

Sommerprosjekt 2025

BB kode og Tegl kode med analog kvantefeilkorrigerings

Anton Brekke^{1,*} og Brage A. Trefjord^{2,†}

¹*Institutt for frie studier, Universitetet i Oslo, Norge*

²*Fysisk institutt, Universitetet i Oslo, Norge*

(Dated: August 19, 2025)

Vi presenterer en uformell oppsummering av vårt arbeid i sammenheng med et sommerprosjekt, der vi har undersøkt bruk av analog informasjon om feilgenerering til å forbedre feilkorrigerings for to nye klasser kvantekode: Bivariate Bicycle (BB) og Tile koder. Vi fant at bruk av analog informasjon til dekoding vil forbedre feilkorrigerings med opp til flere prosent. I tillegg har vi sammenliknet BB koden, Tile koden og overflatekoden i et oppsett med $n = 360$ og $k = 8$ fysiske og logiske qubits, der vi fant at Tile kode og BB kode vil være en bedre kode å bygge enn overflatekoden på en generell basis.

I. INTRODUKSJON

Noen oppgaver er det tilnærmet umulig å gjennomføre på en vanlig datamaskin, i alle fall innenfor noen rimelig tidsramme. Noen av disse oppgavene kan derimot løses ganske raskt med en kvantedatamaskin. For å bygge en kvantedatamaskin trenger man mange qubits, men disse er svært ustabile og mottakelige for påvirkning. Man ønsker derfor å bruke mange slike qubits som jobber sammen for å lage færre, men mer stabile *logiske* qubits. Disse *logiske* qubitsene blir mer stabile ved at man gjør *feilkorrigerings*, hvor de mange *fysiske* qubitsene kontinuerlig sjekker hverandre for, og retter opp, feil. Vi har brukt denne sommeren til å utforske slik feilkorrigerings av qubits ved å simulere den mest standard kvantekoden for feilkorrigerings: “overflatekoden”, i tillegg til to nyere modeller: “BB kode” og “fliskode” (som på engelsk heter henholdsvis “BB code” og “Tile codes”). Vi oppsummerer noe av det vi har lært i denne rapporten, inkludert både teori og resultater fra prosjektet. Simuleringene våre er skrevet i `python` ved hjelp av pakken `PanQEC`[1], og er tilgjengelig på våre Github-sider.¹

II. FEIL OG KORRIGERING

Det er hovedsakelig to typer feil som kan oppstå på en qubit[2]:

- Bitflips (X-feil) endrer $|0\rangle$ til $|1\rangle$ og vice versa:

$$\hat{X}(\alpha|0\rangle + \beta|1\rangle) = \alpha|1\rangle + \beta|0\rangle.$$

- Faseflips (Z-feil) skifter fortegn på $|1\rangle$, men lar $|0\rangle$ forbli uendret:

$$\hat{Z}(\alpha|0\rangle + \beta|1\rangle) = \alpha|0\rangle - \beta|1\rangle.$$

For å forenkle notasjonen utelater vi hatten over operatører videre i rapporten. Merk at $|0\rangle$ og $|1\rangle$ er egen-tilstander til operatoren Z , men ikke til X . Det er noen ganger nyttig å bruke egentilstandene X gitt ved $|\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$.

For å gjøre feilkorrigerings av qubits må vi først kunne se om det har skjedd en feil i utgangspunktet. Når vi måler en qubit kollapse dens bølgefunksjon, og vi mister tilstanden den hadde i utgangspunktet, for eksempel $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \xrightarrow{\text{måling}} |0\rangle$. Vi kan altså ikke sjekke om en qubit er riktig ved å måle den direkte. For å gjøre feilkorrigerings bruker vi derfor flere “fysiske” qubits for å representere én enkelt “logisk” (eller “nyttig”) qubit, og gjør målinger på flere fysiske qubits på en slik måte at vi unngår å ødelegge informasjonen som er kodet inn i den “logiske” qubiten. Målingene vi bruker for å finne feil kalles *stabilisatorer*.

Et eksempel på dette er repetisjonskode, hvor vi definerer logiske qubits ved å repetere qubiten: $|\psi\rangle_L = \bigotimes_{i=1}^n |\psi\rangle$. Dersom vi for eksempel har tre fysiske qubits, og ønsker en logisk 0-tilstand kan vi skrive dette som $|0\rangle_L = |000\rangle$.² Videre definerer vi stabilisatorene våre til å være $S_1 = Z_1 Z_2$ og $S_2 = Z_2 Z_3$, hvor Z_i gjør en Z -måling på den i 'te qubiten. Disse stabilisatorene sjekker om de fysiske qubitene er parvis like. Hvis de to første fysiske qubitene begge er $|0\rangle$ eller begge er $|1\rangle$ vil S_1 gi en positiv (+1) måling, mens den gir en negativ (−1) måling hvis den ene fysiske qubiten er $|0\rangle$ og den andre $|1\rangle$. Vi kan tenke oss at det skjer en X -feil på én av qubitsene, for eksempel $|0\rangle_L^{\text{feil}} = X_3 |0\rangle_L = |001\rangle$. Vi bruker stabilisatorene til å finne $S_1 |001\rangle = +|001\rangle$ og $S_2 |001\rangle =$

* antonabr@uio.no

† brageat@uio.no

¹ Anton (fokus på BB code): <https://github.com/AntonBrekke/BBCODE-QEC-2025>.

Brage (fokus på Tile Code): <https://github.com/Bragit123/QEC>.

² Her bruker vi notasjonen $|\psi_1 \dots \psi_n\rangle := |\psi_1\rangle \otimes \dots \otimes |\psi_n\rangle$

$-|001\rangle$. Siden S_1 ga positiv og S_2 negativ måling vet vi at de to første qubitene er like mens de to siste er ulike. Fra dette konkluderer vi med at det sannsynligvis er den siste qubiten som er feil, og retter opp med en X_3 -operator: $|0\rangle_L^{\text{korrigert}} = X_3 = |0\rangle_L^{\text{feil}} = |000\rangle = |0\rangle_L$. Vi bruker altså flertallet av qubitsene til å korrigere den logiske qubiten.

Et par ting som er verdt å merke seg her:

- Merk at dette ikke ville fungert om det skjedde en feil på flere qubits. For eksempel ville $|0\rangle_L^{\text{feil}} = |101\rangle$ blitt korrigert til $|0\rangle_L^{\text{korrigert}} = |111\rangle \neq |0\rangle_L$. Ved å bruke flere fysiske qubits for å kode informasjon i en logisk qubit kan vi minke sannsynligheten for at feilkorrigeringen gir galt resultat, men det kommer med en pris ettersom vi trenger flere qubits for hver logiske qubit.
- Målingene vi gjør med stabilisatorene resulterer i enten $+1$ (ingen feil) eller -1 (feil), men endrer ikke tilstanden til den logiske qubiten. Vi får dermed målt om det har skjedd en feil uten å ødelegge qubiten vår.³

III. TORISK OVERFLATEKODE

Vi oppsummerer den toriske overflatekoden i grove trekk. For en fullstendig introduksjon til topologiske overflatekoder henviser vi leseren til [3].

1. Homologi

Den toriske overflatekoden er en topologisk kode som benytter seg av torusens *homologigruppe* for å beskytte qubits. Kort oppsummert består homologigruppen av løkker på torusen som ikke kan skrives som randen av en overflate på torusen (ikke-trivielle løkker).

