Promatch: Extending the Reach of Real-Time Quantum Error Correction with Adaptive Predecoding

Narges Alavisamani*
Georgia Tech
USA

Suhas Vittal
Georgia Tech
USA

Ramin Ayanzadeh
Georgia Tech
USA

Poulami Das
University of Texas, Austin
USA

Moinuddin Qureshi
Georgia Tech
USA

Abstract

Fault-tolerant quantum computing relies on *Quantum Error Correction (QEC)*, which encodes logical qubits into data and parity qubits. Error decoding is the process of translating the measured parity bits into types and locations of errors. To prevent a backlog of errors, error decoding must be performed in *real-time* (i.e., within 1μs on superconducting machines). *Minimum Weight Perfect Matching (MWPM)* is an accurate decoding algorithm for surface code, and recent research has demonstrated real-time implementations of MWPM (RT-MWPM) for a distance of up to 9. Unfortunately, beyond d=9, the number of flipped parity bits in the syndrome, referred to as the Hamming weight of the syndrome, exceeds the capabilities of existing RT-MWPM decoders. In this work, our goal is to enable larger distance RT-MWPM decoders by using adaptive predecoding that converts high Hamming weight syndromes into low Hamming weight syndromes, which are accurately decoded by the RT-MWPM decoder.

An effective predecoder must balance both accuracy (as any erroneous decoding by the predecoder contributes to the overall Logical Error Rate, termed as LER) and coverage (as the predecoder must ensure that the hamming weight of the syndrome is within the capability of the final decoder). In this paper, we propose Promatch, a real-time adaptive predecoder that predecodes both simple and complex patterns using a locality-aware, greedy approach. Our approach ensures two crucial factors: 1) high accuracy in prematching flipped bits, ensuring that the decoding accuracy is not hampered by the predecoder, and 2) enough coverage adjusted based on the main decoder's capability given the time constraints. Promatch represents the first real-time decoding framework capable of decoding surface codes of distances 11 and 13, achieving an LER of 2.6×10^{-14} for distance 13. Moreover, we demonstrate that running Promatch concurrently with the recently proposed Astrea-G achieves LER equivalent to MWPM LER, 3.4×10^{-15}, for distance 13, representing the first real-time accurate decoder for up-to a distance of 13.

ACM Reference Format:
Narges Alavisamani, Suhas Vittal, Ramin Ayanzadeh, Poulami Das, and Moinuddin Qureshi. 2024. Promatch: Extending the Reach of Real-Time Quantum Error Correction with Adaptive Predecoding . In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3620666.3651339

1 Introduction

The inherent noisiness of quantum computers limits the execution of many promising applications in quantum chemistry, cryptanalysis, and machine learning [3, 14, 25, 33, 34, 37, 42, 47, 50, 54, 68]. Furthermore, many of these applications require extremely low error rates ($< 10^{-12}$) unlikely to be achieved on physical devices. *Quantum Error Correction (QEC)* can enable these applications by forming fault-tolerant *logical qubits* from multiple physical qubits [8, 22, 23, 36, 41, 53, 56]. These logical qubits have lower error rates than their constituent physical qubits, and by increasing the level of redundancy, or *code distance* (d), the logical error rate (LER) can be reduced. Logical qubits are encoded using a combination of *data* qubits, which encode a quantum state, and *parity* qubits, which detect errors on the data qubits [8, 22, 23, 36, 41, 53, 56]. To identify any errors on the data qubits, *Fault-Tolerant Quantum Computers (FTQCs)* that use Quantum Error Correction (QEC) periodically execute syndrome extraction. This process involves executing a quantum circuit that measures the parity qubits. A measurement result of '1' indicates a parity-check failure. The parity qubit measurements are then compiled into a bitstring known as a *syndrome*, which is subsequently sent to a classical *decoder*.

*The corresponding author can be reached at narges.alavisamani@gatech.edu.

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA

ACM ISBN 979-8-4007-0386-7/24/04…$15.00
https://doi.org/10.1145/3620666.3651339

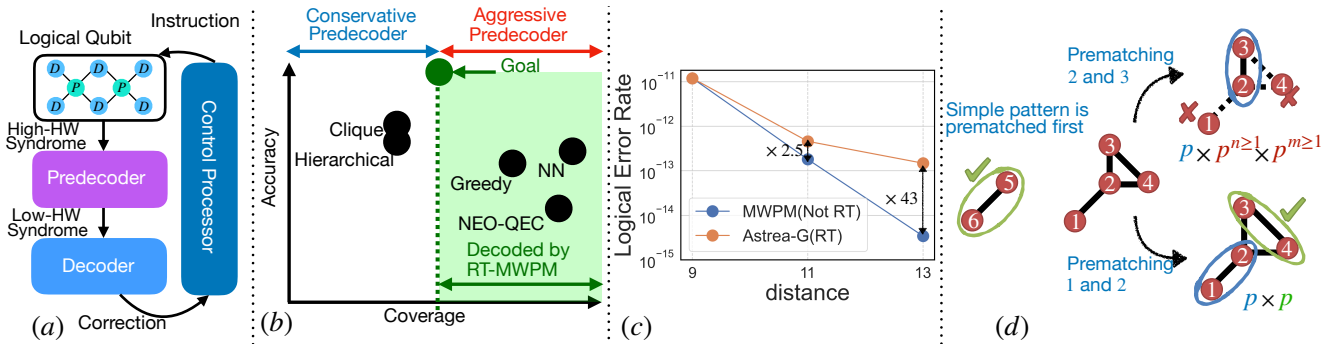

Figure 1. (a) The overview of QEC process in the presence of a predecoder. (b) The tradeoff between accuracy and coverage for Hierarchical predecoer [20], Clique [49], and the greedy predecoder [55]. Existing predecoders either have low accuracy or low coverage. (c) The gap between current RT-MWPM and Non-RT MWPM decoders indicating a 43x higher LER (d) An overview of insights using in Promatch for the locality-aware greedy algorithm. Promatch checks the neighborhood of flipped parity bits and finds the correct matching (nodes 1 and 2), which enables additional correct prematching (nodes 3 and 4). These two prematchings result in a lower weight, having a higher probability (P^2 compared to at most P^3).

The decoder analyzes the syndrome to identify where errors have occurred on the logical qubit and computes a correction, which is sent to the control processor to update future operations. However, as the syndrome extraction operations are themselves faulty, decoders must analyze syndromes across d rounds to accurately determine the location of errors.

In recent years, several software-based implementations of MWPM have been proposed based on Blossom algorithm [29, 30, 38, 69] aiming to accelerate decoding and reduce average decoding time. However, they remain impractical due to the complexity of Blossom algorithm, which fails to guarantee 1μs window. To achieve the required decoding speed, the decoding algorithms need to be implementable on hardware, such as FPGA, which serves as the primary focus of this work. Astrea [66], the state-of-the-art real-time MWPM (RT-MWPM) decoder, proposes using a brute-force method for decoding syndromes that has up to 10 flipped parity bits. Number of flipped parity bits in the syndrome is referred to as *Hamming weight (HW)* of the syndrome. Astrea is limited to HW$\leq$ 10 due to the exponential growth of number of matchings when HW increases. This fast growth makes applying brute-force method impossible for $d > 7$. To address this challenge, a *predecoding* method can be applied to predecode syndromes, accordingly reduce the HW of syndromes, and then send them to the main decoder, as shown in Figure 1(a). To have an effective predecoding process, the predecoder needs to 1) be accurate, not limiting the accuracy of the decoder 2) sufficiently cover enough number of flipped bits, making sure that the rest is manageable by the decoder. The goal of this paper is to develop a larger distance RT-MWPM using an adaptive predecoder that is accurate and has enough coverage.

In the context of predecoding methods, accuracy refers to the correctness of the predecoding process in matching flipped bits before they are passed to the main decoder. If predecoding is inaccurate, it results in erroneous decoding even if the main decoder is capable of accurately handling the remaining bits. Consequently, achieving high accuracy in predecoding is essential to ensure the reliability of the entire error correction process. In addition to accuracy, coverage is another important factor for predecoders, which refers to the number of pairs of flipped bits that are predecoded compared to the total number of pairs required to be decoded. Insufficient coverage means that too many error pairs are left for the main decoder which cannot be handled within the time constraints. On the other hand, excessively high coverage underutilizes the main decoder's capability and may potentially compromise accuracy.

Prior predecoders tend to prioritize either coverage or accuracy [20, 49, 55], as shown in Figure 1(b). For example, Clique [49] and Hierarchical decoder [20] which uses a simple predecoding structures for simple patterns and using MWPM for complex patterns. Therefore, these predecoders do not reduce the complexity requirement of the main decoder, and they are still reliant on having a main decoder that can decode high Hamming weight syndromes, which is impractical to perform in real-time. Furthermore, any inaccuracy of the predecoder still contributes to the overall logical error-rate. On the other hand, Smith et al.'s work [55] proposes a Greedy predecoder that has high coverage but low accuracy. Therefore, the overall logical error rate of the combined decoder is more than two orders of magnitude higher than the MWPM decoder alone. Another work Astrea-G [66] adopts a greedy method that matches bit flips until the main decoder can handle the remaining decoding, guaranteeing sufficient coverage, which leads to similar LER as MWPM LER for distance 9 but reduced accuracy for higher distances (43× higher LER at $d = 13$), as shown in Figure 1(c). The tradeoff between accuracy and coverage makes it challenging to achieve an ideal solution. Low coverage may lead to

the main decoder being unable to accurately decode the remaining bits. Conversely, high coverage might utilize the main decoder's capacity inefficiently, but could come at the cost of reduced overall accuracy due to incorrect matches. Promatch strikes the right balance between accuracy and coverage ensuring precise decoding and efficient utilization of the main decoder's capabilities.

Promatch utilizes a locality-aware greedy algorithm, carefully considering the consequences of each decision made during the predecoding process. It matches and removes the flipped bits from the syndrome until the main decoder can find the exact MWPM solution for the remaining flipped bits before reaching $1\mu s$. In some cases, a group of flipped bits may form a complex pattern that is not immediately apparent how to match efficiently. However, Promatch utilizes the insight that by making *the right initial matching* within a complex pattern, it can be broken down into simpler, more manageable patterns, as shown in Figure 1(d).

