

A note on unitary synthesis, DRAFT

John Bostanci

UC Berkeley & Simons Institute

We present an algorithm for unitary synthesis using $O(n/\log(n))$ queries, based on the algorithm presented by OpenAI, together with an intuitive explanation for how the algorithm works that involves minimal math. Specifically, we show that for every ε with $1/\varepsilon = \text{poly}(n)$, there is an algorithm making $O\left(\frac{n}{\log(n)}\right)$ queries (independent of ε) to a function, such that for every unitary U , there is a function that causes the algorithm to implement U up to ε -error in diamond norm. We also discuss implications for unitary complexity theory, quantum cryptography, and quantum algorithms.

Disclaimer: To be clear, the human author had little to no original contribution to the result of this work, this is just meant to be a more intuitive explanation of the original manuscript produced by OpenAI. The goal is to not just explain the “math”, but also explain how a reader might understand what is going on, and what the author believes are the important takeaways from the proof. The human author came to this understanding through repeatedly prompting OpenAI’s ChatGPT 6 Astra, and the proofs in the appendix are generated by ChatGPT 6 Astra.

1 Introduction

This note concerns the *Unitary Synthesis problem*, first posed by Aaronson and Kuperberg [AK20], as a way to potentially relate quantum and classical oracle separations to each other. The unitary synthesis problem asks whether *all* quantum computation can be reduced to *some* (however difficult) classical function, in addition to polynomial time quantum computation. In the last decade, however, work on fully-quantum cryptography has given a new perspective on the unitary synthesis problem, as people hoped that certain fully-quantum cryptographic primitives could exist even if all classical cryptography was broken. Proving the unitary synthesis problem *false* (that there is no way to synthesize all unitaries) would mean that certain hard quantum tasks might remain hard, even if *all* classical computation was free [LMW23]. Unlike unitary synthesis, the related problem of *state* synthesis has long been known to have very efficient algorithms. Aaronson presented a folklore $O(n)$ -query algorithm in his Barbados lecture [Aar16] (although this algorithm can easily be made $O(n/\log n)$ -query), but the query complexity was improved over a number of works [INN⁺22, Ros26], to get an efficient one-query algorithm for state synthesis. The lower bound of [LMW23] showed that such a 1-query algorithm for unitary synthesis, on the other hand, is impossible, potentially showing that unitary synthesis might be a fundamentally different problem than state synthesis.

Recently, however, OpenAI claimed that an internal model had found an algorithm for unitary synthesis. Although the author believes that the original manuscript [Ope26] is correct, it is not prepared in a state that is easily read by humans. The goal of this note is to explain the key ideas from the original manuscript in a slightly different, but hopefully easier to understand way, and push the query count in the original manuscript to its natural stopping point.

2 Overview

In this section, we present a high level overview of the unitary synthesis algorithm. To begin, it will be useful to have in our minds the original state synthesis algorithm from [Aar16]. The algorithm starts by decomposing a state as

$$|\psi\rangle = \alpha|0\rangle|\psi_0\rangle + \beta|1\rangle|\psi_1\rangle.$$

Here, we are letting $|\psi_0\rangle$ and $|\psi_1\rangle$ be normalized quantum states. The idea of Aaronson’s algorithm is to first prepare the state $\alpha|0\rangle + \beta|1\rangle$, and then recursively prepare both $|\psi_0\rangle$ and $|\psi_1\rangle$, controlled on the first qubit. Taking this as inspiration, we might try to find a “recursive” unitary synthesis algorithm. Exploring this idea will lead to a unitary synthesis algorithm, but we begin by explaining ideas in the context of a

related problem, “permutation synthesis”. This will make the algorithm easier to understand and almost directly translate to unitary synthesis.

Truncating a transformation

Say that we have a permutation $P = \sum_{x \in \{0,1\}^n} |\pi(x)\rangle\langle x|$, and we somehow want to write down an $n - 1$ -bit permutation P_0 that “represents” the permutation on just strings that start with 0. There is really only one option that makes sense, which we describe with the pseudocode below:

```

 $P_0(x)$ :
   $z \leftarrow 0||x$ 
  while  $P(z)$  does not start with 0:
     $z \leftarrow P(z)$ 
  return  $P(z)_{[1:]}$ 

```

Here, we are using the notation $s_{[1:]}$, where s is an n -bit string to denote the final $n - 1$ bits of s . Basically, when $P(0||x)$ would map to another string that starts with 0, for example $0||y$, we can output the string y , but if it maps to a string of the form $1||y$, we should “keep going”, and apply P to it again until we see the first string that starts with a 0.

Now, we can think about how to write this down in math. Similar to Aaronson’s recursive algorithm, we can write down any permutation (or unitary) as follows:

$$P = |0\rangle\langle 0| \otimes A + |1\rangle\langle 0| \otimes B + |0\rangle\langle 1| \otimes C + |1\rangle\langle 1| \otimes D.$$

Here, we note that the matrices (A, B, C, D) are *not* permutations (or unitaries, if P was a unitary instead) themselves, they sometimes zero out states. For example, for a $n - 1$ bit string x , if $P(0||x)$ starts with 1, then we will have that $A|x\rangle = 0$. The way that we should think about these matrices is as follows. A looks at an input x and, if $P(0||x)$ starts with a 0, it outputs $P(0||x)$, otherwise it kills the input, B looks at an input and, if $P(0||x)$ starts with a 1, outputs $P(0||x)$, otherwise it kills the input, and so on.

Then if we thought about how we could implement P_0 from above as a mathematical expression using the decomposition of P , we might end up at the following expression for the truncated permutation:

$$P_0 = A + CB + CDB + CD^2B + \dots$$

Here, the first term corresponds to strings $0||x$ such that $0||x$ maps to $0||y$. The second term accounts for strings such that x maps to $1||z$, and then $1||z$ would map to another string $0||y$, the third term accounts for strings that need to get mapped to another 1-string before going back to the 0 subspace, and so on. This expression actually generalizes to unitaries, and is sometimes called the Schur characteristic of P , which appears in the original OpenAI manuscript, which (to the best of the author’s understanding) comes from the work of Fritzsche, Katsnelson, and Kirstein [FKK08].

Recording intermediate vertices

The next step is to understand how doing P_0 can be helpful for implementing P . Intuitively, if our friend implemented P_0 for us, we might know that we started from a string $0||x$ and the result of applying P_0 is a string $0||y$, but we don’t know how $0||x$ got to $0||y$. For example, if the permutation mapped $0||x$ to $0||y$ directly, then we would be done (and output $0||y$). But if $0||x$ had a longer loop through $0||x \mapsto 1||y_1 \mapsto \dots \mapsto 0||y$, we really want to output the first string, $1||y_1$.

The most naive thing we might try is to have our friend who is implementing P_0 also write down the intermediate strings it went through. Following through with this, we would end up with the following pseudocode for a “recording version” of P_0 :

```

 $P_0^{\text{Rec}}(x, S)$ :
   $z \leftarrow 0||x$ 
  while  $P(z)$  does not start with 0:
     $z \leftarrow P(z)$ 
    Insert  $z$  into  $S$ 
  return  $(P(z)_{[1:]}, S)$ 

```

If we had a friend who implemented this, and we ran it on a string $0||x$, we would now at least be able to tell how many iterations the algorithm took, and maybe we would have some hope to partially recover the first step on the path.

Now we can think about writing this down as math. Say that we have a unitary Rec that acts as follows:

$$\text{Rec} |x, S\rangle \mapsto |x, S + \{x\}\rangle .$$

Here, we are thinking of S as a multiset, and adding two multisets increases their multiplicities (the idea of taking multisets will become important for generalizing this to unitaries, but for permutations we actually only need to take sets). Then we would write out this “recording version” of P_0 , as a mathematical expression, as follows:

$$P_0^{\text{Rec}} = A + C \text{Rec} B + C \text{Rec} D \text{Rec} B + C \text{Rec} (D\text{Rec})^2 B + \dots$$

Up until this point, it actually looks like we have made things much worse. P_0^{Rec} is an isometry that acts on *more* qubits than the original unitary (because it has to touch the set register), and implicitly runs an infinite loop. At this point, we might be better off just implementing U in exponential time. But, hopefully you (the reader) feel like we have made some conceptual progress on breaking down the problem into a reasonable sub-problem. Surprisingly, the actual unitary synthesis algorithm will follow from only one more main idea, and then a bunch of algorithmic improvements!

Dealing with the 1 subspace

From the previous subsection, we might have some vague understanding of how to handle permutation synthesis for strings of the form $0||x$, but it’s not clear a priori how to deal with strings of the form $1||x$. We could imagine directly plugging x into the permutation P_0 , but this would tell us about what happens to $0||x$, which could be completely unrelated to what happens to $1||x$. The next idea is to first “backtrack” (we are going to call the actual isometry **Backward**) to some other vertex such that running P_0 from that vertex would end up encountering $1||x$. This way, we have a sensible way to map $1||x$ to a string that P_0 expects to act on, and during the execution of P_0 , we will see our original input $1||x$, and thus we may be able to extract P from this.

We can think about what **Backward** should look like. Before writing it out, we can establish the convention that when we traverse in the forward direction, we “add” elements to the multiset, and when we traverse in the backwards direction, we “remove” elements from the multiset, and we are allowed to have negative multiplicities. Then we might write down the following pseudocode:

```

Backward( $x$ ):
   $S \leftarrow \{\}$ 
  while  $x$  does not start with 0:
    Delete  $x$  from  $S$ 
     $x \leftarrow P^\dagger(x)$ 
  return  $(x, S)$ 

```

To extend this to unitaries, it will be useful to describe what **Backward** needs to do in terms of the sub-matrices (A, B, C, D) , as we did before. The thing that matters is not the actual mathematical form of **Backward**, just the fact that this algorithm has a reasonable extension to the unitary case. In math, we have the following:

$$\text{Backward} = (|0\rangle\langle 0| \otimes \text{id} + |0\rangle\langle 1| \otimes (B^\dagger \text{Rec}^\dagger + B^\dagger \text{Rec}^\dagger D^\dagger \text{Rec}^\dagger + B^\dagger \text{Rec}^\dagger (D^\dagger \text{Rec}^\dagger)^2 + \dots)) (\text{id} \otimes |\{\}\rangle) .$$

This might look confusing, but it just describes literally what the pseudocode is doing: Create an empty set register, when the input starts with $|0\rangle$ do nothing, otherwise keep applying $D^\dagger$ (mapping between strings starting with 1) and at the end apply $B^\dagger$ (mapping from strings starting with 1 to those starting with 0), recording along the way.

