Slide 1: Title (about 1:30)

[Next slide.]

Slide 2: From Classical Coding Theory to Quantum Error Correction (about 3:00)

In classical coding theory, / we have reached a mature stage.
There are code families, / LDPC codes being a prime example, / where belief propagation decoding achieves performance close to channel capacity, / with complexity essentially linear in the block length.

So the natural question is:
Can we transfer these powerful principles, / sparsity, message passing, and capacity-approaching performance, / to quantum error correction?

Now, the key lessons we take from classical LDPC codes are the following.

First, belief propagation works because the parity-check matrix is sparse.
The decoder iteratively updates local beliefs using local parity constraints.

Second, density evolution, / which is an analytical tool that tracks how message distributions evolve through iterations of belief propagation in the large-block-length limit, / shows that even simple regular degree distributions can yield excellent waterfall performance.
Third, in binary random sparse codes, requiring column weight at least three is closely connected to linear growth of the minimum distance.

And fourth, the girth, / the shortest cycle length in the Tanner graph, / strongly affects both decoding dynamics and distance-related behavior.

Here is the key observation for today's talk.
Non-binary LDPC codes over finite field extensions can show very strong BP performance even with column weight two.
This is interesting for us, / because column weight two leads to an ultra-sparse parity-check structure, / which is highly attractive under quantum constraints, such as commutativity and scalable decoding, / if we can make it work.

[Next slide.]

Slide 3: Numerical Results (about 3:00)

I will show numerical performance results for qubit quantum error correction based on non-binary LDPC codes, when both the code construction and the decoding algorithm are carefully designed.

This plot shows the frame error rate over the depolarizing channel as a function of the physical error rate p_D.
The most important feature is a clear threshold-like waterfall behavior, which approaches the hashing bound.

Let me emphasize several key points.
First, belief propagation decoding has linear-time complexity in the code length.
We do not rely on heavy post-processing methods, such as ordered statistics decoding.

Second, no error floor is observed down to at least 10^{-4} in frame error rate.
Third, we achieve remarkably high coding rates, at least one third, even for very large block lengths.
Importantly, these results are obtained for extremely large-scale codes,

with the number of logical qubits reaching several hundred thousand.

And finally, a GPU implementation enables a decoding throughput on the order of one million qubits per second.

So the empirical message is quite strong.
We can achieve near-hashing-bound performance with scalable, practical decoding even at the scale of hundreds of thousands of logical qubits.
In the rest of this talk, I will explain how to construct these codes.
[Next slide.]

Slide 4: General Procedure of Error Correction (about 3:00)

Before going into the details of the code construction, I would like to precisely compare what a decoder is asked to do in the classical and quantum settings.

First, let me start with the classical case.
We have a single parity-check matrix H.
A noise realization is represented by a binary vector e.
We measure the syndrome $s = He$.
The decoder then finds an estimate $\hat{e}$ such that $H\hat{e} = s$.
The success criterion is strict: / the estimate $\hat{e}$ must exactly equal the true error.

Now, the quantum CSS case is fundamentally different.
We need two sparse parity-check matrices, H_X and H_Z, which must satisfy the commutation condition
$H_X H_Z^{\mathsf{T}} = 0$ over $\mathbb{F}_2$.

Physical noise is described by two binary vectors, x and z.

For the depolarizing channel, these two components are statistically correlated.
We measure two syndromes:
$s_X = H_X z$, / and $s_Z = H_Z x$.

The decoder outputs estimates $\hat{x}$ and $\hat{z}$ that are consistent with the measured syndromes.
However, successful decoding does not require exact recovery.
Because of degeneracy, it is sufficient that the residual error belongs to the appropriate dual code.

Thus, quantum decoding is more constrained in terms of code construction, / but at the same time it has extra freedom in what counts as a successful correction.

In this work, we assume perfect syndrome measurements.
Extending the framework to noisy and repeated syndrome measurements is an important direction for future work.
[Next slide.]

Slide 5: General Construction of Quantum LDPC Codes (about 3:00)

This slide explains a standard construction approach, and why it can naturally introduce small logical operators.

Because the condition $H_X H_Z^{\mathsf{T}} = 0$ must be satisfied, the design of H_X and H_Z is tightly coupled.
This often pushes us toward highly structured constructions.

A common approach is to start from fully orthogonal sparse "full" matrices, $H_X^{(\mathrm{ful})}$ and $H_Z^{(\mathrm{ful})}$, satisfying full orthogonality.

However, such full constructions often yield zero quantum rate.
To increase the rate, we remove some check rows.

