Scaling up DNA digital data storage by efficiently predicting DNA hybridisation using deep learning

David Buterez

Department of Computer Science and Technology
University of Cambridge, Cambridge, UK

Abstract. Deoxyribonucleic acid (DNA) has shown great promise in enabling computational applications, most notably in the fields of DNA digital data storage and DNA computing. Information is encoded as DNA strands, which will naturally bind in solution, thus enabling search and pattern-matching capabilities. Being able to control and predict the process of DNA hybridisation is crucial for the ambitious future of Hybrid Molecular-Electronic Computing. Current tools are, however, limited in terms of throughput and applicability to large-scale problems.

We present the first comprehensive study of machine learning methods applied to the task of predicting DNA hybridisation. For this purpose, we introduce an in silico-generated hybridisation dataset of over 2.5 million data points, enabling the use of deep learning. Depending on hardware, we achieve a reduction in inference time ranging from one to over two orders of magnitude compared to the state-of-the-art, while retaining high fidelity. We then discuss the integration of our methods in modern, scalable workflows.

Keywords: DNA hybridisation · Annealing · DNA storage · DNA computing · Deep learning

The use of DNA to facilitate computation is an active area of research, dating back to 1994 when Leonard Adleman solved a seven-node instance of the Hamiltonian path problem, using the toolbox of (DNA) molecular biology [1]. This pioneering work opened the gate to many interesting questions: can molecular machines be used to solve intractable problems? Is DNA suitable for long-term storage of digital information? More recently, are such methods scalable in the era of Big Data?

The focus has gradually changed from solving difficult computational problems to exploiting desirable properties of DNA, leading to the development of DNA digital data storage. It is now generally accepted that the amount of digital data is doubling at least every two years. Predictions from Seagate estimate that this quantity, called the Global Datasphere, will grow from 33 ZB (zetabytes) in 2018 to 175 ZB by 2025 [2]. Furthermore, the dominant storage mediums are traditional, with 59% of the storage capacity expected to come from hard disk drives and 26% from flash technologies. Synthetic DNA has been argued to be an attractive storage medium, for at least three prominent reasons [3]:

1. Density – The theoretical maximum information density of DNA is $10^{18}$ B/mm$^3$. Comparatively, this is a 7 orders of magnitude increase over tape storage.
2. Durability – Depending on storage conditions, DNA can be preserved for at least a few hundred years. Fossil studies reveal that DNA has a half-life of 521 years [4]. In appropriate conditions, researchers estimate that digital information can be recovered from DNA stored at the Global Seed Vault (at -18°C) after over 1 million years [5].
3. Future-proofing – Next-generation sequencing and technologies such as Oxford Nanopore [6] have made reading and writing DNA more accessible than ever. Furthermore, since DNA is the fundamental building block of life, it will be relevant for as long as humans exist.

In silicon-based computers, information is loaded and stored from unique locations denoted by a numerical address or index (random access). Implementing this property in DNA storage applications requires

* Work conceptualised and partially implemented while at Imperial College London.
a short, arbitrary sequence of DNA acting as the index, enabling selection and amplification by polymerase chain reaction (PCR) [7], [8]. Using this design, the index DNA sequence acts as both a unique identifier and a PCR primer, thus it must fulfil standard primer length requirements, often between 18 and 26 nucleotides [9], and annealing and melting temperature requirements, often between 30°C and 65°C [10], [11], [12]. As the scale of the primer library increases, designing sequences that strongly interact with their desired targets and weakly (preferably never) interact with non-intended sequences becomes progressively more difficult. Such sequence libraries are called orthogonal. It is also possible to exploit non-precise hybridisation to enable fuzzy or similarity search, by encoding closely-related entities (by some distance metric) in DNA, leading to similar datasets.

Successful and precise DNA hybridisation is required to implement computational features directly at the level of DNA. The OligoArchive project [7] introduced a technique capable of performing SQL operations such as selection, projection and join directly on the DNA molecules, by means of hybridisation and finding appropriate encodings for primary keys and attributes. Stewart et al. [13] used an approximation for thermodynamic analysis that is differentiable (necessary for backpropagation) in the form of a modified sigmoid function, with the goal of creating a content-addressable DNA database. Unfortunately, the estimation is far from perfect and the authors remarked that an important future direction is a more accurate approximation for thermodynamic yield.

Zhang et al. [14] predicted hybridisation kinetics from 36-nucleotide sequence pairs derived from human genes (non-synthetic), on a much more limited scale (100 pairs) using non-neural methods. We first introduced the concept of using CNNs to predict DNA hybridisation as a poster in June 2019, at the 25th International Conference on DNA Computing and Molecular Programming (DNA25), reporting significant gains in inference time. Close to our research direction, in a recent preprint Bee et al. [15] present a similarity search DNA database that employs a hybridisation predictor, a CNN trained on one-hot encodings of DNA. The authors state that hybridisation yields are either close to 0 or close to 1, suggesting that they do not consider intermediate values during training.

