SALSA: Attacking Lattice Cryptography with Transformers

Emily Wenger[*]
University of Chicago

Mingjie Chen[*]
University of Birmingham

Francois Charton[†]
Meta AI Research

Kristin Lauter[†]
Meta AI Research

Abstract

Currently deployed public-key cryptosystems will be vulnerable to attacks by full-scale quantum computers. Consequently, "quantum resistant" cryptosystems are in high demand, and lattice-based cryptosystems, based on a hard problem known as Learning With Errors (LWE), have emerged as strong contenders for standardization. In this work, we train transformers to perform modular arithmetic and mix half-trained models with statistical cryptanalysis techniques to propose SALSA: a machine learning attack on LWE-based cryptographic schemes. SALSA can fully recover secrets for small-to-mid size LWE instances with sparse binary secrets, and may scale to attack real-world LWE-based cryptosystems.

1 Introduction

The looming threat of quantum computers has upended the field of cryptography. Public-key cryptographic systems have at their heart a difficult-to-solve math problem that guarantees their security. The security of most current systems (e.g. [58, 28, 50]) relies on problems such as integer factorization, or the discrete logarithm problem in an abelian group. Unfortunately, these problems are vulnerable to polynomial time quantum attacks on large-scale quantum computers due to Shor's Algorithm [62]. Therefore, the race is on to find new post-quantum cryptosystems (PQC) built upon alternative hard math problems.

Several leading candidates in the final round of the 5-year NIST PQC competition are *lattice-based cryptosystems*, based on the hardness of the Shortest Vector Problem (SVP) [2], which involves finding short vectors in high dimensional lattices. Many cryptosystems have been proposed based on hard problems which reduce to some version of the SVP, and known attacks are largely based on lattice-basis reduction algorithms which aim to find short vectors via algebraic techniques. The LLL algorithm [43] was the original template for lattice reduction, and although it runs in polynomial time (in the dimension of the lattice), it returns an exponentially bad approximation to the shortest vector. It is an active area of research [22, 48, 5] to fully understand the behavior and running time of a wide range of lattice-basis reduction algorithms, but the best known classical attacks on the PQC candidates run in time exponential in the dimension of the lattice.

In this paper, we focus on one of the most widely used lattice-based hardness assumptions: Learning With Errors (LWE) [56]. Given a dimension n, an integer modulus q, and a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$, the learning with errors problem is to find the secret given noisy inner products with random vectors. LWE-based encryption schemes encrypt a message by *blinding* it with a noisy inner product. Given a random vector $\mathbf{a} \in \mathbb{Z}_q^n$, the noisy inner product is $b := \mathbf{a} \cdot \mathbf{s} + \mathbf{e} \mod q$, where $\mathbf{e}$ is an "error" vector sampled from a narrow Gaussian distribution (so its entries are small, thus the reference to noise).

[*]Co-first authors, work done while at Meta AI Research
[†]Co-last authors

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Interestingly, the assumption for cryptographic applications is that the Learning With Errors problem is hard: given a lot of noisy inner products of random vectors with a secret vector, it should be hard to learn the secret vector. However, in Machine Learning we make the opposite assumption: given a lot of noisy data, we can still learn patterns from it. So in this paper we investigate the possibility to train ML models to learn from LWE samples.

To that end, we propose **SALSA**, a technique for performing **S**ecret-recovery **A**ttacks on **L**WE via **S**equence to sequence models with **A**ttention. SALSA trains a language model to predict b from $\mathbf{a}$, and we develop two algorithms to recover the secret vector $\mathbf{s}$ using this trained model.

Our paper has three main **contributions**. We demonstrate that **transformers can perform modular arithmetic** on integers and vectors. We show that transformers trained on LWE samples can be used to distinguish LWE instances from random. This can be further turned into **two algorithms that recover binary secrets**. We show how these techniques yield a practical attack on LWE based cryptosystems and demonstrate its efficacy in the **cryptanalysis of small and mid-size LWE instances with sparse binary secrets**. Our code is available at `https://github.com/facebookresearch/SALSA`.

2 Lattice Cryptography and LWE

2.1 Lattices and Hard Lattice Problems

An integer lattice of dimension n over $\mathbb{Z}$ is the set of all integer linear combinations of n linearly independent vectors in $\mathbb{Z}^n$. In other words, given n such vectors $\mathbf{v}_i \in \mathbb{Z}^n, i \in \mathbb{N}_n$, we define the lattice $\Lambda(\mathbf{v}_1, ..\mathbf{v}_n) := \{\sum_{i=1}^n a_i \mathbf{v}_i \mid a_i \in \mathbb{Z}\}$. Given a lattice Λ, the Shortest Vector Problem (SVP) asks for a nonzero vector $\mathbf{v} \in \Lambda$ with minimal norm. Figure 1 depicts a solution to this problem in the trivial case of a 2-dimensional lattice, where $\mathbf{b}_1$ and $\mathbf{b}_2$ generate a lattice Λ and the green vector is the shortest vector in Λ.

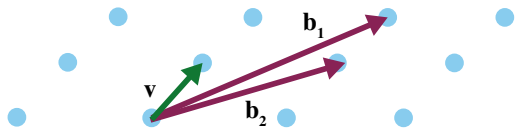

Figure 1: The dots form a lattice Λ, generated by vectors $\mathbf{b}_1$, $\mathbf{b}_2$. $\mathbf{v}$ is the shortest vector in Λ.

The best known algorithms to find exact solutions to SVP take exponential time and space with respect to n, the dimension of the lattice [49]. There exist lattice reduction algorithms to find approximate shortest vectors, such as LLL [43] (polynomial time, but exponentially bad approximation), or BKZ [22]. The shortest vector problem and its approximate variants are the hard mathematical problems that serve as the core of lattice-based cryptography.

2.2 LWE

The Learning With Errors (LWE) problem, introduced in [56], is parameterized by a dimension n, the number of samples m, a modulus q and an error distribution χ (e.g., the discrete Gaussian distribution) over $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$. Regev showed that LWE is at least as hard as quantumly solving certain hard lattice problems. Later [53, 46, 14], showed LWE to be classically as hard as standard worst-case lattice problems, therefore establishing a solid foundation for building cryptographic schemes on it.

LWE and RLWE. The LWE distribution $\mathcal{A}_{\mathbf{s},\chi}$ consists of pairs $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^n$, where $\mathbf{A}$ is a uniformly random matrix in $\mathbb{Z}_q^{m\times n}$, $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q$, where $\mathbf{s} \in \mathbb{Z}_q^n$ is the secret vector sampled uniformly at random and $\mathbf{e} \in \mathbb{Z}_q^m$ is the error vector sampled from the error distribution χ. We call the pair $(\mathbf{A}, \mathbf{b})$ an LWE sample, yielding n LWE instances: one row of $\mathbf{A}$ together with the corresponding entry in $\mathbf{b}$ is one *LWE instance*. There is also a ring version of LWE, known as the Ring Learning with Errors (RLWE) problem (described further in Appendix A.1).

Search-LWE and Decision-LWE. We now state the LWE hard problems. The search-LWE problem is to find the secret vector $\mathbf{s}$ given $(\mathbf{A}, \mathbf{b})$ from $\mathcal{A}_{\mathbf{s},\chi}$. The decision-LWE problem is to distinguish $\mathcal{A}_{\mathbf{s},\chi}$ from the uniform distribution $\{(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^n$: $\mathbf{A}$ and $\mathbf{b}$ are chosen uniformly at random$\}$. [56] provided a reduction from search-LWE to decision-LWE . We give a detailed proof of this reduction in Appendix A.2 for the case when the secret vector $\mathbf{s}$ is binary (i.e. entries are 0 and 1). In Section 4.3, our Distinguisher Secret Recovery method is built on this reduction proof.

(Sparse) Binary secrets. In LWE based schemes, the secret key vector $\mathbf{s}$ can be sampled from various distributions. For efficiency reasons, binary distributions (sampling in $\{0, 1\}^n$) and ternary distributions (sampling in $\{-1, 0, 1\}^n$) are commonly used, especially in homomorphic encryption [4]. In fact, many implementations use a sparse secret with Hamming weight h (the number of 1's

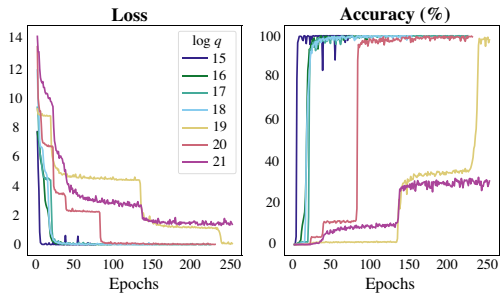

Figure 2: **Learning modular multiplication for various moduli** q. Test loss and accuracy for q with $\lceil \log_2(q) \rceil$ from 15 to 21. 300,000 training examples per epoch. One layer transformers with 512 dimensions, 8 attention heads, integers encoded in base 81.

Table 1: **Size of the training sets required for learning modular inversion.** Base-2 logarithm of the number of examples needed to reach 95% accuracy, for different values of $\lceil \log_2(q) \rceil$ and bases. '-' means 95% accuracy not attained after 90 million examples.

$\lceil \log_2(q) \rceil$	Base								
	2	3	5	7	24	27	30	**81**	128
15	23	21	23	22	20	23	22	**20**	20
16	24	22	22	22	22	22	22	**22**	21
17	-	23	25	22	23	24	22	**22**	22
18	-	23	25	23	23	24	25	**22**	22
19	-	23	-	25	25	24	-	**25**	24
20	-	-	-	-	24	25	24	**24**	25
21	-	24	-	25	-	-	-	-	25
22	-	-	-	-	-	25	-	-	25
23	-	-	-	-	-	-	-	-	-

in the binary secret). For instance, HEAAN uses $n = 2^{15}$, $q = 2^{628}$, ternary secret and Hamming weight 64 [23]. For more on the use of sparse binary secrets in LWE, see [6, 25].

3 Modular Arithmetic with Transformers

Two key factors make breaking LWE difficult: the presence of error and the use of modular arithmetic. Machine learning (ML) models tend to be robust to noise in their training data. In the absence of a modulus, recovering $\mathbf{s}$ from observations of $\mathbf{a}$ and $b = \mathbf{a} \cdot \mathbf{s} + e$ merely requires linear regression, an easy task for ML. Once a modulus is introduced, attacking LWE requires performing linear regression on an n-dimensional torus, a much harder problem.

Modular arithmetic therefore appears to be a significant challenge for an ML-based attack on LWE. Previous research has concluded that modular arithmetic is difficult for ML models [52], and that transformers struggle with basic arithmetic [51]. However, [17] showed that transformers can compute matrix-vector products, the basic operation in LWE, with high accuracy. As a first step towards attacking LWE, we investigate whether these results can be extended to the modular case.