We write

$$H_X^{(\mathrm{ful})} = \begin{bmatrix} H_X \\ H_X^{(\mathrm{res})} \end{bmatrix}, \qquad H_Z^{(\mathrm{ful})} = \begin{bmatrix} H_Z \\ H_Z^{(\mathrm{res})} \end{bmatrix}.$$

The final code uses only H_X and H_Z.

Here is the key point.
A removed row, for example a row z from $H_X^{(\mathrm{res})}$, automatically satisfies $H_Z z^{\mathsf{T}} = 0$.
So z lies in C_Z.
But in general, z does not belong to $C_X^{\perp}$.
Therefore, z acts as a Z-type logical operator.

As a result, increasing the rate by removing checks can naturally introduce low-weight logical operators.
This is a major reason why quantum LDPC code design often suffers from error floors or limited minimum distance.

So the key challenge is the following: / Can we increase the code rate while still controlling the emergence of small logical operators?

There are alternative constructions, such as those based on hypergraph product (HGP) codes.
However, in such constructions, short cycles are essentially unavoidable, and the degree distributions are difficult to control.
As a consequence, belief-propagation decoding does not perform particularly well, and heavy post-processing, such as ordered statistics decoding, is often required.

[Next slide.]

Slide 6: Key Ingredients (about 3:00)

To address the challenge I just described, / let me now summarize the key ingredients behind our proposed quantum LDPC codes.
This slide provides the roadmap for the remainder of the talk.

First, we adopt a generalized Hagiwara–Imai construction.
Second, we control commutativity using affine permutation matrices.
Third, we focus on ultra-sparse structures with column weight $J = 2$, / which allows large girth.

Fourth, in the $J = 2$ regime, codewords are cycle-based.
We exploit this by carefully assigning non-binary symbols so that these cycles do not create small logical errors.

Fifth, we use joint belief propagation decoding to exploit X–Z correlations.
And sixth, we correct the dominant rare trapping events using a post-processing algorithm with essentially constant complexity with respect to n.

Now I will unpack these components in order, starting with the construction backbone: the Hagiwara–Imai framework.
[Next slide.]

Slide 7: Hagiwara–Imai Construction (about 3:00)

In this slide, I explain the Hagiwara–Imai construction, which provides a systematic method to construct orthogonal LDPC matrix pairs with

column weight J and row weight L.

This construction was proposed by Manabu Hagiwara who made pioneering contributions to coding theory and to the development of algebraic constructions for quantum error-correcting codes.
And I should also mention that Hagiwara-san is well known in the community for leading a craft beer project at Chiba University, which has inspired many informal research discussions.

The basic idea of the construction is to arrange $P \times P$ binary permutation blocks in a cyclic, block-circulant manner.
Here, the block size P is assumed to be large, typically on the order of several thousand to several tens of thousands, which allows us to construct very long codes while keeping the local structure simple.

Let $\{F_i\}$ and $\{G_i\}$ be $P \times P$ binary permutation matrices, or more generally, affine permutation matrices.

The full parity-check matrices $H_X^{(\mathrm{ful})}$ and $H_Z^{(\mathrm{ful})}$ are block-circulant: each block row is obtained by a one-step cyclic shift of the previous one.

To obtain the final LDPC matrices H_X and H_Z, we take the upper J block rows of the full matrices.

With this choice, the product $H_X H_Z^{\mathsf{T}}$ contains $L/2$ pairs of products of the underlying permutation matrices.
The key idea of the construction is that if each such pair of permutation matrices commutes, their contributions cancel out, and the overall product becomes zero.

In the original Hagiwara–Imai construction, each permutation matrix is chosen to be a circulant shift permutation matrix.
As a consequence, all permutation matrices commute, and the orthogo-

nality condition is automatically satisfied.

In the generalized Hagiwara–Imai construction, however, we allow more general permutation matrices, including affine permutation matrices.
In this setting, the permutation matrices do not necessarily commute.
Therefore, orthogonality is no longer automatic and must be carefully enforced.

In the next slide, I will explain how orthogonality is explicitly controlled within the Hagiwara–Imai construction.
[Next slide.]

Slide 8: Commutativity Controls Orthogonality (about 3:00)

In the original Hagiwara–Imai construction, all constituent permutation matrices are chosen to commute.
As a result, the full matrices $H_X^{(\mathrm{ful})}$ and $H_Z^{(\mathrm{ful})}$ are perfectly orthogonal.

However, this strong form of orthogonality comes at a cost.
Because all block interactions are constrained to commute, the construction inevitably introduces low-weight logical errors, typically with weight proportional to L.
This leads to a small minimum distance and limits the error-correcting performance.