Unfortunately, NUPACK, the state-of-the-art in the analysis and design of nucleic acid systems [16], cannot be used for large-scale applications. Firstly, while NUPACK employs efficient dynamic programming algorithms, these still require $O(N^4)$ time and $O(N^2)$ memory, rendering multi-million applications intractable. Secondly, NUPACK 3 is only available as a collection of standalone executables, massively slowing down the performance due to repeated I/O operations. Thirdly, and perhaps most importantly, the computation of NUPACK is not differentiable, thus it cannot be used in neural networks. We acknowledge that the very recently introduced NUPACK 4 [17] has completely removed the second drawback. However, while NUPACK 4 provides faster, vectorised operations, the first point is only partially mitigated, as the massive speed improvements only apply to large complexes of long sequences. The third point still applies in full.

We recognise the need for scalable and accurate prediction of DNA hybridisation, necessarily as a differentiable operation that can be seamlessly plugged into neural network models. Hybridisation of two single-stranded sequences is quantified by the equilibrium concentration of the DNA duplex, here referred to as yield. We propose a modern, scalable deep learning approach for predicting DNA hybridisation (schematically illustrated in Figure 1), designed and evaluated on an in silico-generated hybridisation dataset of over 2.5 million single-stranded DNA sequence pairs with ground truth yields computed at 37.0°, 42.0°, 47.0°, 52.0°, 57.0°, and 62.0° (Celsius), capturing a wide range and largely mirroring the temperatures used in [18], [14], making the following contributions:
Figure 1: A high-level overview on how to integrate hybridisation prediction into DNA storage workflows. A trained machine learning model can be used as a standalone tool to assemble orthogonal or similar libraries of DNA sequences (left half of the figure). Alternatively, the neural network can be seamlessly integrated into a larger machine learning model as a subcomponent. The presented example is content-based search in a DNA database, where document features are extracted by a neural network (CNN for images, but text, video or audio inputs are conceivable) in a pairwise manner, another neural component generates appropriate encodings (usually one-hot) and the hybridisation predictor outputs the expected yield of the pair. Such a model is trained to associate similar documents to similar single stranded DNA sequences that form stable duplexes with the query sequence (right half of the figure).
1. **Integration with neural networks** – We investigate Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNN) and Transformers. Naturally, we formulate the problem such that backpropagation can be applied.

2. **Speed and accuracy** – A decrease in inference time of one to over two orders of magnitude compared to non-neural methods, while we empirically show that our best models can reach over 0.965 in $F_1$ score per class after binarising the predicted yields.

**Results**

**Designing a diverse hybridisation dataset**

To achieve a high-quality, comprehensive hybridisation dataset, the following two criteria must be satisfied: **size** – large enough to allow a broad range of machine learning methods (including deep learning) and **variety** – capture as many DNA interactions as possible, not only those corresponding to maximum or minimum yield. Unfortunately, we are not aware of any hybridisation dataset of biological origin that satisfies these conditions. Therefore, we introduce a synthetic dataset, where the DNA sequence pairs are generated from scratch and the yield computations are carried out by NUPACK.

Following the procedures presented in *Methods*, a final dataset of 2,556,976 sequence pairs or just over 2.5 million data points is assembled, with sequence lengths varying in the range 18–26. By examining the properties of extreme and intermediary samples, a machine learning model should ideally be able to deduce the Watson-Crick base-pairing rules and how these affect the yield. The yield distribution is summarised visually in Figure 2. For evaluation, we used stratified splits of 2,045,579 for training, 255,696 for validation and 255,701 for testing.

![Histogram of the hybridisation dataset.](image1.png)

**Figure 2:** Visual summary of the yield distribution at 57°C (the temperature used throughout the paper). For a discussion on the behaviour of the yield at different temperatures, see Supplementary Information 14. The yield is binned in 10 groups, each spanning a 0.1 interval. Brighter colours correspond to higher yield and are shared for the two subfigures. **a**, Low and high values are the most numerous, which is expected considering our generative procedure. **b**, The highest density is achieved at the extremes of 0, respectively 1. Given how sensitive the molecules are even to a 1-base change, we count the entire intermediate range of yields [0, 1] as one entity when considering how balanced the dataset is. In this regard, 1,058,364 pairs achieve low yields (< 0.1), 769,750 achieve very high yields (≥ 0.9) and 728,862 are in-between. It is important that samples with extremely low or high yields are well represented. In particular, there are virtually endless combinations of base pairs resulting in minimum yield.
To establish a baseline and study the difficulty of the task, we extend the dataset with extracted features for each sequence, enabling classical machine learning techniques to be applied. Alignment scores reveal the relatedness of two biological sequences and are hypothesised to be good predictors of DNA annealing even for arbitrary, artificially-generated sequences that are much shorter than typical naturally-occurring DNA fragments. As such, the first extracted feature is given by alignment scores, computed according to Methods. Furthermore, calculations such as the minimum free energy (MFE) of secondary structures indicate potentially undesirable behaviour. Motivating examples are provided in Figure 3.

Overall, we extend the dataset with the alignment score, secondary structure MFE, concentration of the single-stranded and double-stranded forms, and the GC content percentage of each sequence (ablation study in Supplementary Information 12).