The **Backward** transformation allows us to imagine running the algorithm P_0^{Rec} on inputs that start with a 1, but it still does not tell us what to do with the outputs of P_0^{Rec} . Imagine that we got an input from P_0^{Rec} . In multiset notation, the output will contain a string of the form $0||y$, as well as *part* of a path (since the negative path from **Backward** will have cancelled some of the elements that P_0^{Rec} added). If we want to figure out the element immediately after our starting element, we just have to trace backwards through the set until we have emptied the set. It turns out, this will better be described as the “inverse” of going forward through the set, which motivates us to write down the **Forward** operation:

```

Forward( $x$ ):
   $S \leftarrow \{\}$ 
  while  $x$  does not start with 0:
    Insert  $x$  into  $S$ 
     $x \leftarrow P(x)$ 
  return ( $x, S$ )

```

In math, we have a similar expression for **Forward**. Again, the actual form is not important, as when we go to implement **Forward**, we will not use anything about what it actually does, but it is important to *be able* to write down **Forward** for arbitrary unitaries.

$$\text{Forward} = (|0\rangle\langle 0| \otimes \text{id} + |0\rangle\langle 1| \otimes (C \text{ Rec} + C \text{ Rec } D \text{ Rec} + C \text{ Rec } (D \text{ Rec})^2 + \dots)) (\text{id} \otimes |\{\}\rangle) .$$

With these functions, we can actually describe the entire permutation synthesis algorithm as simply running **Backward**, recursively calling P_0^{Rec} , and then calling **Forward** † , the inverse of the forward operation. In pseudocode, we have

```

P( $x$ ):
  ( $x, S$ )  $\leftarrow$  Backward( $x$ )
  ( $x, S$ )  $\leftarrow P_0^{\text{Rec}}$ ( $x, S$ )
   $x \leftarrow \text{Forward}^\dagger(x, S)$ 
  return  $x$ 

```

Of course, at this point it is entirely not clear why this helps us do unitary synthesis, or how we can actually implement any of the operations described here. In the next subsection, we describe how to actually do efficient unitary synthesis.

Implementation 1: Removing infinite loops

The first algorithmic improvement we make is to fix all of the infinite loops in the algorithms, which will in turn allow us to use a $\text{poly}(n)$ -sized register to store our sets. We first note the following: If you want to implement a fixed permutation P , you can instead sample a random permutation Q , and implement both $Q^\dagger P$ and then Q , and the same goes for unitaries. Thus, a unitary synthesis algorithm that works for a Haar random unitary, with probability greater than $1/2$, will imply an algorithm for unitary synthesis in general. As a cool aside, note that the sub-permutation (or unitary), P_0 will also be a uniformly random (or Haar random) permutation, so we can recursively apply this argument to all of the sub-synthesis algorithms we make along the way.

We want to argue that for a random permutation, the length of the path that P_0^{Rec} needs to write down will never be much larger than $O(n)$. However, we can note that for any element, the probability that $P(x)$ maps to a string starting with 0 is exactly $1/2$. Thus, the probability that a path of length ℓ never has a string starting with 0 is at most $2^{-\ell}$ (note that if we condition on the path not going to a 0-string, we actually are slightly *more* likely than $1/2$ to end up at a 0-string). Taking $\ell = O(n)$ and using the union bound, we see that with overwhelming probability over a random permutation, P_0^{Rec} never writes down a

path of length longer than $O(n \log n)$. For a random unitary, a slightly more involved argument is required, but the main idea is that we can bound the norm, $\|D^2\|$ by a constant bounded away from 1, meaning that if we truncate the infinite sum at $O(n)$, our total error will be negligible in n . In the rest of the overview, we will use ℓ to denote the length of paths that we store.

Implementation 2: Implementing the recording oracle

The second algorithmic improvement we make is to change the way we store the multiset S . Recall the compressed oracle technique from [Zha18]. The main idea is that if we start from a uniform superposition over functions $f : \{0, 1\}^n \mapsto \mathbb{F}_q$, the following operations are equivalent up to a quantum Fourier transform on the purifying register:

$$\mathcal{O} |x, f\rangle \mapsto \omega_q^{f(x)} |x, f\rangle \quad \text{and} \quad \text{CompO} |x, S\rangle \mapsto |x, S + \{x\} \bmod q\rangle,$$

Here, the $\bmod q$ acts on the multiplicities (so, when you get q copies of a string x , you have the same state as if you had stored 0 copies of x). Notice that the second picture is exactly the recording (if we take q to be large) operation we had in our pseudocode, and that the first picture is just a controlled phase operation. Since we only need to store sets of size ℓ , we can actually replace the uniform superposition over all functions with the uniform superposition over ℓ -degree polynomials (Note that we can perfectly store ℓ points using degree ℓ polynomials). We also note that we can write down a degree ℓ polynomial by writing down all of its coefficients (of which there are $\ell + 1$ of them), and the uniform superposition over the coefficients will give us the uniform superposition over polynomials too.

Thus, we can replace our definition of P_0^{Rec} with the following phase version:

PhP₀(x, p):

```

 $z \leftarrow 0 \parallel x$ 
while  $P(z)$  does not start with 0:
     $z \leftarrow P(z)$ 
    Apply a phase of  $\omega_q^{p(z)}$ 
return  $P(z)_{[1:]}$ 

```

Here, p is a description of a low-degree polynomial, so $p(x) = \sum_i p_i x^i$, which is the form that appears in the original manuscript. From our recording oracle intuition, this is *equivalent* to the original P_0^{Rec} , up to a quantum Fourier transform in an external register that stores the set S , as long as it is used to record fewer than ℓ elements.

Implementation 3: Implementing the backward and forward oracles

Now, a reader might be wondering if we can do a similar phase-trick to remove the recording oracles from **Backward** and **Forward**. However, to implement these, we will do an even crazier thing.

The idea is that whenever you have a unitary U , and an extraction **Ext** that, given $U |i\rangle$ writes down i with probability $\geq p$ (for all i), you can implement U with roughly $O(\sqrt{1/p})$ queries to a state synthesis oracle, $|i\rangle \mapsto |i\rangle \otimes U |i\rangle$. From the work of Rosenthal [Ros26], we can implement the isometry with a constant number of queries, try to use our extractor to un-compute $|i\rangle$, and then use amplitude amplification to approximately post-select on having un-computed $|i\rangle$, after which we just have a copy of $U |i\rangle$.

To see why this applies to us, we note that the operations **Backward** and **Forward** both write down sets S , and those sets are *always* short (at most $O(n)$ elements), and *always* have the initial x in them (unless x started with 0, in which case we don't have to do anything to implement either of them). Thus, our extractor can simply output a uniformly random element of the set S , and it will guess the original input with probability at least $1/n$. Thus, we do not actually need to know anything about what the paths actually look like, we just ask the oracle to help us synthesize some state, controlled on our input, and then un-compute our original input given the state.

Putting these together, we get a relatively simple algorithm that performs unitary synthesis, making $O(n^{3/2})$ queries to an oracle (we need exactly n -rounds, and in each round we use $O(\sqrt{n})$ queries).

Disclaimer: For a “Pareto optimal” reading experience, we recommend not reading the rest of the implementation details. The algorithm so far does solve unitary synthesis, and involves all of the main ideas. Beyond this, we do much more work to get a modest saving in query complexity of the algorithm. This is also where the contents of this note diverge from the actual algorithm presented in [Ope26], which implements unitary synthesis using the amplitude amplification technique in this subsection.

Optimization 1: Finding a “polynomial action” unitary

First, we can recall how to turn the original Aaronson state synthesis algorithm into an algorithm that makes $O(n/\log n)$ queries. Instead of only using the first qubit, we can instead write our state as

$$|\psi\rangle = \sum_{x \in \{0,1\}^{\log(n)}} \alpha_x |x\rangle |\psi_x\rangle .$$

Then, our state synthesis algorithm can first prepare the state $\sum_x \alpha_x |x\rangle$ using a constant number of queries. Since there are n many amplitudes, we can ask the oracle to write them all down, prepare this state, and un-compute the parameters. Thus, making the algorithm run in $O(n/\log n)$ queries is as simple as making the part we control on $\log n$ bits instead of 1, but importantly, requires that we can synthesize the entire $\log n$ -qubit state in a single (or constant) query!

In our algorithm so far, we have some parameter ℓ that tells us the maximum length of the sets we want to store (or equivalently, when we truncate the infinite sums in the definitions of **Backward**, **Forward**, and P_0^{Rec}), and we implement the **Backward** and **Forward** isometries using $O(\sqrt{\ell})$ queries to the oracle. The parameter ℓ implicitly determines the error of our algorithm, since when we truncate at D^ℓ , we incur error that is something like $\|D^\ell\| \approx c^\ell$ for some constant c . The math works out so that ℓ has to be chosen to be at least $\text{poly}(1/\delta)$ if we want the dimension of P_0 to be $\delta 2^n$. Imagine, instead, that we could implement **Backward** and **Forward** with a constant number of queries, no matter how large ℓ was taken to be. Then the natural idea would be to take the size of D (the part we are removing in every recursive step) to be as large as possible so we need as few rounds of recursion, while still keeping δ inverse polynomial in n (because δ affects our choice of ℓ , which determines how much space our algorithm needs). Thus, if we had a constant query implementation of **Forward** and **Backward**, we would take δ to be, say $1/n$, meaning we only need $n/\log n$ rounds (and $O(n/\log n)$ queries) to implement the algorithm.

