvensen p_{est} genom att numeriskt maximera logaritmen av sannolikhetsfördelningen med avseende på p.

3.2 Träning av grafneurala nätverk

Logisk noggrannhet α_L, vilket beskriver andelen korrekt klassificerade grafer, definieras för att utvärdera prestandan hos både grafneurala nätverk och MWPM. Låt N beteckna det totala antalet grafer från en körning och k antalet av avkodaren korrekt klassificerade grafer. Av de N graferna kommer ett antal n att vara triviala, det vill säga utan detektionshändelser. Dessa avkodas inte utan klassificeras direkt som korrekta. Således definieras logisk noggrannhet som

$$\alpha_L = \frac{k+n}{N} \in [0,1]$$

och den *logiska felsannolikheten* som $p_L = 1 - \alpha_L$. För att kvantifiera osäkerheten i skattningen kan varje körning ses som N stycken oberoende Bernoulli-försök. Standardfelet i α_L ges då av $\mathrm{SE}(\alpha_L) = \sqrt{\frac{\alpha_L(1-\alpha_L)}{N}}$ och analogt för p_L.

De grafneurala nätverken tränades på beräkningsklustret Alvis med CUDA-accelererade beräkningar på Nvidias A40-GPU:er. Inför träningen delades syndromen upp i två delar: en träningsdel motsvarande 90% av syndromen och en valideringsdel motsvarande 10% av syndromen. Träningsdelen används sedan för att träna nätverket, medan valideringsdelen används för att löpande utvärdera den logiska noggrannheten på syndrom som nätverket inte tränats på.

För samarbete och versionshantering av kodbasen användes Chalmers GitLab, tillsammans med tjänsten *WANDB* som sammanställer träningshistorik på ett lättöverskådligt sätt via en webbsida. Den använda kodbasen finns tillgänglig via [23] och inkluderar bland annat funktioner som sparar de tränade grafneurala nätverken, vilket medför att nätverken vid senare tillfällen kan testas på olika dataset eller användas som förtränade nätverk vid vidare träning.

3.2.1 Optimering av hyperparametrar

Ofta krävs det ett omfattande antal försök där varje parameterkombination undersöks och utvärderas individuellt för att undersöka vilka parameterkombinationer som ger bäst presterande nätverk. Då träning av neurala nätverk är både tids- och kostnadsmässigt krävande krävs därför en noggrant utvald försöksplanering för att maximera informationen för varje vald parameterkombination. Trots försöksplaneringen begränsades projektet av den tillgängliga beräkningskapaciteten och därmed optimerades endast parametrar för avkodning av repetitionskoder med kodavstånd $d = 5$ och antal tidsrepetitioner $d_t = 5$.

Hyperparameteroptimeringen inleds med ett 2^3-faktorförsök där partistorlek, inlärningstakt och nätverksstorlek varieras mellan en fördefinierad hög och låg nivå, vilka presenteras i appendix A.1 där nivåerna valdes något större samt längre än de hyperparametrar presenterade i tidigare arbeten [12]. Varje parameterkombination tränades vardera tre gånger under 300 epoker på ett dataset bestående av $N = 2000000$ datapunkter. För att utvärdera nätverken isolerades 10% av datapunkterna för att användas som valideringsdata. De tränade nätverkens logiska noggrannhet på valideringsdatan redovisas i appendix A.1. Resultatet av 2^3-faktorförsöket indikerade att samspelseffekten av nätverkets storlek och inlärningstakt var negativt proportionell mot prestandan, det vill säga att prestandan ökar om nätverksstorlek och inlärningstakt är på olika nivå. Koefficienterna för effekterna visas

i tabell 3.1. Utöver detta kunde inga signifikanta slutsatser kring partistorlekens påverkan dras. Den slutgiltiga faktormodellen blev således

$$\hat{y}(A,B) = \beta_0 + \beta_A A + \beta_B B + \beta_{AB} AB \tag{3.2}$$

där $\hat{y}(A,B)$ är den predikterade modellresponsen och $\beta_0, \beta_A, \beta_B$ samt β_{AB} är faktorkoefficienter. Notera särskilt att huvudeffekterna, A och B, tas med på grund av hierarkiprincipen [22].

Tabell 3.1: Faktorkoefficienter och p-värden för 2^3-faktorförsöket. De fetmarkerade koefficienterna är både de statistiskt signifikanta faktorkoefficienter med tillhörande huvudeffekter. De modellkonfigurationer, deras hyperparametrar samt slutgiltiga logiska noggrannhet som användes för att genomföra 2^3-faktorförsöket finns redovisade i appendix A.1.

Term	**Faktorkoefficient** ($\beta_i = \frac{E_i}{2}$)	p**-värde**
Nätverksarkitektur (A)	**−0,0087**	**0,895**
Inlärningstakt (B)	**0,0121**	**0,853**
Partistorlek (C)	−0,0051	0,938
A × B	**−0,1390**	**0,043**
A × C	0,1034	0,125
B × C	0,0916	0,172
A × B × C	−0,0157	0,811

Ovan kvalitativa samband indikerar att en samverkan av inlärningstakt och storleken av nätverksarkitekturen har högst påverkan på prestandan. Med hjälp av faktorkoefficienterna i tabell 3.1 samt prediktionsmodellen $\hat{y}(A,B)$ (ekv. 3.2) fås att parameterkombinationen med högst predikterad noggrannhet är $\hat{y}(A = -,B = +)$, det vill säga ett litet nätverk med hög inlärningstakt. För den valda aktiveringsfunktionen rekommenderas en inlärningstakt på 0,001 [20] vilket tillsammans med resultatet från 2^3-faktorförsöket indikerar att ett välfungerande nätverk bör ha hög inlärningstakt och liten nätverksstorlek.

Slutligen användes ett rutnätsförsök i området med hög inlärningstakt och liten nätverksstorlek för att heuristiskt finna en högpresterande parameterkombination. Under försöket tränades nätverken på samma datapunkter som i 2^3-faktorförsöket under 300 epoker. Då 2^3-faktorförsöket misslyckades med att förkasta nollhypotesen att partistorleken hade signifikant påverkan på träffsäkerheten användes den högre partistorleken på 1024 då en större partistorlek medför att träningen av nätverken går snabbare. De använda försökskonfigurationerna samt deras noggrannhet på valideringsdatan redovisas i appendix A.2. Den parameterkombination som hade högst α_L på träningsdatan utan att uppvisa tecken på överanpassning i valideringsdatan redovisas i 3.2. Den tränades i ytterligare 300 epoker. Vid ytterligare träning efter 600 epoker visades ingen förbättrad prestanda. I figur 4.1b i resultatavsnittet redovisas träningsförloppet för det slutgiltiga nätverket. På grund av begränsade beräkningsresurser och projektets omfattning användes de från $d = 5, t = 5$-försöket optimala parametrarna även till andra kodavstånd d och antal tidsrepetitioner d_t.

3.2.2 Träning av slutgiltiga grafneurala nätverk

För att undersöka de grafneurala nätverkens prestanda och egenskaper vid olika kodavstånd och antal tidsrepetitioner tränades ytterligare 13 nätverk. Dessa nätverk kan delas upp i två

serier, en serie där kodavståndet är fixerat till $d = 3$ och en serie där antal tidsrepetitioner är fixerat till $d_t = 3$. I tabell 3.3 redovisas de nätverk som har tränats. Varje nätverk tränades under 600 epoker med 400000 syndrom (shots) varav 10% användes som valideringsdata. De hyperparametrar som togs fram i avsnitt 3.2.1 användes vid träningen av samtliga nätverk, men eftersom optimeringen inte gav några slutsatser kring vilken partistorlek som gav bäst resultat användes i arbetet först en partistorlek på 1024 på samma sätt som i föregående avsnitt. Under träningens gång upptäcktes det att en partistorlek på 256 presterade bättre för nätverken med $d = 3$, $d_t \in \{3, 5, 7, 9, 11, 13\}$ och $d \in \{5, 7, 9, 11\}$, $d_t = 3$. En partistorlek på 256 gav dock instabila nätverk som hade stor sannolikhet att klassificera alla syndrom som samma klass för $d \in \{13, 15\}$, $d_t = 3$ och därför användes en partistorlek på 2048 då det var den lägsta partistorlek där nätverken var stabila.

För att underlätta träningen och minska det totala antalet epoker som krävs för att träna varje nätverk användes sekventiell överföringsinlärning, det vill säga nätverken tränades i stegvis ökande svårighetsgrad. I slutändan tränades dock alla nätverk 600 epoker oavsett om deras noggrannhet på valideringsdatan slutat öka vid ett lägre antal epoker för att hålla träningen så konsistent som möjligt. Vikterna från det färdigtränade nätverket för $d = 5$, $d_t = 5$ i avsnitt 3.2.1 användes för att träna nätverken där $d \leq 7$ eller $d_t \leq 7$. Återstående nätverk initialiserades sedan med vikterna från nätverket med en nivå lägre d eller d_t, till exempel tränades nätverket med $d_t = 3$, $d = 9$ genom att använda vikterna från $d_t = 3$, $d_t = 7$. Avslutningsvis tränades ytterligare tre grafnätverk där grafernas samtliga kantvikter $\forall(i,j) \in E : e_{ij} = 1$. Träningarna utfördes för $d_t = 3$, $d \in \{7, 11, 15\}$ i 150 epoker vardera.