We begin with the one-dimensional case, training models to predict $b = as \mod q$ from a for some fixed unknown value of s when $as \in \mathbb{Z}_q$. This is a form of modular inversion since the model must implicitly learn the secret s in order to predict the correct output b. We then investigate the n-dimensional case, with $\mathbf{a} \in \mathbb{Z}_q^n$ and $\mathbf{s}$ either in $\mathbb{Z}_q^n$ or in $\{0,1\}^n$ (binary secret). In the binary case, this becomes a (modular) subset sum problem.

3.1 Methods

Data Generation. We generate training data by fixing the modulus q (a prime with $15 \leq \lceil \log_2(q) \rceil \leq 30$, see the Appendix B), the dimension n, and the secret $\mathbf{s} \in \mathbb{Z}_q^n$ (or $\{0,1\}^n$ in the binary case). We then sample $\mathbf{a}$ uniformly in $\mathbb{Z}_q^n$ and compute $b = \mathbf{a} \cdot \mathbf{s} \mod q$, to create data pair (a, b).

Encoding. Integers are encoded in base B (usually, B=81), as a sequence of digits in $\{0, \ldots B-1\}$. For instance, $(a, b) = (16, 3)$ is represented as the sequences `[1,0,0,0,0]` and `[1,1]` in base 2, or `[2,2]` and `[3]` in base 7. In the multi-dimensional case, a special token separates the $\mathbf{a}$ coordinates.

Model Training. The model is trained to predict b from $\mathbf{a}$, for an unknown but fixed value of $\mathbf{s}$. We use sequence-to-sequence transformers [67] with one layer in the encoder and decoder, 512 dimensions and 8 attention heads. We minimize a cross-entropy loss, and use the Adam optimizer [39] with a learning rate of 5×10^{-5}. At epoch end (300000 examples), model accuracy is evaluated over a test set of 10000 examples. We train until test accuracy is 95% or loss plateaus for 60 epochs.

3.2 Results

One-Dimensional. For a fixed secret s, modular multiplication is a function from $\mathbb{Z}_q$ into itself, that can be learned by memorizing q values. Our models learn modular multiplication with high accuracy for values of q such that $\lceil \log_2(q) \rceil \leq 22$. Figure 2 presents learning curves for different values of $log_2(q)$. The loss and accuracy curves have a characteristic step shape, observed in many of our experiments, which suggests that "easier cases" (small values of $\lfloor as/q \rfloor$) are learned first.

The speed of learning and the training set size needed to reach high accuracy depend on the problem difficulty, i.e. the value of q. Table 1 presents the $\lceil \log_2 \rceil$ of the number of examples needed to reach 95% accuracy for different values of $\lceil \log_2(q) \rceil$ and base B. Since transformers learn from scratch, without prior knowledge of numbers and moduli, this procedure is not data-efficient. The number of examples needed to learn modular multiplication is between $10q$ and $50q$. Yet, these experiments prove that transformers can solve the modular inversion problem in prime fields.

Table 1 illustrates an interesting point: learning difficulty depends on the base used to represent integers. For instance, base 2 and 5 allow the model to learn up to $\lceil \log_2(q) \rceil = 17$ and 18, whereas base 3 and 7 can reach $\lceil \log_2(q) \rceil = 21$. Larger bases, especially powers of small primes, enable faster learning. The relation between representation base and learning difficulty is difficult to explain from a number theoretic standpoint. Additional experiments are in Appendix B.

Multidimensional integer secrets. In the n-dimensional case, the model must learn the modular dot product between vectors $\mathbf{a}$ and $\mathbf{s}$ in $\mathbb{Z}_n$. The proves to be a much harder problem. For $n = 2$, with the same settings, small values of q (251, 367 and 967) can be learned with over 90% accuracy, and $q = 1471$ with 30%. In larger dimension, all models fail to learn. Increasing model depth to 2 or 4 layers, or dimension to 1024 or 2048 and attention heads to 12 and 16, improves data efficiency (less training samples are needed), but does not scale to larger values of q or $n > 2$.

Multidimensional binary secrets. Binary secrets make n-dimensional problems easier to learn. For $n = 4$, our models solve problems with $\lceil \log_2(q) \rceil \leq 29$ with more than 99.5% accuracy. For $n = 6$ and 8, we solve cases $\lceil \log_2(q) \rceil \leq 22$ with more than 85% accuracy. But we did not achieve high accuracy for larger values of n. So in the next section, we introduce techniques for recovering secrets from a partially trained transformer. We then show that these additional techniques allow recovery of sparse binary secrets for LWE instances with $30 \leq n \leq 128$ (so far).

4 Introducing SALSA: LWE Cryptanalysis with Transformers

Having established that transformers can perform integer modular arithmetic, we leverage this result to propose SALSA, a method for **S**ecret-recovery **A**ttacks on **L**WE via **S**eq2Seq models with **A**ttention.

4.1 SALSA Ingredients

SALSA has three modules: a **transformer model** $\mathcal{M}$, a **secret recovery algorithm**, and a **secret verification procedure**. We assume that SALSA has access to a number of LWE instances in dimension n that use the same secret, i.e. pairs $(\mathbf{a}, b)$ such that $b = \mathbf{a} \cdot \mathbf{s} + e \mod q$, with e an error from a centered distribution with small standard deviation. SALSA runs in three steps. First, it uses LWE data to train $\mathcal{M}$ to predict b given $\mathbf{a}$. Next SALSA runs a secret recovery algorithm. It feeds $\mathcal{M}$ special values of $\mathbf{a}$, and uses the output $\tilde{b} = \mathcal{M}(\mathbf{a})$ to predict the secret. Finally, SALSA evaluates the guesses $\tilde{\mathbf{s}}$ by verifying that residuals $r = b - \mathbf{a} \cdot \tilde{\mathbf{s}} \mod q$ computed from LWE samples have small standard deviation. If so, s has been recovered and SALSA stops. If not, SALSA returns to step 1 and iterates.

4.2 Model Training

SALSA uses LWE instances to train a model that predicts b from $\mathbf{a}$ by minimizing the cross-entropy between the model prediction b' and b. The model architecture is a universal transformer [27], in which a shared transformer layer is iterated several times (the output from one iteration is the input to the next). Our base model has two encoder layers, with 1024 dimensions and 32 attention heads, the second layer iterated 2 times, and two decoder layers with 512 dimensions and 8 heads, the second layer iterated 8 times. To limit computation in the shared layer, we use the copy-gate mechanism from [24]. Models are trained using the Adam optimizer with $lr = 10^{-5}$ and 8000 warmup steps.

For inference, we use a beam search with depth 1 (greedy decoding) [40, 65]. At the end of each epoch, we compute model accuracy over a test set of LWE samples. Because of the error added when computing $b = \mathbf{a} \cdot \mathbf{s} + e$, exact prediction of b is not possible. Therefore, we calculate *accuracy within tolerance* τ *(*acc_τ*)*: the proportion of predictions $\tilde{b} = \mathcal{M}(\mathbf{a})$ that fall within τq of b, i.e. such that $\|b - \tilde{b}\| \leq \tau q$. In practice we set $\tau = 0.1$.

4.3 Secret Recovery

We propose two algorithms for recovering $\mathbf{s}$: direct recovery from special values of $\mathbf{a}$, and distinguisher recovery using the binary search to decision reduction (Appendix A.2). For theoretical justification of these, see Appendix C.

Direct Secret Recovery. The first technique, based on the LWE search problem, is analogous to a chosen plaintext attack.For each $i \in \mathbb{N}_n$, a guess of the i-th coordinate of $\mathbf{s}$ is made by feeding model $\mathcal{M}$ the special value $\mathbf{a_i} = K\mathbf{e_i}$ (all coordinates 0 except the i-th), with K a large integer. If $s_i = 0$, and the model $\mathcal{M}$ correctly approximates $b_i = \mathbf{a}_i \cdot \mathbf{s} + e$ from $\mathbf{a_i}$, then we expect $\tilde{b}_i := \mathcal{M}(\mathbf{a}_i)$ to be a small integer; likewise if $s_i = 1$ we expect a large integer. This technique is formalized in Algorithm 1. The $binarize$ function in line 7 is explained in Appendix C. In SALSA, we run direct recovery with 10 different K values in order to yield 10 $\mathbf{s}$ guesses.

Algorithm 1 Direct Secret Recovery

1: **Input:** $\mathcal{M}, K, n$
2: **Output:** secret s
3: $p = \mathbf{0}^n$
4: **for** $i = 1, \ldots, n$ **do**
5: $\quad a = \mathbf{0}^n;\ a_i = K$
6: $\quad p_i = \mathcal{M}(a)$
7: $s = binarize(p)$
8: **Return:** s

Distinguisher Secret Recovery. The second algorithm for secret recovery is based on the decision-LWE problem. It uses the output of $\mathcal{M}$ to determine if LWE data $(\mathbf{a},\ b)$ can be distinguished from randomly generated pairs $(\mathbf{a_r}, b_r)$. The algorithm for distinguisher-based secret recovery is shown in Algorithm 2. At a high level, the algorithm works as follows. Suppose we have t LWE instances $(\mathbf{a},\ b)$ and t random instances $(\mathbf{a_r}, b_r)$. For each secret coordinate s_i, we transform the $\mathbf{a}$ into $a'_i = a_i + c$, with $c \in \mathbb{Z}_q$ random integers. We then use model $\mathcal{M}$ to compute $\mathcal{M}(\mathbf{a}')$ and $\mathcal{M}(\mathbf{a_r})$. If the model has learned $\mathbf{s}$ and the ith bit of $\mathbf{s}$ is 0, then $\mathcal{M}(\mathbf{a}')$ should be significantly closer to b than $\mathcal{M}(\mathbf{a_r})$ is to b_r. Iterating on i allows us to recover the secret bit by bit. SALSA runs the distinguisher recovery algorithm when model $acc_{\tau=0.1}$ is above 30%. This is the theoretical limit for this approach to work.

4.4 Secret Verification.

At the end of the recovery step, we have 10 or 11 guesses $\tilde{\mathbf{s}}$ (depending on whether the distinguisher recovery algorithm was run). To verify them, we compute the residuals $r = \mathbf{a} \cdot \tilde{\mathbf{s}} - b \mod q$ for a set of LWE samples $(\mathbf{a}, b)$. If $\mathbf{s}$ is correctly guessed, we have $\tilde{\mathbf{s}} = \mathbf{s}$, so $r = \mathbf{a} \cdot \mathbf{s} - b = e \mod q$ will be distributed as the error e with small standard deviation σ. If $\tilde{\mathbf{s}} \neq \mathbf{s}$, r will be (approximately) uniformly distributed over $\mathbb{Z}_q$ (because $\mathbf{a} \cdot \tilde{\mathbf{s}}$ and b are uniformly distributed over $\mathbb{Z}_q$), and will have standard deviation $\sigma(r) \approx q/\sqrt{(12)}$. Therefore, we can verify if $\tilde{\mathbf{s}}$ is correct by calculating the standard deviation of the residuals: if it is close to σ, the standard deviation of error, the secret was recovered. In this paper, $\sigma = 3$ and $q = 251$, so the standard deviation of r will be around 3 if $\tilde{\mathbf{s}} = \mathbf{s}$, and 72.5 if not.