Formelt kan vi plassere qubits på et rutenett, som anvist i Fig. 2, der qubits er plassert på kantene i rutenettet, og vertekser (plaketter) er markert med blå (røde) kvadrater. Fra rutenettet kan vi danne 3 abelske grupper under addisjon modulo 2: gruppen av hjørner/punkter/noder (0-kjeder) C_0 , gruppen av kjeder (1-kjeder) C_1 , og gruppen av flater (2-kjeder) C_2 . Noder, kjeder og flater kan være en arbitrær sum av vertekser, kanter og plaketter jf. (6). Mer generelt kan vi danne i -kjeder tilhørende den abelske gruppen C_i .

Dersom man også kan finne et sett med homomorfier $\{\partial_i : C_i \rightarrow C_{i-1}\}_{i=1}^N$ som tilfredsstiller $\partial_i \circ \partial_{i+1} = 0$

mellom kjedegruppene kan man danne et *kjede-kompleks* $(C_\bullet, \partial_\bullet)$ slik at

$$C_\bullet : \dots \longrightarrow C_{n+1} \xrightarrow{\partial_{n+1}} C_n \xrightarrow{\partial_n} C_{n-1} \xrightarrow{\partial_{n-1}} \dots \quad (1)$$

Slike homomorfier i kjede-komplekset kalles *randoperatorer*, og brukes til å konstruere den n -te homologi-gruppen H_n definert som kvotientgruppen

$$H_n := \ker(\partial_n) / \text{im}(\partial_{n+1}). \quad (2)$$

Mer generelt dannes en kvotientgruppe fra en gruppe G og en normal undergruppe $H \triangleleft G$. En normal undergruppe er en undergruppe $H \leq G$ som tilfredsstiller $ghg^{-1} \in H$ for alle $g \in G, h \in H$, som er nødvendig for at gruppeoperasjoner i kvotientgruppen skal være veldefinert. Dersom G er abelsk (og dermed også H) vil alle undergrupper være normale, fordi $ghg^{-1} = gg^{-1}h = h \in H$. Da kan man danne kvotientgruppen

$$G/H := \{gH \mid g \in G\}, \quad gH := \{gh \mid h \in H\}. \quad (3)$$

Kvotientgruppen deles dermed inn i klasser av g , med avbildningen $\pi : G \rightarrow G/H$ som $g \mapsto gH$. Gruppemultiplikasjon i kvotientgruppen er definert som

$$(aH)(bH) = (ab)H, \quad a, b \in G, \quad (4)$$

som er veldefinert nettopp fordi H er normal. Dersom vi velger $b = e$ til å være identitets-elementet, ser vi at $(aH)(H) = aH$ – med andre ord blir H sett på som identitets-elementet i kvotientgruppen. Konstruksjonen inducerer også ekvivalensrelasjonen $a \sim b \Leftrightarrow a^{-1}b \in H \Leftrightarrow b \in aH \Leftrightarrow b = ah$ – med andre ord sier vi at b er ekvivalent med a dersom $b = ah$ for $h \in H$. Dermed vil alle elementene i gH være ekvivalente, og vi kaller dermed settet gH for en *ekvivalensklasse*, som vi betegner $[x]$ for en hvilken som helst representant $x \in gH$.

Som et konkret eksempel kan vi se på hva som skjer dersom vi danner en kvotientgruppe av \mathbb{Z} med undergruppen $3\mathbb{Z} = \{3k \mid k \in \mathbb{Z}\}$. Da vil

$$\begin{aligned} \mathbb{Z}/3\mathbb{Z} &= \{n + 3\mathbb{Z} \mid n \in \mathbb{Z}\} \\ &= \{\{n + 3k \mid k \in \mathbb{Z}\} \mid n \in \mathbb{Z}\} \\ &= \{\{\dots, -6, -3, 0, 3, 6, \dots\}, \{\dots, -5, -2, 1, 4, 7, \dots\}, \\ &\quad \{\dots, -4, -1, 2, 5, 8, \dots\}\} \\ &= \{[0], [1], [2]\}. \end{aligned}$$

Vi ser at vi har delt gruppen \mathbb{Z} inn i 3 ekvivalensklasser med ekvivalensrelasjon $n \sim m \Leftrightarrow m = n + 3k, n, m, k \in \mathbb{Z}$, der addisjon i kvotientgruppen er f.eks $[1] + [2] = [1 + 2] = [3] = [0]$ (kan også skrives $(1 + 3\mathbb{Z}) + (2 + 3\mathbb{Z}) = (1+2) + 3\mathbb{Z}$ etc.). Dette er det eksakt samme som gruppen av heltall med addisjon modulo 3, \mathbb{Z}_3 , og det finnes en naturlig isomorfi $\phi : \mathbb{Z}/3\mathbb{Z} \rightarrow \mathbb{Z}_3$ som sender $[n] \rightarrow n$. Dermed er $\mathbb{Z}_3 \cong \mathbb{Z}/3\mathbb{Z}$.

Homologigruppen arver en abelsk gruppestruktur fra C_i , og vi kan generelt gi elementene koeffisienter over en ring R for å danne en modul $H_n(X; R)$ der X er rommet

³ Merk at repetisjonskoden vår kun beskytter mot X -feil. For å lage en tilsvarende kode som beskytter mot Z -feil kan vi bruke $|0\rangle_L = |+++\rangle$ og $|1\rangle_L = |--\rangle$ i stedet.

vi jobber på. Dersom vi gir elementene koeffisienter over en kropp K , blir $H_n(X; K)$ et vektorrom. I vårt tilfelle vil vi alltså bruke Galois kroppen \mathbb{F}_2 (også kalt $\text{GF}(2)$) på homologigruppen, slik at $H_n = H_n(X; \mathbb{F}_2)$.

La oss konkretisere dette i kontekst av den toriske overflatekoden, der $H_n = H_n(T^2; \mathbb{F}_2)$. Der har vi kjede-komplekset og homologigruppen

$$0 \xrightarrow{\partial_3} C_2 \xrightarrow{\partial_2} C_1 \xrightarrow{\partial_1} C_0 \xrightarrow{\partial_0} 0, \quad (5)$$

$$H_1 = \ker(\partial_1)/\text{im}(\partial_2)$$

I dette tilfellet vil ∂_2 ta inn en overflate $f \in C_2$ og gi tilbake randen av flaten, mens ∂_1 vil ta inn en kjede $c \in C_1$ og gi tilbake endepunktene av kjeden. Da vil også $\partial_1 \circ \partial_2 = 0$ være oppfylt, fordi randen av en flate er en kjede uten endepunkter (sykel/løkke). Dermed blir $Z_1 := \ker(\partial_1) = \{c \in C_1 \mid \partial_1 c = 0\}$ settet av alle løkker, og $B_1 := \text{im}(\partial_2) = \{c = \partial_2 f \mid f \in C_2\}$ blir settet av alle løkker som kan skrives som randen av en flate i rutenettet (også bare kalt rand). Homologigruppen $H_1 := Z_1/B_1$ dannes dermed ved å ta alle løkker på torusen, og “dele ut” rendene – altså er B_1 identitets-elementet i homologigruppen. Det vi står igjen med er ekvivalensklasser for løkkene på torusen som ikke er en rand av en flate, som for torusen blir løkkene illustrert i Fig. 1. I rutenettet vil de gå på tvers, vist i Fig. 3.