Thus, the final algorithmic improvement is a way to implement **Backward** and **Forward** using a constant number of queries. The way that the author knows how to do it is a bit complicated and makes heavy use of the particular form of the states we are trying to prepare. Let us begin by thinking about the actual isometry we want to implement. Say that the isometry **Backward** can be written in the following way:

$$\text{Backward} |x\rangle = \sum_{S \ni x} |v_{x,S}\rangle |S\rangle .$$

Here, we are just re-writing the overall state in terms of sub-normalized states $|v_{x,S}\rangle$, which will be the result of applying $B^\dagger$ to the result of going through $D^\dagger$ many times in the definition of **Backward**. Then if we imagine re-grouping the terms, so that $\beta_{x,S} = \| |v_{x,S}\rangle \|$, we can re-write our state as follows:

$$\sum_{S \ni x} \left(\frac{|v_{x,S}\rangle}{\beta_{x,S}} \right) \beta_{x,S} |S\rangle .$$

Thus, we might imagine an algorithm that does the following: First, for every x , prepare the superposition over S , $\sum_S \beta_{x,S} |S\rangle$, which would give us a state proportional to

$$|x\rangle \sum_{S \ni x} \beta_{x,S} |S\rangle .$$

With this state, we might hope that we can implement a transformation that is *controlled on S* that can map x to the normalized version of $|v_x\rangle$:

$$V_S : |x\rangle \mapsto \frac{|v_{x,S}\rangle}{\beta_{x,S}} .$$

However, this mapping might be quite complicated, and it acts over a large space. The next important observation is the following: Once we have prepared S , the set actually has a lot of information about the original input, x , since it has to be an element of S (as an aside: S also has information about the output vector, since it must be a linear combination of vectors $B^\dagger |y\rangle$ for $y \in S$). So, we should be able to compress the first register into a much smaller space. Consider the following isometry that compresses the first register, controlled on S .

$$\text{Comp}_S : |x\rangle \mapsto |\text{pos}_S(x)\rangle .$$

Here, $\text{pos}_S(x)$ is the position of x in S , if we ordered the elements of S by lexicographical order (so, for the multiset $\{1, 1, 20, 100\}$, the string 1 would get mapped to 0, 20 would get mapped to 1, and 100 would get mapped to 2).

Up to this compression isometry, the map V_S we want to implement is some map on an ℓ -dimensional subspace (note, this is ℓ *dimensions*, not qubits! So, this is an extremely tiny transformation), which is referred to as V_S having “polynomial action” by [RY21]. The paper of Rosenthal and Yuen also gives a reduction showing how to implement “polynomial action” unitaries and isometries to ε error via state synthesis (by synthesizing $O(\ell^2/\varepsilon^2)$ copies of a program state) using the density matrix exponentiation algorithm of [LMR14]. Thus, if V_S were an isometry, we could apply the algorithm of Rosenthal and Yuen to synthesize it using a constant number of oracle queries.

Optimization 2: Coarse graining to make things work

However, we are unfortunately not done yet, because the transformation we are trying to apply in the previous subsection is actually not even an isometry. Intuitively, the operation **Backward** is trying to delete a large fraction of indices, and trying to map a state $|x\rangle$ to its “projection” onto the remaining subspace. Unlike the case of permutations, two different states might end up getting mapped onto similar states, *even* when they record the same set S . Indeed, it is not hard to find cases when the inner products between the different choices of $|v_{x,S}\rangle$ are very high for different choices of x and x' . This forces us to do something somewhat un-natural (but that does work).

The last idea that we need comes from statistics, called “coarse graining”. Instead of implementing the transformation V_S for a single S , we should try to implement the transformation for many S ’s at the same time. Since we know that the *overall* transformation is an isometry, applying the transformation on a large collection of S ’s should interfere with each other and begin to look like the larger isometry. If we think about how we should coarse grain, a natural idea comes up. For a pair, S and S' of recorded sets, we can notice that if S and S' do not share any elements, then their isometries act on different subspaces, and never mix, so we probably should not do those together. So what we should do is take large blocks, T , and implement an approximate isometry V_T that represents V_S for all of the $S \subseteq T$ (except, re-weighted).

Thus, the actual algorithm starts by preparing (controlled on the input $|x\rangle$) a superposition over big sets T , and then compresses the input into a position in T , and uses the compressed register to prepare the *entire* superposition over $|y, S\rangle$ (including the paths S), and finally uncomputes the big sets T from the sets S . In our case, taking the size of T to be $O(\ell^2)$ will work (i.e., V_T will be an approximate isometry), and keep the transformation small enough to apply the algorithm of Rosenthal and Yuen.

3 Discussion and philosophizing

So now we know that unitary synthesis is possible. In some sense, the actual algorithm is not important at all, in comparison to the following question: “Why should unitary synthesis be possible?” Hopefully the community of scientists who care about quantum computing will be able to come to a better answer to this question than the author, however, I offer one way that I think about this algorithm.

We already knew how to implement isometries of the form $|i\rangle \mapsto |i\rangle |\psi_i\rangle$, and the gap between doing this and performing unitary synthesis is really the task of *uncomputing* $|i\rangle$ from $|\psi_i\rangle$. This algorithm tells us that there was actually an additional state we could have prepared, $|i\rangle \mapsto |i\rangle |\psi_i\rangle |\text{path}_i\rangle$, with the property that this magical additional state contains all the information about *both* $|i\rangle$ and $|\psi_i\rangle$ (and thus can be un-computed from either of them), and is not too long. To understand this state, we should “unroll” the entire algorithm. After all of the recursive steps in the algorithm, the algorithm will have “squeezed” all of

the information about the original unitary into the subspace where the first $n - \log n$ bits of the input are 0, it will apply a small unitary in that subspace, and then it will “reverse” the entire squeezing.

The additional state is similar to the following: Fix a small subspace Π (in our case, the subspace of the first $n - \log n$ qubits being $|0\rangle$) and output the path an arbitrary state takes to get to Π when you run $U, U^2, \dots, U^\ell$ for some fixed ℓ . For a Haar random unitary, we hope that every state will pass through Π (if it’s not too small), and will pass through in not too much time. Furthermore, this path encodes both where the state started from (i.e., $|i\rangle$), and the first step in the process (i.e., $|\psi_i\rangle$), so we can use it to uncompute $|i\rangle$, and we can use $|\psi_i\rangle$ to uncompute this path. However, it is probably too much to hope that just applying $U, U^2, \dots, U^\ell$ will map you into Π with good probability. Instead of doing this, the actual algorithm we use slowly extracts the path, and allows us to do amplitude amplification a bunch of times along the way so that we can actually do this path synthesis efficiently.

3.1 Open questions

The author believes this is just the beginning of our understanding of unitary synthesis, and this algorithm actually presents more questions than it answers. Here are a few of the questions the author had, in light of this discovery.

Optimal query unitary synthesis? Perhaps the most direct open question remaining is how many queries do you need to do unitary synthesis exactly? This paper matches the original Aaronson state synthesis algorithm, and we know from the work of Lombardi, Ma, and Wright, that there is no 1-query algorithm, but this leaves a big gap. Can we improve unitary synthesis algorithms to get below $n/\log n$, maybe even constant, query unitary synthesis? Can we improve our lower bounds to rule out 2-query non-clean algorithms?

Unitary implications below PSPACE? In the algorithm presented here, when the unitary being synthesized is in UnitaryPSPACE, it is not hard to see that the oracle itself can be implemented in PSPACE. [BEM⁺25] showed that the existence of any falsifiable quantum cryptography requires that UnitaryBQP is not UnitaryPSPACE, which, combined with this algorithm, implies that the existence of falsifiable quantum cryptography implies $P \neq PSPACE$. They also showed that certain concrete assumptions, like EFI pairs, require that UnitarySZK is not equal to UnitaryBQP. What is the complexity required to implement the oracle in this note, when the unitary is in UnitarySZK? What is the connection between the existence of EFI pairs and classical complexity theory? Can we break EFI pairs with oracle access to PP, what about QSZK?

Other unitary synthesis algorithms? While the note from OpenAI provides the first known unitary synthesis algorithm, it is unlikely that this recursive approach is the only way to do unitary synthesis. In state synthesis, for example, algorithms making a constant number of queries first prepare a *phase state*, and then apply an efficient transformation to map it to a target state. Is there a similarly non-recursive algorithm for unitary synthesis (even if it makes more than a constant number of queries)? Additionally, the final step of the algorithm presented in this note is certainly not the cleanest way to achieve $O(n/\log n)$ queries, is there a more conceptually nice way to modify this algorithm to achieve the same query bound?

4 Preliminaries

Here, we define a few operations related to the unitary synthesis algorithm.

Definition 4.1 (Phased truncation of a unitary). *Given a unitary U , subspace Π , and diagonal unitary Z , write the unitary $U = \begin{pmatrix} A & C \\ B & D \end{pmatrix}$, where A acts in the Π subspace, we define the phase truncation of U , $\text{Trunc}_{U,\Pi}(Z)$ as*

$$\text{Trunc}_{U,\Pi}(Z) = A + \sum_{k=0}^{\infty} CZ(DZ)^k B.$$

Next we define the notation around polynomials and polynomial set states. We use the notation $\mathcal{P}_{\leq \ell}$ to denote the set of univariate polynomials of degree $\leq \ell$. For a polynomial p , written as $p(x) = a_0 + a_1x + a_2x^2 + \dots + a_\ell x^\ell$, we write the polynomial basis state $|p\rangle = |a_0, \dots, a_\ell\rangle$.