Promatch can be added to any decoder, however for our evaluations and getting low enough LER, we use Astrea [66]. Our design incorporates Astrea's exact MWPM solution for low-HW syndromes and employs an adaptive matching approach to predecode flipped bits of the syndrome till reduce the HW to a value that allows us to apply Astrea within the remaining time before reaching $1\mu s$. Promatch decodes surface codes of $d = 11$ and $d = 13$ in real time with LER of 4.5×10^{-13}, and 2.6×10^{-14}, respectively. The LER gained by Promatch, is the lowest LER in the literature that a real-time decoding process has achieved for distance 13. Further, we observed the gap between these LER values and MWPM LER can be closed by running Astrea-G [66] in parallel with Promatch. Thus, our combined proposal represents the first decoder proposal that is both accurate (similar to MWPM) and real-time for up-to distance of 13.

Overall, this paper makes the following contribution:

- *Accurate Predecoder*: Promatch introduces a locality-aware greedy method that utilizes information from each flipped bit's neighborhood to provide an adequate number of accurate matching while avoiding inaccurate patterns.
- *Adaptive Predecoder:* Promatch is an adaptive predecoder which increases the complexity of patterns that are pre-matched based on whether the resulting hamming weight is within the capability of the main decoder.
- *Real-time Decoding for Large Distances:* Promatch is the first real-time decoder capable of decoding surface codes of distance 11 and 13 with the highest accuracy in the literature.
- *Achieving MWPM LER for Large Distances:* When run concurrently with Astrea-G, Promatch makes it possible to achieve MWPM LER for up to $d = 13$ for the first time.

2 Background and Motivation

2.1 Quantum Error Correction Using Surface Codes

Surface codes are regarded as the most promising QEC code due to their high threshold (about 1%) and grid structure. Surface codes encode a logical qubit of distance d onto a lattice with d^2 data qubits and $d^2 - 1$ parity qubits [22, 23, 36, 59], as shown in Figure 2(a). Errors on data qubits are projected into Pauli errors on their adjacent parity qubits through the periodic measurement of these parity qubits. The output of this parity qubit measurement is called a syndrome, and the measurement process is known as syndrome extraction. During this process, each parity qubit measures a four-qubit operator, called a stabilizer, involving its four neighboring data qubits and extracts information about errors on them. Surface codes use two types of stabilizers (Z and X) to detect bit-flip (X) and phase-flip (Z) errors, respectively. Errors lead to failed parity checks producing *non-zero* syndromes. The code distance is the measure of the redundancy of the code as well as its error-correcting capability. A distance d code can correct all error chains of at most length $\lfloor \frac{d-1}{2} \rfloor$.

2.2 Error Decoding

QEC uses decoders that analyze the syndromes to identify the location and type of errors by *matching* or *pairing* the non-zero syndrome bits or failed parity checks. The problem can be reduced to a matching problem on a two-dimensional graph, known as the *decoding graph*, where each node denotes a parity qubit and each edge denotes a data qubit. The pairing step matches the non-zero nodes and assigns errors onto the data qubits corresponding to the edges connecting them, as illustrated in Figure 2(b). However, in reality, syndromes are imperfect due to gate and measurement errors that occur during syndrome extraction. These errors result in failed parity checks across consecutive syndrome extraction rounds. To tolerate these errors, the decoder analyzes d consecutive syndromes, resulting in a matching problem on a three-dimensional graph, as shown in Figure 2(b). Decoders must accurately identify errors in real-time to prevent the accumulation of errors. The typical latency that must be met is about $1\mu s$ on superconducting systems (which corresponds to the time it takes to extract a syndrome).

2.3 Minimum Weight Perfect Matching (MWPM)

The MWPM decoder is widely regarded as the gold standard for decoding surface codes. Each edge on the decoding graph is associated with a *weight* that denotes the *probability of error* causing the two nodes connecting the edge to flip. Therefore, constructing a fully-connected weighted graph comprising of the non-zero syndrome bits and *perfectly matching* the nodes such that the *total weight is minimized* enables us to determine the highest probability error event. Although

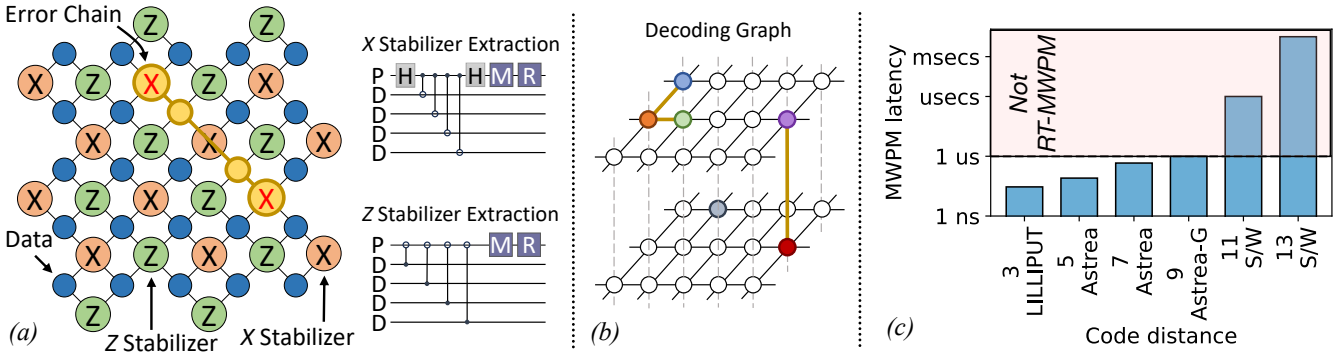

Figure 2. (a) Illustration of a $d = 5$ surface code logical qubit lattice, alongside the associated Z and X stabilizer extraction circuits. (b) Two-round decoding graph example for the X stabilizer. (c) Real-time decoders, under $1\mu s$, exist for up to distance 9 while, for higher distances, 11 and 13, we need to rely on software-based decoders which have high latency.

the accuracy of MWPM is desirable, implementing it in real-time is challenging owing to the complexities of the inherent Blossom algorithm used to compute the MWPM.

Recent works solve real-time MWPM (RT-MWPM) decoding using alternate approaches. As shown in Figure 2(c), LILLIPUT achieves RT-MWPM in 29 ns and 42 ns for d=3 and d=5 (for only two syndrome rounds) respectively using lookup tables [17, 59]. However, the size of the tables grows exponentially with the distance, limiting its scalability. Astrea achieves RT-MWPM up to d=7 within 456ns [66]. As each error chain (irrespective of its length) only flips up to two syndrome bits, the *Hamming weight* of the syndromes corresponding to correctable errors remain within 10 for d=7. The number of possible matchings for syndromes of Hamming weight 10 is 945. Astrea searches through them using an accelerated brute-force search in hardware. However, brute-force search is not scalable for larger code distances as the Hamming weight increases due to the increased redundancies. Astrea-G extends Astrea by searching greedily and prioritizing certain matchings [66]. It achieves RT-MWPM in $1\mu s$ for up to d=9. However, this greediness causes inaccuracies beyond d=9 and the logical error rate of Astrea-G is higher than the idealized MWPM, for example 43× for 13. Beyond d=9, we must rely on software implementations [28, 31, 39] to achieve MWPM accuracy or approximate solutions (such as AFS decoder) [18, 32, 35, 52, 61, 63] that trade-off accuracy for real-time decoding. Although a recent variant of the *Blossom* algorithm has significantly lower complexity compared to the original implementation, the worst-case latencies are still a few hundred microseconds to milliseconds. *Ideally, we want to expand the reach of RT-MWPM beyond d=9.*

2.4 Prior Works on Predecoding

The decoding complexity grows with the code distance due to the increased number of syndrome bit flips in the larger decoding graph. High Hamming weight syndromes are more complex and take longer to decode due to the increase in the number of possible pairings. The complexity of decoding can be reduced by *prematching* or *predecoding* a subset of the non-zero syndrome bits. The first proposals for predecoding focused on reducing syndrome transmission bandwidth between the quantum substrate and the decoders [20, 49]. These implementations attempt to quickly decode the entire syndrome by matching non-zero syndrome bits to their geometrically local neighbors. If the predecoder is able to do so, no information is sent to the main decoder (often an MWPM or Union-Find decoder). We refer to these predecoders as *Non-Syndrome-Modified (NSM)* predecoders because they do not modify the syndrome before sending it to the main decoder, as shown in Figure 3(a).

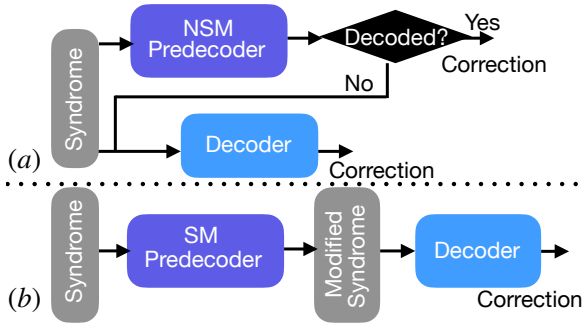

Figure 3. (a) NSM Predecoders attempt to decode the entire syndrome. If they fail, then the entire syndrome is sent to the main decoder. (b) SM Predecoders decode a subset of the syndrome and send the remainder to the main decoder.

More recent predecoders use an orthogonal approach that focuses on minimizing the complexity of the decoding task handled by the main decoder [9, 11, 55, 62]. These *Syndrome-Modified (SM)* predecoders reduce the Hamming weight of the syndromes by matching a subset of the syndrome bits and sending the remaining unmatched syndrome with a lower Hamming weight to the main decoder, as illustrated in Figure 3(b).

2.5 Limitations of Prior Works on Predecoding

The limitation of NSM predecoders is that these predecoders do not reduce the decoding complexity of the main decoder and therefore, the overall decoding performance is still constrained by the main decoder. If the main decoder is a software MWPM decoder, real-time decoding is not feasible [49], whereas the accuracy is reduced if the main decoder is a Union-Find decoder [20]. Concurrently, SM predecoders are limited by their accuracy: such predecoders may err when matching, and this error will cause a logical error[1].

Thus, we observe a tradeoff between predecoder *accuracy* and *coverage*. NSM predecoders avoid predecoding syndromes with potential non-local matchings, resulting in subpar coverage. In contrast, Syndrome-Modification predecoders achieve good coverage by reducing the Hamming weight of the syndrome, but may incur inaccuracy while doing so. We further note that coverage is strongly correlated to *latency*, and there is an inherent trade-off between accuracy and decoding latency. A predecoder that optimizes for accuracy will avoid reducing the Hamming weight too significantly so the main decoder decodes the majority of the syndrome. However, this increases overall decoding latency. For example, the Clique predecoder is limited by the latency of software MWPM [49]. On the other hand, a predecoder that optimizes for coverage may predecode too much of the syndrome to relieve the burden of the main decoder. This approach reduces decoding latency but also reduces accuracy. This phenomenon is observed in the predecoders proposed by Chamberland et al. and Smith et al. [11, 55]. Unfortunately, prior predecoders optimize for either higher accuracy or coverage. Note that it is possible to have very accurate and high coverage predecoding using complex algorithms such as belief propagation but incur long latencies that may not converge in real-time [9].