Algorithm 2 Distinguisher Secret Recovery

Input: $\mathcal{M}, n, q, acc_\tau, \tau$
Output: secret s
$s = \mathbf{0}^n$
$advantage, bound = acc_\tau - 2 \cdot \tau, \tau \cdot q$
$t = min\{50, \frac{2}{advantage^2}\}$
$\mathbf{A_{LWE}}, \mathbf{B_{LWE}} = LWESamples(t, n, q)$
for $i = 1, \ldots n$ **do**
$\quad \mathbf{A_{unif}} \sim \mathcal{U}\{0, q-1\}^{n \times t}$
$\quad \mathbf{B_{unif}} \sim \mathcal{U}\{0, q-1\}^{t}$
$\quad \mathbf{c} \sim \mathcal{U}\{0, q-1\}^{t}$
$\quad \mathbf{A'_{LWE}} = \mathbf{A_{LWE}}$
$\quad \mathbf{A'_{LWE}}[:, i] = (\mathbf{A_{LWE}}[:, i] + \mathbf{c}) \mod q$
$\quad \widetilde{\mathbf{B_{LWE}}} = \mathcal{M}(\mathbf{A'_{LWE}})$
$\quad \widetilde{\mathbf{B_{unif}}} = \mathcal{M}(\mathbf{A_{unif}})$
$\quad dl = |\widetilde{\mathbf{B_{LWE}}} - \mathbf{B_{LWE}}|$
$\quad du = |\widetilde{\mathbf{B_{unif}}} - \mathbf{B_{unif}}|$
$\quad c_{LWE} = \#\{j \mid dl_j < bound, j \in \mathbb{N}_t\}$
$\quad c_{unif} = \#\{j \mid du_j < bound, j \in \mathbb{N}_t\}$
$\quad$ **if** $(c_{LWE} - c_{unif}) \leq advantage \cdot t/2$ **then**
$\quad\quad s_i = 1$
Return: s

5 SALSA Evaluation

In this section, we present our experiments with SALSA. We generate datasets for LWE problems of different sizes, defined by the dimension and the density of ones in the binary secret. We use gated universal transformers, with two layers in the encoder and decoder. Default dimensions and attention heads in the encoder and decoder are 1024/512 and 16/4, but we vary them as we scale the problems. Models are trained on two NVIDIA Volta 32GB GPUs on an internal cluster.

5.1 Data generation

We generate LWE data for SALSA training/evaluation is randomly given the following parameters: dimension n, secret density d, modulus q, encoding base B, binary secret s, and error distribution χ. For all experiments, we use $q = 251$ and $B = 81$ (see §3.1), fix the error distribution χ as a discrete Gaussian with $\mu = 0, \sigma = 3$ [4], and generate a random s.

We vary the problem size n (the LWE dimension) and the density d (the proportion of ones in the secret) to test attack success and to observe how it scales. For problem size, we experiment with

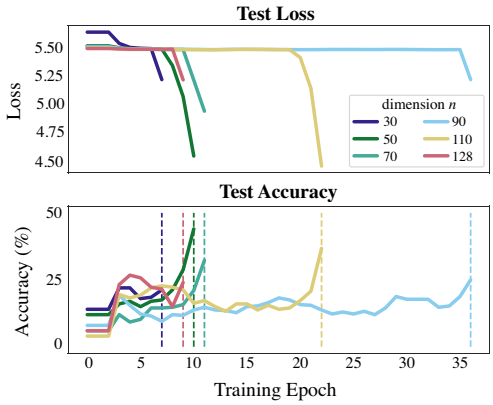

Figure 3: **Full secret recovery**: Curves for loss and $acc_\tau = 0.1$, for varying n with Hamming weight 3. For $n < 100$, model has 1024/512 embedding, 16/4 attention heads. For $n \geq 100$, model has 1536/512 embedding, 32/4 attention heads.

Table 2: **Full secret recovery.** Highest density values at which the secret was recovered for each n, $q = 251$. For $n < 100$, the model has 1024/512 embedding, 16/4 attention heads. For $n \geq 100$, the model has 1536/512 embedding, 32/4 attention heads. For comparison, we include the log_2 number of possible secrets at each n/d level.

Dim. n	**Density** d	log_2 **samples**	log_2 **secrets**	**Runtime (hours)**
30	0.1	21.9	12.0	1.2
	0.13	24.8	14.7	21
50	0.06	22.4	14.3	5.5
	0.08	25.6	17.8	18.5
70	0.04	22.5	15.7	4.5
90	0.03	24.1	16.8	35.0
110	0.03	21.5	17.7	32.0
128	0.02	22.3	18.4	23.0

$n = 30$ to $n = 128$. For density, we experiment with $0.002 \leq d \leq 0.15$. For a given n, we select d so that the Hamming weight of the binary secret ($h = dn$), is larger than 2. Appendix D.2 contains an ablation study of data parameters. We generate data using the RLWE variant of LWE, described in Appendix A. For RLWE problems, each $\mathbf{a}$ is a line of a circulant matrix generated from an initial vector $\in \mathbb{Z}_q^n$. RLWE problems exhibit more structure than LWE due to the use of the circulant matrix, which may help our models learn.

5.2 Results

Table 2 presents problem sizes n and densities d for which secrets can be fully recovered, together with the time and the logarithm of number of training samples needed. SALSA can recover binary secrets with Hamming weight 3 for dimensions up to 128. Hamming weight 4 secrets can be recovered for $n < 70$. For context, we include a column "$\log_2$ secrets" listing the number of possible secrets for the corresponding Hamming weight. Exhaustive search for the secret may run faster than our current experiments for low Hamming weight. However, an exhaustive search requires knowing the exact hamming weight in advance, or iterating through all possible Hamming weights. Although we have only succeeded unconditionally with low Hamming weight so far, SALSA does not a priori know the secret Hamming weight h. It remains to be seen how SALSA will scale to larger n and h.

Interestingly, the number of samples required for SALSA to succeed for a fixed Hamming weight is relatively flat as the dimension n increases. However the time needed to recover the secret for a fixed Hamming weight increases with n, partly because the length of the input sequence fed into the model is proportional to n even though the number of samples needed remains stable as n grows. This is an important result, because all the data used for training the model must be collected (e.g. via eavesdropping), making sample size an important metric.

For a given n, scaling to higher densities requires more time and data, and could not be achieved with the architecture we use for $n > 50$. As n grows, larger models are needed: our standard architecture, with $1024/512$ dimensions and $16/4$ attention heads (encoder/decoder) was sufficient for $n \leq 90$. For $n > 90$, we needed $1536/512$ dimensions and $32/4$ attention heads.

Figure 3 illustrates model behavior during training. After an initial burn-in period, the loss curve (top graph) plateaus until the model begins learning the secret. Once loss starts decreasing, model accuracy with $0.1q$ tolerance (bottom graph) increases sharply. Full secret recovery (vertical lines in the bottom graph) happens shortly after, often within one or two epochs. Direct secret recovery accounts for 55% of recoveries, while the distinguisher only accounts for 18% of recoveries (see Appendix C.3). 27% of the time, both methods succeed simultaneously.

One key conclusion from these experiments is that the secret recovery algorithms enable secret recovery long before the transformer has been trained to high accuracy (even before training loss settles at a low level). Frequently, the model only needs *to begin* to learn for the attack to succeed.

Table 3: **Architecture Experiments** We test the effect of model layers, loops, gating, and encoder dimension and report the log_2 samples required for secret recovery ($n = 50$, Hamming weight 3).

Regular vs. UTs (1024/512, 16/4, 8/8)		**Ungated vs. Gated** (1024/512, 16/4, 8/8)		**UT Loops** (1024/512, 16/4, **X/X**)			**Encoder Dim.** (**X**/512, 16/4, 2/8)			**Decoder Dim.** (1024/**X**, 16/4, 2/8)			
Regular	UT	Ungated	Gated	2/8	4/4	8/2	512	2048	3040	256	768	1024	1536
26.3	22.5	26.5	22.6	23.5	26.1	23.2	23.3	20.1	19.7	22.5	21.8	23.9	24.3

5.3 Experiments with model architecture

SALSA's base model architecture is a Universal Transformer (UT) with a copy-gate mechanism. Table 3 demonstrates the importance of these choices. For problem dimension $n = 50$, replacing the UT by a regular transformer with 8 encoder/decoder layers, or removing the copy-gate mechanism increases the data requirement by a factor of 14. Reducing the number of iterations in the shared layers from 8 to 4 has a similar effect. Reducing the number of iterations in either the encoder or decoder (i.e. from 8/8 to 8/2 or 2/8) may further speed up training. Asymmetric transformers (e.g. large encoder and small decoder) have proved efficient for other math problems, e.g. [37], [17], and asymmetry helps SALSA as well. Table 3 demonstrates that increasing the encoder dimension from 1024 to 3040, while keeping the decoder dimension at 512, results in a 7-fold reduction in sample size. Additional architecture experiments are presented in Appendix D.1.

5.4 Increasing dimension and density

To attack real-world LWE problems, SALSA must handle larger dimension n and density d. Our experiments with architecture suggest that increasing model size, and especially encoder dimension, is the key factor to scaling n. Empirical observations indicate that scaling d is a much harder problem. We hypothesize that this is due to the subset sum modular addition at the core of LWE with binary secrets. For a secret with Hamming weight h, the base operation $\mathbf{a} \cdot \mathbf{s} + e \mod q$ is a sum of h integers, followed by a modulus. For small values of h, the modulus operation is not always necessary, as the sum might not exceed q. As density increases, so does the number of times the sum "wraps around" the modulus, perhaps making larger Hamming weights more difficult to learn. To test this hypothesis, we limited the range of the coordinates in $\mathbf{a}$, so that $a_i < r$, with $r = \alpha q$ and $0.3 < \alpha < 0.7$. For $n = 50$, we recovered secrets with density up to 0.3, compared to 0.08 with the full range of coordinates (see Table 4). Density larger than 0.3 is no longer considered a sparse secret.