Definition 4.2 (Polynomial phase unitary). *Given a polynomial p from $\{0, 1\}^n \mapsto \mathbb{F}_q$, where q is a large prime, we define the polynomial phase unitary to be the diagonal unitary that acts as follows,*

$$F_p |x\rangle \mapsto \omega_q^{p(x)} |x\rangle.$$

Definition 4.3 (Polynomial phase recording set). *Given a multiset S of we use the notation $m_S(x)$ to denote the multiplicity of x in S (which can be negative). Let $\sum_{x \in \text{supp}(S)} |m_S(x)| \leq \ell$, and let q be a prime larger than 2ℓ , then we define the multiset recording basis to be*

$$|S\rangle = \frac{1}{q^{(\ell+1)/2}} \sum_{p \in \mathcal{P}_{\leq \ell}} \omega_q^{\sum_{x \in \text{supp}(S)} m_S(x)p(x)} |p\rangle.$$

As a matter of notation, in the rest of this paper we will split the n -bit strings into the first $2^n/n$ that start with $0^{\log n}$, and the remaining. We call this remaining set Del (for deleted). We now present the intended Backward operation, which has an infinite sum.

Definition 4.4 (Ideal backward isometry). *Given a unitary U , written as $U = \begin{pmatrix} A & C \\ B & D \end{pmatrix}$, where A is a $\frac{1}{n}2^n \times \frac{1}{n}2^n$ matrix. Then the intended Backward acts on an input $x \in \{0, 1\}^n$ as*

$$\text{Backward} |x\rangle = \begin{cases} |x\rangle \otimes |\{\}\rangle, & x_{[\log(n)]} = 0^{\log(n)}, \\ \sum_{m=1}^{\infty} \sum_{\substack{z_1, \dots, z_m \in \text{Del} \\ z_1 = x}} \left(\prod_{j=1}^{m-1} \langle z_{j+1} | D^\dagger | z_j \rangle \right) B^\dagger |z_m\rangle \otimes |-\{z_1, \dots, z_m\}\rangle, & \text{otherwise.} \end{cases}$$

Here, the second register is an infinite dimensional register.

Then we can define the truncated backwards isometry, which has the infinite sum stopping at ℓ for some parameter ℓ .

Definition 4.5 (Truncated backward isometry). *Given a unitary U with the same decomposition and a positive integer ℓ , the truncated backward isometry, Backward_ℓ is given by*

$$\text{Backward}_\ell |x\rangle = \begin{cases} |x\rangle \otimes |\{\}\rangle, & x_{[\log(n)]} = 0^{\log(n)}, \\ \sum_{m=1}^{\ell} \sum_{\substack{z_1, \dots, z_m \in \text{Del} \\ z_1 = x}} \left(\prod_{j=1}^{m-1} \langle z_{j+1} | D^\dagger | z_j \rangle \right) B^\dagger |z_m\rangle \otimes |-\{z_1, \dots, z_m\}\rangle, & \text{otherwise.} \end{cases}$$

The forward operation is defined similarly, using D and C in place of $D^\dagger$ and $B^\dagger$, and recording the visited strings with positive multiplicities.

Definition 4.6 (Ideal forward isometry). *Given a unitary U with the same decomposition, the intended Forward acts on an input $x \in \{0, 1\}^n$ as*

$$\text{Forward} |x\rangle = \begin{cases} |x\rangle \otimes |\{\}\rangle, & x_{[\log(n)]} = 0^{\log(n)}, \\ \sum_{m=1}^{\infty} \sum_{\substack{z_1, \dots, z_m \in \text{Del} \\ z_1 = x}} \left(\prod_{j=1}^{m-1} \langle z_{j+1} | D | z_j \rangle \right) C |z_m\rangle \otimes |\{z_1, \dots, z_m\}\rangle, & \text{otherwise.} \end{cases}$$

Here, the second register is an infinite dimensional register.

Then we can define the truncated forward isometry, which has the infinite sum stopping at ℓ for some parameter ℓ .

Definition 4.7 (Truncated forward isometry). *Given a unitary U with the same decomposition and a positive integer ℓ , the truncated forward isometry, Forward_ℓ is given by*

$$\text{Forward}_\ell |x\rangle = \begin{cases} |x\rangle \otimes |\{\}\rangle, & x_{[\log(n)]} = 0^{\log(n)}, \\ \sum_{m=1}^{\ell} \sum_{\substack{z_1, \dots, z_m \in \text{Del} \\ z_1 = x}} \left(\prod_{j=1}^{m-1} \langle z_{j+1} | D | z_j \rangle \right) C |z_m\rangle \otimes |\{z_1, \dots, z_m\}\rangle, & \text{otherwise.} \end{cases}$$

5 The unitary synthesis algorithm

As stated in the overview, the entire unitary synthesis algorithm is very easy to describe in terms of two subroutines, **Backward** and **Forward**. We will fix parameters ℓ and r , and we assume that the classical oracle $\mathcal{O}$ has $n/\log(n)$ many inputs of length r . We use the notation $\mathcal{O}(x, \cdot)$ to denote oracle access to the function that corresponds to $\mathcal{O}$ with its first input hard-coded to x .

Synthesize $^\mathcal{O}$ (A):

If the dimension of A is less than n^{100} :
 Ask the oracle for an explicit description of U and implement it
 Return A
 Run $\text{Backward}_{\ell, r}^\mathcal{O}(A)$ to get a pair of registers AB
 Controlled on a polynomial p appearing in B , run controlled-Synthesize $^{\mathcal{O}(p, \cdot)}(A_{[\log(n):]})$
 Run $\text{Forward}_{\ell, r}^{\dagger, \mathcal{O}}(AB)$ to get a register A
 Return A

We can first prove that if there is an algorithm that implements Backward_ℓ and Forward_ℓ from Definitions 4.5 and 4.7, then the unitary synthesis algorithm succeeds.

Theorem 5.1. *Let $\ell = n^4$ and $r = O(\ell^2)$. Then there exists an oracle such that $\text{Backward}_{\ell, r}^\mathcal{O}(A)$ and $\text{Forward}_{\ell, r}^\mathcal{O}(A)$ implement the isometries Backward_ℓ and Forward_ℓ , to error ε/n using $\text{poly}(n, 1/\varepsilon)$ space and time, and the controlled synthesis algorithm implements $\text{Trunc}_{U, \text{id}-\text{Del}}(F_p)$, the phased truncation of U that uses the polynomial phase unitary. Relative to this oracle, Synthesize $^\mathcal{O}$ implements U to error ε .*

Proof. This follows from the definitions of the idealized operators **Backward**, **Forward**, and $\text{Trunc}_{U, \text{id}-\text{Del}}(F_p)$, up to a Fourier transform in the multiset recording register. From Lemma A.1, the operator norm distance between Backward_ℓ and **Backward** (and the same for **Forward**) for our choice of $\ell = n^4$ is bounded by $2^{-\Omega(n/\log n)}$. $\square$

We also argue that the unitary synthesis algorithm succeeds with high probability when given a Haar random unitary to synthesize. From the argument in the overview, this means that for *every* n -qubit unitary U , we can choose some V and synthesize both V and $V^\dagger U$, so applying this algorithm twice does synthesize every unitary.

Theorem 5.2. *The synthesis algorithm here succeeds with probability at least 0.9 over a Haar random unitary.*

Proof. As long as the unitary the algorithm is trying to synthesize has a small bottom corner, and the implementation of **Backward** succeeds, for the original U and all of the sub-unitaries, the algorithm will successfully synthesize U . Lemma A.2 shows that each of the sub-unitaries will be a Haar random unitary of dimension at least n^{100} (because the algorithm stops at n^{100}), so we can take a union bound over all the small unitaries, along with Lemma A.1 and Lemma A.4, to bound the total failure probability by (say) 0.1 over a Haar random unitary. $\square$

The main interesting parts come from implementing **Backward** and **Forward**, and proving that there does exist a classical function such that for every unitary U , this algorithm implements U . We first describe the implementation of **Backward** and **Forward**.

5.1 The Backward and Forward subroutines

As alluded to at the *end* of the overview, the **Backward** and **Forward** isometries are going to be quite complicated in order to get them to only make a constant number of queries to their oracle. From the perspective of the synthesizer, both algorithms are actually the same (just that the oracle queries will do different things in each case). We describe the algorithm in pseudocode below.

Backward $_{\ell,r}^{\mathcal{O}}(\mathbf{A})$, and Forward $_{\ell,r}^{\mathcal{O}}(\mathbf{A})$:

Prepare a register in the state $|\perp\rangle$ and perform a constant number of rounds of amplitude amplification using the following subroutine:

Controlled on **A**:

Run the algorithm from [Ros26] to synthesize a state on **TM**

Apply the isometry $\text{Comp}_{\text{AT}} : |x, T\rangle \mapsto |\text{pos}_T(x), T\rangle$

Controlled on registers **TM** being in the state $|T, m\rangle$:

Query the oracle to receive an $r \times r$ matrix $M_{T,m}$ and initialize a new register **D** in the state $|\perp\rangle$

Apply the unitary $|\psi\rangle \mapsto M_{T,m} |\psi, \top\rangle + \sqrt{1 - M_{T,m}^\dagger M_{T,m}} |\psi, \perp\rangle$ to the **A** register

Uncompute the matrix $M_{T,m}$

Run the algorithm from [RY21], with [Ros26] for state synthesis to get registers **AS**

Apply the isometry $|-S\rangle_S ||S\rangle_M \mapsto \frac{1}{\sqrt{N_{|\text{supp}(S)|,r}}} \sum_{\substack{T \subseteq \text{Del}, |T|=r \\ \text{supp}(S) \subseteq T}} |T\rangle_T \mapsto |-S\rangle_S$ to uncompute the **TM** registers.