Supervised learning using extracted features

The choice of machine learning algorithms is motivated by previous work (LDA in [18]), scalability to large problems (millions of samples) and widespread adoption and recognition. The four algorithms we evaluate are Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Random Forests (RF) and Neural Networks (NN). Our criteria prohibit the use of some well-known algorithms such as support-vector machines (SVM) due to limited scalability. For all chosen methods we perform hyperparameter optimisation, with the exception of QDA (very small number of tunable hyperparameters). For additional details see Supplementary Information 2.

The results of Figure 4c, 4e confirm that random forests and neural networks are the top performers on this classification task. Furthermore, the more granular performance report from Figure 4c indicates that all chosen methods perform well on the classification task. A general trend, consistent across all algorithms, can be identified. Precision for the Low class is high, indicating that when a low prediction is made, it is very likely correct. However, the recall for this class is relatively low, signalling that Low samples are often classified incorrectly as High. Almost all members of High are identified correctly (high recall). This trend is at its extreme for LDA, with the others (QDA, RF, NN) becoming more and more balanced across the two classes.

We acknowledge two important limitations of this approach. Firstly, generality – ideally, yields should be approximated directly, as a regression task, since different applications might require different cut-offs for the classification labels. Unfortunately, the methods presented so far are unable to approximate yields directly on our challenging dataset. Secondly, performance and scalability – while the actual inference times of the algorithms in Figure 4c are low, predictions require 9 pre-computations. Preferably, ML methods should be applied directly to sequences.

To address these limitations we introduce the use of deep learning methods to predict DNA hybridisation directly from pairs of sequences.

Deep learning strategies for DNA hybridisation

We employ two different deep learning paradigms: image-based, or more generally, models that operate on 2-dimensional grids with explicitly defined coordinates, and sequence-based. For the first category, we choose Convolutional Neural Networks (CNNs) as a representative, while for sequence models we investigate both Recurrent Neural Networks (RNNs) and Transformers.

One of the first challenges in designing CNN models is to find a 2-dimensional representation of DNA sequences that enables convolutions. Our inputs are pairs of single-stranded DNA represented using the alphabet \{A, C, G, T\}. The proposed 2D representation is as $4 \times N \times 2$ images ($height \times width \times channels$), where $N$ is the maximum sequence length.
Figure 3: Alignment and thermodynamic properties for various DNA sequences as reported by NUPACK.

a. Pair with alignment score 31.
b. Pair with alignment score 77.
c. A sequence with no remarkable secondary structure.
d. A sequence that exhibits self-complementarity.
e. Equilibrium concentration of the ssDNA is only 0.17 µmol.
f. Equilibrium concentration of the dsDNA is 0.92 µmol.

A sequence that is expected to be stable (ACTACAAGTGAGGACAGAAGATA).
A sequence that is expected to be more problematic in hybridisation reactions (TTTGCACGGAGCAGGAGCTCGGTTA).
A sequence can be similar enough to its reverse complement that it is more probable to find it in duplex formations with different instances of itself rather than in the normal single-stranded state. Illustrated is the sequence CCATGGAGGCGCGCTTT in a complex of size 2, each strand initially present in solution at concentration 1 µmol. The duplex formation of this sequence (f) is more than 5 times as abundant as the single-stranded conformation (e).
A simplified, schematic overview of the CNN architecture is provided in Figure 4a. Our CNN implementation is described in Methods and the precise architectural details and hyperparameters are discussed in Supplementary Table 3 and Supplementary Information 5.

By studying sequence-based algorithms, the existing knowledge from language processing can be transferred to DNA storage and computing. The string representation of DNA sequences is naturally fit for RNNs. Here, we propose a character-level RNN based on bi-directional Long Short-Term Memory (LSTM) [19] layers.

Compared to CNNs, which capture the spatial information within a receptive field and learn pixel-level filters, RNNs process the input data (characters) sequentially and maintain an internal memory state. Here, we hypothesise that the LSTM can effectively model the interaction between hybridising DNA strands, which requires capturing long-distance nucleobase relationships. Our choice of encoding DNA pairs also naturally suggests using a bi-directional implementation (see Figure 4b). A simplified overview of the RNN architecture is provided in Figure 4b, and the architectural details and hyperparameters are provided in Supplementary Information 5.

Finally, we turn our attention to the Transformer [20], an architecture now prevalent in deep learning, with its origins in Natural Language Processing. We refer the readers to [21] for an authoritative review, and to Supplementary Information 6 for a short summary. For this study, we adapt an established Transformer implementation, RoBERTa [22]. Our use falls within the encoder-only category: in language processing jargon, we formulate our task as a sentence-pair regression problem, where the inputs are two “sentences” (DNA sequences) with the goal of outputting a numerical value (hybridisation yield). At a high level, the workflow is very similar to Figure 4b, in the sense that the inputs are tokenised and propagated through multiple Transformer blocks. Distinctly from other models, RoBERTa requires two separate training phases: pre-training and fine-tuning. Our implementation is described in Methods.