5.5 Increasing error size

Table 5: log_2 **samples needed for secret recovery when** $\sigma = \lfloor \sqrt{n} \rfloor$**.** Results averaged over 6 SALSA runs at each n/σ level.

n/σ	30/5	50/7	70/8	90/9
logSamples	18.0	18.5	19.3	19.6

Theoretically for lattice problems to be hard, σ should scale with $\sqrt{n}$, although this is often ignored in practice, e.g. [4]. Consequently, we run most SALSA experiments with $\sigma = 3$, a common choice in existing RLWE-based systems. Here, we investigate how SALSA performs as σ increases. First, to match the theory, we run experiments where $\sigma = \lfloor \sqrt{n} \rfloor$, $h = 3$ and found that SALSA recovers secrets even as σ scales with n (see Table 5, same model architecture as Table 2). Second, we evaluate SALSA's performance for fixed n/h values as σ increases. We fix $n = 50$ and $h = 3$ and evaluate for σ values up to $\sigma = 24$. Secret recovery succeeds for all tests, although the number of samples required for recovery linearly increases (see Figure 7 in Appendix). For both sets of experiments, we reuse samples up to 10 times.

6 SALSA in the Wild

Problem Size. Currently, SALSA can recover secrets from LWE samples with n up to 128 and density $d = 0.02$. It can recover higher density secrets for smaller n ($d = 0.08$ when $n = 50$). As mentioned in Section 2.2, sparse binary secrets are used in real world LWE homomorphic encryption, and attacking these implementations is a future goal for SALSA. Admittedly, SALSA must scale to attack larger n before it can break full-strength homomorphic encryption implementations. However, other parameters of full-strength homomorphic encryption such as secret density (the secret vector in HEAAN has $d < 0.002$) and error size ([4] recommends $\sigma = 3.2$) are within SALSA's reach.

Table 4: **Secret recovery when max a value is bounded.** Results shown are fraction of the secret recovered by SALSA for $n = 50$ with varying d when a values are $\leq p \cdot Q$. Green means that s was fully recovered. Yellow means all of the 1 bits were recovered, but not all 0 bits. Red means SALSA failed. For completeness, we also note the log_2 number of possible secrets at each d level.

d	log_2 **secrets**	**Max a value as fraction of q**						
		0.35	0.4	0.45	0.5	0.55	0.6	0.65
0.16	29.0	1.0	1.0	1.0	1.0	1.0	1.0	0.88
0.18	31.2	1.0	1.0	1.0	1.0	0.82	0.86	0.84
0.20	33.3	1.0	1.0	1.0	1.0	1.0	0.82	0.82
0.22	35.1	0.98	1.0	1.0	0.98	0.80	0.78	0.86
0.24	36.8	1.0	1.0	1.0	0.98	0.78	0.78	0.80
0.26	38.4	1.0	1.0	0.88	0.92	0.76	0.76	0.76
0.28	39.8	0.98	1.0	0.80	0.74	0.74	0.76	0.74
0.30	41.0	0.98	1.0	0.93	0.76	0.72	0.74	0.74

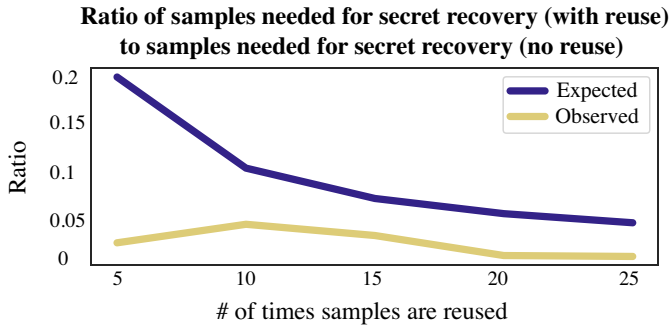

Figure 4: **Reusing LWE samples yields a significant decrease in the number of samples needed for secret recovery.** Shown here is the ratio of samples required for secret recovery with reuse to the samples required for secret recovery without reuse, both expected (top curve) and observed (bottom curve, better than expected).

Other LWE-based schemes use secret dimensions that seem achievable given our current results. For example, in the LWE-based public key encryption scheme Crystal-Kyber [9], the secret dimension is $k \times 256$ for $k = \{2, 3, 4\}$, an approachable range for SALSA based on initial results. The LWE-based signature scheme Crystal-Dilithium has similar n sizes [29]. However, these schemes don't use sparse binary secrets, and adapting SALSA nonbinary secrets is an avenue for future work.

Sample Efficiency. A key requirement of real-world LWE attacks is sample efficiency. In practice, an attacker will only have access to a small set of LWE instances $(\mathbf{a}, b)$ for a given secret s. For instance, in Crystal-Kyber, there are only $(k+1)n$ LWE instances available with $k = 2$, 3 or 4 and $n = 256$. The experiments in [20, 11] use less than 500 LWE instances. The TU Darmstadt challenge provides n^2 LWE instances to attackers.

The $\log_2$ *samples* column of Table 2 lists the number of LWE instances needed for model training. This number is much larger than what is likely available in practice, so it is important to reduce sample requirements. Classical algebraic attacks on LWE require LWE instances to be linearly independent, but *SALSA does not have this limitation.* Thus, we can reduce SALSA's sample use in several ways. First, we can reuse samples during training. Figure 4 confirms that this allows secret recovery with fewer samples. Second, we can use integer linear combinations of given LWE samples to make new samples which have the same secret but a larger error σ. Appendix E contains the formula for the number of new samples we can generate with this method (up to 2^{42} new samples from 100 samples).

Comparison to Baselines. Most existing attacks on LWE such as uSVP and dual attack use an algebraic approach that involves building a lattice from LWE instances such that this lattice contains an exceptionally short vector which encodes the secret vector information. Attacking LWE then involves finding the short vector via lattice reduction algorithms like BKZ [22]. For LWE with sparse binary secrets, the main focus of this paper, various techniques can be adapted to make algebraic attacks more efficient. [20, 11] and [23] provide helpful overviews of algebraic attacks on sparse binary secrets. More information about attacks on LWE is in Appendix A.3.

Compared to existing attacks, SALSA's most notable feature is its novelty. We do not claim that to have better runtime, neither do we claim the ability to attack real-world LWE problems (yet). Rather, we introduce a new attack and demonstrate with non-toy successes that transformers can be used to attack LWE. Given our goal, no serious SALSA speedup attempts have been made so far, but a few simple improvements could reduce runtime. First, the slowest step in SALSA is model training, which can be greatly accelerated by distributing it across many GPUs. Second, our transformers are trained from scratch, so pre-training them on such basic tasks as modular arithmetic could save time and data. Finally, the amount of training needed before the secret is recovered depends in large part on the secret guessing algorithms. New algorithms might allow SALSA to recover secrets faster.

Since SALSA does not involve finding the shortest vector in a lattice, it has an advantage over the algebraic attacks – with all LWE parameters fixed and in the range of SALSA, SALSA can attack the LWE problem for a smaller modulus q compared to the algebraic attacks. This is because the target vector is relatively large in the lattice when q is smaller and is harder to find. For instance, in [20], their Table 2 shows that when the block size is 45, for $n = 90$, their attack does not work for q less than 10 bits, but we can handle q as small as 8 bits (Table 20).

7 Related Work

Use of ML for cryptanalysis. The fields of cryptanalysis and machine learning are closely related [57]. Both seek to approximate an unknown function $\mathcal{F}$ using data, although the context and techniques for doing so vary significantly between the fields. Because of the similarity between the domains, numerous proposals have tried to leverage ML for cryptanalysis. ML-based attacks have been proposed against a number of cryptographic schemes, including block ciphers [3, 63, 38, 10, 30, 12, 21], hash functions [31], and substitution ciphers [1, 64, 8]. Although our work is the first to use recurrent neural networks for lattice cryptanalysis, prior work has used them for other cryptographic tasks. For example, [32] showed that LSTMs can learn the decryption function for polyalphabetic ciphers like Enigma. Follow-up works used variants of LSTMs, including transformers, to successfully attack other substitution ciphers [1, 64, 8].

Use of transformers for mathematics. The use of language models to solve problems of mathematics has received much attention in recent years. A first line of research explores math problems set up in natural language. [59] investigated their relative difficulty, using LSTM [34] and transformers, while [33] showed large transformers could achieve high accuracy on elementary/high school problems. A second line explores various applications of transformers on formalized symbolic problems. [42] showed that symbolic math problem could be solved to state-of-the-art accuracy with transformers. [68] discussed their limits when generalizing out of their training distribution. Transformers have been applied to dynamical systems [18], transport graphs [19], theorem proving [54], SAT solving [61], and symbolic regression [13, 26]. A third line of research focuses on arithmetic/numerical computations and has had slower progress. [52] and [51] discussed the difficulty of performing arithmetic operations with language models. Bespoke network architectures have been proposed for arithmetic operations [35, 66], and transformers were used for addition and similar operations [55]. [17] showed that transformers can learn numerical computations, such as linear algebra, and introduced the shallow models with shared layers used in this paper.

Learnability of LWE. Shalev-Shwartz, Shamir and Shammah [60] showed that some problems of modular arithmetic cannot be solved by gradient descent. Because it uses a transformer to learn the secret, by predicting b from a, SALSA's scalability hinges on the solvability of LWE by gradient descent. Here, we provide our perspective on this question.

The LWE problem amounts to (discrete) linear regression on an n-dimensional torus with radius q. When q is infinite, LWE is pure linear regression, and gradient methods will succeed. For finite q, gradients are informative unless the model's prediction lies on the opposite side of the torus from the true solution (e.g. $q = 100$, true value is 0, prediction is 50). At this point, the projection of the gradient along the coordinate axis is uninformative, since either direction on the torus points towards the true solution. For large q, this situation is uncommon, so gradient methods should work. For very small q, this will happen more often, and gradient methods will be perturbed. In the degenerate case $q = 2$, featured in [60], gradient methods will always fail.

For the value $q = 251$ used in SALSA, gradient methods can recover secrets (i.e. solve the modular linear regression problem) for n up to 128 so far. Except for the case of very small q, it is unlikely

that LWE belongs to the "failing gradient" class of problems described in [60]. Any learnability problems should disappear altogether as q increases.

Interestingly, this intuition about the size of q also appears in the classical lattice reduction approach to solving LWE. Laine and Lauter [41] use LLL to demonstrate concrete polynomial-time attacks against lattice problems for large q. They also explore the boundary of where these attacks start to fail for smaller q. The intuition is that when q is large, LLL directly finds a vector which is "small enough" compared to q to break the system. When q is small, LLL does not find a short enough vector. This is further explored in [20], which gives concrete running times for generalizations of LLL and quantifies the size of q where these attacks start to fail.

8 Conclusion

Discussion. In this paper, we demonstrate that transformers can be trained to perform modular arithmetic. Building on this capability, we design SALSA, a method for attacking the LWE problem with binary secrets, a hardness assumption at the foundation of many lattice-based cryptosystems. We show that SALSA can break LWE problems of medium dimension (up to $n = 128$), which is in the same range as problems in the Darmstadt challenge [15], although we restrict to the easier case of sparse binary secrets. This is the first paper to use transformers to solve hard problems in lattice-based cryptography. Future work will attempt to scale up SALSA to attack higher dimensional lattices with more general secret distributions.