2.6 Goal: Enabling Higher Distance RT-MWPM Decoders by Using Adaptive SM Predecoding

Currently, there is a gap between RT decoders and Non-RT MWPM for $d > 9$, 2.5 times and 43 times higher than logical error rate of Non-RT MWPM, as illustrated in Figure 4. Notably, a contemporary RT MWPM decoder, named Astrea, can decode syndromes that have low Hamming weight. Motivated by this capability, our objective is to construct a SM predecoder that can precisely predecode and reduce high Hamming weight syndromes. This reduction enables the RT MWPM decoder to process the modified syndromes effectively. Our envisioned SM predecoder operates adaptively, predecoding syndromes to achieve a level of *sufficient* coverage based on the present abilities of the RT MWPM decoder. With this approach, we aim to design an accurate RT predecoder that, when combined with Astrea, can bridge the existing gap between RT and Non-RT MWPM decoders.

[1]We note that the accuracy of any predecoder, both NSM and SM, is crucial as any inaccuracies will cause a logical error.

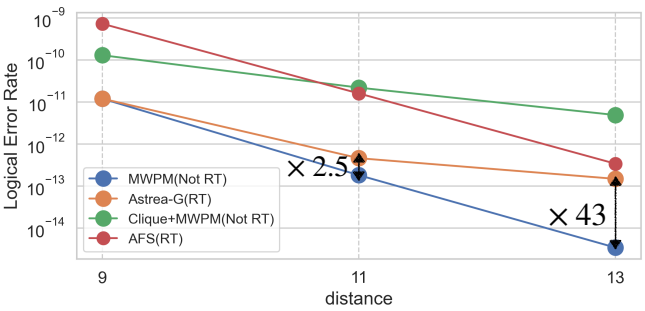

Figure 4. Logical error rate trends for MWPM, Astrea-G, Clique+MWPM, and AFS as code distance d increases, considering a physical error rate of 10^{-4}.

3 Promatch: Key Insights

Promatch focuses on decoding syndromes that have more than 10 flipped bits (HW > 10), which we refer to as high Hamming weight syndromes as Astrea can accurately decode all syndromes with HW ≤ 10 in real-time. Like many other predecoding approaches, the core idea of this work revolves around the observation that most of the flipped bits in the syndrome are matched to their neighbors in the decoding graph (indicating a chain of length one).

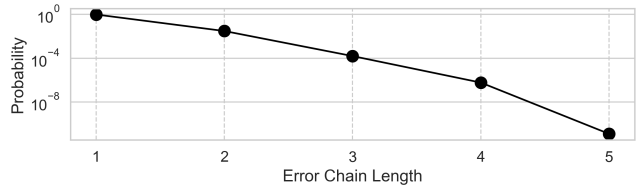

Figure 5. More than 90% of error chains, based on MWPM decoder, has length of 1. This means more than 90% of flipped bits are matched to their neighbors. This plot is for distance 13 and physical error rate 10^{-4}.

Figure 5 shows the frequency of different error chain lengths for the high Hamming weight syndromes. As error chains of length 1 are extremely common, most predecoders attempt to remove such errors [20, 55]. However, only greedily predecoding errors via error chains of length 1 leads to a loss in accuracy. In this section, we present our insights regarding how incorrect decisions can be avoided during predecoding. In our insights, we leverage the *decoding subgraph*, which is the subgraph of the larger decoding graph containing only the nonzero bits in a syndrome bit and any edges between them, as shown in Figure 6.

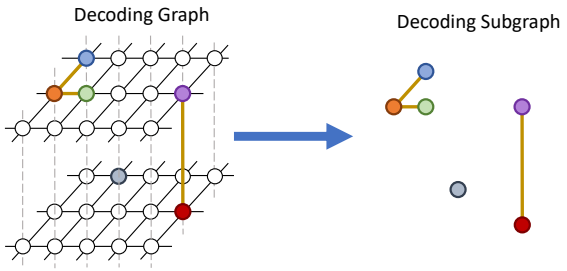

Figure 6. Decoding subgraph created from flipped parity bits and edges of which both nodes (parity bits) are flipped.

3.1 High Accuracy: Singletons contribute more weights to the MWPM solution

During predecoding, every prematching removes two nonzero syndrome bits from the decoding subgraph, consequently removing any edges incident to these syndrome bits. As edges are removed from the graph, this can result in syndrome bits unconnected to the rest of the decoding subgraph. We call these unconnected bits *singletons*. As these singletons are not connected to any other syndrome bit, the only way to match such bits to other syndrome bits is by correcting along an error chain with length $L \geq 2$. However, such error chains are unlikely, as their probability of occurrence is about p^{-L}. Thus, our first insight is that *predecoding should minimize the number of singletons created during prematching.*

In Figure 7, we illustrate the structure of four connected flipped bits observed in the simulations. In this scenario, we have six possible options for matching, and the correct choice is to match 1 with 2 and 3 with 4. However, if we make a mistake and match 2 with 3, in the next step, we will have two singletons of 1 and 4, and we will have no other option but to match them. This incorrect matching solution results in a total weight of approximately 12, whereas the correct match has a weight of around 8. It is important to note that the correct match is the only one that does not generate any new singletons, making it the optimal choice.

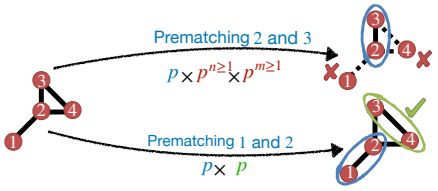

Figure 7. Avoiding generation of singletons results in lower-cost matchings in future steps. Additionally, a correct matching (nodes 1 and 2) enables additional correct prematching (nodes 3 and 4).

3.2 Sufficient Coverage: Use multiple simple steps

Prior predecoders treat prematching as a monolithic process: the predecoder prematches once, and the remainder of the syndrome, regardless of the remaining Hamming weight, is sent to the main decoder. However, we observe that prematching decisions often reveal new possible prematchings by removing nodes and edges from the decoding subgraph and consequently reducing the complexity of the decoding subgraph. Thus, our second insight is that *prematching decisions enable additional prematching decisions* [2]

As shown in Figure 7, the structure of four connected flipped bits may appear complex at first glance. However, by following the key insight of not generating new singletons, we can make the optimal matches. For instance, we match 1 with 2, leaving us with the simple match of 3 with 4 as a standalone pair. In this way, we can successfully match an error structure of four flipped bits in just two consecutive simple steps. This demonstrates the effectiveness of our approach in handling complex error patterns while minimizing the adverse impact of matching decisions.

4 Promatch Design

This section explains the algorithm and hardware design of Promatch. Figure 8 shows an overview of Promatch. Promatch is designed in such a way that it initiates the matching process with the least risky pairs. If necessary, it incrementally adjusts the risk level until a sufficient coverage is achieved. It leverages the insight that initial matching of complex patterns into simpler ones is crucial. Specifically, during stages 2, 3, and 4, which tackle complex patterns, Promatch matches one pair at a time before reassessing. This approach can simplify patterns, allowing earlier stages to address them. For example, in Figure 7, prematching bits 1 and 2 breaks the complex pattern which results in trivially matching 3 and 4. After each matching that Promatch applies, it checks if the main decoder can decode the modified syndrome in the remaining time. If yes, it gives the syndrome to the main decoder. If not, Promatch predecodes more bits.

4.1 Promatch Algorithm

At each round of Promatch, we extract the data of the decoding subgraph. Each flipped bit that has not yet been matched in the syndrome acts as a node in the subgraph. We gather the following data for each node of the subgraph at each round of the algorithm: 1) The degree of the node. For node i, this is denoted by deg_i. 2) For each node, the number of neighbors possessing degree 1. For node i, this number is denoted by $\#dependent_i$. $\#dependent_i$ indicates the number

[2] We assume independent errors (no spatial correlation). If multiple errors happen far away from each other then such isolated errors can be matched easily, and decoding becomes an easy problem. However, the harder problem for decoding is when multiple errors happen to occur near to each other and this occurs with non-negligible probability even with random errors. Handling such patterns is a requirement to achieve low logical error rate (for example, if such patterns happen with a 1 in a billion probability and the decoder is unable to handle, then the LER will always be at-least 10^{-9}, whereas we seek LER of as low as 10^{-15}), so we need to handle such patterns.

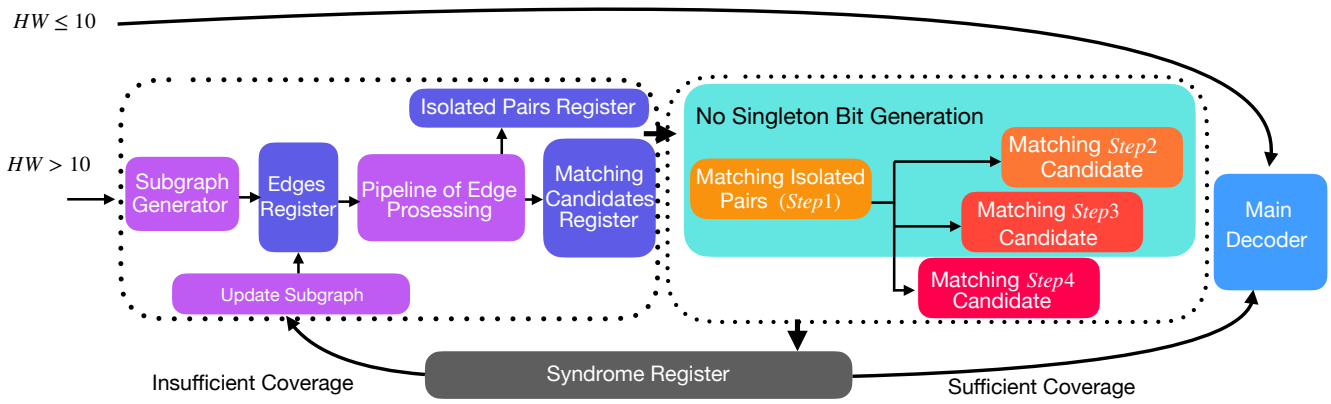

Figure 8. Overview of Promatch

of flipped bits in the syndrome that rely exclusively on node i for a match with an error chain of length 1. If these neighboring bits do not get matched with node i, they become singletons. For instance, in Figure 9, flipped bit a has four neighbors, b, c, d, and e. Three out of these four neighbors have a degree of 1, namely b, c, and d. This means that if these three nodes intend to form a match in an error chain of length 1, bit a is their sole option. For example, if a and b get matched and removed from the syndrome, the other *degree-1* neighbors, c and d, will have no neighbors in their neighborhood with only 1 edge and will become singleton bits. However, e still has f in its neighborhood, preventing it from becoming a singleton.