Deep learning accurately predicts hybridisation

We evaluate three models termed simply CNN, RNN and RoBERTa and additionally a fourth convolutional model with a reduced number of convolutions and filter sizes: CNNLite. Performance metrics are summarised in Figure 4d, 4e. With the exception of RoBERTa, we report a marked increase in classification performance over the baseline methods, which completely failed in approximating yields.

We interpret the results at the level of false positives (FP) and false negatives (FN) as they enable a more fine-grained discussion. Examining the performance of the two best models (CNN and RNN), we find that the CNN model is slightly better at avoiding false negatives, at 1,174 FNs compared to 1,740 for the RNN. However, only 413 FNs are shared, meaning that a large portion of the FNs which are wrongly labelled by one model would be correctly classified by the other. It is important to note that the accurate labelling of the positives (High class) is an easier task than for negatives on our dataset, as indicated by the baseline methods.

More interestingly, the RNN offers a pronounced decrease in false positives, at 5,871 compared to 8,981 for the CNN. In this case, 4,732 FPs are shared, suggesting that, in general, the RNN makes the same mistakes as the CNN, but less often. False positives are the area where both the baseline and deep learning models struggled the most. FNs and FPs for the RNN indicate a trade-off, where the model loses some power in predicting true positives but significantly reduces false positives.

The CNNLite performs worse, on the whole, compared to the deeper CNN and the RNN. Generally, the erroneously classified samples are close to being subsets of the corresponding FNs and FPs of the CNN. While CNNLite is not as strong a performer as the first two models, its benefits are in inference times, which will be presented shortly.
Figure 4: **a, b.** Simplified representations of the deep learning architectures. **a.** Convolutional Neural Network overview. Each of the columns in the 2-channel grid (image) corresponds to a one-hot encoding of the four nucleobases for that particular strand position, while each channel represents an entire strand. 2D convolutions on the 2-channel one-hot encoded DNA strands are followed by 1D convolutions (only 3 channels shown) and fully-connected layers. **b.** Recurrent Neural Network overview. LSTM layers are widely used and recognised for their performance in language modelling tasks [23], as well as other sequence-based tasks [24]. For readability, we represent bi-directional interactions with two-headed arrows between the sequence elements. **c, d, e.** Classification and regression results for the evaluated machine learning models. The choice of metrics is explained in **Methods.** **c, d.** Graphical summary of precision, recall and F_1 score for the two classes of machine learning models. The **Low** class is represented by dark grey and the **High** class corresponds to the four bright colours (one colour for each method cases readability). **c.** The four baseline ML algorithms. Exact numerical values are provided in Supplementary Table 1. **d.** Classification metrics for the four deep learning models, after yield binarisation (with a threshold of 0.2). The numerical values are provided in Supplementary Table 2. **e.** The AUROC and MCC summarise the classification performance of all evaluated machine learning techniques; additionally, the MSE (Mean Squared Error) is reported for the four deep learning models.
Table 1: MCC of the four deep learning models on the selected temperatures. The results suggest that the models perform reasonably well even on temperatures they were not trained on.

| Model    | 37.0°C | 42.0°C | 47.0°C | 52.0°C | 62.0°C |
|----------|--------|--------|--------|--------|--------|
| RNN      | 0.862  | 0.872  | 0.888  | 0.916  | 0.833  |
| CNN      | 0.884  | 0.893  | 0.908  | 0.929  | 0.811  |
| CNN Lite | 0.893  | 0.901  | 0.914  | 0.929  | 0.801  |
| RoBERTa  | 0.925  | 0.930  | 0.930  | 0.914  | 0.762  |

The Transformer model, RoBERTa, has significant difficulties in discerning positives and exhibits the highest number of false positives, at 16,212. About half of these are overlapping with those reported by the CNN model, and about a third are shared with the RNN. Thus, the dataset contains difficult examples that are not correctly classified by any model and this is exacerbated for RoBERTa. The number of false negatives is low, at 1,174, a trend observed by the other models as well. These mediocre results could be partially explained by Transformers being notoriously data-hungry. It is possible that not even our training dataset of over 2.5 million sequence pairs is enough to fully exploit the architecture.

**Pre-trained models allow estimation on different temperatures**

Studying the performance of the deep learning approach on different temperatures can prove or refute the generalisability of the models. Furthermore, for time and cost-saving purposes, it is essential to investigate if existing trained models can be repurposed for different temperature ranges that were not seen during training.

For brevity, we report only the Matthews Correlation Coefficient in Table 1. Additional metrics are provided in Supplementary Table 6 and Supplementary Table 7.

We evaluate the models of Figure 4d on the five additional temperatures. Table 1 and Supplementary Table 6 suggest that the models perform reasonably well even on temperatures they were not trained on. The MSE continues to increase as the temperature is lowered and the distance from the training setting increases. For RNN, CNN and CNN Lite, this also translates to a progressive alteration of the classification metrics. Overall, the RNN model is less generalisable, i.e., performs worse on different temperatures compared to the CNNs. The most dramatic drop in performance is for 62.0°C, as high-scoring pairs tend to achieve much lower yields at this temperature.