Tabell 3.2: Slutgiltig nätverksstruktur. d_{in} och d_{ut} är respektive ingångs- och utgångsdimensioner för varje lager.

Lager	d_{in}	d_{ut}
GraphConv_1	2	64
GraphConv_2	64	128
Fullt anslutet_1	128	64
Fullt anslutet_2	64	4
Fullt anslutet_3	4	1
Inlärningstakt	0,001	
Partistorlek	1024	

Tabell 3.3: De kombinationer av kodavstånd d och antal tidsrepetitioner d_t för vilka grafneurala nätverk har tränats utöver $d = 5$, $d_t = 5$.

Fixerat kodavstånd		**Fixerat # tidsrepetitioner**	
d	$\mathbf{d_t}$	**d**	$\mathbf{d_t}$
3	3	3	3
3	5	5	3
3	7	7	3
3	9	9	3
3	11	11	3
3	13	13	3
		15	3

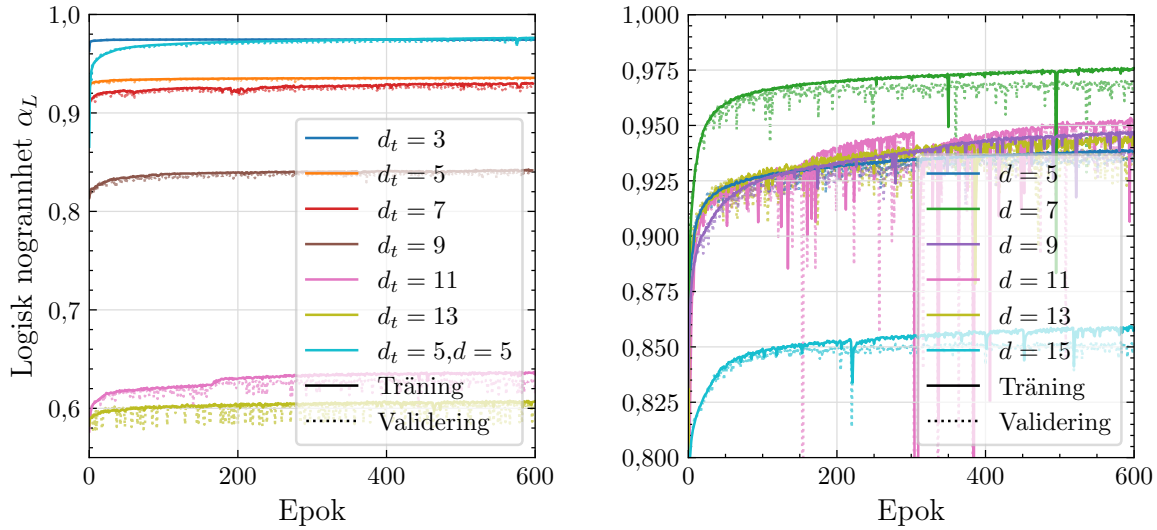
4

Resultat

Här redovisas resultaten från träning av GNN-avkodaren samt dess förmåga att avkoda syndrommätningarna för olika kombinationer av kodavstånd och antal tidsrepetitioner i jämförelse med MWPM. Jämförelserna delas upp med avseende på optimal logisk noggrannhet, generaliserbarhet och tidskomplexitet för de olika avkodningsmetoderna. Slutligen presenteras även intrinsiska egenskaper hos de experimentellt uppmätta syndrommätningarna och en estimering av den logiska felfrekvensen för de använda kvantbitarna.

4.1 Träning av de grafneurala nätverken

I figur 4.1 presenteras den logiska noggrannheten α_L under träningsförloppet av de grafneurala nätverken. Nätverken tränades under 600 epoker med i regel $N = 400000$ syndrom varav 10% användes för validering. Partistorleken var 256 för samtliga nätverk förutom de med kodavstånden $d \in \{13, 15\}$ där partistorleken var 2048. Notera att noggrannheten inte är mer än en procentenhet lägre för valideringsdatan jämfört med träningsdatan för samtliga nätverk förutom de med $d_t \in \{11, 13\}$. Träningsförloppet för nätverken som tränades med grafer med vikter $\forall(i,j) \in E : e_{ij} = 1$ redovisas i appendix B.2. Dessa nätverk presterade lika bra som motsvarande nätverk som tränades med varierande kantvikter.



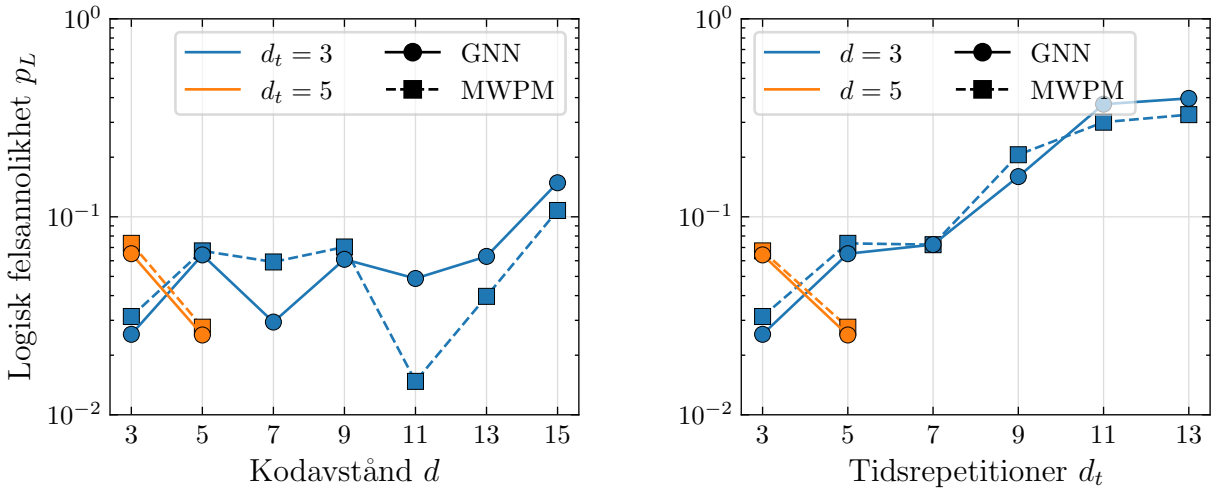
(a) Fixerat $d = 3$ samt $d = d_t = 5$.

(b) Fixerat $d_t = 3$

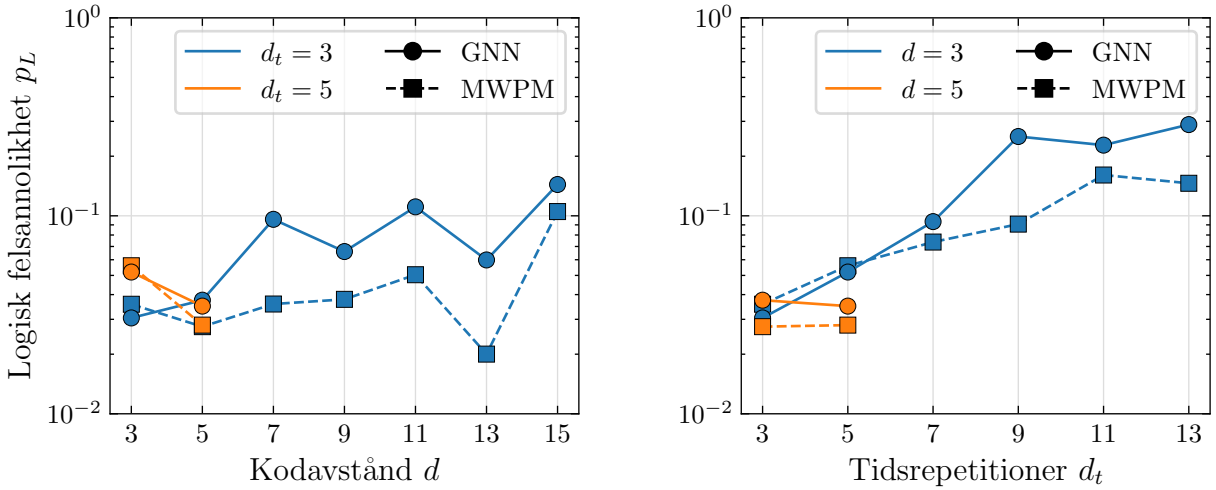
Figur 4.1: Tränings- och valideringsnoggrannhet per träningsepok för de grafneurala nätverken. I **(a)** redovisas de nätverk med kodavstånd $d = 3$ uppdelat på antal tidsrepetitioner d_t och samt nätverket tränat för $d = d_t = 5$. I **(b)** redovisas de nätverk med antal tidsrepetitioner $d_t = 3$ uppdelat på kodavstånd d.