Elementene i homologigruppen er ekvivalensklasser $\bar{z} := z + B = \{z + b \mid b \in B_1\} \in H_1, z \in Z_1$ – altså ser man på to løkker som ekvivalente dersom forskjellen kun er en rand. Addisjon i homologigruppen arves fra Z_1 , og gjøres ved $\bar{z} + \bar{z}' = \overline{z + z'}$.

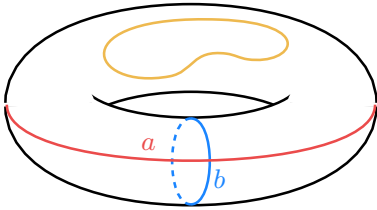


FIG. 1. Torus over \mathbb{Z}_2 med sine to ikke-trivielle løkker i rødt og blått, og en triviell løkke i oransje. De ikke-trivielle løkkene omslutter ikke en flate på torusen, og kan dermed ikke “snurpes igjen”. Her er homologigruppen $H_1(T^2; \mathbb{F}_2) = \{0, a, b, a + b\}$ fordi vi jobber over modulus 2 slik at $a + a = 2a = 0 \pmod{2}$, men for $H_1(T^2; \mathbb{Z})$ vil også $2a \in H_1$. Vi ser at $|H_1| = 4 = 2^2$, mens H_1 har basis $\{a, b\}$ som gir $\dim H_1 = 2$ som et vektorrom over \mathbb{F}_2 .

Gitt en kjede $c \in C_1$ bestående av kanter $E = \{e_i\}$, er det naturlig å definere den korresponderende kvantetilstanden ved

$$c = \sum_i c_i e_i \rightarrow |c\rangle = \bigotimes_i |c_i\rangle, \quad c_i = \begin{cases} 1, & e_i \in E \\ 0, & e_i \notin E \end{cases}, \quad (6)$$

hvor indeksen i nå løper over alle kantene i rutenettet.

Tilsvarende Pauli-operatorer for c blir

$$X_c := \bigotimes_i X_i^{c_i}, \quad X_c X_{c'} = X_{c+c'}, \quad (7)$$

$$Z_c := \bigotimes_i Z_i^{c_i}, \quad Z_c Z_{c'} = Z_{c+c'}, \quad (8)$$

hvor også $X_{c'}|c\rangle = |c+c'\rangle$. Selve definisjonen av koderommet for overflatekoden er underrommet spent ut av tilstandene fra homologigruppen

$$|\bar{z}\rangle := \sum_{b \in B_1} |z + b\rangle, \quad \bar{z} \in H_1. \quad (9)$$

Antallet logiske qubits blir da

$$\begin{aligned} k = \dim H_1 &= \dim \ker(\partial_1) - \dim \text{im}(\partial_2) \\ &= \dim C_1 - \text{rk}(\partial_1) - \text{rk}(\partial_2) \\ &= n - \text{rk}(\partial_1) - \text{rk}(\partial_2) \end{aligned} \quad (10)$$

hvor vi har brukt rang-nullitetsteoremet, og n er antallet fysiske qubits i rutenettet. For torusen kan vi finne at $\text{rk}(\partial_1) = \text{rk}(\partial_2) = n/2 - 1$ ved å bruke at $\ker(\partial_0) = C_0$ og $\text{im}(\partial_3) = \{0\}$, $\ker(\partial_2) = \{0, f_{T^2}\}$ og dermed $\dim H_2 = 1 = n/2 - \text{rk}(\partial_2)$ og $\dim H_0 = 1 = n/2 - \text{rk}(\partial_1)$ ($\dim H_0 = 1$ fordi alle noder i rutenettet er koblet sammen). Dimensjonen på koderommet er $|H_1| = 2^k$. Dette er en nyttig definisjon av koderommet, fordi homologien ikke er sensitiv til deformasjoner. Først og fremst vil $\langle \bar{z}' | \bar{z} \rangle = 0$ dersom $\bar{z} \neq \bar{z}'$ fordi de tilhører ulike ekvivalensklasser. Om vi først betrakter $c \in C_1$ slik at $\partial_1 c \neq 0$, vil vi få

$$\langle \bar{z}' | X_c | \bar{z} \rangle = \langle \bar{z}' | \overline{z + c} \rangle = 0 \quad (11)$$

fordi $\partial_1 c \neq 0 \Rightarrow \partial_1(z + c) \neq 0$, og dermed $\overline{z + c} \neq \bar{z}'$. Elementet $\overline{z + c} \notin H_1$, og X_c har “sparket oss ut” av koderommet. Dersom vi heller får bitflips på kjeder der $\partial_1 c = 0$ (sykler/løkker), vil ikke feilen kunne detekteres fordi $\overline{z + c} = \bar{z}'$. Om vi heller ser på $b \in B_1$, vil vi få

$$X_b |\bar{z}\rangle = \sum_{b' \in B_1} |z + b' + b\rangle = \sum_{b \in B_1} |z + b\rangle = |\bar{z}\rangle, \quad (12)$$

og kodetilstanden vil ikke endre seg. Altså vil bitflips på render virke trivielt på kodetilstandene. Dette er noe av kraften bak den toriske overflatekoden – vi trenger kun å feilkorrigere opp til homologi, fordi det ikke har noe å si om vi korrigerer feil med en rand.

2. Stabilisatorer på en torus

Stabilisatorene for den toriske koden er

$$X_f = \bigotimes_{e \in \partial_2 f} X_e^{c_e}, \quad Z_v = \bigotimes_{e \in \partial_1 v} Z_e^{c_e}, \quad (13)$$

og er operatorer med egenverdier ± 1 . Koderommet er per definisjon $+1$ egenverdi-rommet av stabilisatorene. Fordi

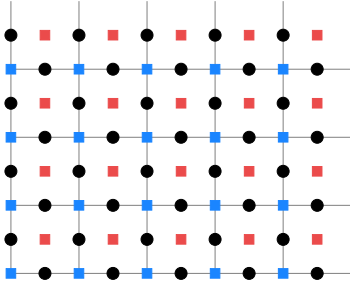


FIG. 2. Qubits, her visualisert som sorte sirkler på hver kant, kan organiseres i et rutenett og legges rundt en torus. Vertekser vises med blå kvadrater som representerer Z -sjekker/stabilisatorer, mens plaketter vises med røde kvadrater som representerer X -sjekker/stabilisatorer.

egenverdiene er ± 1 vil projektorene ned på ± 1 egenrommene være $P_{X/Z}^{\pm} = \frac{1}{2}(1 \pm X_f/Z_v)$. La oss fokusere på $+1$ egenrommet på en kjede-tilstand $|c \in C_1\rangle$:

$$|\tilde{c}\rangle = \bigotimes_f \frac{1 + X_f}{2} \bigotimes_v \frac{1 + Z_v}{2} |c\rangle. \quad (14)$$