Perhaps surprisingly, RoBERTa performs better in terms of classification metrics on the lower temperatures. In fact, this agrees with our previous observation that RoBERTa exhibits many false positives: as the temperature is lowered, low-scoring pairs are more likely to truly have high yields.

**Deep learning reduces prediction time by orders of magnitude**

Short inference times are crucial in scaling DNA hybridisation computations for the zettabyte future. To study this aspect, we perform a comprehensive empirical evaluation of the time required for the forward pass, i.e., prediction time. The choice of experimental platforms is described in detail in Supplementary Information 8, Supplementary Information 9 and the approach taken to measure time in Methods and Supplementary Information 10. Results are reported in Table 2.

Depending on the hardware used, we observe a speedup of one to over two orders of magnitude over multi-threaded NUPACK. When trained on a GPU, the previously identified best models (RNN and
Table 2: The execution of all deep learning methods is timed on the test dataset of 255,701 samples. The average execution time and the standard deviation are reported in seconds. Each deep learning method is run 10 times, after an initial warm-up run. The time elapsed to load the dataset into memory is not taken into account and the batch size was chosen to maximise inference time. All deep learning models use consumer hardware or openly-available hardware (the TPU platform is completely free to use).

| Model          | # Params. | Batch | Hardware      | Time (s)   | Speedup |
|----------------|-----------|-------|---------------|------------|---------|
| NUPACK 3       | N/A       | N/A   | 64-core VM    | 372.59     | × 1.00  |
| RoBERTa        | 6.1M      | 1024  | RTX 3090      | 388.44 ± 0.32 | × 0.96 |
| RNN            | 249K      | 8192  | RTX 3090      | 15.87 ± 0.10 | × 23.47|
|                |           | 4096  | TPUv2         | 3.60 ± 0.11 | × 103.5 |
| CNN            | 2.8M      | 512   | RTX 3090      | 23.84 ± 0.08 | × 15.63|
| CNNLite        | 470K      | 512   | RTX 3090      | 9.01 ± 0.00 | × 41.34|
|                |           | 4096  | TPUv2         | 1.28 ± 0.15 | × 290.21|

CNN) are around ×23 and respectively ×16 faster in making predictions, while TPUs almost trivialise inference times. The RNN’s speed is comparable to the CNN’s despite having less than one tenth of the number of trainable parameters. This is expected, as the sequential nature of RNNs prohibits the massively parallel optimisations possible for convolutions. For this reason, we introduced CNNLite, whose architecture is described in Supplementary Table 4. CNNLite still offers competitive performance (Figure 4e), while being almost twice as fast as the RNN. Due to the complex architecture and large number of parameters, RoBERTa (GPU) is slightly slower than NUPACK, while also underperforming in prediction accuracy. Given these two observations, it is difficult to currently recommend Transformers.

The inference time increases proportionally with the dataset size (Supplementary Information 13, Supplementary Figure 1).

Integrating fast approximation into large-scale workflows

We concentrate on the task of designing a set of orthogonal sequences. A brute-force approach to this problem quickly becomes unfeasible: checking every pair of a set is a quadratic problem in the length of the set: for a set of 100,000 initial sequences, over 5 billion pairs need to be checked. No matter how efficient the yield computation, we cannot tackle this problem using just brute-force. We can, however, devise a method that reduces the use of heuristics as much as possible and utilises our efficient yield approximation.

Concretely, we determine an appropriate pairwise Longest Common Substring (not subsequence, i.e., the characters have to be consecutive) - the LCS. This is motivated by the fact that a sequence has to have at least some similarity to its reverse complement for them to anneal. In particular, consecutive regions are much more stable than individual, separated nucleobases that match. We demonstrate our workflow using a randomly generated 20-nucleobases sequence set of size 100,000. We use MMseqs2 [25] to cluster sequences that achieve a minimum target LCS (in our case, 5). The full details are provided in Methods. After processing with MMseqs2, we arrive at a set of 20,033,335 pairs that passed the LCS
threshold (i.e., have a consecutive overlap of at least 5 nucleobases) and that need to be further analysed by the yield computation.

Compared to the brute-force approach, we have reduced the number of checks needed from over 5 billion to just 0.004%. Note that we do not lose anything in terms of accuracy, since the sequence pairs that we eliminate are so structurally different that it is meaningless to include them in the hybridisation computation.

For NUPACK, we estimate a running time of over 62 hours – two and a half days. Comparatively, based on our results from Table 2 and Supplementary Information 13 we estimate that the RNN prediction time is around 6 minutes for second generation TPUs, with the CNNs expected to be even faster. This kind of workflow was previously impossible, and earlier strategies relied only on human-designed heuristics to filter sequences. With our approach, we apply precise estimations at a considerably larger scale than before.