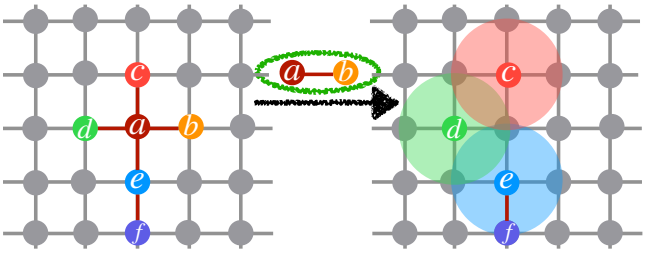

Figure 9. There are three nodes (b, c, and d) in the decoding subgraph, whose only neighbor is node a. If a and b get matched with each other, it creates singleton bits c and d (while e pairs with f).

After updating all the $\#dependent_i$ and deg_i values for all nodes, Promatch starts finding matchings with prioritizing pairs that: (a) do not generate singletons after matching, and (b) have the highest probability, meaning they minimally increase the overall weight in the MWPM solution. To ensure high accuracy, Promatch begins by matching *isolated pairs*—simple patterns of flipped bits with only one neighbor, which are more common and less risky. To achieve sufficient coverage, it also engages with complex patterns, involving flipped bits that have multiple neighbors and, consequently, multiple matching options. These patterns can also contain extant singletons, which are complicated by their lack of adjacent matching options. Despite these complex patterns being riskier and less frequent, Promatch dedicates part of its design to these patterns to ensure enough coverage. In the following parts, we describe the steps of Promatch, ordering them by their priorities. In addition, we explain the reasons behind their prioritization order. These steps are also elaborated in Algorithm 1.

1. Promatch begins by matching isolated pairs primarily for two key reasons: First, based on our insight about how singletons can increase the weight of the MWPM solution, Promatch prioritizes matching isolated pairs because matching them does not create singletons and accordingly ensures that the decoding process remains efficient by not introducing higher weights. Second, if any of the flipped bits among the two of bits in an isolated pairs gets matched to another node, the other node becomes a singleton bit. Therefore, to prevent either of flipped bits in an isolated pair from becoming a singleton bit, we require to match flipped bits of an isolated pairs with each other.
2. In this step, Promatch matches two neighboring flipped parity bits, that creates a non-isolated pairs, only if the matching does not generate any singleton bits.
   2.1. First, Promatch prioritizes a pairs of neighboring nodes with the lowest weight in the decoding graph (highest probability). In this step, it prioritizes the pair of which one of the nodes have degree 1. Similar to step 1, it is important to prioritizing matching such pair because for one of the bits, this matching is the only option that prevents it from becoming a singleton bit.
   2.2. Second, if there is no pair of which the degree of the nodes is 1, it chooses the pair that has the lowest weight in the decoding graph (highest probability).
3. Promatch employs this particular step only when there are no viable matching candidates left for step 2. This

situation arises when pairs of neighboring flipped bits cannot be matched without leading to the creation of a singleton bit. In such cases, Promatch opts to match an extant singleton bit with another flipped bit, choosing the one that forms the shortest path or, equivalently, the error chain with the highest probability. In this step, the condition of not creating singleton bit is still necessary. it is important to note that, Promatch requires to search among a fewer number of paths, compared to Astrea's brute-force method. This is because the number of singleton bits is low. Additionally, this step is utilized only after Step 1 and 2 have been applied and if Step 1 and Step 2 have not been enough to reach the sufficient coverage. Therefore, there are a few number of paths left, which makes this step fast enough.

4. The algorithm uses this step only if no flipped bit can be matched in Step 1, Step 2, and Step 3. This step is similar to Step 2, having similar substeps 4.1 and 4.2, without the condition of not generating singleton bits, and is the only one that adds singleton bits to the decoding subgraph. Therefore, in this step, Promatch takes the riskiest decision to ensure that the main decoder can decode the modified syndrome within remaining time before reaching $1\mu s$. Note that, similar to Step 2 and Step 3, Promatch may break complex patterns to simple patterns allowing less risky decisions in future.

4.2 Hardware Implementation of Promatch

Promatch iterates over the edges of decoding subgraph. The weights of the edges of the decoding graph are stored in a Edge Table in an on-chip memory on the FPGA. To avoid impacting memory latency on the decoding time, the data is loaded from the memory gradually while the syndrome is extracted and previous syndrome is being decoded. Note that Promatch's locality-aware approach makes loading all the required data during the syndrome extraction possible, which is not the case for exhaustive search approaches. There are four steps in Promatch algorithm design, detailed in Section 4.1. Each of these steps has its own matching candidates. The candidates of Step 1, isolated pairs, are stored in a separate register from the rest of the candidates because this step can have multiple candidates. This setup allows all these candidates to be applied to the syndrome simultaneously, a notable difference from other steps that match and modify the syndrome based on a single pair at a time. The candidates of other steps are stored in Matching Candidate Register.

4.2.1 Implementation of the Decoding Subgraph. The decoding subgraph information is stored in this format: A vertex array is utilized to store the index of the flipped parity bits in the syndrome. A neighbor array is also utilized to specify the neighbors of each vertex. Each neighbor contains the weight of the edge connecting it to the vertex. Additionally, each vertex i has two vertex property array, namely the degree array and the *dependency* array. which contains deg_i and $\#dependent_i$, respectively.

Algorithm 1 Promatch Algorithm

```
Input: Decoding Subgraph (V, E)
while HW is not low enough do
    while isolated pairs exist and HW is not low enoug
do
        match isolated pairs
    if HW is low enough then break
    for e_ij in E
        if matching i and j does not create Singleton
          if min(deg_i, deg_j) == 1
            if w_ij < weight of current Step2.1 candidate
              then set (i, j) as Step2.1 candidate
          else if w_ij < weight of current Step2.2 candi-
date
              then set (i, j) as Step2.2 candidate
        else                                   ▷ Risky step
          if min(deg_i, deg_j) == 1
            if w_ij < weight of current Step4.1 candidate
              then set (i, j) as Step4.2 candidate
          else if w_ij < weight of current Step4.2 candi-
date
              then set (i, j) as Step4.2 candidate
        end if
    end for
    if Step2.1 and Step2.2 candidate is empty
    and ∃ Singleton ∈ V
      for every node i and Singleton node j in V
        if matching i and j does not create new Singleton
          and path weight of i and j is less than path
            weight of Step3 candidate
              then set (i, j) as Step3 candidate
        end if
      end for
    end if
    Match only 1 pair among the candidates prioritizing
    Step2.1, Step2.2, Step3, Step4.1, and then Step4.2.
end while
```

4.2.2 The Pipeline of Finding Matching Candidates For Each Step. Figure 10 depicts the pipeline stages for identifying matching candidates within the decoding subgraph. The first stage examines each edge's node degrees, signaling which of the four steps in Promatch could consider the edge as a matching candidate. The subsequent stage evaluates if pairing through this edge would result in a singleton bit; if it does not, the edge is flagged as a potential candidate for Step 2 of Promatch. The third stage determines the appropriate candidate register for updates based on this edge. Lastly, it compares this edge's weights with the existing candidates,

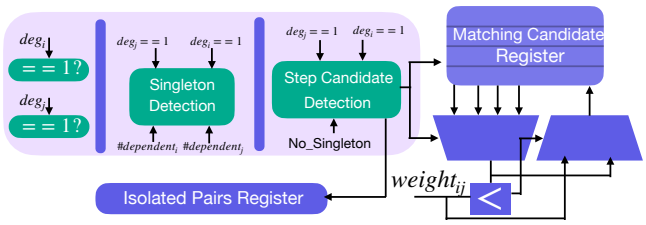

Figure 10. Pipeline of iterating over the edges of subgraph to find matching candidates for Promatch.

updating the candidate register if this edge offers a lower weight (higher probability match).

Figure 11 shows the simple logics for singleton detection and step candidate detection, which makes Promatch fast enough compatible with the time limitation that the decoding process has.

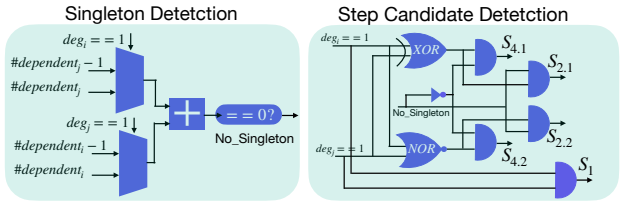

Figure 11. Logic for Singleton Selection and Step Selection of Promatch's Algorithm.

In Step 3, Promatch calculates the lowest-weight paths between existing singleton bits and the remaining flipped bits. It uses an $n \times n$ weight table, with each cell containing 8 bits, to store path weights among all n syndrome bits. This table, kept in on-chip memory, is processed in parallel to and independently from the main pipeline of Figure 10.

4.2.3 Promatch || Astrea-G. Promatch demonstrates a high proficiency in decoding a range of syndrome patterns, from simple to complex, prioritizing simpler matchings of isolated pairs or nodes with a single degree. This approach, however, faces challenges with more intricate error structures, especially those involving multiple distinct components located comparably close to each other, resulting in having multiple closely good matching options. AstreaG, on the other hand, use MWPM graph which is a complete graph comprising all flipped parity bits. In this graph, the edges represent the shortest path between each pair of flipped bits. AstreaG filters out higher weight pairs by pruning edges of the MWPM graph with error chain probabilities below the LER, followed by employing a greedy-based near-exhaustive search method for decoding. When run in parallel, Astrea-G helps Promatch by providing an exhaustive search strategy. Promatch and Astrea-G each have their specific strengths in handling different types of error patterns which we explain in the following parts.

Both Succeed in Sparse Error Patterns: Both Promatch and Astrea-G are successful in decoding scenarios with sparse flipped parity bits. Promatch's greedy approach works well in these cases, as it efficiently prioritizes isolated pairs or nodes with a single degree. Astrea-G also performs effectively here, as its searching method benefits from the sparsity, as it can prune more number of edges allowing for rapid convergence.