Legg merke til at $Z_v|c\rangle = -|c\rangle \Rightarrow |\tilde{c}\rangle = 0$ dersom $v \in \partial_1 c$, og $Z_v|c\rangle = |c\rangle$ dersom $v \notin \partial_1 c$. Dermed må $\partial_1 c = 0$ for at $|\tilde{c}\rangle \neq 0$, som betyr at $c = z \in Z_1$. Videre kan vi ekspandere

$$\begin{aligned} \bigotimes_f (1 + X_f) &= \sum_{\{f_i\}} \bigotimes_i X_{\partial_2(f_i)} \\ &= \sum_{\{f_i\}} X_{\partial_2(\sum_i f_i)} \\ &= \sum_{c_2 \in C_2} X_{\partial_2(c_2)}, \end{aligned} \quad (15)$$

hvor sum over $\{f_i\}$ betyr at vi summerer over alle delmengder av flater på rutenettet, og vi har brukt (7) til å skrive $X_f = X_{\partial_2 f}$ i andre og tredje likhet. Vi kan nå bruke at enhver $b \in B_1$ kan skrives på formen $b = \partial_2(c_2)$ for $c_2 \in C_2$. Vi har den inverse avbildningen $\partial_2^{-1}(b) = \{c_2 \mid \partial_2(c_2) = b \in B_1\}$, og dermed

$$\sum_{c_2 \in C_2} X_{\partial_2(c_2)} = \sum_{b \in B_1} \sum_{\substack{c_2 \in C_2 \\ \partial_2(c_2) = b}} X_b = |\ker(\partial_2)| \sum_{b \in B_1} X_b \quad (16)$$

hvor vi har brukt at $\partial_2(c_2) = b$ og at $\sum_{c_2: \partial_2(c_2) = b} = |\partial_2^{-1}(b)| = |\ker(\partial_2)|$ (fordi $k \mapsto c_2 + k$ hvor $\partial_2(c_2) = b$ er en bijeksjon mellom $\ker(\partial_2) \rightarrow \partial_2^{-1}(c_2)$). Dermed er

$$|\tilde{c}\rangle = \frac{|\ker(\partial_2)|}{2^{n/2}} \sum_{b \in B_1} X_b |z\rangle = 2^{-n/2} |\bar{z}\rangle \quad (17)$$

der n er antallet fysiske qubits i rutenettet, og $|\ker(\partial_2)| = 1$. Vi ser nå at X_f og Z_v projiserer tilstander inn i koderommet. Rollen til Z_v er å sørge

for at tilstander forblir uten endepunkter, og stabiliserer rommet som spennes ut av $|z \in Z_1\rangle$. Rollen til X_f er å sørge for at tilstander forblir en homogen superposisjon av tilstander tilhørende samme homologiklasse. På denne måten stabiliseres overflatekoden av torusens underliggende topologi.

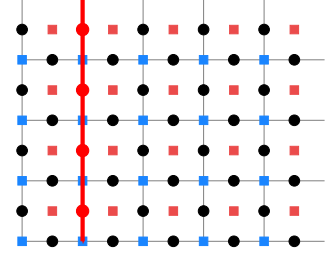


FIG. 3. Én av de to ikke-trivielle løkkene til torusen vist i rutenettet. Den andre vil strekke seg på tvers av rutenettet i horisontal retning. Her er hver verteks markert med en Z -sjekk som et blått kvadrat, og hver plakett markert med en X -sjekk som et rødt kvadrat. Qubits er representert som sorte sirkler på kantene i rutenettet.

IV. BB KODE

Bivariat Bisyklisk kode (engelsk: Bivariate Bicycle code) eller BB kode (BB code) er en kvantekode som hører til klassen av LDPC-koder, og foreslått for første gang i [4].

Idéen bak denne koden er å bruke matrisepolynomer til å spesifisere hvilke qubits som skal brukes i stabilisatoren for å sjekke feil i rutenettet.

1. BB kode – Stabilisatorer

Vi definerer S_ℓ som en sirkulær skiftmatrise (en undergruppe av sykliske permutasjonsmatriser)

$$S_\ell := \sum_{i=1}^{\ell} |i\rangle\langle i+1|, \quad (18)$$

som sender $|i+1\rangle$ til $|i\rangle$. Det kan f.eks være

$$S_3 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \quad S_4 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \quad |2\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix}. \quad (19)$$

La I_m være $m \times m$ identitetsmatrisen. Videre definerer vi skift-matriser

$$x = S_\ell \otimes I_m, \quad y = I_\ell \otimes S_m, \quad (20)$$

som tilfredsstillers $xy = yx$ og $x^\ell = y^m = I_{\ell m}$. Dersom vi kan indeksere tilstanden på rutenettet i vertikal og horisontal retning som $|c\rangle = |i, j\rangle := |i\rangle \otimes |j\rangle$ vil

$$x|i, j\rangle = |i-1, j\rangle, \quad y|i, j\rangle = |i, j-1\rangle \quad (21)$$

slik at vi kan “gå på rutenettet” i x og y -retning.

En BB kode er definert av to $(\ell m) \times (\ell m)$ matriser $A(x, y)$ og $B(x, y)$, som vil være arbitrære polynomer av x, y matrisene. De kan for eksempel se ut som

$$\begin{aligned} A(x, y) &= I + y + x^2 + y^2 + x^5, \\ B(x, y) &= I + x^3 + y^5 + y^6 + y^9. \end{aligned} \quad (22)$$

Vi bruker matrisepolynomene videre til å definere *paritetssjekk matrisene*

$$H_X := [A|B], \quad H_Z := [B^T|A^T], \quad (23)$$

som forteller oss hvilke qubits i rutenettet som skal være med i X og Z stabilisatorene. Merk at H_X og H_Z har dimensjon $(\ell m) \times (2\ell m) = (n/2) \times (n)$, der $n = 2\ell m$ er antallet fysiske qubits, og $n/2$ er antallet vertekser/flater i rutenettet.

Dermed vil hver rad i H_X og H_Z representere en verteks/flate som X/Z -stabilisatoren virker på, og hver kolonne representerer hvilke qubits i rutenettet som skal være med i stabilisator-sjekk. Dersom kolonne j , rad i har verdien 1 i $H_{X/Z}$ skal qubit $f(i, j)$ være med i X/Z -stabilisatoren, der $f(i, j)$ er en funksjon som spesifiserer indekseringen av rutenettet. Et eksempel på indeksering kan sees i Fig. 4.

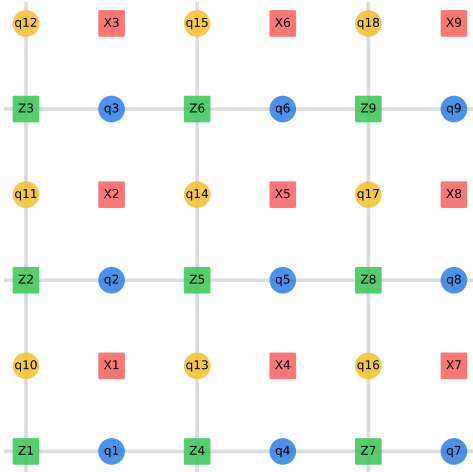


FIG. 4. Eksempel på hvordan et rutenett kan indeksere qubits og X/Z -sjekker.