4.2 Avkodningsnoggrannhet för tränade nätverk

I figur 4.2 illustreras hur olika grafneurala nätverk, tränade på varierande kombinationer av d och d_t, presterar i jämförelse med MWPM. I figur 4.2 (a) visas resultaten för valideringsdata som utgörs av en 10/90-fördelning av samma exekvering som användes för träningen, inklusive triviala syndrom. Eftersom validerings- och träningsdata härrör från samma uppsättning fysiska kvantbitar, representerar detta scenario de mest gynnsamma förutsättningarna för GNN:erna. I figur 4.2 (b) utvärderas istället avkodarnas prestanda på nya syndrommätningar som genererats vid ett senare tillfälle och på andra fysiska kvantbitar. Överlag presterar GNN:er marginellt bättre eller liknande jämfört med MWPM under optimala förutsättningar för låga d och d_t. Dock uppvisar GNN konsekvent högre p_L när de testas på nya syndrommätningar, förutom för $(d,d_t) = (3,3)$, $(3,3)$, $(3,5)$.



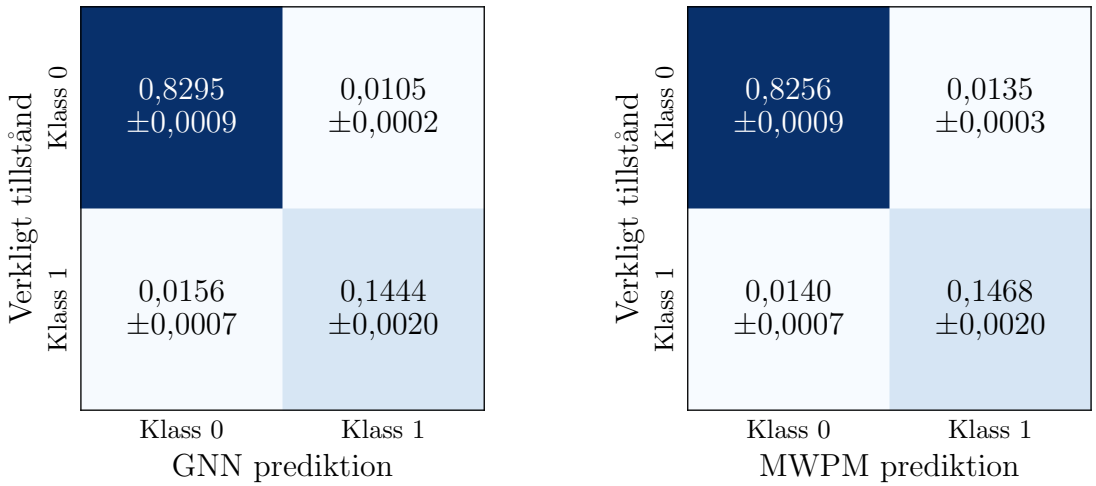
(a) Jämförelse med avseende på valideringsdata från träningen



(b) Jämförelse med avseende på nya syndrommätningar från andra fysiska kvantbitar.

Figur 4.2: Logisk felsannolikhet p_L för GNN och MWPM som funktion av kodavstånd d (vänster) respektive antal tidsrepetitioner d_t (höger), uppdelat på d_t respektive d för repetitionskoden. Standardfelet är mindre än storleken på markörerna. I **(a)** utvärderas båda avkodarna på valideringsdata från samma exekvering som användes vid träningen (typiskt $N = 40000$). I **(b)** utvärderas de på nya syndrommätningar från andra fysiska kvantbitar, insamlade vid ett senare tillfälle ($N = 20000$).

I figur 4.3 visas fördelningen av α_L för nätverket med $d = d_t = 5$ i form av en förvirringsmatris (eng. confusion matrix) i jämförelse med MWPM. Denna visualisering ger en bild av hur klassificeringsfelen fördelar sig med avseende på det verkliga logiska tillståndet (typ 1- och typ 2-fel) .



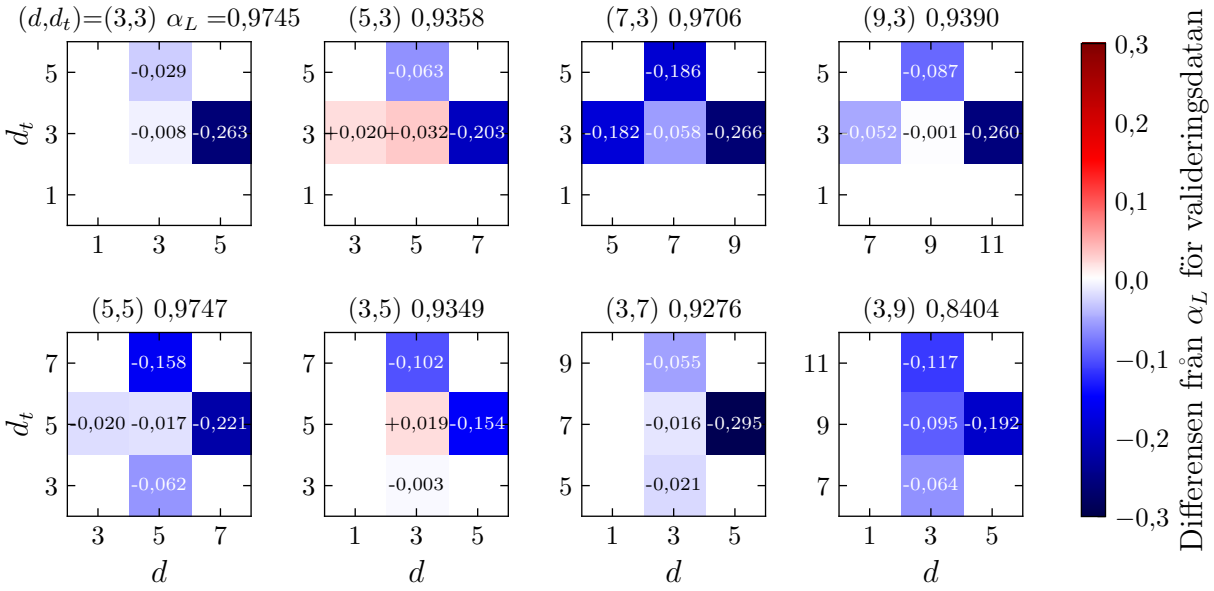
Figur 4.3: Förvirringsmatriser som visar α_L vid avkodning av syndrom med $d = d_t = 5$ för GNN och MWPM. Avkodarna utvärderas på valideringsdata från samma exekvering som användes för träning ($N = 200000$), inklusive triviala syndrom.

4.3 Generaliserbarhet för olika (d, d_t)

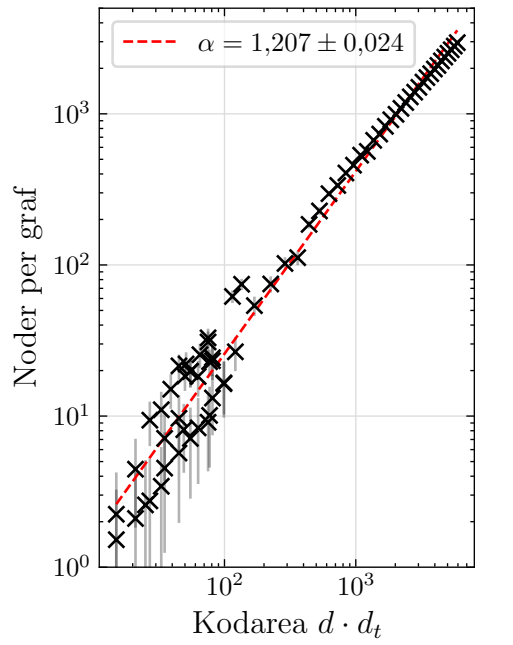
En begränsning med GNN är att nätverket behöver tränas för en specifik kombination av (d, d_t) innan det kan avkoda syndrommätningarna. Dock är det möjligt att avkoda syndrommätningar med andra (d, d_t) än vad det är tränat för eftersom vi normaliserar positions- och tidskoordinaterna i graferna. I figur 4.4 redovisas GNN:s förmåga att generalisera till $(\pm 2d, \pm\, 2d_t)$. De olika rutorna motsvarar ett specifikt nätverk och i deras celler visas differensen mellan α_L för GNN utvärderat på valideringsdatan jämfört med helt nya syndrommätningar med $(\pm 2d, \pm\, 2d_t)$. I centerrutorna redovisas differensen för samma (d, d_t) vilket motsvarar differensen mellan datapunkterna i figur 4.2 (a) och figur 4.2 (b).