Promatch Succeeds, Astrea-G Struggles: In situations where decoding subgraph components are closely spaced but has just enough number of simple matchings, such as isolated pairs, Promatch outperforms Astrea-G. Astrea-G's exhaustive search method becomes less effective under real-time constraints in these dense environments. It struggles prune the MWPM graph. Therefore, it cannot effectively navigate through the tightly packed, yet non-interconnected components within the strict time limit.

Promatch, on the other hand, excels in these scenarios by prioritizing safer and easier matchings. Its local approach focuses on quickly resolving simpler pairings in the immediate vicinity. The remaining, potentially complicated parts of the syndrome are then managed by the main decoder, within the remaining time until reaching the 1-μs threshold. In Figure 12, we depict an example of these cases, which components are close to each other but the correct matching does not have any pairs among separate components. In this case, Promatch correctly predecodes three pairs, circled in blue in Figure 12(a), and sends the rest to the main decoder. On the other hand, as shown in Figure 12(b), Astrea-G's solution contains matching among components due to the closeness of the components which prevent pruning of the MWPM graph.

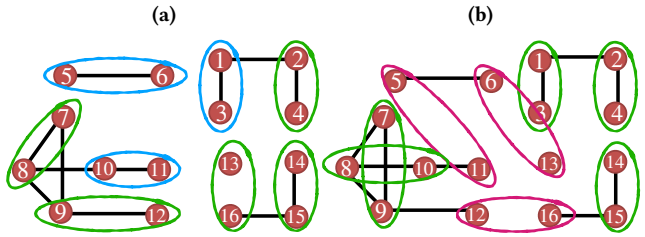

Figure 12. The flipped parity bits are closely spaced. (a) Promatch localy predecode 3 pairs (circled in blue) and sends the rest to the main decoder (b) Astrea-G could not prune the paths among components, resulting in incorrectly matching flipped bits across components.

Astrea-G Succeeds, Promatch Struggles: While Astrea-G also employs a greedy algorithm, it shows proficiency in decoding scenarios where some components of the decoding subgraph are distantly placed, facilitating the pruning process and reducing the size of the search space. This is particularly advantageous in cases where certain components are far enough apart to allow for effective pruning, yet others

are close enough to provide lower matching solutions overall. Especially, if there exists components with an odd number of parity bits, which necessitate cross-component matchings. Our experiments showed that almost all of samples that Promatch fails to decode (99.9% for Promatch compared to 61% for Astrea-G) contains components with odd number of nodes. In such instances, Astrea-G's ability to balance between pruning distant components and exploring close pairings becomes crucial.

Promatch, with its focus on local pairings, may not efficiently decode these error patterns. It excels in scenarios with more localized error structures but struggles when the optimal decoding path involves considering a wider spread of error components (more than 10 flipped bits so that no all of them can be passed to the main decoder). Astrea-G's method of selectively pruning the search space, while still considering broader pairings, allows it to uncover decoding paths that Promatch might miss due to its local focus. See Figure 13 as an example for such scenarios.

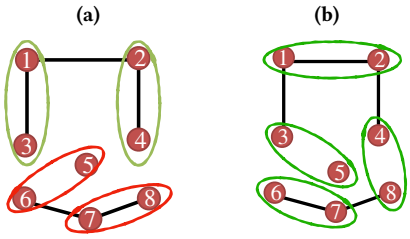

Figure 13. There exist multiple matchings across components (a) Promatch predecode two pairs incorrectly, since node 5 is not close enough to any of the remaining nodes (b) Astrea-G near-exhaustive search finds the correct solution.

Both Face Challenges: Despite their respective strengths, Promatch and Astrea-G both encounter difficulties in certain complex scenarios. These challenging situations involve dense error patterns that require exhaustive searches beyond what either algorithm can handle. In these instances, the error patterns are so densely packed that they exceed the search capabilities of both Promatch and Astrea-G within the required timeframe. While such scenarios are infrequent, they contribute to a slight increase in the LER (e.g. 10^{-17} for $d = 13$). However, this impact on the LER is not significant enough to notably affect the overall decoding performance. The rarity of these complex cases ensures that the effectiveness of parallel running of Promatch and Astrea-G in real-time decoding remains high.

4.3 Comparison of Promatch with Prior Predecoders

In the realm of predecoders for surface codes, various methodologies have been explored. The hierarchical predecoder [20] and Clique [49] adopt a greedy strategy, matching each flipped parity bit with its neighbors. It operates on a non-syndrome-modified basis: if it fails to decode all flipped bits, it forwards the entire syndrome unaltered to the main decoder. These predecoders do not adequately address the issue of high Hamming weight syndromes, as they do not modify these complex syndromes before passing them to the main decoder. Promatch, in contrast, modifies syndromes in such a way that the main decoder can process them within the remaining time, thus easing the decoding process for high distance codes ($d > 9$).

Neural network-based predecoders, represented by Chamberland et al. [11] and NEO-QEC [62], modify syndromes but are not designed for real-time decoding due to latency issues on FPGAs [11] and relying on emerging technologies [62]. Promatch offers an advantage over these methods by being optimized for real-time decoding on FPGAs, solving the latency issues inherent in these approaches.

Belief propagation represents a novel approach in predecoding proposed by Caune et al. [9], targeting high accuracy and coverage. However, its effectiveness relies on the algorithm's ability to converge to a solution. When it does not achieve convergence, the algorithm partially decodes the syndrome, modifies it, and adjusts the weights of the decoding graph before sending the remainder to the main decoder. This process, involving complex message passing among the nodes of the decoding graph, presents significant challenges for real-time implementation. To date, a real-time execution of belief propagation on hardware, particularly on FPGA, has not been demonstrated. Promatch, in contrast, offers a simpler and more reliable solution. It employs a straightforward logic that has been effectively implemented on FPGA, ensuring consistent predecoding performance without the complexities of message passing.The qualitative differences among various predecoders are summarized in Table 1, with respect to accuracy, coverage, and real-time (RT) capability.

Table 1. Comparison of Promatch with Prior Predecoders

Predecoder	Accuracy	Coverage	RT
Promatch	High	Sufficient	Yes
Clique [49][1]	High	Low	Yes
Hierarchical [20][1]	High	Low	Yes
Smith et al. [55]	Low	High	Yes
Chamberland et al. [11]	Low	High	No
NEO-QEC [62]	Low	High	No
Belief Propagation [9]	High	High	No

[1] *Despite having high accuracy, due to very low coverage, the accuracy of final LER of these methods is very low, due to the limitations of RT-MWPM.*

5 Evaluation Methodology

5.1 Surface Code

We consider rotated surface codes for distance 11 and 13. Given that the recent Astrea-G decoder achieves RT-MWPM up to $d = 9$, we seek to achieve RT-MWPM up to $d = 13$.

5.2 Evaluation Baseline

As our goal is to demonstrate RT-MWPM up to $d = 13$, we use idealized MWPM as a baseline. The closer a decoder's logical error rate is to MWPM's logical error rate, the better. We also use Astrea-G [66] decoder, Clique [49], and Smith et al. predecoders [55] as other baselines.

5.3 Noise Model and Simulation Infrastructure[3]

We adopt a uniform circuit-level noise model with a physical error rate ranging from $p = 10^{-4}$ to 5×10^{-4}. This model includes (1) start-of-round depolarizing errors (with equal probabilities for I, X, Y, Z) on data qubits, (2) depolarizing errors following gate operations on all qubit operands, (3) measurement errors, and (4) reset initialization errors, each occurring with probability p. The use of a uniform physical error rate is widely acknowledged in quantum-error-correction research, as evidenced by several studies [10, 18, 23, 26, 45, 49, 66]. Additionally, employing a circuit-level error model is considered reflective of real-device performance, aligning with the standards set in recent research [1, 23, 24, 27, 41, 66]. We use Google's Stim framework [24] for our evaluations due to its status as an industry-scale simulator, well-established in quantum error correction studies [26, 45, 66]. This ensures the reliability and relevance of our experimental approach. In these state-preservation, or memory, experiments [2, 13, 17, 18, 32, 51, 57, 61, 63, 66], we initialize a logical qubit in the $|0\rangle$ state and perform syndrome extraction over d rounds, followed by measurement in the computational basis. The success of each experiment, and the estimation of the decoder's LER, is based on whether the measurement outcome aligns with the decoder's correction. Repeating these experiments across millions of trials allows for an accurate estimation of logical error rates[4].

To simulate the expected low LER (i.e. in the order of 10^{-15}), which would otherwise require trillions of trials, we use an alternate approach [48]. For up to $k = 24$ random error injections, we generate millions of syndromes and calculate the decoding failure probability, $P_f(k)$, and the occurrence probability, $P_o(k)$, of these errors. The logical error rate is then estimated using Equation (1), based on our error model.

$$\text{Logical Error Rate} = \sum_k P_o(k) \times P_f(k) \tag{1}$$

[3]The source code of our implementation is available for access at https://github.com/nargesalavi/Promatch.

[4]In this paper, we use only Z memory experiments, equivalent to X experiments with qubit initialization to $|+\rangle$ and Hadamard basis measurement.

6 Evaluations

We evaluate Promatch and Promatch || AG for $d = 11$ and $d = 13$, for $p = 10^{-4}$ to 5×10^{-4}.

6.1 Logical Error Rate of Promatch

Table 2 shows the LER of Promatch, idealized MWPM, Astrea-G [66], and Smith et al. [55] predecoder for $d = 11$ and $d = 13$. Smith et al. predecoder is only applied to high-Hamming weight (HW > 10) syndromes and uses Astrea as the main decoder (same as Promatch). Smith predecoder cannot improve the LER of Astrea beyond $d = 11$ due to its low accuracy and not guaranteeing enough coverage. For $d = 11$, Promatch gains similar LER to Astrea-G for $p = 10^{-4}$. For $d = 13$, Promatch outperforms Astrea-G by 5.6×. Furthermore, we observe parallel running of Promatch and Astrea-G achieves practically identical performance to MWPM, outperforming the setting which Smith is running in parallel to Astrea-G.