Our key idea is to relax this requirement.
Instead of enforcing commutativity for all constituent matrices, we require commutativity only for the subset of matrices that is necessary to guarantee orthogonality between H_X and H_Z.

The remaining matrices are intentionally chosen to be non-commutative. By doing so, we preserve the orthogonality condition $H_X H_Z^{\mathsf{T}} = 0$, while at the same time breaking the unwanted algebraic structure that gives rise to small logical errors.

In this way, commutativity becomes a design parameter that controls orthogonality, rather than a rigid global constraint.

This leads to a practical design challenge: how do we design permutations with prescribed commutativity patterns?
This is exactly why we introduce affine permutation matrices.
[Next slide.]

Slide 9: Affine Permutation Matrices (about 3:00)

On the previous slide, we reduced orthogonality to commutation relations among permutation blocks.
Now I explain how affine permutation matrices make these constraints manageable.

An affine permutation on $\mathbb{Z}_P$, / that is, the set of integers modulo P representing the block indices, / is defined by $f(x) = ax + b \pmod{P}$, where a is invertible modulo P.

The key benefit is that commutativity becomes a simple algebraic condition.
For two affine maps f and g, / the condition $f \circ g = g \circ f$ reduces to a modular equation in a and b.

So, with APMs, we can design commutativity and non-commutativity explicitly through parameters, rather than searching combinatorially over

general permutations.

Next, APMs also help us detect and avoid short cycles systematically, especially when $J = 2$.
[Next slide.]

Slide 10: Cycle Detection via Composition, $J = 2$ (about 3:00)

On the previous slide, we introduced APMs to control commutativity.
Now I explain how APMs make cycle detection algebraic.

For $J = 2$, a length-4 cycle in a 2×2 block submatrix is equivalent to the existence of a fixed point of a certain composite permutation.

If all blocks are affine, the composite map is again affine.
So checking for a fixed point reduces to a simple gcd-based condition.

Therefore, avoiding 4-cycles becomes a number-theoretic check rather than a graph search.
This approach generalizes to longer cycles.
And exploiting the ultra-sparse $J = 2$ structure, we can systematically avoid short cycles and achieve girth at least 12.

So far, we have controlled the binary support pattern and the girth.
Now we use non-binary symbols to further improve performance and eliminate small logical errors.
[Next slide.]

Slide 11: Non-Binary Extension (about 3:00)

On the previous slide, we discussed how we avoid short cycles and guarantee large girth.
Now I introduce the non-binary extension.

We extend the parity-check matrices to the finite field $\mathrm{GF}(2^e)$, with $e = 8$ in this work.
Each field element is represented in the binary parity-check matrix by an $e \times e$ companion-matrix representation.

So the proposed code remains a qubit code, not a qudit code.

Non-binary belief propagation updates probability distributions over $\mathrm{GF}(2^e)$.
This typically strengthens iterative decoding and provides extra design freedom.

Most importantly for us, non-binary symbol assignment lets us control rank properties of cycle-based substructures.
This is essential in the $J = 2$ regime, as I explain next.
[Next slide.]

Slide 12: Eliminating Small Logical Errors for $J = 2$ (about 3:00)

On the previous slide, we introduced the non-binary extension.
Now I explain how it is used to eliminate small logical errors when $J = 2$.

When the column weight is two, every codeword is generated by a cycle or a union of cycles in the Tanner graph.
Therefore, cycle-based structures in the removed rows can naturally lead

to small-weight codewords, and these can become dominant sources of logical errors.

Our strategy is to carefully assign non-binary symbols so that these cycles become full rank.
If the submatrix associated with a cycle is full rank, then that cycle cannot support a nonzero codeword.

As a result, the cycle-based structures no longer form valid codewords, which effectively eliminates small logical errors.

Now we turn to decoding, where the main tool is joint belief propagation.
[Next slide.]

Slide 13: Joint BP Decoding (about 3:00)

On the previous slide, we completed the construction story by eliminating small logical errors.
Now I explain the decoding algorithm: joint belief propagation.

We measure the syndromes $s = H_Z x$ and $t = H_X z$.

Joint BP decoding iteratively estimates $\hat{x}$ and $\hat{z}$ simultaneously.
This is well matched to the depolarizing channel, / where X and Z components are correlated through Y errors.

Error correction is regarded as successful if and only if

$$x + \hat{x} \in C_X^{\perp}, \qquad z + \hat{z} \in C_Z^{\perp}.$$

In most cases, joint BP converges and decoding finishes with linear-time complexity.