The important part of the algorithm is filling in what state and matrix the oracle $\mathcal{O}$ should help us prepare. We present those below in a sequence of definitions in the next section.

5.2 Describing the oracle

We can define a sequence of transformations and oracles that implement the “coarse graining” that the overview talks about. For a fixed string x , we imagine re-writing the result of applying the truncated Backward_ℓ to $|x\rangle$, $\text{Backward}_\ell |x\rangle$ as follows:

$$\text{Backward}_\ell |x\rangle = \sum_{|S| \leq \ell} |v_{x,S}\rangle |-S\rangle.$$

Here, $|v_{x,S}\rangle$ captures all of the information about the paths through $D^\dagger$ that is in Definition 4.5. We can now define the following map (note that this will not be an isometry, as stated in the overview), that “uncompresses” a tiny string into this state. Here, the input i is meant to be read as an index into a big set T (which is how we do the coarse-graining).

Definition 5.3 (Uncompress map). *We define the map Uncompress , for a fixed large set T and m , as follows:*

$$\text{Uncompress}_{T,m} |i\rangle \mapsto \sum_{\substack{S \subseteq T \\ |S|=m}} \frac{1}{\sqrt{N_{|\text{supp}(S)|,r}}} |v_{T_i,S}\rangle |-S\rangle.$$

Here, T_i is the i 'th element of T in lexicographical ordering, and the sum over S is over all multi-subsets of T with total size m , but the normalization inside the sum depends on the size of the support of S (as a set).

Note that $\text{Uncompress}_{T,m}$ acts on an r -dimensional input register, and outputs two registers, the first of which is n qubits, and the second one stores multisets of total size m . With the map Uncompress , we can now give the definition of the correction matrix $M_{T,m}$ that the algorithm actually applies.

Definition 5.4 (Coarse-graining correction matrix). *For a fixed T and m , we first define the Gram matrix, $\text{Gram}_{T,m}$ as follows:*

$$\text{Gram}_{T,m} = \text{Uncompress}_{T,m}^\dagger \text{Uncompress}_{T,m}.$$

This Gram matrix captures how far from being an isometry the uncompress map is on the input r -dimensional subspace. For entries of $\text{Gram}_{T,m}$ and $\text{Weight}_{T,m}$, $|x\rangle$ with $x \in T$ denotes the compressed basis vector $|\text{pos}_T(x)\rangle$. We also define a diagonal weight matrix, $\text{Weight}_{T,m}$ as follows:

$$\langle x | \text{Weight}_{T,m} | x \rangle = \langle x | \text{Gram}_{T,m} | x \rangle + \sum_{z \in T \setminus \{x\}} |\langle z | \text{Gram}_{T,m} | x \rangle|.$$

We define the diagonal total-weight matrix W on the deleted input space by

$$\langle x | W | x \rangle = \sum_{m=1}^{\ell} \sum_{\substack{T \subseteq \text{Del} \\ |T|=r, x \in T}} \langle x | \text{Weight}_{T,m} | x \rangle.$$

Then the intended matrix $M_{T,m}$ applied in the algorithm $\text{Backward}_{\ell,r}$, controlled on T and m , is given by

$$M_{T,m} = \text{Gram}_{T,m}^{1/2} \cdot \text{Weight}_{T,m}^{-1/2} \cdot \frac{\text{Comp}_T W^{1/2} \text{Comp}_T^\dagger}{\sqrt{2}}.$$

The last factor is $W^{1/2}/\sqrt{2}$ restricted to T in the compressed basis. We take $\text{Weight}_{T,m}^{-1/2}$ to be zero on zero-weight coordinates.

At a high level, we would really like to have implemented the map $\text{Uncompress}_{T,m}$. However, this map by itself will not be an isometry, so the idea is that we first re-weight the input state according to Gram and Weight , which removes the “non-isometry” part of $\text{Uncompress}_{T,m}$, and after that, there will be some isometry we can apply in our small space that gives us the result of applying $\text{Uncompress}_{T,m}$. At this point, however, we have not even defined the initial state that the algorithm should synthesize (in the first state synthesis step). We define that below,

Definition 5.5 (Initial coarse-graining state). *For $x \in \text{Del}$ with positive total weight, we define the initial state that the algorithm should synthesize (controlled on x) as follows:*

$$|\text{Init}_x\rangle = \frac{1}{\sqrt{\langle x | W | x \rangle}} \sum_{m=1}^{\ell} \sum_{\substack{T \subseteq \text{Del} \\ |T|=r, x \in T}} \sqrt{\langle x | \text{Weight}_{T,m} | x \rangle} |T, m\rangle.$$

Retained inputs use a separate empty-record branch, with success amplitude $1/\sqrt{2}$ and identity action on the input.

We also define the following operation, Expand , that takes a set S and outputs the uniform superposition over big sets T that it is contained in.

Definition 5.6 (Set expanding isometry). *For every m and r , we define the number $N_{m,r} = \binom{(1-1/n)2^n - m}{r-m}$, which represents the number of sets T of size r (with elements from Del) that contain a given set of size m . The set expanding isometry, Expand_r , acts on a register storing a multiset S of size m as follows,*

$$\text{Expand}_r | -S \rangle \mapsto | -S \rangle \otimes \frac{1}{\sqrt{N_{|\text{supp}(S)|,r}}} \sum_{\substack{T \subseteq \text{Del} \\ |T|=r \\ \text{supp}(S) \subseteq T}} |T, m\rangle$$

Here, $m = |S|$ counts multiplicities, whereas the normalization counts supersets of the support of S .

This set expanding isometry is the operation that the algorithm inverts at the end of $\text{Backward}_{\ell,r}^\mathcal{O}$, but it is never actually called in the forward direction. The idea is that the state synthesis (preparing $|\text{Init}_x\rangle$) has already had Expand_r applied to it.

The main theorem we have to prove is that (a) controlled on having applied $M_{T,m}$, the remaining transformation is an isometry, and (b) that the probability of seeing $\top$ in the algorithm is a constant. We prove these in the appendix since they are mostly math. Applying these gives us the following theorem.

Theorem 5.7 ($\text{Backward}_{\ell,r}$ succeeds). *When U is a Haar random unitary, with high probability over U , there exists a choice of oracle $\mathcal{O}$ that causes the algorithm $\text{Backward}_{\ell,r}^{\mathcal{O}}$ to approximately implement the truncated backward isometry, Backward_{ℓ} , only making a constant number of queries.*

Proof. Conditioned on measuring $\top$ after the amplitude amplification, the algorithm will have applied the matrix $M_{T,m}$ to the state it prepared ($|\text{Init}\rangle$), after which, by inspection, it will have a state that is exactly proportional to

$$\text{Gram}_{T,m}^{-1/2} \cdot \text{Uncompress}_{T,m}^{\dagger} \cdot \text{Expand}_r \cdot \text{Backward}_{\ell} |x\rangle$$

By the definition of the Gram matrix, the isometry $V_{T,m} = \text{Uncompress}_{T,m} \cdot \text{Gram}_{T,m}^{-1/2}$ will be an isometry, and will leave us with exactly the result of applying Expand_r to the desired state, so uncomputing expand will get the desired state.

Lemma A.3 and Lemma A.4 show that with high probability over a Haar random U , the number of rounds of amplitude amplification to get $\top$ is a constant. $\square$

An analogous statement holds for $\text{Forward}_{\ell,r}$ for the same choice of ℓ and r . We also note that we can choose the error ε to be an arbitrary inverse polynomial error, because the only time ε appears is when we apply the polynomial action unitary synthesis algorithm from [RY21], which requires $O(\text{poly}(n)/\varepsilon^2)$ copies of the program state (which we can synthesize in a constant number of queries, independent of ε).

References

- [Aar16] Scott Aaronson. The complexity of quantum states and transformations: From quantum money to black holes, 2016. [1](#)
- [AK20] Scott Aaronson and Greg Kuperberg. Quantum versus classical proofs and advice, 2020. [1](#)
- [BEM⁺25] John Bostanci, Yuval Efron, Tony Metger, Alexander Poremba, Luowen Qian, and Henry Yuen. Unitary complexity and the Uhlmann transformation problem, 2025. [8](#)
- [FKK08] Bernd Fritzsche, Victor Katsnelson, and Bernd Kirstein. The Schur algorithm in terms of system realization, 2008. [2](#)
- [INN⁺22] Sandy Irani, Anand Natarajan, Chinmay Nirkhe, Sujit Rao, and Henry Yuen. Quantum search-to-decision reductions and the state synthesis problem. volume 234, pages 5:1–5:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. [1](#)
- [KKS16] Mario Kieburg, Arno B. J. Kuijlaars, and Dries Stivigny. Singular value statistics of matrix products with truncated unitary matrices. *International Mathematics Research Notices*, 2016(11):3392–3424, 2016. [19](#)
- [LMR14] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, 2014. [7](#)
- [LMW23] Alex Lombardi, Fermi Ma, and John Wright. A one-query lower bound for unitary synthesis and breaking quantum cryptography. Cryptology ePrint Archive, Paper 2023/1602, 2023. [1](#)
- [Mec19] Elizabeth S. Meckes. *The Random Matrix Theory of the Classical Compact Groups*, volume 218 of *Cambridge Tracts in Mathematics*. Cambridge University Press, 2019. [14](#), [19](#)
- [Ope26] OpenAI. Polynomial-Time Unitary Synthesis from a Boolean Oracle. OpenAI Math Release preprint [OAI:Polynomial-Time-Unitary-Synthesis-from-a-Boolean-Oracle-October-5-2026](#), 2026. [1](#), [6](#)
- [Ros26] Gregory Rosenthal. Efficient quantum state synthesis with one query, 2026. [1](#), [5](#), [11](#)
- [RY21] Gregory Rosenthal and Henry Yuen. Interactive proofs for synthesizing quantum states and unitaries, 2021. [7](#), [11](#), [13](#)
- [Zha18] Mark Zhandry. How to record quantum queries, and applications to quantum indistinguishability. Cryptology ePrint Archive, Paper 2018/276, 2018. [5](#)

A AI generated math facts

Here, we fill in some math facts needed to make the algorithm work. Most of them involve using the strong concentration properties of Haar random unitaries.