På denne måten kan vi også se på H_X og H_Z som avbildninger $H_X : C_1 \rightarrow C_0$, og $H_Z^T : C_2 \rightarrow C_1$, hvor $H_X \leftrightarrow \partial_1$ og $H_Z^T \leftrightarrow \partial_2$. På samme måte som den toriske overflatekoden kan vi danne den første homologigruppen

og finne



$$\begin{aligned} k &= \dim H_1 = n - \text{rk}(H_X) - \text{rk}(H_Z) \\ &= n - 2 \cdot \text{rk}(H_X) \\ &= n - 2 \cdot (n/2 - \dim \ker(H_X)) \\ &= 2 \cdot \dim(\ker(A) \cap \ker(B)), \end{aligned} \quad (24)$$

hvor $\text{rk}(H_{X/Z})$ er bestemt av hva slags paritetssjekker vi velger å gjøre med stabilisatorene. Dermed kan H_X/H_Z lage flere ikke-trivielle løkker på torusen. For eksempel vil $A(x, y) = 1 + y$ og $B(x, y) = 1 + x$ gi tilbake den toriske overflatekoden. I BB koden gjøres ansatsen

$$\begin{aligned} A(x, y) &= A_1 + A_2 + A_3, \\ B(x, y) &= B_1 + B_2 + B_3, \end{aligned} \quad (25)$$

slik at alle stabilisatorene har vekt $w = 6$ (bruker 6 qubits per sjekk, 3 fra $A(x, y)$ og 3 fra $B(x, y)$). Vi kommer for det meste til å bruke

$$\begin{aligned} A(x, y) &= x^3 + y + y^2, \\ B(x, y) &= y^3 + x + x^2. \end{aligned} \quad (26)$$

Legg merke til hvilke qubits de to matrisepolynomene spiller sin rolle på. I sjekk-matrisen $H_X = [A|B]$ vil $A(x, y)$ virke på de $n/2$ første qubitene, mens $B(x, y)$ vil virke på de $n/2$ siste qubitene. De qubitene som $A(x, y)$ virker på kaller vi for “**L-qubits**” , og qubitene som $B(x, y)$ virker på kaller vi for “**R-qubits**” . Dersom vi indekserer rutenettet slik at alle L -qubits er horisontale kanter og alle R -qubits er vertikale kanter, vil stabilisatoren for matrisene i (26) se ut som i Fig. 5

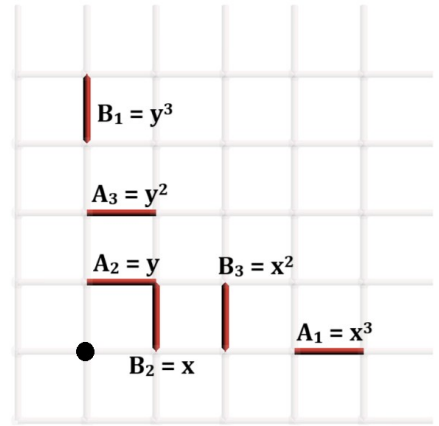


FIG. 5. X -stabilisator for $A(x, y), B(x, y)$ i (26) der L -qubits er horisontale kanter og R -qubits er vertikale kanter.

2. BB kode – Tanner graf

En annen nyttig måte å visualisere koder på er ved å bruke Tanner grafen i stedet for rutenettet direkte.

Tanken er ikke så langt ifra rutenettet – nå ser vi på qubits $\bigcirc = \{ \text{blue circle}, \text{orange circle} \}$ og X/Z -sjekker $\text{red X}, \text{green Z}$ som noder i en graf. Dersom en sjekk bruker en qubit i stabilisatoren sin, vil qubiten of sjekken være koblet sammen med en kant. I overflatekoden er hver qubit med i 2 X -sjekker, og 2 Z -sjekker.

En graf $G = (V, E)$ er et sett med noder V og et sett med kanter $E = \{(x, y) \mid x, y \in V, x \neq y\}$ som er uordnede par av noder. En Tanner graf er en todelt graf, som betyr at nodene kan separeres inn i to disjunkte og uavhengige sett U og W . Med andre ord vil enhver kant koble en node i U til en node i W , men aldri en node i U til en annen node i U (og tilsvarende for W). Her vil U være qubit noder, mens W vil være sjekk-noder. I Fig. 6 kan vi se Tanner grafen til overflatekoden.

For BB koden kan vi splitte opp Tanner grafen inn i to delgrafer $G_A = (V, E_A)$, $G_B = (V, E_B)$ der

$$\begin{aligned} \text{Tanner graf } G_A: \quad & H_X^A = [A_2 + A_3 | B_3] \\ & \text{og } H_Z^A = [B_3^T | A_2^T + A_3^T] \\ \text{Tanner graf } G_B: \quad & H_X^B = [A_1 | B_1 + B_2] \\ & \text{og } H_Z^B = [B_1^T + B_2^T | A_1^T] \end{aligned} \quad (27)$$

er sjekk-matrisene som brukes til å generere Tanner grafene. Dermed vil en node i f.eks G_A være koblet til 3 kanter som er generert av A_2, A_3 og B_3 , som vi tegner med stiplete linjer. I tillegg kan vi legge til en ekstra linje på være node, generert av B_2 og tegnet som en heltrukket linje. Resultatet blir grafen i Fig. 7. Vi kan gjøre tilsvarende med G_B der kanter generert av A_1, B_1 og B_2 blir tegnet med heltrukne linjer, og en ekstra kant generert av A_2 tegnet som stiplet linje som vist i Fig. 8.

Merk at sirkelgrafene i Fig. 7, 8 ikke lenger ligger på en torus, men nå er planare grafer i et plan. Hver sirkel er klippet ut av en stripe fargelagt som grå, og på torusen vil det være flere slike striper. Vi kan potensielt legge hver sirkel på utsiden av hverandre, og koble sirkelgrafene med kantene B_2 og A_2 for å bygge en større planar sirkelgraf.

Til sammenlikning med overflatekoden, sitter vi nå igjen med en ekstra graf for BB koden. Et annet alternativ vi har er å fokusere på grafen G_A , og få med oss de manglende kantene A_1 og B_1 ved å tegne dem inn som “langdistansekanter”. Resultatet av denne grafen for $A(x, y), B(x, y)$ gitt i (26) er vist i Fig. 9. Der kan vi se at hver sjekk bruker naboqubits, i tillegg til to langdistanse-qubits. Dermed er essensen av BB koden “overflatekode med ekstra langdistansesjekker”.

3. BB kode – Kodeimplementasjon

For full implementasjon og bruk av BB koden kan du sjekke ut min GitHub. Koden har blitt implementert i `python`, og bruker i hovvedsak pakken `panqec` (Pancake). Koden har blitt implementert ved bruk av objektorientert programmering, og lagt i et generelt rammeverk som

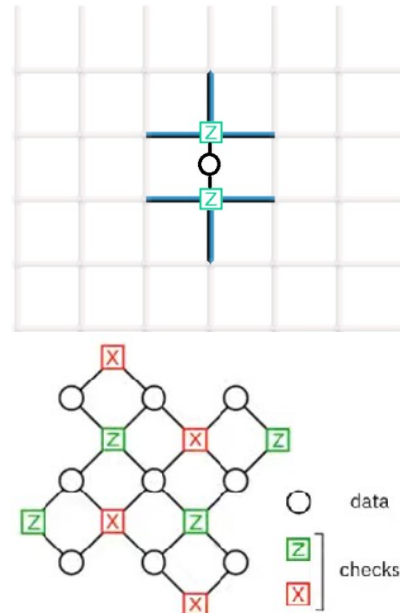


FIG. 6. Tanner graf til overflatekoden. Hver qubit er koblet til to X -sjekker, og to Z -sjekker. I øverste panel er dette visualisert i rutenettet – en qubit markert som en hvit sirkel er brukt i to Z -sjekker.

gjør at man skal kunne implementere sin egen kode uten å trenge å endre på noe annet enn hvordan man lager stabilisatorer/sjekk-matriser.