The key to scaling up to larger lattice dimensions seems to be to increase the model size, especially the dimensions, the number of attention heads, and possibly the depth. Large architectures should scale to higher dimensional lattices such as $n = 256$ which is used in practice. Density, on the other hand, is constrained by the performance of transformers on modular arithmetic. Better representations of finite fields could improve transformer performance on these tasks. Finally, our secret guessing algorithms enable SALSA to recover secrets from low-accuracy transformers, therefore reducing the data and time needed for the attack. Extending these algorithms to take advantage of partial learning should result in better performance.

Ethics and Broader Impact. The primary value of this work is in alerting the cryptographic and ML communities to the risk of ML-based attacks on PQC. Even if current attacks do not succeed, we believe that providing early warning of potential threats is critical. However, we emphasize that SALSA represents a proof of concept that cannot be used against real-world implementations (i.e. the PQC schemes which NIST standardized on July 5, 2022). Additional scaling work will be necessary before these techniques will be relevant to attacking real-world cryptosystems.

References

[1] Ezat Ahmadzadeh, Hyunil Kim, Ongee Jeong, and Inkyu Moon. A Novel Dynamic Attack on Classical Ciphers Using an Attention-Based LSTM Encoder-Decoder Model. IEEE Access, 2021.

[2] Miklós Ajtai. Generating hard instances of lattice problems. In Proc. of the 28th annual ACM symposium on Theory of Computing, 1996.

[3] Mohammed M Alani. Neuro-cryptanalysis of DES and triple-DES. In Proc. of NeurIPS, 2012.

[4] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, et al. Homomorphic encryption standard. In Protecting Privacy through Homomorphic Encryption, pages 31–62. Springer, 2021.

[5] Martin Albrecht, Carlos Cid, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. On the complexity of the BKW algorithm on LWE. Designs, Codes and Cryptography, 2015.

[6] Martin R. Albrecht. On dual lattice attacks against small-secret lwe and parameter choices in helib and seal. In EUROCRYPT, 2017.

[7] Martin R. Albrecht, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Lazy Modulus Switching for the BKW Algorithm on LWE. In Public-Key Cryptography, 2014.

[8] Nada Aldarrab and Jonathan May. Can sequence-to-sequence models crack substitution ciphers? arXiv preprint arXiv:2012.15229, 2020.

[9] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber (version 3.02) – Submission to round 3 of the NIST post-quantum project. 2021.

[10] Seunggeun Baek and Kwangjo Kim. Recent advances of neural attacks against block ciphers. In Proc. of SCIS, 2020.

[11] Shi Bai and Steven D. Galbraith. Lattice Decoding Attacks on Binary LWE. In Information Security and Privacy, 2014.

[12] Adrien Benamira, David Gerault, Thomas Peyrin, and Quan Quan Tan. A deeper look at machine learning-based cryptanalysis. In Proc. of Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2021.

[13] Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. arXiv preprint arXiv:2106.06427, 2021.

[14] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical Hardness of Learning with Errors. In Proc. of the Forty-Fifth Annual ACM Symposium on Theory of Computing, 2013.

[15] Johannes Buchmann, Niklas Büscher, Florian Göpfert, Stefan Katzenbeisser, Juliane Krämer, Daniele Micciancio, Sander Siim, Christine van Vredendaal, and Michael Walter. Creating Cryptographic Challenges Using Multi-Party Computation: The LWE Challenge. In Proc. of the 3rd ACM International Workshop on ASIA Public-Key Cryptography, 2016.

[16] Johannes A. Buchmann, Florian Göpfert, Rachel Player, and Thomas Wunderer. On the Hardness of LWE with Binary Error: Revisiting the Hybrid Lattice-Reduction and Meet-in-the-Middle Attack. In Proc. of AFRICACRYPT, 2016.

[17] François Charton. Linear algebra with transformers. arXiv preprint arXiv:2112.01898, 2021.

[18] François Charton, Amaury Hayat, and Guillaume Lample. Learning advanced mathematical computations from examples. arXiv preprint arXiv:2006.06462, 2020.

[19] François Charton, Amaury Hayat, Sean T. McQuade, Nathaniel J. Merrill, and Benedetto Piccoli. A deep language model to predict metabolic network equilibria. arXiv preprint arXiv:2112.03588, 2021.

[20] Hao Chen, Lynn Chua, Kristin E. Lauter, and Yongsoo Song. On the Concrete Security of LWE with Small Secret. IACR Cryptol. ePrint Arch., 2020:539, 2020.

[21] Yi Chen and Hongbo Yu. Bridging Machine Learning and Cryptanalysis via EDLCT. Cryptology ePrint Archive, 2021.

[22] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better Lattice Security Estimates. In Proc. of ASIACRYPT 2011, 2011.

[23] Jung Hee Cheon, Minki Hhan, Seungwan Hong, and Yongha Son. A Hybrid of Dual and Meet-in-the-Middle Attack on Sparse and Ternary Secret LWE. IEEE Access, 2019.

[24] Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. The Neural Data Router: Adaptive Control Flow in Transformers Improves Systematic Generalization. arXiv preprint arXiv:2110.07732, 2021.

[25] Benjamin R. Curtis and Rachel Player. On the feasibility and impact of standardising sparse-secret LWE parameter sets for homomorphic encryption. In Proc. of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, 2019.

[26] Stéphane d'Ascoli, Pierre-Alexandre Kamienny, Guillaume Lample, and François Charton. Deep symbolic regression for recurrent sequences. arXiv preprint arXiv:2201.04600, 2022.

[27] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. arXiv preprint arXiv:1807.03819, 2018.

[28] Whitfield Diffie and Martin Hellman. New directions in cryptography. IEEE transactions on Information Theory, 1976.

[29] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium – Algorithm Specifications and Supporting Documentation (Version 3.1). 2021.

[30] Aron Gohr. Improving attacks on round-reduced speck32/64 using deep learning. In Proc. of Annual International Cryptology Conference, 2019.

[31] Sergij V Goncharov. Using fuzzy bits and neural networks to partially invert few rounds of some cryptographic hash functions. arXiv preprint arXiv:1901.02438, 2019.

[32] Sam Greydanus. Learning the enigma with recurrent neural networks. arXiv preprint arXiv:1708.07576, 2017.

[33] Kaden Griffith and Jugal Kalita. Solving Arithmetic Word Problems with Transformers and Preprocessing of Problem Text. arXiv preprint arXiv:2106.00893, 2021.

[34] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 1997.

[35] Łukasz Kaiser and Ilya Sutskever. Neural GPUs learn algorithms. arXiv preprint arXiv:1511.08228, 2015.

[36] Ravi Kannan. Minkowski's Convex Body Theorem and Integer Programming. Mathematics of Operations Research, 12(3):415–440, 1987.

[37] Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. Deep Encoder, Shallow Decoder: Reevaluating the Speed-Quality Tradeoff in Machine Translation. arXiv preprint arXiv:2006.10369, 2020.

[38] Hayato Kimura, Keita Emura, Takanori Isobe, Ryoma Ito, Kazuto Ogawa, and Toshihiro Ohigashi. Output Prediction Attacks on SPN Block Ciphers using Deep Learning. IACR Cryptol. ePrint Arch., 2021.

[39] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[40] Philipp Koehn. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In Conference of the Association for Machine Translation in the Americas. Springer, 2004.

[41] Kim Laine and Kristin Lauter. Key recovery for lwe in polynomial time. Cryptology ePrint Archive, 2015.

[42] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. arXiv preprint arXiv:1912.01412, 2019.

[43] H.W. jr. Lenstra, A.K. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. Mathematische Annalen, 261:515–534, 1982.

[44] Richard Lindner and Chris Peikert. Better Key Sizes (and Attacks) for LWE-Based Encryption. In Topics in Cryptology – CT-RSA 2011, 2011.

[45] Mingjie Liu and Phong Q. Nguyen. Solving BDD by Enumeration: An Update. In Ed Dawson, editor, Topics in Cryptology – CT-RSA 2013, 2013.

[46] Vadim Lyubashevsky and Daniele Micciancio. On Bounded Distance Decoding, Unique Shortest Vectors, and the Minimum Distance Problem. In Advances in Cryptology, 2009.

[47] Daniele Micciancio. On the Hardness of Learning With Errors with Binary Secrets. Theory of Computing, 2018.

[48] Daniele Micciancio and Oded Regev. Lattice-based cryptography. Post-Quantum Cryptography, pages 147–191, 2009.

[49] Daniele Micciancio and Panagiotis Voulgaris. Faster exponential time algorithms for the shortest vector problem. Proc. of the 2010 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2010.

[50] Victor S Miller. Use of elliptic curves in cryptography. In Conference on the Theory and Application of Cryptographic Techniques. Springer, 1985.

[51] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks. arXiv preprint arXiv:2102.13019, 2021.

[52] Theodoros Palamas. Investigating the ability of neural networks to learn simple modular arithmetic. 2017.

[53] Chris Peikert. Public-Key Cryptosystems from the Worst-Case Shortest Vector Problem: Extended Abstract. In Proc. of the Forty-First Annual ACM Symposium on Theory of Computing, 2009.

[54] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. arXiv preprint arXiv:2009.03393, 2020.

[55] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets. arXiv preprint arXiv:2201.02177, 2022.

[56] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Proc. of the Thirty-Seventh Annual ACM Symposium on Theory of Computing, 2005.

[57] Ronald L Rivest. Cryptography and machine learning. In International Conference on the Theory and Application of Cryptology, pages 427–439. Springer, 1991.

[58] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 1978.

[59] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing Mathematical Reasoning Abilities of Neural Models. arXiv preprint arXiv:1904.01557, 2019.

[60] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In Proc. of ICML, 2017.

[61] Feng Shi, Chonghan Lee, Mohammad Khairul Bashar, Nikhil Shukla, Song-Chun Zhu, and Vijaykrishnan Narayanan. Transformer-based Machine Learning for Fast SAT Solvers and Logic Synthesis. arXiv preprint arXiv:2107.07116, 2021.

[62] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994.

[63] Jaewoo So. Deep learning-based cryptanalysis of lightweight block ciphers. Security and Communication Networks, 2020.

[64] Shivin Srivastava and Ashutosh Bhatia. On the Learning Capabilities of Recurrent Neural Networks: A Cryptographic Perspective. In Proc. of ICBK, 2018.

[65] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Proc. of NeurIPS, 2014.

[66] Andrew Trask, Felix Hill, Scott Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. arXiv preprint arXiv:1808.00508, 2018.