Discussion

We presented the first comprehensive study of machine learning techniques applied to the problem of predicting DNA hybridisation, aiming to improve DNA storage applications both qualitatively and quantitatively, and to encourage further research into this expanding area, especially by harnessing the power of machine learning for large, near-data processing-enabled systems. As part of this investigation, we introduced a carefully designed dataset of more than 2.5 million sequence pairs with hybridisation scores for 6 different temperatures. We evaluated a range of classical machine learning methods as a baseline, and followed by introducing deep learning architectures capable of accurately predicting yields while reducing inference time by one to over two orders of magnitude. We believe that this combination of high-fidelity and massive throughput is necessary for scaling up emerging DNA storage applications. Our work can act as a building block for new neural architectures, or on their own, the deep learning models can be used for DNA database library design of a scope that was previously impossible. Future studies could focus on generative methods that produce pairs matching a given yield, predicting base-pair probabilities (i.e., at the character level) or applying Graph Neural Networks on DNA strands represented as graphs, for example for edge prediction (topology of the duplex).

Methods

Dataset design

We start with a small number (e.g. 1,000 or 2,000) of randomly generated sequences where long repeats (≥ 3) of the same nucleobase are forbidden. We then perform three types of mutations: insertions, deletions and substitutions on each sequence. We incorporate both minor mutations (generally limited to a single type of mutation out of the three and which is applied once or twice) and severe mutations (all three types of mutations occur in the same sequence, possibly five or more times). To cover a wide range of thermodynamic interactions, we also apply the mutation operations on the reverse complement of the sequence. Thus, for each sequence, the mutation procedure generates a set of related sequences. We form a set of the initial randomly-generated sequences and all of their mutations, remove duplicates and generate all possible pairings. We repeat the procedure until the target size and variety constraints are achieved. To automate the yield computation process, we wrote Python wrappers for the NUPACK stand-alone executables. Each pair of inputs is written to a .in file, which is then fed to the complexes executable. In this work the default temperature is 57°C, instead of the NUPACK default 37°C; also see the discussion in Pre-trained models allow estimation on different temperatures. As with the length criterion, this is based on the consensus in molecular biology that primers should be designed broadly in the 50°−65°C range. The output of complexes is then used as input for the concentrations executable.
Finally, we can read the result as a float and incorporate it into our dataset. Based on this numerical value, we can assign labels to each record. A good threshold, as suggested by [18], is 0.2. Thus, pairs reported to be below 0.2 are labelled Low, otherwise High.

**Alignment scores**
When used as a proxy for DNA annealing, we have to reverse complement one of the sequences. A high alignment score indicates that a sequence is very similar to the other’s reverse complement. In this case, it is reasonable to assume that they will anneal. On the other hand, if the alignment score is low, then the sequence is dissimilar to the other’s complement and we would not expect them to bind. However, as DNA interactions are very sensitive, it is possible to encounter pairs that do not strictly follow these rules. A few examples are illustrated in Supplementary Information 1, Figure 3a and Figure 3b. To perform alignments, we use parasail [26], a fast implementation of the Smith-Waterman and Needleman-Wunsch algorithms for local, and respectively global and semi-global sequence alignment. Out of the three alignment techniques, the one that fits our task is semi-global alignment. This is because we would expect that two compatible sequences can be aligned from end to end, perhaps with mismatches and gaps in-between. It is also possible that considerable (> 4 nucleobases) deletions and insertions happened at the end of the sequences. This means that just global alignment is too weak, while local alignment is not optimal as we are not interested in matching isolated fragments of the sequences. The command used to compute the semi-global alignment of two sequences is:

```
parasail.sg_scan_16(str(s1), str(s2), 5, 2, parasail.dnafull)
```

where the input sequences `s1` and `s2` are Biopython [27] Seq objects. Also note that we are using a DNA substitution matrix, a gap open penalty of 5 and a gap extend penalty of 2. The graphical representation of the alignment can be computed by enabling the traceback feature:

```
parasail.sg_trace_scan_16(str(s1), str(s2), 5, 2, parasail.dnafull)
```

The resulting alignments, however, are not always representative of the actual binding that NUPACK predicts, as can be seen when comparing Supplementary Information 1 to the NUPACK outputs of Figure 3a and Figure 3b.

**Classification metrics**
The overall classification performance can be expressed through two metrics, the Matthews Correlation Coefficient (MCC) and Area Under the Receiver Operating Characteristics (AUROC). The AUROC is well-known and widely used in classification tasks, while the MCC has been recently argued to be more informative and reliable in binary classification tasks compared to popular metrics like accuracy and F₁ score [28], [29]. We do not reproduce the mathematical formulae here; however, they can be accessed through the cited studies.