Table 2. Logical Error Rate for $d = 11$ and $d = 13$ at $p = 10^{-4}$

Decoder	$d = 11$	$d = 13$
MWPM (*Ideal*)	1.8×10^{-13}	3.4×10^{-15}
Promatch[1] || AG	1.8×10^{-13} (1×)	3.4×10^{-15} (1×)
Promatch + Astrea	4.5×10^{-13} (2.5×)	2.6×10^{-14} (7.7×)
Astrea-G (AG)	4.5×10^{-13} (2.5×)	1.4×10^{-13} (43×)
Smith[1] || AG	2.5×10^{-13} (1.3×)	1.5×10^{-14} (4.5×)
Smith + Astrea	4.4×10^{-11} (240×)	6.9×10^{-11} (20412×)

[1] *The main decoder in this setting is Astrea. In other words, structure of this design is (predecoder + Astrea) || AG*

We also experimented with the Clique [49] predecoder in two settings: 1) Astrea as the main decoder (Clique + Astrea), 2) Astrea-G as the main decoder (Clique+AG). In both cases, the predecoder is only applied on High-Hamming weight syndromes, as syndromes with low Hamming weights can perfectly be decoded by the main decoder (same as Promatch). Table 3 shows the LER for Clique. Clique+Astrea has a high LER as Clique is a non-modified predecoder that does not handle complex patterns. As distance increases, the probability of encountering complex patterns, particularly at high-Hamming weight syndromes (HW > 10) increases. In these instances, Clique forwards these syndromes to Astrea. Astrea cannot decode any of them because it is optimized for decoding only low-HW (HW ≤ 10) syndromes (this results in the LER of Clique + Astrea becomes too high, in the order of p, for distance 13). The LER of Clique+AG is equal to the LER of AG. This shows Clique cannot improve the performance of main decoders due to its too low coverage. Therefore, we do not analyze Clique any further in our study.

Table 3. Clique's Logical Error Rate for $d = 11$ and $d = 13$, and $p = 10^{-4}$.

Decoder	$d = 11$	$d = 13$
Clique + Astrea	2.2×10^{-5} ($10^8\times$)	$> 10^{-4}$ ($> 10^9\times$)
Clique + AG	4.5×10^{-13} (2.5×)	1.4×10^{-13} (43×)
Astrea-G (AG)	4.5×10^{-13} (2.5×)	1.4×10^{-13} (43×)

6.2 Sensitivity to Physical Error Rate

Figures 14 and 15 present the LER for idealized MWPM, Promatch, Astrea-G (AG), Smith, Smith || AG, and Promatch || AG over physical error rates from 10^{-4} to 5×10^{-4}. Promatch maintains an LER within 5.9× to 202× of MWPM's LER for $d = 11$ and 13, respectively. This performance significantly outperforms Astrea-G, whose LER is up to 22× and 2064× higher than MWPM's for distance 11 and 13, respectively. Promatch || AG remains in 1.1× and 13.9× of MWPM's LER for distance 11 and 13, respectively, significantly outperforms Smith || AG which remains in 25× and 152× of MWPM's LER for distance 11 and 13, respectively.

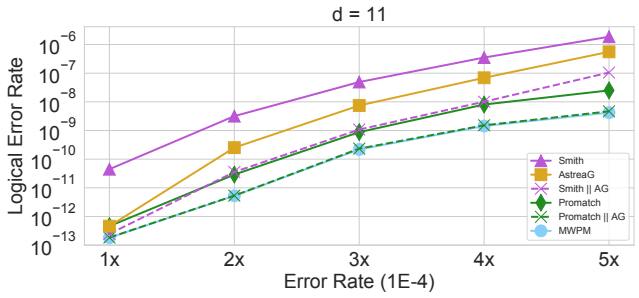

Figure 14. LER of non-real-time MWPM, Promatch, AstreaG(AG), Smith, Smith || AG, Promatch || AG for $10^{-4} \leq p \leq 5 \times 10^{-4}$ for $d = 13$. Promatch || AG remains in 1.1× of MWPM's LER.

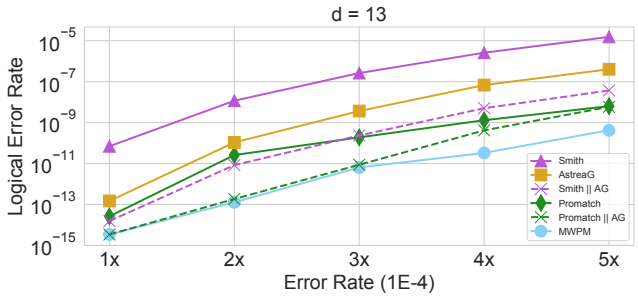

Figure 15. LER of non-real-time MWPM, Promatch, AstreaG(AG), Smith, Smith || AG, Promatch || AG for $10^{-4} \leq p \leq 5 \times 10^{-4}$ for $d = 13$. Promatch || AG remains in 13.9× of MWPM's LER.

6.3 Hamming Weight Reduction of Syndromes

Figures 16 and 17 show the HW distribution before and after applying syndrome-modified predecoding methods, namely Promatch, and Smith et al. [55] for $d = 11$ and $d = 13$. In our experiments, the predecoding only applies to high-HW syndromes ($HW > 10$), as Astrea [66] can decode low-HW syndromes in real-time. Unlike Smith et al [55], Promatch always predecodes high-HW down to HWs of 6, 8, or 10. Promatch consistently achieves sufficient coverage, ensuring Astrea can decode all post-predecoding syndromes.

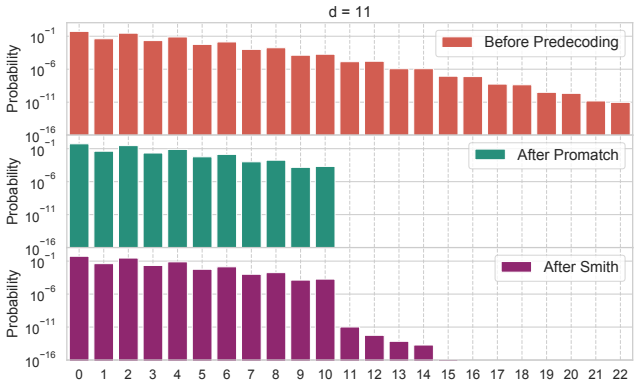

Figure 16. Promatch consistently lowers syndrome Hamming weight to 10 or less, allowing Astrea to accurately decode distance 11 surface codes, unlike the Smith et al. predecoder, for $p = 10^{-4}$

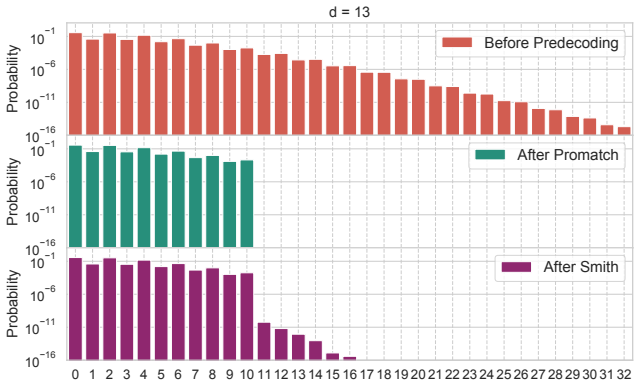

Figure 17. Unlike Smith et al. predecoder, Promatch reduces the syndrome Hamming weight up to 10 such that the main decoder (Astrea) can decode distance 13 surface codes accurately for $p = 10^{-4}$.

6.4 Latency of Promatch and Its Impact on LER

The latency of Promatch depends on the number of edges in the decoding subgraph since the pipeline iterates multiple times over these edges. During each cycle, Promatch's

pipeline analyzes one edge of the subgraph and decides whether or not to match the syndrome bits connected by this edge. If two syndrome bits are matched, the edges linked to the recently matched bits are removed from the decoding graph. Consequently, in the next predecoding round of that syndrome, the Promatch pipeline operates over fewer edges.

Based on this pipeline structure, we estimated the number of consumed cycles for each syndrome by summing the edge numbers in the decoding subgraphs across all predecoding rounds prior to sending the syndrome to the main decoder. When utilizing Step 3 of Promatch, we add the maximum value of the following: the number of paths from singletons to flipped bits or the number of edges.

If the duration exceeds 1μs, it is categorized as a logical error, prompting an abort of Promatch. We have allocated 10 cycles for the final comparison of the solution with Astrea-G [66] in the Promatch || AG design. Thus, in our simulations, Promatch, inclusive of the main decoder, is allotted a time budget of 960ns (operating at 250MHz).

Tables 4 and 5 depict the maximal and average latencies for predecoding and the combined predecoding + main decoder, respectively. It's crucial to note that all provided numbers pertain to decoding high Hamming weight syndromes where HW $\geq$ 10. Is important to note that in Tables 5, the maximum latency shows that there are cases that Promatch reaches its maximum budget, while there is a very low probability 1.5×10^{-17} for $d = 13$ which Promatch exceeds $1\mu s$, negligibly impacts the LER. Moreover, as Promatch is a lightweight predecoder, one can run multiple number of its pipeline in parallel for reducing the latency further.

Table 4. Latency of Predecoding High-HW Syndromes (ns)

Distance	11	13
Max	824	928
Average	68.2	70.0

Table 5. Latency of Decoding High-HW Syndromes using Promatch (ns)

Distance	11	13
Max	904	960
Average	524.2	526.0

6.5 Usage of Each Steps of Promatch

Different steps of Promatch is utilized with different rates because of different probabilities of Hamming weights in the syndrome and different rate of simple and complex patterns. Table 6 illustrates the proportion of samples, for d=11 and d=13, that are processed up to each of four distinct steps in the algorithm. The value associated with each step signifies the frequencies of samples that needed to be processed up to that point. It shows an overview of the efficiency of each step of Promatch, reflecting how often each step is necessary to reach the final LER. For $d = 11$ and 13, 99.56% and 99.83% of samples require only Step 1 of the algorithm, respectively. Nonetheless, other steps of the Promatch play an important role in the predecoding process due to the very low LER values. The last step, which is utilized with the least frequency, is still employed with higher frequency than LER for all the distances, which shows the importance of all the steps of the algorithm in reaching the desired LER.

Table 6. Frequency of each step during the decoding process

Steps	d = 11	d = 13
Step 1	0.9956	0.9983
Step 2	0.00439	0.00167
Step 3	6.1×10^{-11}	7.3×10^{-11}
Step 4	2.4×10^{-11}	1.8×10^{-11}

6.6 FPGA Utilization and Storage Overhead of Promatch

We synthesize Promatch on a Kintex UltraScale+ FPGA. The utilization details are shown in Table 7. Promatch's efficient use of resources underscores Promatch's practicality for near-term real-time decoding.

Table 7. FPGA Utilization of Promatch

Resource	LUT	FF	Frequency
Edge-Processing Pipeline	3%	1%	250 MHz

Table 8 shows the memory needed for storing the Edge table and the Path table of Promatch. For storing the Path Table, we optimize the required memory by categorizing the paths into four groups as Promatch is not sensitive to the exact weight of the paths.