[67] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Proc. of NeurIPs, 2017.

[68] Sean Welleck, Peter West, Jize Cao, and Yejin Choi. Symbolic Brittleness in Sequence Models: on Systematic Generalization in Symbolic Mathematics, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No] At present, we run a proof of concept that cannot be used in real world implementations. Significant additional scaling work will be necessary before these techniques will be relevant to attacking real-world cryptosystems.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The source code will be released after publication of the paper.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See §5
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Since our results are binary (e.g. secret recovered or not), we do not report error bars.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See §5
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]

(c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Supplementary Materials

A Further Details of LWE

A.1 Ring Learning with Errors (§2)

We now define RLWE samples and explain how to get LWE instances from them. Let n be a power of 2, and let $R_q = \mathbb{Z}_q[x]/(x^n+1)$ be the set of polynomials whose degrees are at most $n-1$ and coefficients are from $\mathbb{Z}_q$. The set R_q forms a ring with additions and multiplications defined as the usual polynomial additions and multiplications in $\mathbb{Z}_p[x]$ modulo x^n+1. One RLWE sample refers to the pair

$$(a(x), b(x) := a(x) \cdot s(x) + e(x)),$$

where $s(x) \in R_q$ is the secret and $e(x) \in R_q$ is the error with coefficients subject to the error distribution.

Let $\mathbf{a}, \mathbf{s}$ and $\mathbf{e} \in \mathbb{Z}_q^n$ be the coefficient vectors of $a(x), s(x)$ and $e(x)$. Then the coefficient vector $\mathbf{b}$ of $b(x)$ can be obtained via the formula

$$\mathbf{b} = \mathbf{A}^{\text{circ}}_{a(x)} \cdot \mathbf{s} + \mathbf{e},$$

here $\mathbf{A}^{\text{circ}}_{a(x)}$ represents the $n \times n$ generalized circulant matrix of $a(x)$. Precisely, let $a(x) = a_0 + a_1 x + \ldots + a_{n-2}x^{n-2} + a_{n-1}x^{n-1}$, then $\mathbf{a} = (a_0, a_1, \ldots, a_{n-2}, a_{n-1})$ and

$$\mathbf{A}^{\text{circ}}_{a(x)} = \begin{bmatrix} a_0 & -a_{n-1} & -a_{n-2} & \ldots & -a_1 \\ a_1 & a_0 & -a_{n-1} & \ldots & -a_2 \\ a_2 & a_1 & a_0 & \ldots & -a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1} & a_{n-2} & a_{n-3} & \ldots & a_0 \end{bmatrix}.$$

Therefore, one RLWE sample gives rise to n LWE instances by taking the rows of $\mathbf{A}^{\text{circ}}_{a(x)}$ and the corresponding entries in $\mathbf{b}$.

A.2 Search to Decision Reduction for Binary Secrets (§2)

We give a proof of the search binary-LWE to decisional binary-LWE reduction. This is a simple adaption of the reduction in [56] to the binary secrets case. We call an algorithm a (T, γ)-distinguisher for two probability distributions $\mathcal{D}_0, \mathcal{D}_1$ if it runs in time T and has a distinguishing advantage γ. We use $\mathbf{LWE}_{n,m,q,\chi}$ to denote the LWE problem which has secret dimension n, m LWE instances, modulus q and the secret distribution χ.

Theorem A.1. *If there is a (T, γ)-distinguisher for decisional binary-$\mathbf{LWE}_{n,m,q,\chi}$, then there is a $T' = \widetilde{O}(Tn/\gamma^2)$-time algorithm that solves search binary-$\mathbf{LWE}_{n,m',q,\chi}$ with probability $1 - o(1)$, where $m' = \widetilde{O}(m/\gamma^2)$.*

Proof. Let $\mathbf{s} = (s_1, \ldots, s_n)$ with $s_i \in \{0, 1\}$. We demonstrate the strategy of recovering s_1, and the rest of the secret coordinates can be recovered in the same way. Let $m' = \widetilde{O}(1/\gamma^2)m$, given an LWE sample $(\mathbf{A}, \mathbf{b})$ where $\mathbf{A} \in \mathbb{Z}_q^{m' \times n}, \mathbf{b} \in \mathbb{Z}_q^{m'}$, we compute a pair $(\mathbf{A}', \mathbf{b}')$ as follows:

$$\mathbf{A}' = \mathbf{A} +' \mathbf{c}, \;\; \mathbf{b}' = \mathbf{b}.$$

Here $\mathbf{c} \in \mathbb{Z}_q^{m'}$ is sampled uniformly and the symbol " $+'$ " means that we are adding $\mathbf{c}$ to the first column of $\mathbf{A}$. One verifies by the definition of LWE that if $s_1 = 0$, then the pair $(\mathbf{A}', \mathbf{b}')$ would be LWE samples with the same error distribution. Otherwise, the pair $(\mathbf{A}', \mathbf{b}')$ would be uniformly random in $\mathbb{Z}_q^{m' \times n} \times \mathbb{Z}_q^{m'}$. We then feed the pair $(\mathbf{A}', \mathbf{b}')$ to the (T, γ)-distinguisher for $\mathbf{LWE}_{n,m,q,\chi}$, and we need to running the distinguisher $m'/m = \widetilde{O}(1/\gamma^2)$ times given the number of instances. Since the advantage of this distinguisher is γ with m LWE instances, and we are feeding it m' LWE instances, it follows from the Chernoff bound that if the majority of the outputs are "LWE", then

the pair $(\mathbf{A}', \mathbf{b}')$ is an LWE sample and therefore $s_1 = 0$. If not, $s_1 = 1$. Guessing one coordinate requires running the distinguisher $\widetilde{O}(1/\gamma^2)$ times, therefore, this search to reduction algorithm takes time $T' = \widetilde{O}(Tn/\gamma^2)$. Note that we can use the same m' LWE instances for each coordinate, therefore it requires $m' = \widetilde{O}(m/\gamma^2)$ samples to recover all the secret coordinates. □

A.3 Overview of Attacks on LWE

Typically, attacks on the LWE problem use an algebraic approach and involve lattice reduction algorithms such as BKZ [22]. The LWE problem can be turned into a BDD problem (Bounded Distance Decoding) by considering the lattice generated by LWE instances, and BDD can be solved by Babai's Nearest Plane algorithm [44] or the pruned enumeration [45], this is known as the primal BDD attack. The primal uSVP attack constructs a lattice via Kannan's embedding technique [36] whose unique shortest vector encodes the secret information. The Dual attack [48] finds a short vector in the dual lattice which can be used to distinguish the LWE samples from random samples. Moreover, there are also attacks that do not use lattice reduction. For instance, the BKW style attack [5] uses combinatorial methods; however, this assumes access to an unbounded number of LWE samples.

Binary and ternary secret distributions are widely used in homomorphic encryption schemes. In fact, many implementations even use a sparse secret with Hamming weight h. In [14] and [47], both papers give reductions of binary-LWE to hard lattice problems, implying the hardness of binary-LWE. Specifically, the (n, q)-binary-LWE problem is related to a $(n/t, q)$-LWE problem where $t = O(\log(q))$. For example, if $n = 256$ is a hard case for uniform secret, we can be confident that binary-LWE is hard for $n = 256 \log(256) = 2048$. But [11] refines this analysis and gives an attack against binary-LWE. Their experimental results suggest that increasing the secret dimension by a $\log(\log(n))$ factor might be already enough to achieve the same security level for the corresponding LWE problem with uniform secrets.

Let us now turn to the attacks on (sparse) binary/ternary secrets. The uSVP attack is adapted to binary/ternary secrets in [11], where a balanced version of Kannan's embedding is considered. This new embedding increases the volume of the lattice and hence the chance that lattice reduction algorithms will return the shortest vector. The Dual attack for small secret is considered in [6] where the BKW-style techniques are combined. The BKW algorithm itself also has a binary/ternary-LWE variant [7]. Moreover, several additional attacks are known which can exploit the sparsity of an LWE secret, such as [16, 23] . All of these techniques use a combinatorial search in some dimension d, and then follow by solving a lattice problem in dimension $n - d$. For sparse secrets, this is usually more efficient than solving the original lattice problem in dimension n.

B Additional Modular Arithmetic Results (§3)

Table 6: q values used in our experiments

$\lceil \log_2(q) \rceil$	q	$\lceil \log_2(q) \rceil$	q
5	19, 29	18	147647, 222553
6	37, 59	19	397921, 305423
7	67, 113	20	842779, 682289
8	251, 173	21	1489513, 1152667
9	367, 443	22	3578353, 2772311
10	967, 683	23	6139999, 5140357
11	1471, 1949	24	13609319, 14376667
12	3217, 2221	25	31992319, 28766623
13	6421, 4297	26	41223389, 38589427
14	11197, 12197	27	94056013, 115406527
15	20663, 24659	28	179067461, 155321527
16	42899, 54647	29	274887787, 504470789
17	130769, 115301	30	642234707, 845813581

Here, we provide additional information on our single and multidimensional modular arithmetic experiments from §3.1. Before presenting experimental results, we first highlight two useful tables. Table 6 shows the q values used in our integer and multi-dimension modular arithmetic problems. Table 7 is an expanded version of Table 1 in the main paper body. It shows how the $\log_2$ samples required for success changes with the base representation for the input/output, but includes additional values of base B (secret is fixed at 728).

Table 7: Base-2 logarithm of the number of examples needed to reach 95% accuracy, for different values of $\lceil \log_2(q) \rceil$ and bases.

$\lceil \log_2(q) \rceil$	Base												
	2	3	4	5	7	17	24	27	30	31	63	**81**	128
15	23	21	21	23	22	20	20	23	22	21	21	**20**	20
16	24	22	23	22	22	23	22	22	22	23	22	**22**	21
17	-	23	24	25	22	26	23	24	22	24	23	**22**	22
18	-	23	23	25	23	-	23	24	25	-	23	**22**	22
19	-	23	-	-	25	23	25	24	-	-	25	**25**	24
20	-	-	-	-	-	24	25	24	26	-	-	**24**	25
21	-	24	-	-	25	-	-	-	-	-	-	-	25
22	-	-	-	-	-	-	-	25	-	26	-	-	25
23	-	-	-	-	-	-	-	-	-	-	25	-	-
24	-	-	-	-	-	-	-	-	-	-	-	-	-

Base vs. Secret. We empirically observe that the base B used for integer representation in our experiments may provide side-channel information about the secret s in the 1D case. For example, in Table 8, when the secret value is 729, bases 3, 9, 27, 729 and 3332 all enable solutions with much higher q (8 times higher than the next highest result). Nearly all these are powers of 3^3 as is the secret $729 = 3^6$. In the table, one can see that these same bases provide similar (though not as significant) "boosts" in q for secrets on either side of 729 (e.g. $728, 730$), as well as for $720 = 3^6 - 3^2$. Based on these results, we speculate that when training on (a, b) pairs with an unknown secret s, testing on different bases and observing model performance may allow some insight into s's prime factors. More theoretical and empirical work is needed to verify this connection.