**Convolutional Neural Networks**
As exemplified in Figure 3a and Figure 3b, duplex interactions are complex and often do not correspond to a simple end-to-end alignment. To effectively incorporate this property, we used semi-local alignment for the baseline models. For CNNs, we first perform 2D convolutions with large filters of size $4 \times 9$, covering slightly under 35% of the maximum sequence length. We follow with 1D convolutions of size 9, 3, 3 and finally 1. The large filters in the first two layers ensure that spatial information is captured from a wide area, while the following layers, decreasing in size, can learn more specialised features. Our experiments confirm this choice, as this architecture is superior to stacked convolutions of the same size, e.g. $3 \times 3$ (2D) then 3 (1D) on our regression task. We also note that the small input size of $4 \times N \times 2$ limits the use of very deep models or alternative layers such as residual blocks.
Similar encodings have been used in the literature, although not for pairs of sequences and mostly in different settings, for example in [13], [30], [31]. In principle, an entire architecture can be devised using only 2D convolutions, by applying (zero) padding on the output of the first convolution operator (we used this approach in our work at DNA25). For this study, 1-dimensional convolutions follow the first 2D operation. This change saves millions of learnable parameters while reaching very similar performance. All convolutions are followed by ReLU activation and batch normalisation, with dropout interspersed throughout. Following the convolutional operations, fully-connected layers predict a single numerical value, the hybridisation yield. The entire network is trained under a supervised learning setting with NUPACK yields as ground truth.

Recurrent Neural Networks

Our sequential pair encoding requires the concatenation of two single-stranded sequences, not before inserting a special separator token (\(<s>\) in Figure 4b). Additionally, a special beginning token, represented by \(<b>\) in Figure 4b is prepended to the resulting sequence. Symmetrically, the separator token also marks the ending (last character). Each character is indexed into a vocabulary and then fed through an embedding layer. At this point, multiple recurrent layers can be stacked. The hidden states of the special beginning token (\(<s>\)) are extracted from the last layer and ran through a dense layer that outputs the hybridisation yield. As LSTM layers do not possess the same interpretable hyperparameters as CNNs (e.g. filter sizes), we perform hyperparameter optimisation (see Supplementary Information 5).

Transformers

A short description of a general Transformer architecture and the standard pre-training schemes is provided in Supplementary Information 6. We perform pre-training using the collection of all unique single-stranded DNA sequences in our dataset (totalling 2,579,555 samples). As we formulate our task at the character level, we have no appropriate equivalent for longer sequences of text and thus for the NSP task. Consequently, only MLM is used during pre-training, which we consider useful as predicting nucleobases based on the surrounding context and position is important for learning the structure of DNA sequences. The purpose of the pre-training scheme is learning to model DNA at the primary structure level. During fine-tuning, the pre-trained model is adjusted for the sentence-pair regression task by appending a fully-connected layer and propagating gradients throughout the entire model. We follow our intuition that modelling DNA interactions is not as complex as modelling natural language and since training Transformer models is extremely computationally demanding, we do not perform hyperparameter optimisation; instead, we base our architecture on documented models such as roberta-base. The particular framework we employ is HuggingFace’s Transformers [32].

Measuring inference time

For each platform, we perform 10 trials (repetitions) after one warm-up trial and report the results. We took all precautions we are aware of to ensure that measuring the inference time is as precise and fair as possible. NUPACK was executed on a 64-core Azure VM equipped with the premium SSD option (an important observation since NUPACK outputs must be written and read from disk). Since NUPACK is a single-threaded application, we wrote Python wrappers around NUPACK and employed a ProcessPool from the pathos multiprocessing library to distribute NUPACK logic, reads and writes to disk across all available cores. This dramatically improved NUPACK running time from more than 5 hours when running (in single-threaded mode) on an AMD Ryzen 5950X processor and a PCI Express 4.0 SSD to slightly more than 6 minutes on the Azure VM. This is a more than ×50 improvement over the out-of-the-box NUPACK performance. The execution of the code was timed using the time shell command preceding the script invocation and the wall clock time was read and reported.

When measuring execution times on the GPU, additional measures were taken. More specifically, we
overrode the PyTorch Lightning method `test_step` to perform only the forward pass and no additional computations (such as the loss function), and surrounded the call to the `test` method of the PyTorch Lightning `Trainer` object with two `torch.cuda.Event(enable_timing=True)` objects calling their `record` method. Before calculating the elapsed time, we wait for all kernels in all streams on the CUDA device to complete (`torch.cuda.synchronize`). A representative snippet of timing code for GPUs is provided in Supplementary Information 10. For GPUs, the PyTorch `DataLoader` object does not use multiple workers (`num_workers=0`). For consistency, we use the same setting for TPUs.

Finally, to the best of our knowledge, there currently is no established way to measure execution time on Tensor Processing Units (TPUs). To combat this, we implemented our own timing code which is consistent with the epoch time reported by PyTorch Lightning (note that the epoch time only reports elapsed seconds, i.e., low temporal resolution). The PyTorch Lightning method `test_step` is subjected to the same treatment as above. Our code defines a boolean flag inside the `LightningModule` to be timed; the first time the `forward` method is entered, the flag is set and the timestamp is recorded. The PyTorch Lightning method `on_test_epoch_end` is also overridden to capture a final timestamp and compute the elapsed time. A representative snippet of timing code for TPUs is provided in Supplementary Information 10.