Table 8. Storage Requirements of Promatch

Storage Type	d = 11	d = 13
Edge Table	3.6 KB	6 KB
Path Table	129 KB	345 KB

7 Related Work

We discuss prior work on real-time decoding and compare and contrast them with Promatch.

7.1 Decoders for Surface Code

Error decoding has been an active area of research, with several decoding algorithms proposed in the literature. The different classes of decoding algorithms:

Lookup Table (LUT) Decoder: This method uses a lookup table (LUT) to store corrections for every syndrome. Typical LUT implementations has scalability challenges due to exponential storage overheads.

Minimum Weight Perfect Matching (MWPM): This decoder uses a graph pairing algorithm and is considered to be one of the most effective in terms of accuracy. There has been significant recent research to improve the latency and scalability of MWPM especially for smaller distance codes.

Machine Learning (ML) Decoders: These designs train neural networks with a set of syndrome vectors and the resulting error location and types [4–7, 12, 15, 16, 19, 40, 43, 44, 46, 58, 60, 64, 65, 67]. They key challenges with these designs is the lack of training data for large distance codes and the substantial resource requirements (memory, compute, time), making them unappealing for large distance codes.

7.2 Real-Time Decoding

Real-time decoding is necessary to achieve quantum fault-tolerance, and thus much prior work has been dedicated to achieving accurate decoding in real-time. Recent implementations of MWPM decoding, such as **Sparse-Blossom** and **Fusion-Blossom** [31, 69], have achieved *mean* latencies of 1μs per syndrome extraction round. While such results are impressive, quantum fault-tolerance will likely require a worst case latency of 1μs to avoid unnecessarily stalling the quantum computer, and thus such implementations are insufficient for real-time decoding. Consequently, much recent work has proposed hardware implementations. **LILLIPUT** is capable of decoding up to $d = 5$ with the same accuracy as MWPM [17]. However, the lookup tables of LILLIPUT are difficult to scale beyond $d = 5$. Decoders such as **NISQ+** and **QECOOL**/**QULATIS** leverage superconducting logic to achieve low latencies [32, 61, 63]. However, these decoders cannot handle arbitrary measurement errors, thus causing significant inaccuracy. The **AFS** decoder implements the union-find algorithm [18]. AFS is comparable to MWPM at extremely low error rates, but for the near-term error rate of 10^{-4} evaluated in this paper, AFS is inaccurate compared to MWPM decoding as the underlying union-find algorithm is inaccurate in this regime [21]. **Astrea** is a recent implementation of MWPM decoding restricted to distances 7 and 9, but has poor performance for $d = 11$ and beyond [66]. Promatch achieves real-time decoding up to $d = 13$ with comparable accuracy to MWPM and is the first decoder to do so.

7.3 Predecoding for Reduced Bandwidth

Predecoding has emerged as a strategy for (1) reducing bandwidth requirements and (2) improving the latency of decoders. Delfosse [20] demonstrated that predecoding can offer significant bandwdith reduction without sacrificing accuracy. Recently, the Clique decoder provided an implementation of Delfosse's design in superconducting logic [49]. However, while both Delfosse's proposal and the Clique decoder reduce bandwidth requirements, they do not improve worst-case decoder latency. Smith et al. has proposed a predecoder which greedily filters syndromes sent to the main decoder, allowing for better coverage compared to Delfosse's original proposal [55]. Concurrently, Chamberland et al. proposed a predecoder to filter syndromes, but uses neural networks instead of a greedy strategy [11]; NEO-QEC is an implementation of Chamberland et al.'s proposal in superconducting logic [62]. However, the limitation of these works is that they sacrifice accuracy to improve coverage. In contrast, Promatch adaptively finds the highest accuracy at a good-enough coverage to ensure real-time decoding.

8 Conclusion

Decoders used in quantum error correction must accurately identify errors in real-time (typically within a 1μs on superconducting systems) to prevent the backlog of errors. Although the Minimum Weight Perfect Matching (MWPM) decoder is widely recognized for its effectiveness in decoding surface codes, achieving MWPM accuracy beyond distance 9 remains an open problem because the complexity of decoding grows with the distance of the code. This paper introduces Promatch to expand the reach of real-time MWPM up to distance 13. Promatch uses predecoding to transform high Hamming-weight syndromes into low Hamming-weight that are amenable to real-time MWPM decoders.

Designing an accurate yet high-coverage predecoder is challenging. Aggressive predecoding leads to inaccurate matches, whereas conservative predecoding does not reduce the complexity of the decoding graph substantially and remains bottlenecked by the complexity of the main decoder. Promatch attains a sweet-spot between these two extremes by leveraging the following insights. First, most syndrome bit flips are matched to other flipped syndrome bits in their local neighborhood. Second, predecoding must be performed only until the syndrome Hamming weight reaches a point beyond which it can be fully handled by the MWPM decoder. Promatch enables locality-aware adaptive predecoding that adjusts the number of prematches depending on the syndrome Hamming weight to ensure high accuracy while simultaneously meeting the real-time latency constraints. Promatch achieves logical error rate of 4.5×10^{-13} and 2.6×10^{-14} for distance 11 and 13, respectively. Promatch also achieves MWPM accuracy for up to distance 13 when it runs in parallel with Astrea-G (logical error rate of 3.4×10^{-15}).

Acknowledgement

We thank the reviewers of ASPLOS-2024 for their suggestions and feedback. N.A. thanks Sadegh Shirani for helpful discussions. This work was funded in part by EPiQC, an NSF Expedition in Computing, under grant CCF-173044.

References

[1] Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949):676–681, 2023.

[2] Rajeev Acharya, Igor Aleiner, Richard Allen, Trond I. Andersen, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Juan Atalaya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Joao Basso, Andreas Bengtsson, Sergio Boixo, Gina Bortoli, Alexandre Bourassa, Jenna Bovaird, Leon Brill, Michael Broughton, Bob B. Buckley, David A. Buell, Tim Burger, Brian Burkett, Nicholas Bushnell, Yu Chen, Zijun Chen, Ben Chiaro, Josh Cogan, Roberto Collins, Paul Conner, William Courtney, Alexander L. Crook, Ben Curtin, Dripto M. Debroy, Alexander Del Toro Barba, Sean Demura, Andrew Dunsworth, Daniel Eppens, Catherine Erickson, Lara Faoro, Edward Farhi, Reza Fatemi, Leslie Flores Burgos, Ebrahim Forati, Austin G. Fowler, Brooks Foxen, William Giang, Craig Gidney, Dar Gilboa, Marissa Giustina, Alejandro Grajales Dau, Jonathan A. Gross, Steve Habegger, Michael C. Hamilton, Matthew P. Harrigan, Sean D. Harrington, Oscar Higgott, Jeremy Hilton, Markus Hoffmann, Sabrina Hong, Trent Huang, Ashley Huff, William J. Huggins, Lev B. Ioffe, Sergei V. Isakov, Justin Iveland, Evan Jeffrey, Zhang Jiang, Cody Jones, Pavol Juhas, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Tanuj Khattar, Mostafa Khezri, Mária Kieferová, Seon Kim, Alexei Kitaev, Paul V. Klimov, Andrey R. Klots, Alexander N. Korotkov, Fedor Kostritsa, John Mark Kreikebaum, David Landhuis, Pavel Laptev, Kim-Ming Lau, Lily Laws, Joonho Lee, Kenny Lee, Brian J. Lester, Alexander Lill, Wayne Liu, Aditya Locharla, Erik Lucero, Fionn D. Malone, Jeffrey Marshall, Orion Martin, Jarrod R. McClean, Trevor Mccourt, Matt McEwen, Anthony Megrant, Bernardo Meurer Costa, Xiao Mi, Kevin C. Miao, Masoud Mohseni, Shirin Montazeri, Alexis Morvan, Emily Mount, Wojciech Mruczkiewicz, Ofer Naaman, Matthew Neeley, Charles Neill, Ani Nersisyan, Hartmut Neven, Michael Newman, Jiun How Ng, Anthony Nguyen, Murray Nguyen, Murphy Yuezhen Niu, Thomas E. O'Brien, Alex Opremcak, John Platt, Andre Petukhov, Rebecca Potter, Leonid P. Pryadko, Chris Quintana, Pedram Roushan, Nicholas C. Rubin, Negar Saei, Daniel Sank, Kannan Sankaragomathi, Kevin J. Satzinger, Henry F. Schurkus, Christopher Schuster, Michael J. Shearn, Aaron Shorter, Vladimir Shvarts, Jindra Skruzny, Vadim Smelyanskiy, W. Clarke Smith, George Sterling, Doug Strain, Marco Szalay, Alfredo Torres, Guifre Vidal, Benjamin Villalonga, Catherine Vollgraff Heidweiller, Theodore White, Cheng Xing, Z. Jamie Yao, Ping Yeh, Juhwan Yoo, Grayson Young, Adam Zalcman, Yaxing Zhang, and Ningfeng Zhu. Suppressing quantum errors by scaling a surface code logical qubit, 2022.

[3] Sashwat Anagolum, Narges Alavisamani, Poulami Das, Moinuddin Qureshi, Eric Kessler, and Yunong Shi. Élivágar: Efficient quantum circuit search for classification. *arXiv preprint arXiv:2401.09393*, 2024.

[4] Philip Andreasson, Joel Johansson, Simon Liljestrand, and Mats Granath. Quantum error correction for the toric code using deep reinforcement learning. *Quantum*, 3:183, 2019.

[5] Paul Baireuther, MD Caio, B Criger, Carlo WJ Beenakker, and Thomas E O'Brien. Neural network decoder for topological color codes with circuit level noise. *New J. Phys.*, 21, 2019.

[6] Paul Baireuther, Thomas E O'Brien, Brian Tarasinski, and Carlo WJ Beenakker. Machine-learning-assisted correction of correlated qubit errors in a topological code. *Quantum*, 2018.

[7] Nikolas P Breuckmann and Xiaotong Ni. Scalable neural network decoders for higher dimensional quantum codes. *Quantum*, 2, 2018.

[8] A Robert Calderbank and Peter W Shor. Good quantum error-correcting codes exist. *Physical Review A*, 54(2):1098, 1996.

[9] Laura Caune, Joan Camps, Brendan Reid, and Earl Campbell. Belief propagation as a partial decoder. *arXiv preprint arXiv:2306.17142*, 2023.

[10] Christopher Chamberland and Andrew W Cross. Fault-tolerant magic state preparation with flag qubits. *Quantum*, 3:143, 2019.