4.4 Graf- och tidskomplexitet för kodarean

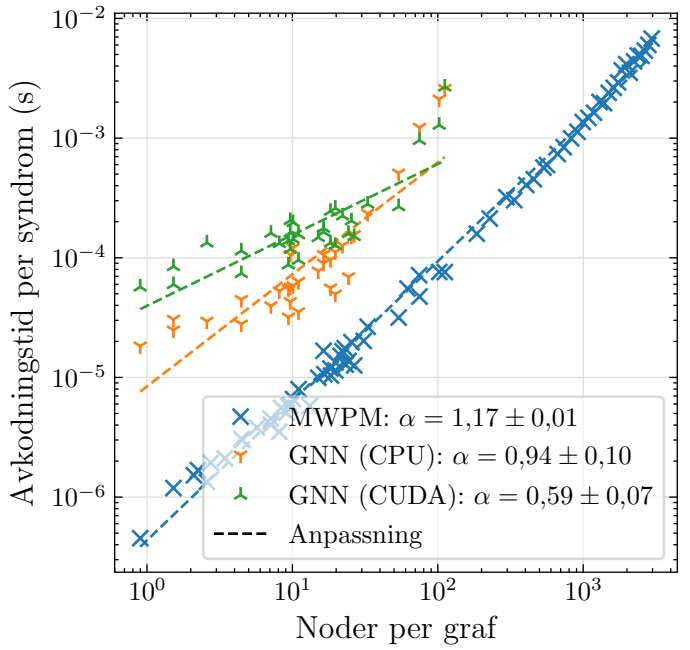
Eftersom kvantfelskorrigering i praktiken kommer att kräva stora d och d_t är det relevant att analysera hur både grafstorleken och avkodningstiden skalar med *kodarean* dd_t. I figur 4.5 visas hur antalet noder per graf, vilket är identiskt för MWPM och GNN, skalar som funktion av dd_t. Varje datapunkt motsvarar en exekvering med $N = 20000$ grafer. I figur 4.6 presenteras en jämförelse av avkodningstiden per syndrom som funktion av dd_t för GNN och MWPM. Med avkodningstid avses den tid det tar att avkoda grafen, exklusive tiden för att konstruera och läsa in grafen i avkodaren. Mätningarna för GNN (CPU) och MWPM utfördes på en Intel Core i7-4790K vid 4,00 GHz, medan mätningarna för GNN (CUDA) genomfördes på ett Nvidia GTX 970, alla med partistorlek 512. Grafer med kodareor större än $dd_t > 441$ kunde inte avkodas av GNN, eftersom den använda implementationen genererade grafrepresentationer vars minneskrav överskred tillgänglig hårdvarukapacitet.



Figur 4.4: Översikt av olika GNNs förmåga att generalisera avkodningen till andra kombinationer av (d, d_t) än vad de är tränade för. I cellerna i varje ruta skrivs differensen ut för α_L för valideringsdatan relativt α_L för nya syndrommätningen. Alla syndrommätningar, inklusive centerrutan, är genererade på andra kvantbitar än valideringsdatan.



Figur 4.5: Skalning av antalet noder per graf, motsvarande antalet detektionshändelser från syndrommätningarna, som funktion av kodarean dd_t. Storlekssambandet skattas enligt relationen $N_{\text{noder}} \propto (dd_t)^{\alpha}$. Standardavvikelsen representeras av grå felstaplar.

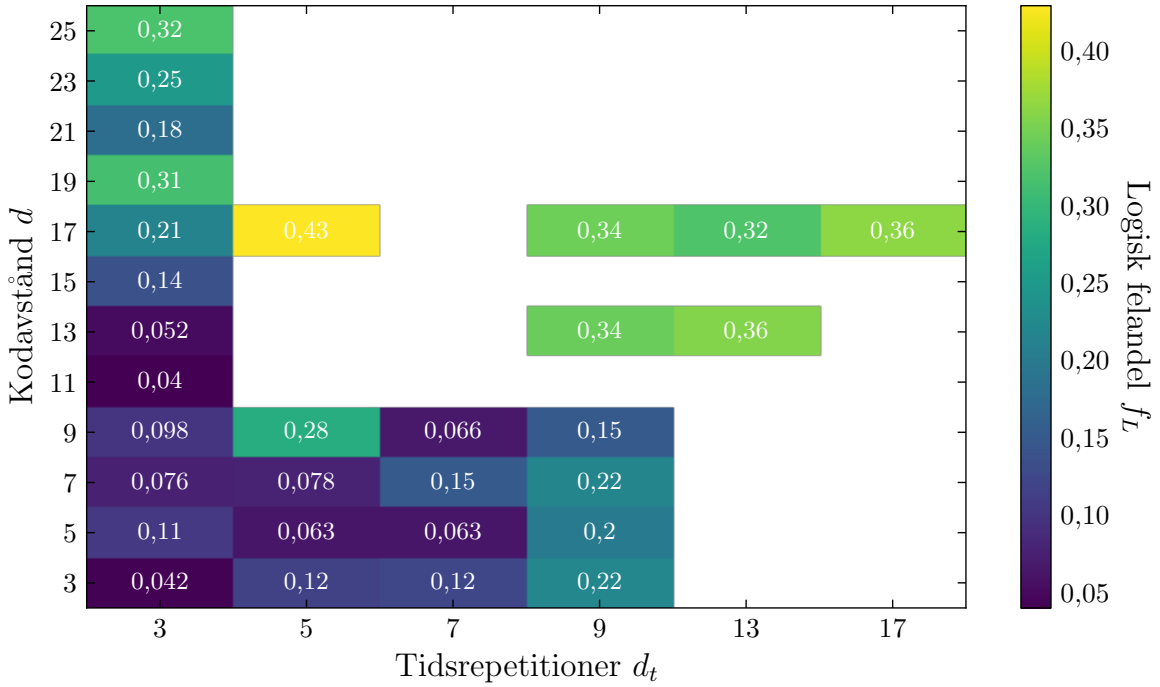


Figur 4.6: Skalning av avkodningstid per syndrom som funktion av antalet noder per graf N_{noder} för både grafneurala nätverk (GNN) och MWPM implementerat med `PyMatching 2` [11]. Tidskomplexiteten skattas enligt sambandet $T_{\text{avkoding}} \propto (N_{\text{det}})^{\alpha}$. Varje datapunkt representerar en körning för en specifik kombination av kodavstånd d och antal tidsrepetitioner d_t.

4.5 Egenskaper hos experimentell syndrommätning

I följande delavsnitt analyseras datan från de experimentella mätningarna från IBM Marrakesh. Detta för att ge en inblick i hur uppmätt logisk felandel f_L (avsnitt 3.1.3) och koherens beror på d och d_t samt på mappningen till de fysiska bitarna på kvantprocessorn. Även den fysiska felfrekvensen p estimeras med en förenklad felmodell som presenteras i avsnitt 3.1.3.

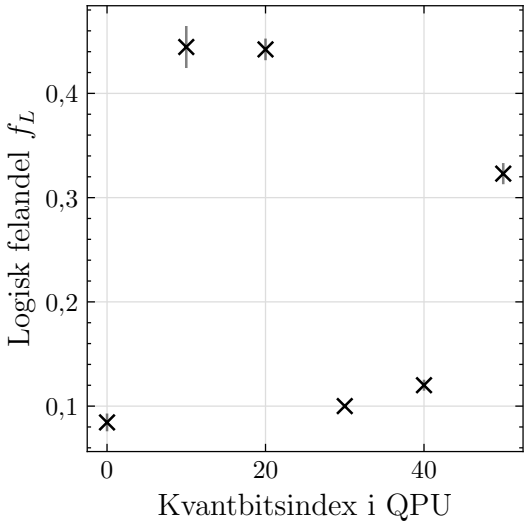
I figur 4.7 presenteras den logiska felandelen f_L, det vill säga andelen gånger en logisk fasflipp har skett vid mätningen av databitarna, för olika d och d_t. Datasetet som analyserats är samma dataset som använts för träning av GNN och testning av såväl GNN- som MWPM-avkodaren. Avsaknaden av vissa kombinationer av d och d_t beror på att de inte behövts i samband med träning och därmed aldrig körts.



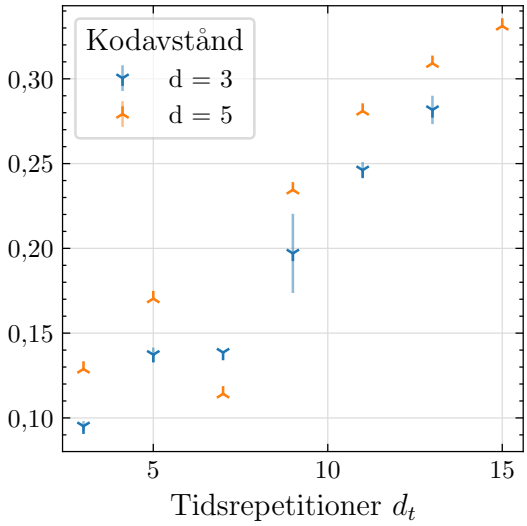
Figur 4.7: Värmekarta över hur uppmätt logisk felandel f_L beror på kodavstånd d och antal tidsrepetitioner d_t. Datasetet som används för att beräkna felfrekvensen är samma dataset som används för träning och testning av avkodarna och är med varierande antal separata körningar och shots.

Som vi ser ökar f_L för högre d_t, vilket är väntat i och med att databitarna genomgår fler syndrommätningar, och således fler kvantgrindar, innan de mäts. Mindre väntat är att f_L också ser ut att öka med d. Teoretiskt sett borde högre d innebära mer redundans för den logiska kvantbiten och således lägre f_L.