Ablation over transformer parameter choices. We provide additional experiments on model architecture, specifically examining the effect of model layers, optimizer, embedding dimension and batch size on integer modular inversion performance. Tables 9-12 show ablation studies for the 1D modular arithmetic task, where entries are of the form (best $\log_2(q)$/$\log_2(samples)$), e.g. the highest modulus achieved and the number of training samples needed to achieve this. The best results, meaning the highest q with the lowest $\log_2(samples)$, are in **bold**. For all experiments, we use the base architecture of 2 encoder/decoder layers, 512 encoder/decoder embedding dimension, and 8/8 attention heads (as in Section 3.1) and note what architecture element changes in the table heading.

We find that shallower transformers (e.g 2 layers, see Table 9) work best, allowing problem solutions with a much higher q especially when the base B is large. The AdamCosine optimizer (Table 10) generally worked best, but required a smaller batch size for success with larger base. For smaller bases, a smaller embedding dimension of 128 performed better (Table 11), but increasing base size and dimension simultaneously yielded good performance. Results on batch size (Table 12) do not show a strong trend.

C Additional information on SALSA Secret Recovery (§4.3)

Here, we provide additional information about SALSA's two secret recovery algorithms.

[3] And 3332 can easily be written out as a sum of powers of 3, e.g. $3332 = 3^8 - 3^7 - 3^6 - 3^5 - 3^4 + 3^2 + 3 - 1$.

Table 8: Relationship between base and secret. Numbers in table represent the highest $\log_2(q)$ value achieved for a particular base/secret combo. Values of $\log_2(q) >= 23$, indicating high performance, are **bold**.

Base	Secret value										
	720	721	722	723	724	725	726	727	728	729	730
2	-	18	16	-	-	-	16	16	-	16	-
3	19	16	18	16	18	18	19	18	21	**24**	20
4	18	18	18	18	18	-	-	-	18		18
5	18	17	16	16	16	18	18	17	16	19	16
7	-	18	-	18	-	20	-	-	19	18	-
9	**23**	18	18	18	18	18	18	21	18	**24**	**23**
11	20	20	-	19	21	21	21	20	-	-	19
17	18	18	-	19	19	18	18		20	20	-
27	**23**	-	**23**	18	18	18	21	22	21	**24**	22
28	-	20	18	-	20	18	-	19	**23**	18	-
49	18	22	18	19	21	18	-	18	19	18	18
63	20	21	18	20	19	19	18	19	18	-	-
128	20	-	18	22	20	-	19	18	19	19	19
729	18	20	19	18	19	21	19	19	18	**25**	18
3332	22	22	22	**23**	22	22	21	22	**23**	**23**	22

Table 9: 1D case: Ablation over number of transformer layers.

Base	# Transformer Layers		
	2	4	6
27	19/24	18/27	20/25
63	18/25	16/25	15/22
3332	**23/26**	23/-	18/22

Table 10: 1D case: Ablation over optimizers. Parenthetical denotes (# warmup steps, learning rate); * = batch size 128.

Base	Optimizer		
	Adam (0, $5e^{-5}$)	Adam (3000, $5e^{-5}$)	AdamCosine (3000, $1e^{-5}$)
27	18/26	19/24	**22/27**
3332	**23/26**	23/27	22/26
3332*	23/26	23/26	**23/25**

Table 11: 1D case: Ablation over embedding.

Base	Embedding Dimension			
	512	256	128	64
3	21/25	21/24	**22/26**	19/-
27	23/26	23/26	**23/25**	19/26
63	**23/27**	18/24	19/27	18/26
3332	**23/25**	23/26	23/26	23/27

Table 12: 1D case: Ablation over training batch size.

Base	Batch size				
	64	96	128	192	256
3	21/26	21/25	21/26	22/26	**23/26**
27	21/25	**24/27**	22/27	23/26	24/28
63	-	20/27	**23/25**	-	23/26
3332	23/26	23/26	**23/25**	**23/25**	23/26

C.1 Direct Secret Recovery

Recovering Secrets from Predictions. During the direct secret recovery phase, we must transform model predictions from sequences on integers in base B into binary secrets. This transformation is denoted by the $binarize$ function on line 7 of Algorithm 1. We first decode the n (one for each special $\mathbf{a}$ input) predictions into integers and concatenate the predictions into one n-long vector $\tilde{\mathbf{s}}$. Then, we use each of the following methods to binarize this vector: mean comparison, softmax mean comparison, and mode comparison.

- The mean comparison method takes the mean of $\tilde{\mathbf{s}}$ and computes *two* potential secrets from it via the following function: $f_{01}(\tilde{\mathbf{s}})$ sets all elements above the mean to be 0 and below it to be 1, and $f_{10}(\tilde{\mathbf{s}})$ sets all elements above the mean to be 1 and below it to be 0.
- To use the softmax mean comparison, we first take the softmax of $\tilde{\mathbf{s}}$. We then take the mean of $\tilde{\mathbf{s}}$ and use the same binarization method as before to get two secret predictions.

- The mode comparison method is similar but instead of taking the mean, it uses the mode of $\tilde{\mathbf{s}}$ as the divider between 0 and 1 methods.

Altogether, these binarization methods produce six secret guesses. In our SALSA evaluation, all of these are compared against the true secret $\mathbf{s}$, and the number of matching bits is reported. If $\tilde{\mathbf{s}}$ fully matches $\mathbf{s}$, model training is stopped. When $\mathbf{s}$ is not available for comparison, the methods in §4.4 can be used to verify $\tilde{\mathbf{s}}$'s correctness.

K values. At the end of each epoch, we use $10\ K$ values for direct secret guessing, 5 of which are fixed and 5 of which are randomly generated. The fixed K values are $K = [239145, 42899, q - 1, 3q + 7, 42900]$, while the random K values are chosen from the range $(q, 10q)$.

C.2 Distinguisher-Based Secret Recovery

Here, we provide more details on the parameters and subroutines used in Algorithm 2.

- τ: This parameter sets the bound on q that will be used for the distinguisher computation. In our experiments, we set $\tau = 0.1$.
- acc_τ: This denotes the distinguisher advantage. Let acc_τ denote the proportion of model predictions which fall within the chosen tolerance (e.g. accuracy within tolerance as described in §4.2), then $advantage = acc_\tau - 2 * \tau$.
- $LWESamples(t, n, q)$ is a subroutine that returns LWE samples $(\mathbf{A}, \mathbf{b}) \in \mathbb{Z}_q^{n \times t} \times \mathbb{Z}_q^t$, note that now columns of $\mathbf{A}$ corresponds to LWE instances.

C.3 Secret Recovery in Practice

Empirically, we observe that direct secret recovery recovers the secret more quickly than the distinguisher-based method. Figure 5 plots counts of which technique provided the successful secret guess for 120 SALSA runs with varying n and d. The direct secret guessing method success $> 90\%$ of the time, but occasionally the distinguisher is able to get the secret first. Occasionally, both methods simultaneously recover the secret.

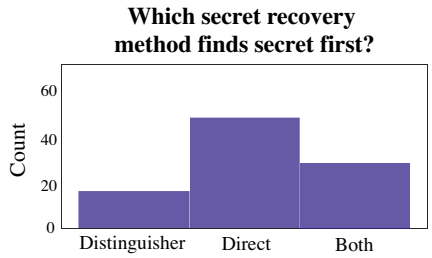

Figure 5: Frequency counts of which secret recovery method succeeds first for 90 successful SALSA runs.

D Additional SALSA Results (§5.2)

D.1 Effect of Architecture for (R)LWE Attacks

Several key architecture choices determine SALSA's ability to recover secrets with higher n and d, namely the encoder and decoder dimension as well as the number of attention heads. Other architecture choices determine the time to solution but not the complexity of problems SALSA could solve. For example, universal transformers (UT) are more sample efficient than regular transformers. Using gated loops in the UT with more loops on the decoder than the encoder reduced both model training time and the number of samples needed. Here, we present ablation results for all these architectural choices.

Universal Transformers vs. Regular Transformers. First, we see if universal transformers improve experimental efficiency or success. We run dueling experiments on medium size problems ($N = 50$, $d = 0.06$, $q = 251$, $B_{in}/B_{out} = 81$). For one experiment, we use regular transformers with between

Table 13: **Transformers vs UTs.** Ratio of training samples required for success for UTs with X/X encoder/decoder loops vs. regular transformers with X/X encoder/decoder layers.

Encoder Loops/Layers	Decoder Loops/Layers 2	4	8
2	1.2	4.7	0.8
4	0.7	0.4	0.6
8	0.1	0.1	0.1

Table 14: **Gated vs Ungated UTs.** Ratio of training samples required for success for gated UTs with X/X encoder/decoder loops vs. ungated UTs with same loop numbers.

Encoder Loops	Decoder Loops 2	4	8
2	1.0	1.3	0.3
4	0.3	0.3	0.3
8	0.1	0.1	0.1

Table 15: **Loops.** Average $\log_2$ of training samples required for $N = 50$, $h = 3$, $q = 251$, $base_{in}/base_{out} = 81$ as loops vary.

Encoder Loops	Decoder Loops 2	4	8
2	23.5	25.4	23.4
4	23.3	24.2	24.4
8	23.1	22.3	22.5

2 and 8 encoder/decoder layers. For the other, we use gated universal transformers with between 2 and 8 loops. We then compare the relative number of training samples needed to achieve success for both methods. As Table 13 shows, when model size increases, universal transformers prove more sample efficient, so we use them exclusively.

Gated vs Ungated UTs. To understand the effect of gating on sample efficiency, we run two experiments with medium-size problems ($N = 50$, $d = 0.06$, $q = 251$, $B_{in}/B_{out} = 81$). For both experiments, we use universal transformers with between 2 and 8 loops on the encoder/decoder. We use gates for one experiment but not for the other. As Table 14 shows, gated UTs are much more sample-efficient.

Number of Loops. Table 15 shows the average number of samples needed to recover the secret for $N = 50$, $h = 3$ with different numbers of encoder/decoder loops (dimension=1025/512, heads=16/4). There is a tradeoff between adding loops and increasing computation time, particularly for encoder loops as n increases. In our experiments, we elect to use 2 encoder loops and 8 decoder loops due to the significant training time needed for high n values with more encoder loops.