Disclaimer: The proofs are entirely AI generated. The author believes they are true and has verified them. However, since their proofs are not actually important for understanding the algorithm, the author did not write them in the same intuitive way as the main body of the note.

Lemma A.1 (Blocks of Haar random unitaries shrink). *There are absolute constants $c, C_0, C_1, C_2 > 0$ with the following property. Let U be an n -qubit Haar random unitary, let $0 < \delta \leq 1/2$, and suppose $2^n \geq C_0 \delta^{-4}$. Let Π be a fixed projection onto $(1 - \delta)2^n$ computational basis states, and let $D = \Pi U \Pi$ restricted to $\text{im}(\Pi)$. For*

$$\beta = \lceil C_1 \delta^{-3} \log(2/\delta) \rceil,$$

we have

$$\Pr[\|D^\beta\| \geq 1/2] \leq \frac{C_2}{\delta} e^{-c\delta 2^n}.$$

Proof. Write $N = 2^n$, $H = \text{im}(\Pi)$, $Q = I - \Pi$, and $k = \delta N$. The idea is that a vector supported on H cannot put all of its spectral mass on a short arc of the unit circle. This will bound the numerical radius

$$w(D) = \sup_{\substack{v \in H \\ \|v\|=1}} |\langle v, Dv \rangle|$$

away from 1. In the proof, c, C denote positive absolute constants that may change from line to line.

Choose $a = a_0 \delta$, where $a_0 > 0$ is a sufficiently small constant. Take $O(1/\delta)$ arcs of angular radius $2a$ whose centers form an $a/4$ -net of the circle. We first show that every such arc contains at most κk eigenvalues, where $\kappa > 0$ is a sufficiently small constant. For each arc J , choose a function f_J taking values in $[0, 1]$, equal to 1 on J , supported on the concentric arc of radius $3a$, and with Lipschitz constant at most C/a . If $\lambda_1, \dots, \lambda_N$ are the eigenvalues of U , set

$$F_J(U) = \frac{1}{N} \sum_{i=1}^N f_J(\lambda_i).$$

Rotational invariance gives $\mathbb{E}F_J \leq Ca$, and the Hoffman–Wielandt inequality gives Lipschitz constant at most $C/(a\sqrt{N})$ in Hilbert–Schmidt distance. Haar concentration [Mec19, Theorem 5.17] says that an L -Lipschitz real function satisfies

$$\Pr[F - \mathbb{E}F \geq t] \leq 2\exp(-cNt^2/L^2).$$

Taking a_0 small enough that $Ca \leq \kappa\delta/2$, the probability that J contains more than κk eigenvalues is therefore at most $2e^{-cN^2\delta^4}$.

Conditional on the eigenvalues, the eigenspace E_J belonging to J is a Haar random subspace. Suppose its dimension is $q \leq \kappa k$. If $q > 0$, an orthonormal frame for E_J has the distribution

$$V = G(G^\dagger G)^{-1/2},$$

where G is an $N \times q$ standard complex Gaussian matrix. Gaussian tail bounds, applied on a fixed-radius net of the unit sphere in $\mathbb{C}^q$, give

$$\sigma_{\min}(QG) \geq c\sqrt{k}, \quad \|G\| \leq C\sqrt{N},$$

except with probability Ce^{-ck} : the net has size $e^{O(q)}$, which is absorbed by the Gaussian tails when κ is small enough. Hence $\sigma_{\min}(QV) \geq c\sqrt{\delta}$. Thus every unit vector in E_J has at least $c\delta$ squared mass in $\text{im}(Q)$. Taking adjoints of the projection from E_J to H gives

$$\|P_{E_J} v\|^2 \leq 1 - c\delta \quad \text{for every unit vector } v \in H.$$

This inequality is also immediate when $q = 0$.

A union bound over the arcs shows that all these conclusions hold except with probability at most

$$\frac{C}{\delta} \left(e^{-cN^2\delta^4} + e^{-c\delta N} \right) \leq \frac{C}{\delta} e^{-c\delta N},$$

where the dimension assumption implies $N\delta^3 \geq C$. On this event, fix a unit vector $v \in H$ and an angle θ . Choose an arc whose center is within $a/4$ of θ . At least $c\delta$ of the spectral mass of v lies outside this arc, where the angular distance from θ is at least a . Since $1 - \cos(a) \geq ca^2$, it follows that

$$\operatorname{Re}(e^{-i\theta} \langle v, Uv \rangle) \leq 1 - c\delta a^2 \leq 1 - c\delta^3.$$

Maximizing over θ and v gives $w(D) \leq 1 - g$, where $g = c\delta^3$.

Finally, on the circle $|z| = \rho := 1 - g/2$, for every unit vector $v \in H$,

$$\|(zI - D)v\| \geq |\langle v, (zI - D)v \rangle| \geq |z| - w(D) \geq g/2.$$

Thus $\|(zI - D)^{-1}\| \leq 2/g$. All eigenvalues of D lie inside this circle, so Cauchy's integral formula yields

$$\|D^\beta\| \leq \frac{2}{g} \rho^{\beta+1} \leq \frac{2}{g} e^{-g(\beta+1)/2} < \frac{1}{2}$$

for sufficiently large C_1 . Combining this with the exceptional probability above proves the claim. $\square$

Lemma A.2 (Truncations are also Haar random unitaries). *Let U be a Haar random unitary and Π be a fixed projector onto any number of computational basis states. Write $U = \begin{pmatrix} A & C \\ B & D \end{pmatrix}$ with respect to $\operatorname{im}(\Pi) \oplus \ker(\Pi)$. For a fixed diagonal unitary Z on $\ker(\Pi)$, $\operatorname{Trunc}_{U,\Pi}(Z)$ is defined almost surely and is a Haar random unitary on $\operatorname{im}(\Pi)$.*

Proof. Write $k = \operatorname{rank}(\Pi)$ and $s = \dim \ker(\Pi)$. The cases $k = 0$ and $s = 0$ are immediate, so assume both are positive and abbreviate the truncation by $F_Z(U)$.

First, the inverse exists almost surely. The unitary $W = U(I_k \oplus Z)$ is Haar random and almost surely has no eigenvalue 1. Indeed, multiplying a Haar unitary by an independent uniform global phase preserves its law, and almost every such phase moves all its eigenvalues away from 1. If $DZh = h$ for some nonzero h , then

$$W \begin{pmatrix} 0 \\ h \end{pmatrix} = \begin{pmatrix} CZh \\ h \end{pmatrix}.$$

Norm preservation forces $CZh = 0$, making $(0, h)$ an eigenvector of W with eigenvalue 1, a contradiction.

Next, for any $v \in \operatorname{im}(\Pi)$, set $h = (I - DZ)^{-1}Bv$. Then

$$U \begin{pmatrix} v \\ Zh \end{pmatrix} = \begin{pmatrix} F_Z(U)v \\ h \end{pmatrix}.$$

Since U and Z are unitary, $\|v\|^2 + \|h\|^2 = \|F_Z(U)v\|^2 + \|h\|^2$. Therefore $F_Z(U)$ preserves norms and, being a square matrix, is unitary.

Finally, let L be any fixed unitary on $\operatorname{im}(\Pi)$. Left multiplication by $L \oplus I_s$ changes the blocks A, C to LA, LC and leaves B, D unchanged. Consequently,

$$F_Z((L \oplus I_s)U) = LA + LCZ(I - DZ)^{-1}B = LF_Z(U).$$

Because $(L \oplus I_s)U$ has the same law as U , the distribution of $F_Z(U)$ is left invariant on $U(k)$. Uniqueness of Haar probability measure proves the claim. $\square$

The final few lemmas show that our coarse graining approach both implements the desired isometry, $\operatorname{Backward}_\ell$, and only requires a constant number of queries, if a certain blowup factor is bounded.

Lemma A.3 (Success probability of the algorithm, before amplitude amplification). *We can define the blowup of a unitary U as follows,*

$$\text{Blowup}_{\ell,r}(U) = \max_{x \in \text{Del}} \sum_{m=1}^{\ell} \sum_{\substack{T \subseteq \text{Del} \\ |T|=r \\ x \in T}} \langle \text{pos}_T(x) | \text{Weight}_{T,m} | \text{pos}_T(x) \rangle.$$

Suppose $\ell \leq r \leq |\text{Del}|$ and

$$\text{Backward}_{\ell}^{\dagger} \text{Backward}_{\ell} \succeq (1 - \eta)I, \quad 0 \leq \eta \leq 1/2.$$

If $\text{Blowup}_{\ell,r}(U) \leq 2$, a single ideal run with the preparation in Definition 5.5 succeeds with probability at least $(1 - \eta)/2 \geq 1/4$ on every normalized input. The same statement holds for Forward_{ℓ} using its corresponding weights and lower bound.