För att undersöka om mappningen till fysiska bitar på processorn har en inverkan på f_L gjordes flera mätningar med samma d och d_t, men med olika kvantbitsindex (se avsnitt 3.1.2). Resultaten från detta presenteras i figur 4.8. Som vi ser får vi för samma kvantkrets mycket olika resultat på f_L och felet ser således ut att bero mycket på vilka kvantbitar kretsen placeras på. I figur 4.9 ser vi hur f_L beror på d_t med $d = 3$ och $d = 5$ när kvantbitsindex fixeras. Förutom för $d_t = 7$ ser vi här en tydlig ökning av f_L med d_t.

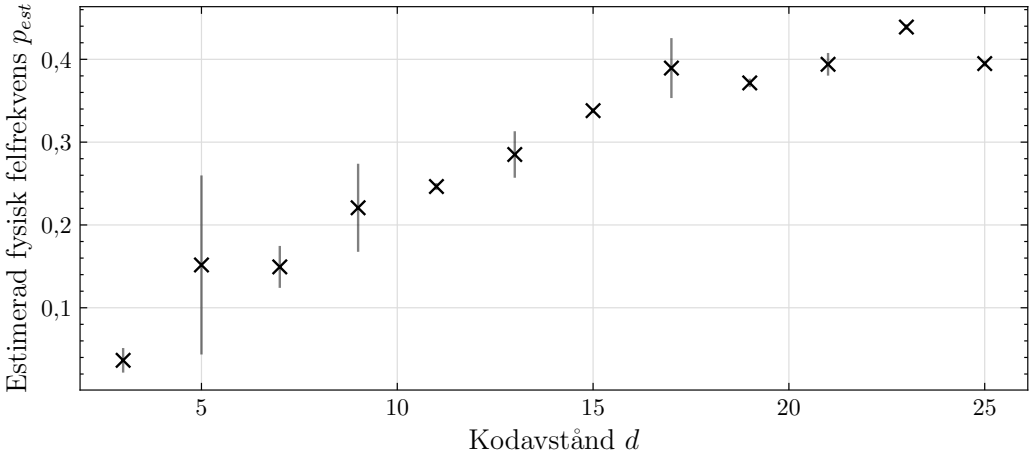


Figur 4.8: Variation av logisk felandel f_L beroende på mappning till fysiska qubits i QPU:n. Datan består av 5 separata körningar med $N = 10000$ shots för varje värde på kvantbitsindex. Samtliga mätningar är med $d = d_t = 5$.



Figur 4.9: Logisk felandel f_L mot tidrepetitioner d_t för data från körningar med kodavstånd $d = 3$ och 5, samt med samma kvantbitindex på QPU:n. Felstaplar representerar standardavvikelsen hos datan.

Hittills har endast den logiska felandelen f_L undersökts för de experimentella mätningarna. En annan aspekt som är intressant är den fysiska felfrekvensen p för kvantbitarna i sig. Estimering av p, p_{est}, gjordes för hela träningsdatasetet med metoden som presenteras i avsnitt 3.1.3 och redovisas i figur 4.10. Observera att felmodellen som används är en förenkling av verkligheten i det att syndrommätningarna antas vara "perfekta", det vill säga att eventuella mätfel ej tas hänsyn till. Detta innebär att resultatet bör tolkas med viss försiktighet.



Figur 4.10: Estimerad fysisk felfrekvens p_{est} för olika kodavstånd d. Datan som används är samma dataset som i figur 4.7, men där endast första syndromet har använts för estimering av p. Felstaplar representerar standardavvikelsen.

5

Diskussion

I kommande del diskuteras kring hur väl det grafneurala nätverket presterade i relation till MWPM. Avsnittet delas upp i diskussion för eventuell felanalys av de experimentella syndrommätningarna, hur felen skalar med kodavstånd samt antal tidsrepetitioner. Vidare redogörs för träningen av det grafneurala nätverket samt jämförelse av GNN och MWPM. Slutligen redogörs för etiska överväganden.

5.1 Träning av grafneurala nätverk

Trots att informationen från hyperparameteroptimeringen optimerats med hjälp av försöksplanering begränsas projektet av de tillgängliga beräkningsresurserna. Av denna anledning är informationen från 2^3-faktorförsöket intrinsiskt begränsad. Slutsatserna dragna från försöket är därför tagna med förbehåll till att eventuella lokala maximum missas. Dessutom bygger 2^3-faktorförsöket på antagandet att parametrarna påverkar nätverkens prestanda linjärt, vilket eventuellt enbart går att motivera för lokala områden. Dock är parametrarnas påverkan generellt sett okända, vilket medför att resultat från en linjär approximation bör beaktas med försiktighet bortom de parameternivåer som 2^3-faktorförsöket genomfördes på. Att nätverken tränades i 300 epoker kan ha gynnat de nätverk med högre inlärningstakt. Detta är dock inte nödvändigtvis negativt då snabb och effektiv inlärning är önskvärda egenskaper för nätverken.

Resultaten från 2^3-försöket kan ses som ointuitivt, det vill säga med hög inlärningstakt och liten nätverksstorlek. Det bör noteras att syndromavkodningen, särskilt vid lägre d och d_t, är av relativt låg komplexitet, vilket skulle kunna gynna enklare nätverksarkitekturer. Dessutom har tidigare arbetens GNN-avkodare [12] också initialt använt inlärningstakten 0,001 vilket talar för det valda värdet.

Då ingen utförlig jämförelse mellan överföringstränade nätverk och nya nätverk genomfördes är det svårt att dra slutsatser kring metodens påverkan på resultatet. Det var dock mycket svårt att undvika att nya ej överföringstränade nätverk endast klassificerade alla syndrom till samma klass. Under arbetets tidiga fas utfördes dock tester på $(d, d_t) = (13, 3)$ där överföringstränade nätverk i regel presterade bättre med minst en procentenhet i termer av logisk noggrannhet på valideringsdatan.

I figur 4.1 kan det observeras att den logiska felsannolikheten inte minskar med ökande kodavstånd d, samt att nätverket med $d = 15$ presterade sämst. Dessutom lyckades inga nätverk tränas för kodavstånd $d < 15$ utan att nätverken klassificerade alla syndrom som den mest förekommande klassen. Detta kan exempelvis bero på att den fysiska felfrekvensen är större än tröskelvärdet, $p > p_{\text{tr}}$. En annan möjlig anledning är att den framtagna modellstrukturen och hyperparametrarna för $(d, d_t) = (5, 5)$ inte är överförbara

till större kodavstånd. Vid större kodareor ökar även antalet detektionshändelser, se figur 4.5, och felet propagerar över fler kvantbitar samt ett större antal tidssteg. Syndromen blir därmed mer komplexa och kräver potentiellt sett ett större och djupare neuralt nätverk, vilket talar för att den optimala modellstorleken för $(d, d_t) = (5, 5)$ inte generaliserar till större kodareor. Detta betyder inte att en optimering vid högre kodavstånd hade givit bättre resultat, utan begränsningar i både avkodaren och informationen i träningsdatan kan vara avgörande. Avslutningsvis är det möjligt att GNN-avkodarens prestanda begränsas av den använda mängden träningsdata. Avkodaren hade troligtvis gynnats av en större träningsdatamängd.

Under träningen observerades det att en liten partistorlek tenderade att ge bättre resultat för små kodareor, medan en större partistorlek presterade bättre för större kodareor. En möjlig förklaring till detta är att längden av syndromen ökar för större kodareor (se figur 4.5), vilket i sin tur medför fler möjliga detektionshändelser. Syndromen blir således mer komplexa och antalet möjliga syndrom ökar. Detta gör i sin tur att bakåtpropagation med en liten partistorlek har svårt att anpassa sig efter de mönster som finns i detektionsgraferna.

5.2 Jämförelse GNN och MWPM

GNN-avkodaren uppvisar marginellt bättre eller likvärdigt α_L jämfört med MWPM utvärderat på valideringsdatan för nätverken $(3, \leq 9)$, $(\leq 9, 3)$ och $(5, 5)$, det vill säga i närområdet kring $(5, 5)$ där hyperparameteroptimeringen genomfördes. För övriga nätverk är även p_L något sämre för GNN jämfört med MWPM, vilket stärker hypotesen att prestandan försämras för konfigurationer av (d, d_t) som ligger långt ifrån den punkt där optimeringen skedde, till följd av begränsad generalisering av hyperparameteroptimeringen. I figur 4.2 (a) kan det även observeras att *båda* avkodarna uppvisar samma övergripande trend, nämligen att p_L inte minskar med ökande (d, d_t). Detta stöder dessutom hypotesen att den observerade icke-minskningen i p_L för valideringsdatan är en konsekvens av att felfrekvensen p ligger över tröskelvärdet p_{tr}, i enlighet med tröskelsatsen.