[11] Christopher Chamberland, Luis Goncalves, Prasahnt Sivarajah, Eric Peterson, and Sebastian Grimberg. Techniques for combining fast local decoders with global decoders under circuit-level noise, 2022.

[12] Christopher Chamberland and Pooya Ronagh. Deep neural decoders for near term fault-tolerant experiments. *Quantum Science and Technology*, 3, 2018.

[13] Zijun Chen, Kevin J Satzinger, Juan Atalaya, Alexander N Korotkov, Andrew Dunsworth, Daniel Sank, Chris Quintana, Matt McEwen, Rami Barends, Paul V Klimov, et al. Exponential suppression of bit or phase flip errors with repetitive error correction. *arXiv preprint arXiv:2102.06132*, 2021.

[14] Andrew M. Childs, Dmitri Maslov, Yunseong Nam, Neil J. Ross, and Yuan Su. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences*, 115(38), sep 2018.

[15] Chaitanya Chinni, Abhishek Kulkarni, and Dheeraj M Pai. Neural decoder for topological codes using pseudo-inverse of parity check matrix. *arXiv:1901.07535*, 2019.

[16] Laia Domingo Colomer, Michalis Skotiniotis, and Ramon Muñoz-Tapia. Reinforcement learning for optimal error correction of toric codes. *arXiv preprint:1911.02308*, 2019.

[17] Poulami Das, Aditya Locharla, and Cody Jones. Lilliput: A lightweight low-latency lookup-table decoder for near-term quantum error correction. In *ASPLOS-27*, 2022.

[18] Poulami Das, Christopher A. Pattison, Srilatha Manne, Douglas M. Carmean, Krysta M. Svore, Moinuddin Qureshi, and Nicolas Delfosse. Afs: Accurate, fast, and scalable error-decoding for fault-tolerant quantum computers. In *HPCA*, 2022.

[19] Amarsanaa Davaasuren, Yasunari Suzuki, Keisuke Fujii, and Masato Koashi. General framework for constructing fast and near-optimal machine-learning-based decoder of the topological stabilizer codes. *arXiv:1801.04377*, 2018.

[20] Nicolas Delfosse. Hierarchical decoding to reduce hardware requirements for quantum computing, 2020.

[21] Nicolas Delfosse and Naomi H Nickerson. Almost-linear time decoding algorithm for topological codes. *arXiv preprint arXiv:1709.06218*, 2017.

[22] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, Sep 2002.

[23] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[24] Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, July 2021.

[25] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, apr 2021.

[26] Craig Gidney, Michael Newman, Austin Fowler, and Michael Broughton. A Fault-Tolerant Honeycomb Memory. *Quantum*, 5:605, December 2021.

[27] Craig Gidney, Michael Newman, and Matt McEwen. Benchmarking the Planar Honeycomb Code. *Quantum*, 6:813, September 2022.

[28] Oscar Higgott. Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching, 2021.

[29] Oscar Higgott. Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching. *ACM Transactions on Quantum Computing*, 3(3):1–16, 2022.

[30] Oscar Higgott and Craig Gidney. Sparse blossom: correcting a million errors per core second with minimum-weight matching. *arXiv preprint arXiv:2303.15933*, 2023.

[31] Oscar Higgott and Craig Gidney. Sparse blossom: correcting a million errors per core second with minimum-weight matching. *arXiv preprint arXiv:2303.15933*, 2023.

[32] Adam Holmes, Mohammad Reza Jokar, Ghasem Pasandi, Yongshan Ding, Massoud Pedram, and Frederic T. Chong. Nisq+: Boosting quantum computing power by approximating quantum error correction. In *ISCA-47*, pages 556–569, 2020.

[33] Hsin-Yuan Huang, Michael Broughton, Jordan Cotler, Sitan Chen, Jerry Li, Masoud Mohseni, Hartmut Neven, Ryan Babbush, Richard Kueng, John Preskill, et al. Quantum advantage in learning from experiments. *Science*, 376(6598):1182–1186, 2022.

[34] Hsin-Yuan Huang, Richard Kueng, and John Preskill. Information-theoretic bounds on quantum advantage in machine learning. *Physical Review Letters*, 126(19):190505, 2021.

[35] Shilin Huang, Michael Newman, and Kenneth R. Brown. Fault-tolerant weighted union-find decoding on the toric code. *Phys. Rev. A*, 102:012419, Jul 2020.

[36] A Yu Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191, dec 1997.

[37] Ian D. Kivlichan, Craig Gidney, Dominic W. Berry, Nathan Wiebe, Jarrod McClean, Wei Sun, Zhang Jiang, Nicholas Rubin, Austin Fowler, Alá n Aspuru-Guzik, Hartmut Neven, and Ryan Babbush. Improved fault-tolerant quantum simulation of condensed-phase correlated electrons via trotterization. *Quantum*, 4:296, jul 2020.

[38] Vladimir Kolmogorov. Blossom v: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, 2009.

[39] Vladimir Kolmogorov. Blossom v: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1(1):43–67, 2009.

[40] Stefan Krastanov and Liang Jiang. Deep neural network probabilistic decoder for stabilizer codes. *Scientific reports*, 7, 2017.

[41] Andrew J. Landahl, Jonas T. Anderson, and Patrick R. Rice. Fault-tolerant quantum computing with color codes, 2011.

[42] Joonho Lee, Dominic W. Berry, Craig Gidney, William J. Huggins, Jarrod R. McClean, Nathan Wiebe, and Ryan Babbush. Even more efficient quantum computations of chemistry through tensor hypercontraction. *PRX Quantum*, 2(3), jul 2021.

[43] Ye-Hua Liu and David Poulin. Neural belief-propagation decoders for quantum error-correcting codes. *PRL*, 122, 2019.

[44] Nishad Maskara, Aleksander Kubica, and Tomas Jochym-O'Connor. Advantages of versatile neural-network decoding for topological codes. *Phys. Rev. A*, 99, 2019.

[45] Satvik Maurya, Chaithanya Naik Mude, William D. Oliver, Benjamin Lienhard, and Swamit Tannu. Scaling qubit readout with hardware efficient machine learning architectures. In *ISCA-50*, 2023.

[46] Xiaotong Ni. Neural network decoders for large-distance 2d toric codes. *arXiv:1809.06640*, 2018.

[47] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5:4213, 2014.

[48] Moinuddin K. Qureshi and Zeshan Chishti. Operating secded-based caches at ultra-low voltage with flair. In *43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2013.

[49] Gokul Subramanian Ravi, Jonathan M Baker, Arash Fayyazi, Sophia Fuhui Lin, Ali Javadi-Abhari, Massoud Pedram, and Frederic T Chong. Better than worst-case decoding for quantum error correction. In *ASPLOS-28, Volume 2*, 2023.

[50] Markus Reiher, Nathan Wiebe, Krysta M. Svore, Dave Wecker, and Matthias Troyer. Elucidating reaction mechanisms on quantum computers. *Proceedings of the National Academy of Sciences*, 114(29):7555–7560, 2017.

[51] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, T. M. Gatterman, S. K. Halit, K. Gilmore, J. A. Gerber, B. Neyenhuis, D. Hayes, and R. P. Stutz. Realization of real-time fault-tolerant quantum error correction. *Phys. Rev. X*, 11:041058, Dec 2021.

[52] T. R. Scruby, D. E. Browne, P. Webster, and M. Vasmer. Numerical Implementation of Just-In-Time Decoding in Novel Lattice Slices Through the Three-Dimensional Surface Code. *Quantum*, 6:721, May 2022.

[53] Peter W Shor. Scheme for reducing decoherence in quantum computer memory. *Physical review A*, 52(4):R2493, 1995.

[54] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 1999.

[55] Samuel C Smith, Benjamin J Brown, and Stephen D Bartlett. Local predecoder to reduce the bandwidth and latency of quantum error correction. *Physical Review Applied*, 19(3):034050, 2023.

[56] Andrew Steane. Multiple-particle interference and quantum error correction. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 452(1954):2551–2577, 1996.

[57] Neereja Sundaresan, Theodore J. Yoder, Youngseok Kim, Muyuan Li, Edward H. Chen, Grace Harper, Ted Thorbeck, Andrew W. Cross, Antonio D. Córcoles, and Maika Takita. Matching and maximum likelihood decoding of a multi-round subsystem quantum error correction experiment, 2022.

[58] Ryan Sweke, Markus S Kesselring, Evert PL van Nieuwenburg, and Jens Eisert. Reinforcement learning decoders for fault-tolerant quantum computation. *arXiv preprint:1810.07207*, 2018.

[59] Yu Tomita and Krysta M. Svore. Low-distance surface codes under realistic quantum noise. *Physical Review A*, 90(6), Dec 2014.

[60] Giacomo Torlai and Roger G Melko. Neural decoder for topological codes. *PRL*, 119, 2017.

[61] Yosuke Ueno, Masaaki Kondo, Masamitsu Tanaka, Yasunari Suzuki, and Yutaka Tabuchi. Qecool: On-line quantum error correction with a superconducting decoder for surface code. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 451–456, 2021.

[62] Yosuke Ueno, Masaaki Kondo, Masamitsu Tanaka, Yasunari Suzuki, and Yutaka Tabuchi. Neo-qec: Neural network enhanced online superconducting decoder for surface codes, 2022.

[63] Yosuke Ueno, Masaaki Kondo, Masamitsu Tanaka, Yasunari Suzuki, and Yutaka Tabuchi. Qulatis: A quantum error correction methodology toward lattice surgery. In *HPCA*, pages 274–287, 2022.

[64] Savvas Varsamopoulos, Koen Bertels, and Carmen G Almudever. Designing neural network based decoders for surface codes. *arXiv:1811.12456*, 2018.

[65] Savvas Varsamopoulos, Ben Criger, and Koen Bertels. Decoding small surface codes with feedforward neural networks. *Quantum Science and Technology*, 3(1):015004, 2017.

[66] Suhas Vittal, Poulami Das, and Moinuddin Qureshi. Astrea: Accurate quantum error-decoding via practical minimum-weight perfect-matching. In *ISCA-50*, 2023.

[67] Thomas Wagner, Hermann Kampermann, and Dagmar Bruß. Symmetries for a high level neural decoder on the toric code. *arXiv:1910.01662*, 2019.

[68] Hanrui Wang, Yongshan Ding, Jiaqi Gu, Yujun Lin, David Z Pan, Frederic T Chong, and Song Han. Quantumnas: Noise-adaptive search for robust quantum circuits. In *HPCA*, 2022.

[69] Yue Wu and Lin Zhong. Fusion blossom: Fast mwpm decoders for qec. *arXiv preprint arXiv:2305.08307*, 2023.