Proof. Recall that Comp_T maps $|x\rangle$ to $|\text{pos}_T(x)\rangle$ for $x \in T$; in the expressions below, we extend it by zero outside T . First, $\text{Weight}_{T,m} \succeq \text{Gram}_{T,m}$: for every vector $|v\rangle = \sum_i v_i |i\rangle$, bounding each off-diagonal term using $2|v_i||v_j| \leq |v_i|^2 + |v_j|^2$ gives

$$\begin{aligned} \langle v | \text{Gram}_{T,m} | v \rangle &\leq \sum_i \left((\text{Gram}_{T,m})_{ii} + \sum_{j \neq i} |(\text{Gram}_{T,m})_{ij}| \right) |v_i|^2 \\ &= \langle v | \text{Weight}_{T,m} | v \rangle. \end{aligned}$$

Thus $\text{Gram}_{T,m}^{1/2} \text{Weight}_{T,m}^{-1/2}$ is a contraction. Since $\|W\| = \text{Blowup}_{\ell,r}(U) \leq 2$, the restriction of $W^{1/2}/\sqrt{2}$ to T is also a contraction. Therefore $M_{T,m}$ has the success/failure completion used in the algorithm. A zero weight forces the corresponding column of $\text{Gram}_{T,m}^{1/2}$ to vanish, so the inverse on positive support causes no difficulty.

Superset normalization gives $\langle x | W | x \rangle \geq \|\text{Backward}_{\ell} |x\rangle\|^2 \geq 1 - \eta > 0$, so each initial state is well defined. For an input $|v\rangle$ supported on Del , the normalized preparation in Definition 5.5 followed by compression is

$$\sum_{T,m} \text{Weight}_{T,m}^{1/2} \text{Comp}_T W^{-1/2} |v\rangle \otimes |T, m\rangle.$$

Since the weight matrix and the restriction of W are both diagonal in the compressed input basis, applying the successful block $M_{T,m}$ cancels both the weights and the initial normalization, leaving

$$\frac{1}{\sqrt{2}} \sum_{T,m} \text{Gram}_{T,m}^{1/2} \text{Comp}_T |v\rangle \otimes |T, m\rangle.$$

To compute its squared norm, note that a multiset S occurs in exactly $N_{|\text{supp}(S)|,r}$ supersets T . This cancels its squared normalization in $\text{Uncompress}_{T,m}$. Also, $|v_{x,S}\rangle = 0$ unless $x \in \text{supp}(S)$. Orthogonality of the record states therefore gives

$$\begin{aligned} \sum_{T,m} \langle v | \text{Comp}_T^{\dagger} \text{Gram}_{T,m} \text{Comp}_T | v \rangle &= \sum_{1 \leq |S| \leq \ell} \left\| \sum_{x \in \text{Del}} \langle x | v \rangle |v_{x,S}\rangle \right\|^2 \\ &= \|\text{Backward}_{\ell} |v\rangle\|^2. \end{aligned}$$

By the definition of the Gram matrix, the map $\text{Gram}_{T,m}^{1/2} |i\rangle \mapsto \text{Uncompress}_{T,m} |i\rangle$ preserves inner products on its support and hence extends to an isometry. Applying this isometry controlled on T, m , then undoing the superset expansion, turns the successful component into $\text{Backward}_{\ell} |v\rangle / \sqrt{2}$. The separate empty-record branch gives the same factor $1/\sqrt{2}$ on retained inputs and is orthogonal to all nonempty records. Hence, for every normalized input,

$$p_{\text{T}}(v) = \frac{1}{2} \|\text{Backward}_{\ell} |v\rangle\|^2 \geq \frac{1 - \eta}{2}.$$

The normalization correction therefore gives the common factor $1/\sqrt{2}$ coherently, including on superpositions of different inputs, and is absorbed into the same matrix-description query. Replacing the backward path vectors by the forward path vectors proves the forward statement.

For completeness, the constant-query amplification uses phases matched to this factor. Let $\mathcal{A}$ be a unitary completion of the boxed attempt, and let Π_{in} and $\Pi_{\top}$ project onto its initialized-input and successful subspaces. Use the phase gates $R_{\text{in}} = I + (i - 1)\Pi_{\text{in}}$ and $R_{\top} = I + (i - 1)\Pi_{\top}$. Since the truncated map is a contraction, its squared singular values satisfy $1 - \eta \leq t \leq 1$. On the corresponding singular mode, the success amplitude of $\mathcal{A}$ is $\sqrt{t/2}$. One phase-matched round $\mathcal{A}R_{\text{in}}\mathcal{A}^{\dagger}R_{\top}\mathcal{A}$, after removing a fixed global phase, has successful coefficient

$$\sqrt{t} \left(1 + \frac{(1-i)(1-t)}{2} \right)$$

and failure amplitude of magnitude $\sqrt{1-t/2}(1-t)$. Its squared distance from the desired successful coefficient $\sqrt{t}$ with no failure component is

$$\frac{t}{2}(1-t)^2 + \left(1 - \frac{t}{2}\right)(1-t)^2 = (1-t)^2 \leq \eta^2.$$

Thus one round, followed by the isometry and uncomputation above, approximates the truncated map within operator norm η , using three calls to the attempt or its inverse. In particular, the query count remains constant. $\square$

Finally, we show that for Haar random unitaries, the blowup factor is small with high probability.

Lemma A.4 (Random unitaries have small blowup). *There are absolute constants $K, c > 0$ such that the following holds for all sufficiently large n . Let $d \geq n^{100}$, $n^2 \leq \ell \leq n^4$, and $r = \lceil K\ell^2 \rceil$. Fix a coordinate subspace of dimension d/n to retain, and let Del be its complement. Use the same definitions of **Uncompress**, **Gram**, and **Weight**, with $N_{a,r} = \binom{|\text{Del}| - a}{r - a}$. For a Haar random d -dimensional unitary U ,*

$$\Pr_U [\text{Blowup}_{\ell,r}(U) \geq 2] \leq e^{-cn^{18}}.$$

The estimate holds for either the backward or the forward path vectors.

Proof. Put $\alpha = 1/n$ and $s = |\text{Del}| = (1 - \alpha)d$; at this active dimension the superset normalization is $N_{q,r} = \binom{s-q}{r-q}$. Constants $c, C > 0$ below are absolute and may change from line to line. We give the argument for Forward, with exit block C and continuation block D ; applying it to the Haar unitary $U^{\dagger}$ gives Backward. Write $K_S |x\rangle = |v_{x,S}\rangle$ for the path vectors in the chosen direction and $\mathcal{T}(U) |v\rangle = \sum_{1 \leq |S| \leq \ell} K_S |v\rangle \otimes |S\rangle$, restricted to deleted inputs. Here $K_S |x\rangle = 0$ unless $x \in \text{supp}(S)$. Index the columns of **Uncompress** $_{T,m}$ and **Gram** $_{T,m}$ by the vertices of T . Sums over T, m below have $T \subseteq \text{Del}$, $|T| = r$, and $1 \leq m \leq \ell$. Exact normalization of the superset expansion gives

$$\sum_{m, T \ni x} (\text{Gram}_{T,m})_{xx} = \sum_{1 \leq |S| \leq \ell} \|K_S |x\rangle\|^2 \leq 1. \quad (1)$$

The last inequality is the telescoping exit-probability identity $C^{\dagger}C + D^{\dagger}D = I$, with different record lengths orthogonal. Thus it suffices to bound

$$\Lambda_x(U) = \sum_{m, T \ni x} \sum_{y \in T \setminus \{x\}} |(\text{Gram}_{T,m})_{xy}|, \quad \text{Blowup}_{\ell,r}(U) \leq 1 + \max_x \Lambda_x(U).$$

Deterministic bounds and sensitivity. We first establish estimates valid for every unitary, including repeated records:

$$\Lambda_x \leq Cr, \quad \text{Lip}(\Lambda_x) \leq Cr\ell^{3/2}, \quad (2)$$

where the distance is the Hilbert–Schmidt norm $\|\cdot\|_{\text{F}}$. For these estimates records may be regarded as formal multisets, whose Fourier transform averages independent unit-modulus phases attached to deleted vertices.

In this basis the length- m exit map is $CZ(DZ)^{m-1}$ for diagonal unitaries Z . Telescoping this product, averaging over the Fourier labels, and using orthogonality of different lengths gives

$$\|\mathcal{T}(U) - \mathcal{T}(U')\|_F \leq \ell^{3/2} \|U - U'\|_F. \quad (3)$$

Indeed, each of the m difference terms has Frobenius norm at most $\|U - U'\|_F$, and $\sum_{m \leq \ell} m^2 \leq \ell^3$. We also need the occupation estimate

$$\sum_{S: x \in \text{supp}(S)} \|K_S\|_F^2 \leq \ell. \quad (4)$$

To prove it, let Q_x count the visits to x in a record, set $M = D\text{Rec}$ and $N = C\text{Rec}$, and let $P_x = |x\rangle \langle x| \otimes I_{\text{rec}}$. On the reachable records, $M^\dagger Q_x M + N^\dagger Q_x N = Q_x + P_x$. Start in the empty record with the unnormalized identity on deleted inputs and sum this identity for ℓ steps. The remaining continuation count is nonnegative; the pre-step occupation at time t is $\mathbb{E}_Z \|\langle x| (DZ)^t\|^2 \leq 1$. Each exit record containing x contributes at least one to its count, proving eq. (4).

For a record of support size q not containing x , its uniformly chosen r -element superset contains x with probability $(r - q)/(s - q) \leq 2r/s$. Consequently, eq. (4) and $\|\mathcal{T}\|_F^2 \leq s$ imply

$$\sum_{m, T \ni x} \|\text{Uncompress}_{T,m}\|_F^2 \leq \ell + 2r, \quad \sum_{m, T \ni x} \|\text{Uncompress}_{T,m} |x\rangle\|_F^2 \leq 1.$$

Cauchy-Schwarz over the labels and comparison columns gives $\Lambda_x \leq \sqrt{r(\ell + 2r)} \leq Cr$. For two unitaries, expand each Gram-entry difference as $\langle a_x - a'_x, a_y \rangle + \langle a'_x, a_y - a'_y \rangle$. Applying the same inequalities, the isometric nature of superset expansion, and eq. (3), bounds its total absolute sum by

$$\sqrt{r}(\sqrt{\ell + 2r} + 1)\ell^{3/2} \|U - U'\|_F.$$

This proves eq. (2). The same deterministic bound applies when only distinct records are retained.