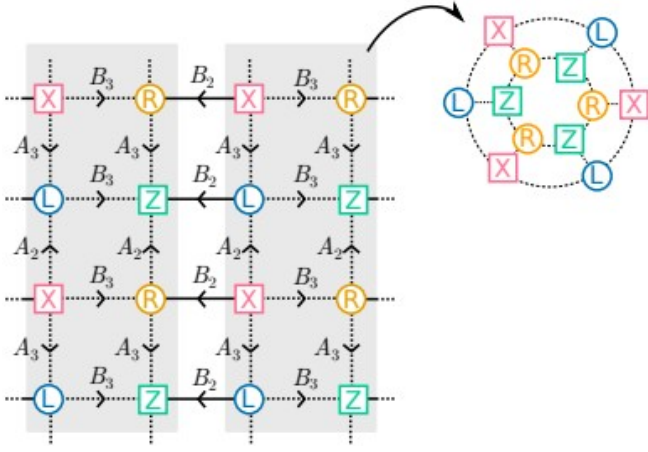


FIG. 7. Tanner graf G_A med ekstra kant generert av B_2 . Vi kan også klippe bort B_2 , og lage moduler av G_A vist som sirkelgrafen til høyre. Mens BB koden er planar på en torus, er sirkelgrafen planar i et plan (ikke-torisk).

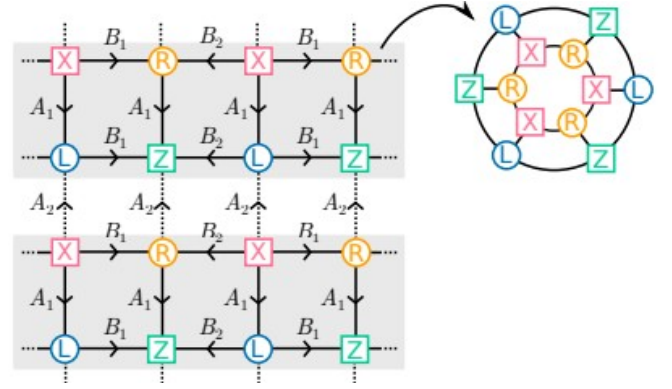


FIG. 8. Tanner graf G_B med ekstra kant generert av A_2 . Vi kan også klippe bort A_2 , og lage moduler av G_B vist som sirkelgrafen til høyre. Mens BB koden er planar på en torus, er sirkelgrafen planar i et plan (ikke-torisk).

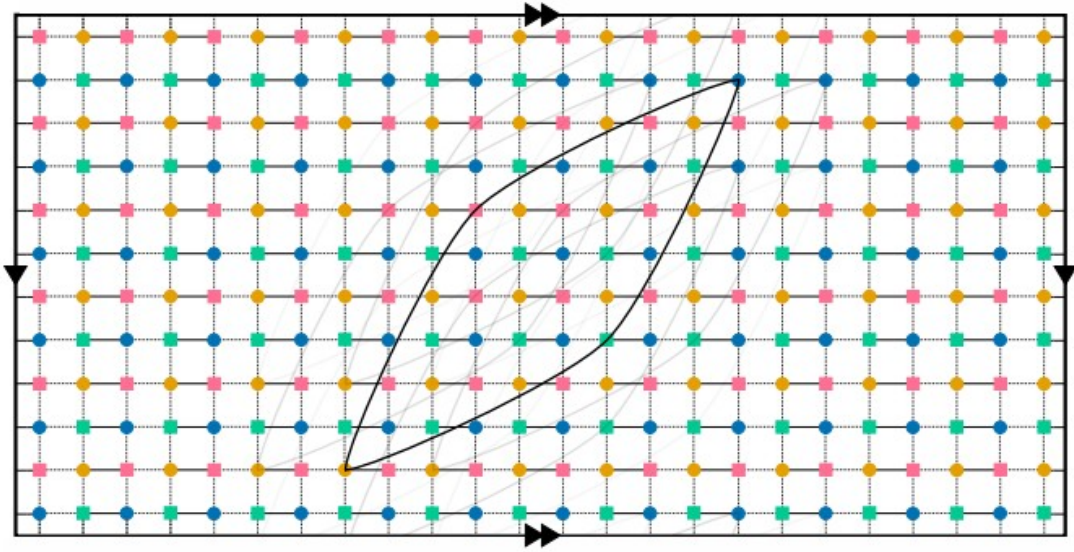
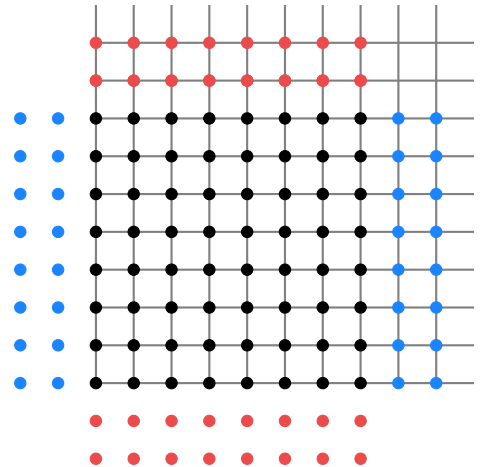


FIG. 9. Full Tanner graf for G_A generert av A_2, A_3 og B_3 i stiplede linjer, med ekstra heltrukken kant i rutenettet generert av B_2 . I tillegg er det to heltrukne langdistansekanter generert av A_1 og B_1 .

V. FLISKODER

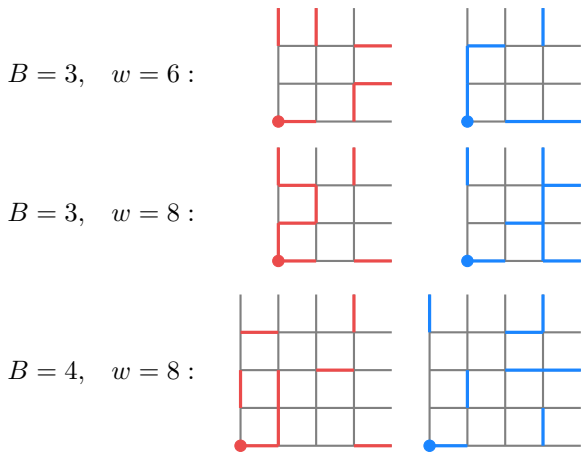
Fliskoder (engelsk: *Tile Codes*) er en type kvantekoder presentert i [5]. Oppsettet ligner den toriske overflatekoden, men overflaten har ikke periodiske grensebetingelser. Qubits er fortsatt markert med linjer i rutenettet vårt, men stabilisatorene våre markeres nå med røde (X -stabilisatorer) og blå punkter (Z -stabilisatorer). De svarte punktene i rutenettet har både en X - og Z -stabilisator.



Stabilisatorene i fliskoden er definert ved X - og Z -målinger på qubits i et lokalt område av størrelse $B \times B$, kalt fliser (engelsk: *tiles*), hvor punktet som markerer stabilisatoren i rutenettet er nederst til venstre. For eksempel ville de vanlige verteks- og plakett-stabilisatorene bli representert i en fliskode som følgende to $B = 2$ fliser:



I vår kode har vi implementert noen slike fliser, spesifikt de tre første eksemplene som gis i Tab. 1 i [5]. Vi gjengir disse her, sammen med flisdimensjonen B og sjekkvekten w (antall qubits stabilisatorene gjør en måling på. Engelsk: *check weight*).