Encoder/Decoder Dimension. Table 3 in §5.2 demonstrates how increasing the encoder size and decreasing the decoder sizes enables secret recovery with fewer samples. Here, we explore the effect of encoder and decoder dimension on the n and hamming weight of secrets SALSA can recover. Our results, shown in Tables 16 and 17, follow the same pattern as before: higher encoder dimension and lower decoder dimension allow us to recover secrets with higher n. Furthermore, for $n = 30$, higher encoder dimension and/or smaller decoder dimension allows recovery of hamming weight 4 secrets.

Table 16: **Ablation over Encoder Dimension.** Proportion of secret bits recovered for varying n and encoder dimension. For all experiments, we fix decoder dimension to be 512, 2/2 layers, 2/8 loops. Green means secret was guessed, yellow means all 1s, but not all 0s, were guessed, and red means SALSA failed.

n	Encoder Dimension (hamming=3) 512	1024	2048	3040	Encoder Dimension (hamming=4) 512	1024	2048	3040
30	1.0	1.0	1.0	1.0	0.87	1.0	1.0	1.0
50	1.0	1.0	1.0	1.0	0.94	0.94	0.94	0.94
70	0.97	1.0	1.0	1.0	0.96	0.97	0.94	0.96
90	0.97	0.98	1.0	1.0	0.96	0.96	0.97	0.97

Attention Heads. We run experiments for varying n with 2 encoder/decoder layers, 1024/512 embedding dimension, 2/8 encoder/decoder loops, and varying attention heads to observe the impact of attention heads on SALSA's success. Increasing the number of encoder attention heads while keeping decoder heads at 4 allows SALSA to recover secrets for $n > 70$ (Table 18), although it slightly increases the number of samples needed for recovery (Table 19). Increasing the number of decoder heads increases the number of samples needed but does not provide the same scale-up for n.

Table 17: **Ablation over Decoder Dimension.** Proportion of secret bits recovered for varying n and encoder dimension. For all experiments, we fix encoder dimension to be 1024, 2/2 layers, 2/8 loops. Green means secret was guessed, yellow means all 1s, but not all 0s, were guessed, and red means SALSA failed.

n	Decoder Dimension (hamming=3)				Decoder Dimension (hamming=4)			
	256	768	1024	1536	256	768	1024	1536
30	1.0	1.0	1.0	1.0	1.0	1.0	0.90	0.87
50	1.0	1.0	1.0	0.94	0.94	0.92	0.92	0.92
70	1.0	1.0	1.0	0.96	0.96	0.94	0.94	0.94
90	1.0	0.97	0.97	0.97	0.97	0.96	0.97	-

Table 18: **Attention Heads: Effect on secret recovery.** Table of success for varying n with hamming 3 for encoder/decoder head combinations. Green means secret was guessed, yellow means all 1s, but not all 0s, were guessed.

N	**Encoder/Decoder Heads**							
	8/8	**16/4**	**16/8**	**16/16**	**32/4**	**32/8**	**32/16**	**32/32**
30	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
50	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
70	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
90	0.97	1.0	0.97	0.99	1.0	0.98	0.98	0.97

D.2 Effect of data parameters for (R)LWE Attacks

Complex relationships between N, q, B, and d affect SALSA's ability to fully recover secrets. Here, we explore these relationships, with success measured by the proportion of secret bits recovered. Table 20 shows SALSA's performance as n and q vary with fixed hamming weight 3. SALSA performs better for smaller and larger values of q, but struggles on mid-size ones across all N values (when hamming weight is held constant). This stands in contrast to traditional attacks on LWE, which only work well when q is large [20]. Table 21 shows the interactions between q and d with fixed $n = 50$. Here, we find that varying q does not increase the density of secrets recovered by SALSA. Finally, Table 22 shows the $\log_2$ samples needed for secret recovery with different input/output bases with $n = 50$ and hamming weight 3. The secret is recovered for all input/output base pairs except for $B_{in} = 17$, $B_{out} = 3$, and using a higher input base reduces the $\log_2$ samples needed for recovery.

D.3 Effect of training parameters on (R)LWE Attacks

We experiment with numerous training parameters to optimize SALSA's performance (e.g. optimizer, learning rate, floating point precision). While most of these settings do not substantively change SALSA's overall performance, we find that batch size has a significant impact on SALSA's sample efficiency. In experiments with $n = 50$ and Hamming weight 3, small batch sizes, e.g. < 50, allow recovery of secrets with much fewer samples, as shown in Figure 6. The same model architecture is used as for $n = 50$ in Table 2.

E Improving SALSA's Sample Efficiency

Here, we provide both theoretical and empirical results highlighting ways to improve SALSA's sample efficiency.

Generating New Samples. We explain how to generate new LWE samples from existing ones via linear combinations. Assume we have access to m LWE samples. Suppose a SALSA model can still learn from samples following a family of Gaussian distributions with standard deviations less than $N\sigma$, where σ is the standard deviation of the original LWE error distribution. The number of new samples we could make is equal to the number of vectors $\mathbf{v} = (v_1, \ldots, v_m)^T \in \mathbb{Z}^m$ such that

Table 19: **Attention Heads: Effect on $\log_2$ samples.** We test the effect of attention heads and report the $\log_2$ samples required to recover the secret in each setting. Experiments are run with $n = 50$, hamming 3.

Attention Heads (1024/512, X/X, 2/8)		
8/8	16/4,8,16	32/4,8,16,32
22.4	22.8, 22.9, 23.2	23.0, 23.1, 23.7, 24.7

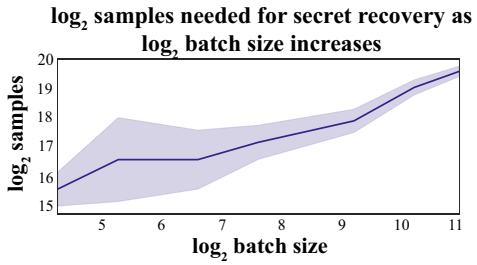

Figure 6: **Batch size and sample efficiency.** Smaller batch sizes allow SALSA to recover secrets faster. Experiments are run with $n = 50$, Hamming weight 3.

$\sum_{i=1}^{m} |v_i| \leq N^2$. For simplicity, assume that $v_i's$ are nonnegative. Then, there are $\sum_{n=1}^{N^2} \frac{(m+n-1)!}{(m-1)!(n)!}$ such vectors, and therefore this many new LWE samples one can generate.

Results on Generated Samples. Next, we show how SALSA performs when we combine different numbers of existing samples to create new ones for model training. We use the above method but do not allow the same sample to appear more than once in a given combination. We fix K, which is the number of samples used in each linear combination of reused samples. Then, we generate K coefficients for the combined samples, where each k_i is randomly chosen from $\{-1, 0, 1\}$. Finally, we randomly select K samples from a pregenerated set of samples, and produce a new sample from their linear combination with the k_i coefficients. These new samples follow error distribution with the standard deviation less than or equal to $\sqrt{K}\sigma$.

We experiment with different values of K, as well as different numbers of times we reuse a sample in linear combinations before discarding it. The $\log_2$ samples required for secret recovery for each (K, times reused) setting are reported in Table 8. The first key result is that the secret is recovered in all experiments, confirming that the additional error introduced via sample combination does not disrupt model learning. Second, as expected, sample requirements decrease significantly as we increase both K and times reused.

Samples vs. Sigma. We observe that the number of samples needed for secret recovery increases linearly with σ, see Figure 7.

Table 20: **N** vs **q**. Results reported are proportion of total secret bits recovered for various N/q combinations. Green cells mean the secret was fully guessed, yellow cells all the 1 bits were correctly guessed during training, and red cells mean SALSA failed. Fixed parameters: $h = 3$, $base_{in} = base_{out} = 81$. $1/1$ encoder layers, $1024/512$ embedding dimension, $16/4$ attention heads, $2/8$ loops.

N	$\log_2(q)$ 6	7	8	9	10	11	12	13	14	15
30	0.90	1.0	1.0	1.0	1.0	1.0	0.9	0.97	1.0	1.0
50	0.94	1.0	1.0	1.0	1.0	1.0	0.94	0.98	1.0	1.0
70	0.96	1.0	1.0	1.0	1.0	1.0	0.96	1.0	1.0	1.0
90	0.97	0.97	1.0	1.0	0.97	1.0	0.97	0.97	0.97	0.99

Table 21: **q** vs **d**. Results reported are proportion of total secret bits recovered for various q/d combinations. Green cells mean the secret was fully guessed, yellow cells all the 1 bits were correctly guessed during training, and red cells mean SALSA failed. Fixed parameters: $N = 50$, $base_{in} = base_{out} = 81$. $1/1$ encoder layers, $3040/1024$ embedding dimension, $16/4$ attention heads, $2/8$ loops.

d	$\log_2(q)$ 6	7	8	9	10	11	12	13	14	15
.06	0.94	1.0	1.0	1.0	1.0	1.0	0.94	0.98	1.0	1.0
.08	0.92	0.92	1.0	0.92	0.94	0.92	0.94	0.94	0.94	0.94
.10	0.90	0.94	0.96	0.90	0.90	0.92	0.90	0.92	0.94	0.92

Table 22: $\mathbf{B_{in}}$ v. $\mathbf{B_{out}}$. Effect of input and output integer base representation on $\log_2$ samples needed for secret recovery. In each row, the **bold** numbers represent the lowest $\log_2$ samples needed for this value of B_{in}. Fixed parameters: $n = 50$, hamming 3, 2/2 encoder layers, 1024/512 embedding dimension, 16/4 attention heads, and 2/8 loops.

B_{in}	B_{out} 3	7	17	37	81
7	25.8	**24.0**	25.4	24.5	24.9
17	-	25.9	27.2	25.6	**25.4**
37	22.8	**22.1**	22.6	22.2	22.9
81	22.2	**22.1**	22.4	21.9	**22.1**

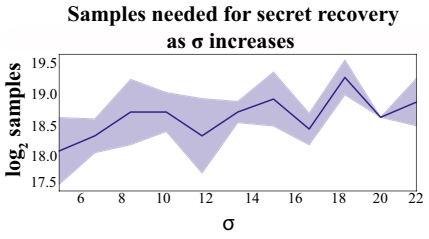

Figure 7: log_2 **samples vs.** σ**, fixed** n**.** As σ increases, the log_2 samples required for a fixed $n = 50$, $h = 3$ increases linearly.

Figure 8: **Effect of Sample Reuse on Sample Efficiency.** Sample reuse via linear combinations greatly improves sample efficiency. The secret is recovered in all experiments, indicating that error introduced by sample combination does not degrade performance. Parameters: $n = 50$, Hamming 3, 2/2 encoder layers, 1024/512 embedding, 16/4 attention heads, 2/8 loops.

K	**Times Samples Reused** 5	10	15	20	25
1	20.42	21.915	20.215	17.610	17.880
2	19.11	20.605	18.695	18.650	16.490
3	20.72	19.825	17.395	18.325	16.200
4	19.11	19.065	17.180	15.405	16.355