However, in rare cases, joint BP stagnates due to trapping in a small set of short cycles.
To remove the resulting error floor, we add a very light post-processing step, which I now describe.
[Next slide.]

Slide 14: Post-Processing Overview (about 2:30)

On the previous slide, we saw that joint BP is the main workhorse.
Now I explain what we do in the rare stagnation events.

Step 1: run joint BP for sufficiently many iterations. In most cases, this is enough.
Step 2: in rare cases, the number of unsatisfied syndromes does not decrease to zero and the process stagnates, typically due to trapping in short cycles, such as length-12 cycles. When this happens, we estimate the trapping cycles.
Step 3: once the cycles are identified, the remaining unknown noise components are estimated by solving a small linear system. The key point is that this system has size independent of the code length.

So the remaining question is how we estimate the trapping cycles efficiently.
[Next slide.]

Slide 15: Estimation of Trapping Cycles (abo 2:30)

On the previous slide, we introduced post-processing and said we first estimate trapping cycles.

Now I explain how we do that.

At each iteration, we track where the estimated noise has changed within the past d iterations. This defines a set $K_d^{(\ell)}$.
We also track where the syndrome values of the estimated noise have changed within the past d iterations. This defines a set $I_d^{(\ell)}$.

Empirically, for sufficiently large ℓ and d, these sets tend to cover the columns and rows of the trapping cycles.
So we can identify candidate trapping cycles from the time evolution of the decoder state, without scanning the entire code structure.

Once the cycle set is identified, we solve a small linear system to correct the remaining unknown noise components.
[Next slide.]

Slide 16: Post-Processing via a Small Linear System (about 2:30)

On the previous slide, we estimated the trapping cycles, obtaining a small set of variable indices K.
Now we solve for the remaining unknown noise on K.

To estimate X-noise, we use the restricted submatrix of H_Z on the columns in K.
We solve a small linear system over the finite field for the unknown x_K, while substituting the BP estimate outside K.

The cardinality of K is independent of the code length, so Gaussian elimination costs $O(|K|^3)$, which is effectively constant.
In our experiments, taking $|K| = 12$ was sufficient.

We do the same for Z-noise symmetrically.

Finally, we declare success if and only if

$$x + \hat{x} \in C_X^{\perp}, \qquad z + \hat{z} \in C_Z^{\perp}.$$

[Next slide.]

Slide 17: Future Works and Conclusions (about 3:00)

On the previous slide, we completed the decoding pipeline with post-processing that removes rare trapping failures.
Now I summarize and give an outlook.

We proposed a quantum LDPC coding framework inspired by non-binary LDPC codes, achieving threshold-like decoding performance close to the hashing bound.

By combining ultra-sparse structures with column weight $J = 2$, affine permutation matrices, and carefully designed non-binary symbol assignments, we successfully eliminated small logical errors.

The companion-matrix representation allows non-binary algebra to be exploited while keeping the physical code as a qubit code.

As an important future direction toward fault-tolerant quantum computation, it is crucial to incorporate realistic fault-tolerant noise models, including circuit-level noise, correlated errors, and noisy and repeated syndrome measurements.

We welcome collaborations on code constructions, decoding algorithms, noise modeling, and experimental validation toward practical

fault-tolerant quantum computation.

Thank you very much for your attention.
I would be happy to take questions.

[Next slide.]

Slide 18: Conditions for Applying Density Evolution (about 2:00)

In this slide, I explain when density evolution can be applied to quantum LDPC codes.

The essential requirement of density evolution is local tree-likeness. For a fixed number of belief-propagation iterations, the computation graph must be locally tree-like, so that incoming messages can be treated as independent and the density evolution recursion is justified.

The girth plays an important role here.
If the girth grows with the block length, this is a sufficient condition to guarantee local tree-likeness.
However, girth growth is not a necessary condition.
What is actually required is that, at any fixed depth, the neighborhood is tree-like with high probability, which can hold even when the girth itself remains bounded.

In quantum LDPC codes, there is an additional structural difficulty. In the $\mathrm{GF}(4)$ Tanner graph of stabilizer codes, commutativity constraints inevitably introduce short cycles, in particular four-cycles, which break the classical assumptions behind density evolution.

In contrast, in the factor graph used in this work, cycles of length $2L$

arising from degenerate errors do exist, but they do not directly cause decoding failures.
Moreover, no explicit upper bound on the girth is currently known, and experimentally we achieve a relatively large girth, for example sixteen.

To conclude, density evolution fundamentally relies on local tree-likeness rather than on the girth itself. When the factor graph is appropriately designed, density evolution remains a meaningful analytical tool for quantum LDPC codes, even in the presence of quantum-specific structural constraints.