I implementasjonen i `python` har vi navngitt fliskodene ut fra verdiene til B og w . Koden med flisene blir da kalt `TileCode_B3W6`.

VI. ANALOGE QUBITS

Vi har sett på en alternativ måte å simulere feilgenerering på, beskrevet nærmere i [6]. hvor qubits blir definert ut fra Gaussiske fordelinger av en kontinuerlig variabel q :

$$|\tilde{0}\rangle \propto \sum_{t=-\infty}^{\infty} \int e^{-2\pi\sigma^2 t^2} e^{-(q-2t\sqrt{\pi})^2/(2\sigma^2)} |q\rangle dq$$

$$|\tilde{1}\rangle \propto \sum_{t=-\infty}^{\infty} \int e^{-\pi\sigma^2 (2t+1)^2/2} e^{-(q-(2t+1)\sqrt{\pi})^2/(2\sigma^2)} |q\rangle dq.$$

Tilstandene $|\tilde{0}\rangle$ og $|\tilde{1}\rangle$ beskrives med mange Gaussiske fordelinger av den kontinuerlige verdien q , hvor fordelingene knyttet til $|\tilde{0}\rangle$ har sentrum i partalls multiplum av $\sqrt{\pi}$:

$$q = 0, \pm 2\sqrt{\pi}, \pm 4\sqrt{\pi}, \dots,$$

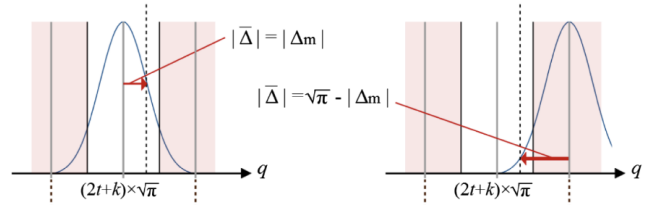


FIG. 10. Illustrasjon av hvordan vi genererer feil hos analoge qubits ved å trekke fra en Gaussisk fordeling. Hvis verdien vi trekker er nærmere nabo-fordelingen skjer det en feil på qubiten. Avstanden fra sentrum av den nærmeste Gausskurven og den målte verdien blir lagret og brukt i dekodningen til å anslå hvor mye vi kan stole på forskjellige qubits. Figuren er hentet fra [6].

og fordelingene til $|\tilde{1}\rangle$ har sentrum i oddetalls multiplum av $\sqrt{\pi}$:

$$q = \pm 1\sqrt{\pi}, \pm 3\sqrt{\pi}, \pm 5\sqrt{\pi}, \dots$$

Når vi genererer feil på disse qubitsene bruker vi forskyvninger av q vekk fra disse Gauss-sentrene. Vi “måler” q , og antar at den kommer fra nærmeste Gausskurve. Hvis q er forskjøvet fra \sin Gauss-kurve så langt at den er nærmere sentrum av nabokurven blir qubiten feil. Vi lagrer dermed avstanden Δ fra nærmeste Gauss-senter, og gir denne videre til dekoderen. Når vi dekode sitter vi da med litt ekstra informasjon om hvor mye vi “stoler” på qubiten. Høy Δ betyr at vi stoler lite på qubiten, siden det er nesten like stor sannsynlighet for at vi trakk fra nabo-Gausskurven. Lav Δ betyr at vi stoler mye på qubiten, siden det er mye mer sannsynlig at vi trakk den fra den antatte Gauss-kurven enn fra nabo-Gauss-kurven. Dette er illustrert i Fig. 10.

VII. RESULTATER

Det er mye man kan si om diverse resultater for BB-Code og TileCode, men vi begrenser oss hovedsaklig til to resultater. I resultatene presentert har vi brukt BB koden fra (26).

I Fig. 11 ser vi resultatet for BB koden simulert med X -Pauli feil generert fra den gaussiske feilmodellen, hvor vi både bruker (heltrukken) og lar være å bruke (stiplet) den analoge informasjonen fra feilgenereringen i dekodningen. Vi plotter logisk feilrate p_L – antall ganger kretsen får en logisk feil på n_{iter} ($= 500$) simuleringer – på y -aksen, og feilraten for fysiske qubits p_{phys} i rutenettet på x -aksen. Vi har simulert for en rekke ulike størrelser på rutenettet, og koden har $k = 12$ logiske qubits. Vi kan se en tydelig forbedring i den logiske feilraten ved bruk av den analoge informasjonen til dekodning, sett som et skift til høyre i plottet. Vi har også inkludert et pseudo-threshold (break-even punkt)

$$p_{\text{pseudo}} = 1 - (1 - p_{\text{phys}})^k \quad (28)$$

som et mål på når det er bedre å bruke k fysiske qubits direkte som logiske qubits i stedet for å lage en mer avansert

kode. Her er p_{phys} den fysiske feilraten – sannsynligheten for at en qubit får feil – slik at $(1 - p_{\text{phys}})^k$ er sannsynligheten for at k qubits ikke får noen feil, så $1 - (1 - p_{\text{phys}})^k$ til slutt er sannsynligheten for at minst én av de k qubitene får en feil.

Vi har i tillegg inkludert Fig. 12 der vi sammenlikner overflatekoden, fliskoden, og BB koden for $k = 8$ logiske qubits med kun X -Pauli feil i feilkanalen. I og med at overflatekoden kun har 2 logiske qubits, har vi simulert 4 overflatekoder og slått dem sammen for å kode inn 8 logiske qubits. Dette gjøres ved

$$p_{L,\text{tot}} = 1 - \prod_{i=1}^4 (1 - p_{L,i}) \quad (29)$$

der $p_{L,i}$ er den logiske feilraten for hver simulerte overflatekode, og $p_{L,\text{tot}}$ blir den totale logiske feilraten for overflatekoden. Alle kodene har samme antall fysiske

qubits $n = 360$, hvor hver overflatekode har rutenett på størrelse 9×5 slik at $n_{\text{overflate,tot}} = 4 \cdot n_{\text{overflate}} = 4 \cdot 2 \cdot 9 \cdot 5 = 360$ og $n_{\text{BB}} = n_{\text{Tile}} = 2 \cdot 15 \cdot 12 = 360$. Dermed vil plottet svare på spørsmålet: “dersom vi kun kan lage n fysiske qubits, burde vi bygge overflatekoder, fliskode, BB kode, eller fysiske qubits direkte?”. Vi anerkjenner samtidig at BB koden har en urettferdig fordel i denne sammenlikningen: med samme antallet fysiske qubits kan man velge en annen måte å organisere rutenettet på (typ. bruke 20×16 i stedet for 32×10 , eller til og med fjerne noen qubits) som kan gi BB koden flere logiske qubits.

Resultatet fra Fig. 12 viser at BB koden ligger med den laveste logiske feilraten – tett fulgt av fliskoden. Deretter krysser kodene pseudo-threshold på rundt $p_{\text{phys}} \approx 0.12$ samtidig. På en mer generell basis observerer vi at koden som gjør det best mellom BB koden og fliskoden kan variere med parameterene, men at tendensen uansett er at begge kodene utkonkurrerer overflatekoden.