Förvirringsmatriserna för GNN och MWPM i figur 4.3 visar att fördelningen av typ 1- och typ 2-fel är mycket snarlika. Dessutom har GNN-avkodaren och MWPM likvärdig logiska felsannolikhet för låga d och d_t. Dessa observationer väcker naturligt frågan om GNN-nätverket imiterar MWPM-algoritmen. Ett sådant generiskt beteende skulle dessutom kunna förklara varför nätverken för låga d och d_t har mycket snarlik logisk noggrannhet mellan träningsdata och valideringsdata. Dock presterar GNN-avkodaren sämre när den testats på ny data, vilket talar emot att den imiterar MWPM. Dessutom presterar GNN-avkodaren bättre än MWPM i vissa fall vilket skulle vara omöjligt om GNN-avkodaren endast imiterar MWPM.

För generalisering till andra exekveringar på andra kvantbitar vid ett senare mättillfälle presterar fortfarande GNN marginellt bättre för $d = d_t = 3$ och $d = 3, d_t = 5$. Dock presterar den avsevärt sämre för alla andra situationer och ännu sämre i figur 4.4 för avkodning av (d, d_t) som den inte är tränad på. Detta utgör en tydlig begränsning i användningen av GNN som avkodare. En fundamental fråga är hur denna begränsning skulle påverka möjligheterna till praktisk implementering. Å ena sidan tränas GNN-avkodaren typiskt för en specifik konfiguration av korrektionskoden, vilket i teorin borde avhjälpa problemet. Å andra sidan är kvantdatorer känsliga system som kräver kontinuerlig kalibrering [14], vilket innebär att felfördelningen förändras över tid. Det är detta fenomen vi försökt

modellera genom att testa avkodaren på syndrommätningar från andra kvantbitar, vilket kan liknas vid en kvantdator där de fysiska egenskaperna varierar över tid. En sådan situation skulle kräva att nätverket regelbundet uppdateras eller vidaretränas för att bibehålla sin prestanda.

Att grafstorleken skalar superlinjärt ($\alpha > 1$) med kodstorleken innebär ökade krav på minneshantering vid avkodning av större koder. Det kan därmed bli nödvändigt att optimera grafrepresentationen, till exempel genom att bortse från kanter med låga vikter. Den större spridningen i figur 4.5 vid låga d och d_t kan bero på en högre andel triviala grafer som inte avkodas. Sådana variationer i förekomsten av triviala fall kan även påverka beräkningen av avkodningstid, eftersom avkodningstiden här definieras som den totala avkodningstiden per exekvering dividerad med antalet fall N.

En intressant observation är att MWPM-avkodaren uppvisar kortare absolut avkodningstid, samtidigt som GNN-metoden uppvisar en mer gynnsam tidskomplexitet. Det är dock värt att notera att spridningen i mätvärdena, och därmed osäkerheten i uppskattningen av α, är större för GNN. Att avkodningstiden skalar exponentiellt enligt $T_{\text{avkdoning}} \propto (N_{\text{det}})^{\alpha}$ är även bara ett antagande, vilket inte nödvändigtvis är korrekt eftersom antydan på högre komplexitet än exponentiell kan ses i figur 4.6. En ytterligare intressant aspekt är att tidskomplexiteten är beroende av implementeringen, exempelvis om GNN körs på GPU via CUDA eller på CPU. För realtidsavkodning kan det vara motiverat att implementera algoritmen på en integrerad krets. Valet av hårdvaruimplementation kan därmed ha stor påverkan på prestandan och bör undersökas mer systematiskt. Resultaten i figur 4.6 bör därför tolkas med viss försiktighet.

Vid användning av det grafneurala nätverk presenterat i ekvation (2.8), det vill säga i det fall där $\forall(i,j) \in E : e_{ij} = 1$, minskar inte den logiska noggrannheten, se figur B.1 appendix B.2. Detta resultat innebär att valet av grafens vikter inte tycks vara betydande för mindre kodavstånd. Vidare reduceras tidskomplexiteten för avkodningen av grafen signifikant. Då den extra matrismultiplikationen för viktmatrisen $\mathbf{E}$ i ekvation (2.6) elimineras kan det vid implementering på en integrerad krets öka avkodningshastigheten redan vid små grafstorlekar.

5.3 Experimentell syndrommätning på QPU

Som redovisats i avsnitt 4.5 varierar den logiska felandelen i datan i hög grad beroende på hur kvantkretsen mappas till den fysiska QPU:n. Således innebär det att vi, om vi låter kvantbitarna allokeras godtyckligt, kan observera olika mängder och typer av fel för samma kodavstånd och antal tidsrepetitioner. Detta kan ha utgjort en begränsning i den tränade GNN-avkodarens generaliserbarhet mellan olika kvantkretsar.

Estimeringen av den fysiska felfrekvensen, p_{est}, visar att större kodavstånd medför en högre felfrekvens. Med hänsyn till tröskelsatsen och ekvation (2.3) innebär felfrekvensens d-beroende att vi för något kodavstånd d troligtvis överskrider tröskelvärdet p_{tr}, vilket motverkar syftet med högre kodavstånd. Tröskelvärdet i sig för repetitionskoden har inte behandlats i detta arbete, men baserat på resultaten från avkodningen finns det inget som tyder på att vi befinner oss under tröskelvärdet. Kvantprocessorns felbenägenhet, särskilt vid högre d, utgör därför en möjlig förklaring till varför avkodarens prestanda inte förbättras vid större kodavstånd.

5.4 Etiska övervägande bortom prestanda

Maskininlärningsbaserade avkodare måste även utvärderas bortom ett rent prestandaorienterat perspektiv. Träning av stora djupinlärningsmodeller kräver omfattande beräkningsresurser och resulterar i betydande energikonsumtion [25]. I många fall kan enklare algoritmer (i vårt fall MWPM) uppnå likvärdig prestanda med avsevärt lägre resursåtgång, vilket aktualiserar frågor om resurseffektivitet och miljöpåverkan. Dessutom riskerar utvecklingen att koncentreras till aktörer med tillgång till stora datacenter, vilket kan förstärka ojämlikheter i tekniktillgång och forskningsmöjligheter. Slutligen finns det en risk att maskininlärning appliceras på problem där det saknas tydlig motivering för dess användning. Om ML-metoder väljs av prestigeskäl snarare än på grundval av faktisk behovsanalys, kan detta leda till ineffektiva lösningar med onödiga alternativkostnader.

Det bör även tas i beaktande att kvantdatorer, särskilt i kombination med artificiell intelligens, kan utgöra en stor säkerhetsrisk. Ett ofta diskuterat exempel är användningen av kvantalgoritmer för dekryptering, vilket kan hota cybersäkerhetsstandarder [26]. När artificiell intelligens används för att effektivisera och styra sådana kvantalgoritmer, ökar risken ytterligare, eftersom processen kan automatiseras och skalas upp på sätt som försvårar upptäckt och motåtgärder. Detta väcker frågor om hur tillgång till sådan teknologi bör regleras, vem som är ansvarig vid missbruk samt hur säkerhetsinfrastrukturer ska utvecklas i takt med den teknologiska utvecklingen.

6

Slutsatser

Detta arbete har undersökt hur väl datadriven kvantfelskorrigering, genom avkodning av repetitionskod med ett grafneuralt nätverk (GNN), presterar i relation till en traditionell matchningsalgoritm (MWPM). Tyngdpunkten låg i att jämföra prestandan för GNN vid klassificering av syndrommätningar från en fasfelsdetekterande repetitionskod med MWPM, med fokus på avkodningsnoggrannhet, generaliserbarhet och tidskomplexitet.

Resultaten visar att GNN-avkodaren, under vissa förutsättningar, kan nå likvärdig eller marginellt högre logisk noggrannhet än MWPM. Detta gäller särskilt nära den kodarea där hyperparameteroptimeringen genomfördes. Notera att korrelationen inte nödvändigtvis tyder på kausalitet och slutsatsen dras därför med förbehåll. Samtidigt indikerar resultaten att GNN:ns generaliseringsförmåga är begränsad, vilket försvårar praktisk tillämpning i miljöer med varierande felfördelningar över tid.

Trots en systematisk optimering av hyperparametrar med ett tvåfaktorförsök begränsades analysens omfattning av tillgängliga resurser. Detta gör att resultaten bör tolkas med viss försiktighet, särskilt med tanke på metodens antagande om linjära samband och den begränsade sökrymden. Dessutom skalar varken modellstorleken eller de valda hyperparametrarna väl till större kodavstånd, vilket antyder att det är möjligt att mer komplexa nätverksarkitekturer kan krävas i dessa fall. För små kodareor presterar däremot de enklare nätverken väl, vilket tyder på att låg grafkomplexitet kan hantera enklare nätverksstrukturer. Emellertid verkar en omskrivning av GNN utan tillskrivna kantvikter i syndromgraferna lovande, vilket hade accelererat hårdvaruprestanda signifikant. Resultaten uppenbarar tröskelsatsen och det faktum att felfrekvensen p förmodligen överstiger det teoretiska tröskelvärdet p_{tr} och att vinsten med större d uteblir för den aktuella QPU:n.