A Gaussian path estimate. Let X be an $r \times r$ matrix of independent standard complex Gaussians. Let $H_m(X)$ have columns $i \in [r]$ and rows (S, a) , where $|S| = m$ and $a \in S$; its entries sum the products along simple directed paths starting at i , visiting exactly S , and ending at a . Put $\mathcal{G}_m = H_m^\dagger H_m$. We claim that, when $r \geq 8m^2$,

$$\mathbb{E}|\mathcal{G}_m(i, j)|^2 \leq Cm^4 r^{2m-5} \quad (i \neq j, m \geq 3), \quad \mathbb{E}\|H_m\|_F^2 = (r)_m \leq r^m. \quad (5)$$

For $m = 1, 2$ the off-diagonal Gram entries vanish. To prove the first bound, expand the squared entry into four paths and fix the two unconjugated paths, starting at j and i , with vertex sets S, T . Both sets contain i, j ; put $t = |S \cap T|$. A surviving Gaussian moment requires equality of the two directed-edge multisets. At a vertex outside $S \cap T$ the assignment of its outgoing edge to the conjugated paths is forced, and at each shared vertex there are at most two assignments. Once these assignments are made, each conjugated path is determined by its start, if it exists at all. There are thus at most 2^t partner pairs. Every edge multiplicity is at most two, and doubled edges have distinct tails in $S \cap T$, so the Gaussian multiplicity moment is at most 2^t as well. Moreover $t \geq 3$: the edge difference of two simple paths on S with distinct starts and common endpoint is nonzero, has zero outgoing row sums, and has different incoming sums at i, j . Matching the second pair forces this difference to be supported on $(S \cap T)^2$; a loop-free matrix supported on $\{i, j\}^2$ with zero row sums must vanish. The number of fixed pairs with intersection size t is at most

$$(m-1)r^{m-2} \binom{m-2}{t-2} \binom{r-m}{m-t} (m-1)! \leq \frac{m^2 m^{2(t-2)}}{(t-2)!} r^{2m-t-2}.$$

Hence their total contribution is at most

$$16m^2 r^{2m-4} (e^{4m^2/r} - 1) \leq Cm^4 r^{2m-5}.$$

For the second bound, distinct simple paths with fixed start and endpoint have different directed-edge multisets, so their cross expectations vanish. There are exactly $(r)_m$ ordered simple paths altogether.

Passing from Gaussian to Haar entries. For a fixed q -coordinate set, J , let $Y = \sqrt{d}U_{J,J}$ and let X_q be a $q \times q$ standard complex Gaussian matrix. The truncated-Haar density [KKS16, equation (4.1)], with its normalization evaluated by successive column integration, is

$$\frac{c_{d,q}}{\pi^{q^2}} \det(I - Y^\dagger Y/d)^{d-2q} \mathbf{1}_{Y^\dagger Y < dI}, \quad c_{d,q} = \prod_{a,b=1}^q \left(1 - \frac{a+b-1}{d}\right) \leq 1.$$

Since $\log(1-u) \leq -u$, any nonnegative homogeneous function f of real degree p satisfies

$$\mathbb{E}f(Y) \leq (1 - 2q/d)^{-(q^2+p/2)} \mathbb{E}f(X_q) \leq 2\mathbb{E}f(X_q) \quad (6)$$

when $q \leq r$, $p \leq 4\ell$, and $d \geq C(r^3 + r\ell)$. The first inequality follows by rescaling the dominating Gaussian integral; the second follows by taking logarithms.

Fix $t_0 \geq 1$ and let $\mathcal{E}$ be the event $\max_{a,b} |(C^\dagger C)_{ab} - \alpha \mathbf{1}_{a=b}| \leq t_0/\sqrt{d}$. The entries are restrictions of a Haar-conjugated rank- αd projection, with means $\alpha \mathbf{1}_{a=b}$ and absolute Lipschitz constants. Haar concentration [Mec19, Theorem 5.17], applied to their real and imaginary parts, gives

$$p_0 := \Pr[\mathcal{E}^c] \leq Cd^2 e^{-ct_0^2}.$$

On $\mathcal{E}$, every restriction to q coordinates differs from αI in operator norm by at most $qt_0/\sqrt{d}$.

Split $\text{Gram}_{T,m}$ into its distinct-record and repeated-record contributions. In the distinct part, replacing $C_T^\dagger C_T$ by αI gives $\alpha \mathcal{G}_m(\sqrt{d}D_T)/(N_{m,r}d^{m-1})$. For $m \leq \ell$ and $r \geq 8\ell^2$,

$$\frac{\binom{s-1}{r-1}}{N_{m,r}} = \frac{(s-1)_{m-1}}{(r-1)_{m-1}} \leq 2(s/r)^{m-1}.$$

Combining this identity, Cauchy-Schwarz in eq. (5), and eq. (6), the expected idealized row-sum budget is at most

$$\frac{C}{\sqrt{r}} \sum_{m \leq \ell} \alpha(1-\alpha)^{m-1} m^2 \leq \frac{C}{\alpha^2 \sqrt{r}}.$$

On $\mathcal{E}$, put $\Delta = rt_0/\sqrt{d}$. The perturbation of a fixed block's Gram matrix has operator norm at most $\Delta \left\| H_m(\sqrt{d}D_T) \right\|_{\text{F}}^2 / (N_{m,r}d^{m-1})$. Bound a row sum by r times this norm and use the second estimate in eq. (5) and the same comparison and multiplicity identity. The expected length- m error, summed over $T \ni x$, is at most $Cr^2\Delta(1-\alpha)^{m-1}$. On $\mathcal{E}^c$ use the deterministic bound Cr . The distinct-record budget therefore has expectation at most

$$\frac{C}{\alpha^2 \sqrt{r}} + \frac{Cr^3 t_0}{\alpha \sqrt{d}} + Crp_0. \quad (7)$$

Repeated records. Define $R_x = \sum_{S \text{ repeated}} |S| \|K_S |x|\|^2$, with lengths at most ℓ . In the Gaussian model, a length- m vertex sequence with fixed start and excess $e = m - |\text{supp}(S)| \geq 1$ has at most $\binom{m-1}{e} m^e s^{m-1-e} \leq m^{2e} s^{m-1-e}/e!$ possibilities. Indeed, choose its repeated positions, the earlier vertex at each such position, and its new vertices. A partner sequence in the squared-amplitude expectation must have the same directed-edge multiset. If d_a are the outgoing degrees of the first sequence, ordering outgoing edges bounds the number of partners by $\prod_a d_a! \leq m^e$, since $\sum_a (d_a - 1)_+ \leq e$. The Gaussian multiplicity moment is also at most $\prod_a d_a! \leq m^e$. Thus the expected repeated-path mass before exit is at most

$$(1-\alpha)^{m-1} \sum_{e \geq 1} \frac{(m^4/s)^e}{e!} \leq \frac{Cm^4}{s} (1-\alpha)^{m-1},$$

provided $s \geq C\ell^4$. Apply eq. (6) separately to each record's supporting block. On $\mathcal{E}$, its exit Gram matrix is at most $(\alpha + \ell t_0/\sqrt{d})I \leq 2\alpha I$. Multiplying by the length weight and summing gives, using $R_x \leq \ell$ on $\mathcal{E}^c$,

$$\mathbb{E}R_x \leq \frac{C\alpha}{s} \sum_{m \geq 1} m^5 (1-\alpha)^{m-1} + \ell p_0 \leq \frac{C}{s\alpha^5} + \ell p_0. \quad (8)$$

This also bounds the expected repeated-record contribution to Λ_x , even though the present construction pads repeated records. Indeed, after the triangle inequality and summation over supersets, that contribution is at most $H_x := \sum_{S \text{ repeated: } x \in \text{supp}(S)} \sum_{y \in \text{supp}(S) \setminus \{x\}} |\langle v_{x,S}, v_{y,S} \rangle|$. Padding weights cancel because each support has exactly $N_{|\text{supp}(S)|,r}$ supersets. Cauchy–Schwarz gives $\sum_x H_x \leq \sum_{S \text{ repeated}} |\text{supp}(S)| \|K_S\|_F^2 \leq \sum_x R_x$. Haar invariance under permutations of deleted coordinates makes the expectations of H_x and of R_x independent of x , so $\mathbb{E}H_x \leq \mathbb{E}R_x$.

Parameters and concentration. Choose $t_0 = A\sqrt{n^{18} + \log d}$ with a sufficiently large absolute A . For fixed sufficiently large K , all preceding conditions hold when n is sufficiently large, $d \geq n^{100}$, $n^2 \leq \ell \leq n^4$, and $r = \lceil K\ell^2 \rceil$. Equation (7) and Equation (8) give, uniformly over these parameters,

$$\mathbb{E}\Lambda_x \leq \frac{C}{\sqrt{K}} + O(K^3 n^{-16}) + O(n^{-95}) + C(r + \ell)d^2 e^{-ct_0^2}.$$

Here $\sqrt{n^{18} + \log d}/\sqrt{d}$ decreases for $d \geq n^{100}$, and A can make the final term tend to zero uniformly in d . Taking K large and then n large gives $\mathbb{E}\Lambda_x \leq 1/4$. By eq. (2) and Haar concentration,

$$\Pr[\Lambda_x \geq 1] \leq 2\exp\left(-\frac{cd}{r^2\ell^3}\right).$$

Finally a union bound over at most d deleted inputs gives

$$\Pr[\text{Blowup}_{\ell,r}(U) \geq 2] \leq 2d\exp\left(-\frac{cd}{r^2\ell^3}\right) \leq e^{-c'n^{18}}.$$

Indeed $r^2\ell^3 \leq C_K n^{28}$ and $d \geq n^{100}$, so the negative exponent dominates both $\log(2d)$ and n^{18} . $\square$