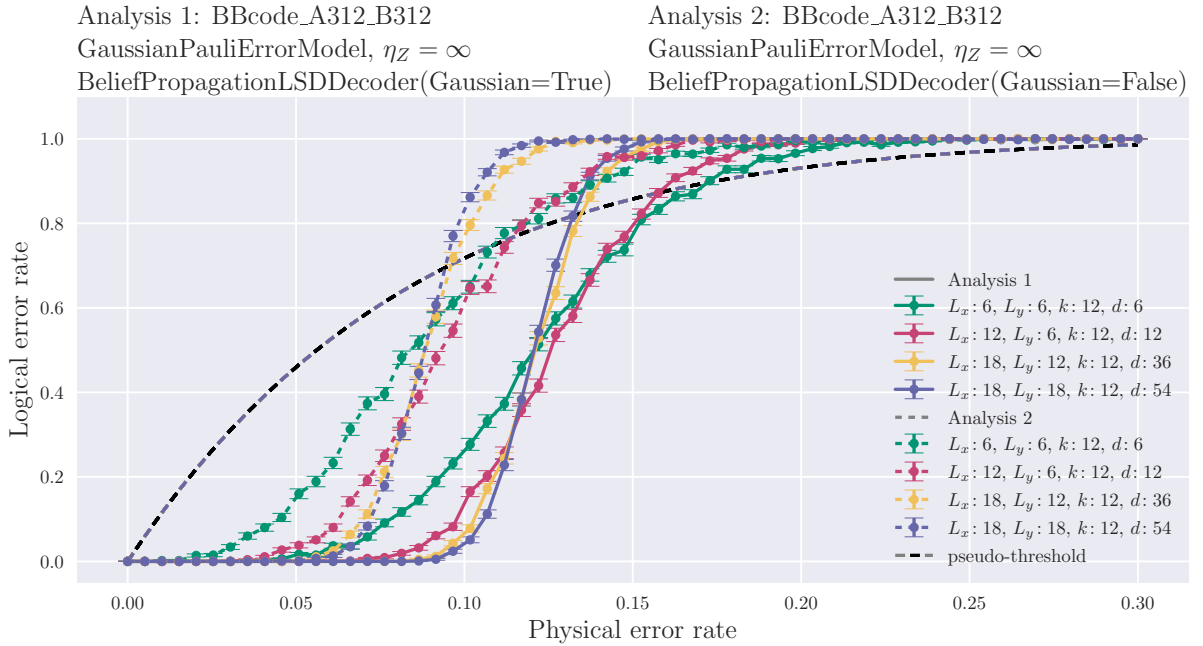


FIG. 11. Simulering av BB koden med gaussisk feilgenerering, der vi både ikke bruker (stiplet) og bruker (heltrukken) analog informasjon til å feilkorrigere. På x -aksen plotter vi feilraten for fysiske qubits, og på y -aksen plotter vi feilraten for logiske qubits. Parameteren η_Z er en bias-parameter, der $\eta_Z = r_{\text{max}}/(1 - r_{\text{max}})$, $r_{\text{max}} = \max\{r_x, r_y, r_z\}$ hvor r_x, r_y, r_z er forholdet av X, Y , og Z -feil ($r_x + r_y + r_z = 1$) på rutenettet. I dette tilfellet betyr $\eta_Z = \infty$ at $r_x = 1, r_y = r_z = 0$.

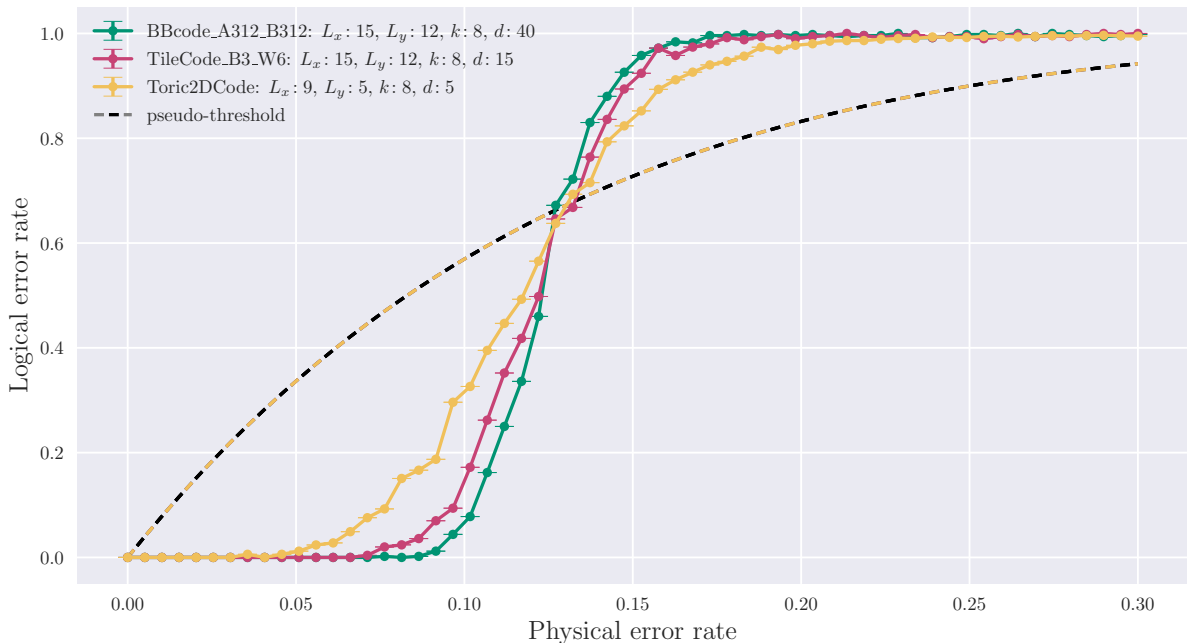


FIG. 12. Plott av både BB kode, fliskode, og overflatekode med $k = 8$ logiske qubits. På x -aksen har vi feilraten for fysiske qubits, og på y -aksen har vi feilraten for logiske qubits. Kodene krysser pseudo-threshold omtrent samtidig rundt $p_{\text{phys}} \approx 0.12$, og den logiske feilraten går raskest mot null for BB koden. I andre parameterkombinasjoner ender fliskoden opp med å gå raskere, men både fliskode og BB kode presterer bedre enn overflatekoden på generell basis.

-
- [1] E. Huang, A. Pesah, C. T. Chubb, M. Vasmer, and A. Dua, Tailoring Three-Dimensional Topological Codes for Biased Noise, *PRX Quantum* **4**, 030338 (2023), arXiv:2211.02116 [quant-ph].
 - [2] A. Chatterjee, K. Phalak, and S. Ghosh, Quantum Error Correction For Dummies, in *2023 International Conference on Quantum Computing and Engineering* (2023) arXiv:2304.08678 [quant-ph].
 - [3] H. Bombin, An Introduction to Topological Quantum Codes, (2013), arXiv:1311.0277 [quant-ph].
 - [4] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, High-threshold and low-overhead fault-tolerant quantum memory, *Nature* **627**, 778 (2024), arXiv:2308.07915 [quant-ph].
 - [5] V. Steffan, S. H. Choe, N. P. Breuckmann, F. R. F. Pereira, and J. N. Eberhardt, Tile Codes: High-Efficiency Quantum Codes on a Lattice with Boundary, (2025), arXiv:2504.09171 [quant-ph].
 - [6] K. Fukui, A. Tomita, A. Okamoto, and K. Fujii, High-Threshold Fault-Tolerant Quantum Computation with Analog Quantum Error Correction, *Phys. Rev. X* **8**, 021054 (2018).