Etiska och resursmässiga aspekter har också diskuterats. GNN erbjuder visserligen förbättrad avkodning i vissa fall, men detta sker till priset av avsevärt högre energiförbrukning. MWPM uppvisar jämförbar prestanda i de flesta testfall, men med betydligt lägre resursåtgång. Detta aktualiserar frågan om när maskininlärning är motiverad, och när det blir ett ineffektivt val drivet av tekniktrender istället för faktisk behovsanalys.

Sammantaget visar arbetet att grafneurala nätverk är lovande i specifikt optimerade fall, särskilt när modellen tränats för den aktuella korrektionskodskonfigurationen. Samtidigt kvarstår utmaningar kring generalisering, skalbarhet, praktisk implementering och resursanvändning. Dessa måste adresseras innan GNN kan konkurrera med matchningsalgoritmer i praktisk kvantdatoranvändning.

Fortsatt arbete kan rikta in sig på att tillämpa GNN-avkodning på mer avancerade korrektionskoder, som ytkoder. När IBM Quantum tillgängliggör hårdvara med lägre p öppnas även möjligheter att studera GNN för $p < p_{\text{tr}}$. Vidare bör alternativa nätverksstrukturer, fler storlekar och mer detaljerade faktorförsök utforskas. Det vore också värdefullt att fortsatt analysera nodvikterna i grafen för att bättre förstå nätverkets beslutsgrunder.

Referenser

[1] OpenAI ChatGPT, *Omslagsbild till rapport: Datadriven avkodning av experimentella syndrommätningar från kvantfasfelkorrigerande repetitionskoder med neurala grafnätverk*, https://chat.openai.com, Genererad med AI-bildverktyget DALL · E via ChatGPT den 2 maj 2025, 2025.

[2] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love och M. Head-Gordon, "Simulated Quantum Computation of Molecular Energies," *Science*, årg. 309, nr 5741, s. 1704–1707, 2005. DOI: 10.1126/science.1113479. [Online]. URL: https://www.science.org/doi/abs/10.1126/science.1113479.

[3] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren och D. Preda, "A Quantum Adiabatic Evolution Algorithm Applied to Random Instances of an NP-Complete Problem," *Science*, årg. 292, nr 5516, s. 472–475, 2001. DOI: 10.1126/science.1057726. [Online]. URL: https://www.science.org/doi/abs/10.1126/science.1057726.

[4] S. Lloyd, "Universal Quantum Simulators," *Science*, årg. 273, nr 5278, s. 1073–1078, 1996. DOI: 10.1126/science.273.5278.1073. [Online]. URL: https://www.science.org/doi/abs/10.1126/science.273.5278.1073.

[5] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Journal on Computing*, årg. 26, nr 5, s. 1484–1509, 1997. DOI: 10.1137/S0097539795293172. [Online]. URL: https://doi.org/10.1137/S0097539795293172.

[6] B. Paredes, F. Verstraete och J. I. Cirac, "Exploiting Quantum Parallelism to Simulate Quantum Random Many-Body Systems," *Phys. Rev. Lett.*, årg. 95, s. 140 501, 14 sept. 2005. DOI: 10.1103/PhysRevLett.95.140501. [Online]. URL: https://link.aps.org/doi/10.1103/PhysRevLett.95.140501.

[7] R. P. Feynman, "Simulating Physics with Computers," *International Journal of Theoretical Physics*, årg. 21, nr 6, s. 467–488, 1 juni 1982, ISSN: 1572-9575. DOI: 10.1007/BF02650179. [Online]. URL: https://doi.org/10.1007/BF02650179.

[8] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Phys. Rev. A*, årg. 52, R2493–R2496, 4 okt. 1995. DOI: 10.1103/PhysRevA.52.R2493. [Online]. URL: https://link.aps.org/doi/10.1103/PhysRevA.52.R2493.

[9] W. K. Wootters och W. H. Zurek, "A single quantum cannot be cloned," *Nature*, årg. 299, nr 5886, s. 802–803, 1982, ISSN: 1476-4687. DOI: 10.1038/299802a0. [Online]. URL: https://doi.org/10.1038/299802a0.

[10] J. Roffe, "Quantum error correction: an introductory guide," *Contemporary Physics*, årg. 60, nr 3, s. 226–245, 2019. DOI: 10.1080/00107514.2019.1667078.

[11] O. Higgott, "PyMatching: A Python Package for Decoding Quantum Codes with Minimum-Weight Perfect Matching," *ACM Transactions on Quantum Computing*, årg. 3, nr 3, 2022. DOI: 10.1145/3505637. [Online]. URL: https://www.scopus.c

om/inward/record.uri?eid=2-s2.0-85176900795&doi=10.1145%2f3505637&partnerID=40&md5=004ee3f58d66ee6cd1ddc5efb2eab43d.

[12] M. Lange, P. Havström, B. Srivastava *m.fl.*, "Data-driven decoding of quantum error correcting codes using graph neural networks," *Phys. Rev. Res.*, årg. 7, s. 023 181, 2 maj 2025. DOI: 10.1103/PhysRevResearch.7.023181. [Online]. URL: https://link.aps.org/doi/10.1103/PhysRevResearch.7.023181.

[13] R. Acharya, L. Aghababaie-Beni, I. Aleiner *m.fl.*, "Quantum error correction below the surface code threshold," *arXiv preprint arXiv:2408.13687*, 2024.

[14] M. Steffen, D. P. DiVincenzo, J. M. Chow, T. N. Theis och M. B. Ketchen, "Quantum computing: An IBM perspective," *IBM Journal of Research and Development*, årg. 55, nr 5, 13:1–13:11, 2011. DOI: 10.1147/JRD.2011.2165678.

[15] M. A. Nielsen och I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[16] Glosser.ca. "Bloch Sphere." (2012), [Online]. URL: https://commons.wikimedia.org/wiki/File:Bloch_Sphere.svg (hämtad 2025-02-15).

[17] C. Forssén. "TIF385 Bayesian inference and machine learning." (2022), [Online]. URL: https://cforssen.gitlab.io/tif385-book/content/Preface/preface-tif385.html (hämtad 2025-05-06).

[18] C. Morris, M. Ritzert, M. Fey *m.fl.*, "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks," *CoRR*, årg. abs/1810.02244, 2018. [Online]. URL: https://arxiv.org/abs/1810.02244.

[19] PyTorch Team, *BCELoss — PyTorch 2.1 documentation*, 2024. [Online]. URL: https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html (hämtad 2025-05-06).

[20] D. P. Kingma och J. Ba, "Adam: A Method for Stochastic Optimization," *CoRR*, årg. abs/1412.6980, 2015. [Online]. URL: http://arxiv.org/abs/1412.6980.

[21] G. E. P. Box, J. S. Hunter och W. G. Hunter, *Statistics for Experimenters: Design, Innovation, and Discovery*, 2. utg. Wiley, 2005, ISBN: 9780471718130.

[22] G. James, D. Witten, T. Hastie och R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R* (Springer Texts in Statistics), 2. utg. New York: Springer, 2021. DOI: 10.1007/978-1-0716-1418-1. [Online]. URL: https://link.springer.com/book/10.1007/978-1-0716-1418-1.

[23] S. Benstorp, L. B. Falkenström, S. Eriksson, A. Jonsson, L. Petersson och V. Petersson. "Kandidatarbete TIFX11-19." (2025), [Online]. URL: https://git.chalmers.se/simoe/kandidatarbete-tifx11-19.

[24] IBM Quantum. "IBM Quantum Services - Resources for ibm_marrakesh." (2025), [Online]. URL: https://quantum.ibm.com/services/resources?system=ibm_marrakesh (hämtad 2025-04-10).

[25] P. Dhar, "The Carbon Impact of Artificial Intelligence," *Nature Machine Intelligence*, årg. 2, nr 8, s. 423–425, 2020, ISSN: 2522-5839. DOI: 10.1038/s42256-020-0219-9. [Online]. URL: https://doi.org/10.1038/s42256-020-0219-9.

[26] V. Mavroeidis, K. Vishi, M. D. Zych och A. Jøsang, "The Impact of Quantum Computing on Present Cryptography," *arXiv preprint arXiv:1804.00200*, 2018.

A

Grafneurala nätverk som använts vid optimering av hyperparametrar

Nedan redovisas de nätverksstrukturer och kombinationer av hyperparametrar som användes i avsnitt 3.2.1 för att träna grafneurala nätverk. Utöver redovisas tränade nätverkens logiska noggrannhet.

A.1 2^3-faktorförsök

I tabell A.1 redovisas de värden på hög och låg nivå som användes för parametrarna i 2^3-faktorförsöket. Prestandan i termer av logisk nogrannhet för varje kombination av parametrar redovisas i tabell A.2.

Tabell A.1: Definition av hög och låg nivå för 2^3-faktorförsöket. Pil $a \to b$ anger ingångs– och utgångsdimension för varje lager i det neurala nätverket.

Parameter	**+ (hög nivå)**	**- (låg nivå)**
Inlärningshastighet	0,0005	0,0001
Partistorlek	1024	256
Modelstorlek	GraphConv_1: $2 \to 64$	GraphConv_1: $2 \to 32$
	GraphConv_2: $64 \to 128$	GraphConv_2: $32 \to 64$
	GraphConv_3: $128 \to 128$	GraphConv_3: $64 \to 128$
	GraphConv_4: $128 \to 256$	Fullt anslutet_1: $128 \to 64$
	Fullt anslutet_1: $256 \to 128$	Fullt anslutet_2: $64 \to 4$
	Fullt anslutet_2: $128 \to 64$	Fullt anslutet_3: $4 \to 1$
	Fullt anslutet_3: $64 \to 32$	
	Fullt anslutet_4: $32 \to 1$	

Tabell A.2: Prestanda i termer av logisk noggrannhet α_L på valideringsdatan för de tränande nätverken i 2^3-faktorförsök. För varje kombination av parametrar tränades minst tre nätverk och deras logiska noggrannheter redovisas separat i tabellen.

ID	Inlärningshastighet	Partistorlek	Modellstorlek	α_L
1	+	+	+	0,965; 0,965; 0,965
2	+	+	–	0,964; 0,969; 0,969; 0,969
3	+	-	+	0,959; 0,956; 0,957; 0,956; 0,957; 0,911
4	+	–	–	0,970; 0,966; 0,967
5	–	+	+	0,969; 0,970; 0,970
6	–	+	–	0,840; 0,960; 0,963; 0,963; 0,965
7	–	–	+	0,967; 0,969; 0,969
8	–	–	–	0,965; 0,967; 0,968

A.2 Rutnätsförsök

De 9 modell- och parameterkonfigurationer som användes i rutnätsförsöket samt deras prestanda redovisas i tabell A.3.

Tabell A.3: Konfigurationer för rutnätsförsöket. Pil $a \to b$ anger lagrets ingångs- och utgångsdimension. Den logiska noggrannhet α_L som redovisas är de tränade nätverkens logiska noggrannhet på valideringsdatan.

ID	Modellstruktur	Inlärningshastighet	Partistorlek	α_L
1	GraphConv_1: $2 \to 32$ GraphConv_2: $32 \to 64$ Fullt anslutet_1: $64 \to 32$ Fullt anslutet_2: $32 \to 4$ Fullt anslutet_3: $4 \to 1$	0,001	1024	0,9694
2	GraphConv_1: $2 \to 32$ GraphConv_2: $32 \to 64$ GraphConv_3: $32 \to 64$ GraphConv_4: $64 \to 128$ Fullt anslutet_1: $128 \to 64$ Fullt anslutet_2: $64 \to 4$ Fullt anslutet_3: $4 \to 1$	0,001	1024	0,9625
3	GraphConv_1: $2 \to 64$ GraphConv_2: $64 \to 128$ GraphConv_3: $64 \to 128$ Fullt anslutet_1: $128 \to 64$ Fullt anslutet_2: $64 \to 4$ Fullt anslutet_3: $4 \to 1$	0,00159	1024	0,9715
4	GraphConv_1: $2 \to 64$ GraphConv_2: $64 \to 128$ Fullt anslutet_1: $128 \to 64$ Fullt anslutet_2: $64 \to 4$ Fullt anslutet_3: $4 \to 1$	0,00041	1024	0,9699

Fortsättning följer på nästa sida

Tabell A.3: Konfigurationer för rutnätsförsöket (forts.)

ID	Modellstruktur	Inlärningshastighet	Partistorlek	α_L
5	GraphConv_1: $2 \to 32$ GraphConv_2: $32 \to 64$ GraphConv_3: $64 \to 128$ Fullt anslutet_1: $128 \to 64$ Fullt anslutet_2: $64 \to 4$ Fullt anslutet_3: $4 \to 1$	0,0015	1024	0,9707
6	GraphConv_1: $2 \to 32$ GraphConv_2: $32 \to 64$ Fullt anslutet_1: $64 \to 32$ Fullt anslutet_2: $32 \to 4$ Fullt anslutet_3: $4 \to 1$	0,00159	1024	0,9713
7	GraphConv_1: $2 \to 32$ GraphConv_2: $32 \to 64$ Fullt anslutet_1: $64 \to 32$ Fullt anslutet_2: $32 \to 4$ Fullt anslutet_3: $4 \to 1$	0,0005	1024	0,9670
8	GraphConv_1: $2 \to 32$ GraphConv_2: $32 \to 64$ GraphConv_3: $64 \to 128$ Fullt anslutet_1: $128 \to 64$ Fullt anslutet_2: $64 \to 4$ Fullt anslutet_3: $4 \to 1$	0,0005	1024	0,9689
9	GraphConv_1: $2 \to 64$ GraphConv_2: $64 \to 128$ Fullt anslutet_1: $128 \to 64$ Fullt anslutet_2: $64 \to 4$ Fullt anslutet_3: $4 \to 1$	0,001	1024	0,9726

B

Omskrivning och logisk noggrannhet av GNN med grafer utan viktade kanter

I nedan sektioner presenteras både en uträkning för den omskrivning av ekvation (2.6) samt utvärdering av logisk noggrannhet för $d_t = 3$, $d \in \{7,11,15\}$ för GNN:et med vikter $\forall(i,j) \in E : e_{ij} = 1$.

B.1 Omskrivning av grafneuralt nätverk med samtliga vikter (FCGNN)

Antag att grafen G är fullt sammankopplad med kantvikter $\forall(i,j) \in E : e_{ij} = 1$. Således blir grafens kantviktmatris $\mathbf{E} = \mathbf{1}\mathbf{1}^T - \mathbf{I}$. Ekvation (2.6) kan således skrivas som

$$\mathbf{X}^{l+1} = f(\mathbf{W}_1^l\mathbf{X}^l + \mathbf{W}_2^l\mathbf{E}\mathbf{X}^l) = f(\mathbf{W}_1^l\mathbf{X}^l + \mathbf{W}_2^l\left(\mathbf{1}\mathbf{1}^T - \mathbf{I}\right))\mathbf{X}^l =$$

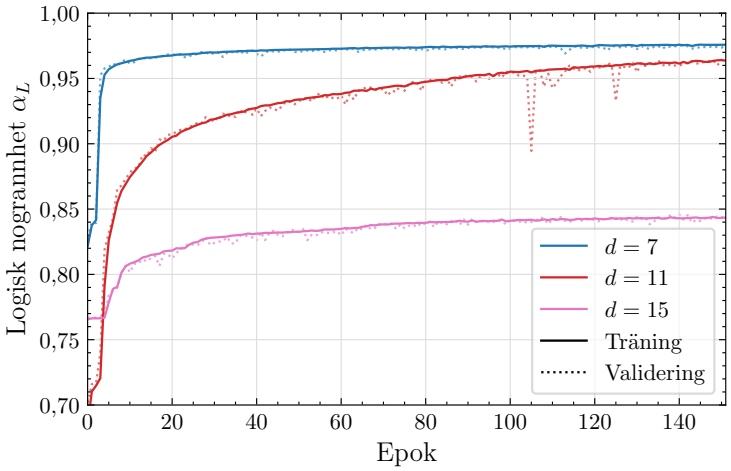
$$= f((\mathbf{W}_1^l - \mathbf{W}_2^l)\mathbf{X}^l + \mathbf{W}_2^l\left(\mathbf{1}\mathbf{1}^T\right)\mathbf{X}^l) = f((\mathbf{W}_1^l - \mathbf{W}_2^l)\mathbf{X}^l + \mathbf{W}_2^l\mathbf{1}\left(\mathbf{1}^T\mathbf{X}^l\right)) = [\mathbf{s} = \left(\mathbf{1}^T\mathbf{X}^l\right)] =$$

$$= f\left((\mathbf{W}_1^l - \mathbf{W}_2^l)\mathbf{X}^l + \mathbf{W}_2^l\mathbf{1}\mathbf{s}\right)$$

Där $\mathbf{X}^l$ är en matris av dimension $M \times F$ där M är antalet noder i grafen och F är antalet egenskaper för varje nod, vilket i initialfallet är tid och rum. $\mathbf{1}$ antas vara av dimension $M \times 1$ och I antas vara av dimension $M \times M$.

B.2 Utvärdering av logisk noggrannhet för FCGNN

I figur B.1 presenteras den logiska noggrannheten α_L under träningsförloppet av de grafneurala nätverken där samtliga grafers kantvikter $\forall(i,j) \in E : e_{ij} = 1$. Istället för att tränas om från noll, användes överföringsinlärning där varje nätverkens vikter initierades med vikterna från nätverket för samma d och d_t fast med varierade kantvikter. Nätverken tränades under 150 epoker med i regel $N = 400000$ syndrom, varav 10% användes för validering. Båda nätverk använde en partistorlek på 256. Notera att nätverken presterar lika bra med kantvikter ansatta till 1 som de nätverk tränade med varierande kantvikter som finns redovisade i figur 4.1a.



Figur B.1: Tränings- och valideringsnoggrannhet per träningsepok för de grafneurala nätverken tränade med grafer där alla kantvikter $\forall(i,j) \in E : e_{ij} = 1$. Nätverken har $d_t = 3$ och är uppdelade på kodavstånd d.

INSTITUTIONEN FÖR FYSIK
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige
www.chalmers.se

CHALMERS