**Finding an appropriate Longest Common Substring**

We start by investigating our 2.5 million pairs dataset. In particular, we looked only at pairs with high yield (above 0.2), which correspond to about 56% of the dataset, or more exactly, 1,437,758 elements. Out of the 1,437,758 high yield pairs, the shortest LCS among pairs is 4, which happens for only 29 pairs, or about 0.00002% of the high yield set. Also, the 29 pairs have an average yield of only 0.37, which is overall still quite low, even though it passes the threshold of 0.2. The pairs with an LCS of 5 are more numerous, at 349 samples, with an average yield of 0.47. We have thus decided that for our next experiment, working with a minimum LCS of 5 is the most appropriate. The running time performance difference between 4 and 5 is significant enough in the clustering stage described next to justify the trade-off. When working with sets of sequences of length 20, this means a minimum 25% identity match between sequences, which we find reasonable. The problem is now to efficiently process a large dataset such that sequences with shared regions of 25% or longer are associated. From our experience, the most appropriate tool for this task is the recently-developed MMseqs2. Briefly, MMseqs2 is described as an "ultra fast and sensitive search and clustering suite", and its performance is said to scale almost linearly with the number of processor cores.

**Clustering with MMseqs2**

Assuming a set of single stranded DNA sequences in FASTA format, the first step is to build a MMseqs2 database, with the following command:

```bash
mmseqs createdb <filename>.fasta <dbname> --dbtype 2 --max-seq-len <N>
```

The `dbtype` option indicates a nucleotide database. For example, in practice the command might look like:

```bash
mmseqs createdb init.fasta seqsDB --dbtype 2 --max-seq-len 20
```

The crucial step is the search. A fully configured search command looks like the following:

```bash
mmseqs search <dbname> <dbname> <alignname> <tmp> -s 7.5 -k 5 --min-seqs 10000 --exact-kmer-matching 1 --spaced-kmer-mode 0 --min-seq-id 0.25 --alignment-mode 4 -e inf --min-aln-len 5 --sub-mat dnafull --max-seq-len 20 --strand 0 --search-type 3
```

A short description is in order. The dbname is the same as used in the database creation step, while alignname and tmp are names for the output. The sensitivity is set to the maximum (7.5). The search is to be done using k-mers of size 5. We allow a maximum of 10,000 related sequences to be returned for each individual query sequence. Exact k-mer matching is enabled, and spaced k-mer mode is disabled to ensure alignment only of ungapped regions. The minimum sequence similarity is defined to be 0.25. Alignment mode 4 is faster, and only considers ungapped alignments. The -e inf option targets similarity, and we want to consider all possibilities. The minimum alignment length is set to 5. The substitution matrix is set to dnafull, the one used for DNA sequences. We again specify a maximum sequence length of 20. The option -strand 0 is important, as it only searches for similar regions in the reverse complements of the set. Finally, a nucleotide search is specified by -search-type 3.

The low running time of this search allows this procedure to be carried out even on a laptop. On our portable experimental platform with 12 CPU threads, the search takes under 15 minutes. We have also verified the search speed on an Azure VM with 64 CPUs and the search finishes in under 5 seconds. The generated output file has 39,432,713 pairs, with a small sample reproduced in Supplementary Table 5.

We notice that it is possible for some sequences to be associated with themselves, i.e., have a pair of identical IDs. We eliminate these from the set, discovering that there are just 7,056 such pairs. We also notice that it is possible for symmetric pairs to occur, i.e., a pair such as (seq10, seq20) and (seq20, seq10). By eliminating one of these duplicates, we are left with 20,033,335 pairs out of the 39,425,657 from the previous filtering step, which is a 51% smaller set.

Data Availability

The author declares that the dataset and the source code supporting the findings of this study are hosted on GitHub at https://github.com/davidbuterez/dna-hyb-deep-learning.

Author contributions statements

D.B. conceptualised the study, ran the experiments, wrote the manuscript text and prepared all the figures.

Competing interests

The author declares no competing interests.
References

1. Adleman, L. M. Molecular Computation of Solutions to Combinatorial Problems. *Science* **266**, 1021–1024. ISSN: 00368075, 10959203. http://www.jstor.org/stable/2885489 (1994).

2. Reinsel, D., Gantz, J. & Rydning, J. The Digitization of the World From Edge to Core tech. rep. (2018). https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

3. Carmean, D. et al. DNA Data Storage and Hybrid Molecular–Electronic Computing. *Proceedings of the IEEE* **107**, 63–72 (2019).

4. Allentoft, M. E. et al. The half-life of DNA in bone: Measuring decay kinetics in 158 dated fossils. *Proceedings of the Royal Society B: Biological Sciences*. ISSN: 14712954 (2012).

5. Grass, R. N., Heckel, R., Puddu, M., Paunescu, D. & Stark, W. J. Robust chemical preservation of digital information on DNA in silica with error-correcting codes. *Angewandte Chemie - International Edition*. ISSN: 15213773 (2015).

6. Jain, M., Olsen, H. E., Paten, B. & Akeson, M. The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome Biology* **17**, 239. ISSN: 1474-760X. https://doi.org/10.1186/s13059-016-1103-0 (2016).

7. Appuswamy, R. et al. OligoArchive: Using DNA in the DBMS storage hierarchy in Conference on Innovative Data Systems Research (CIDR) (2019).

8. Stewart, K. et al. A content-addressable DNA database with learned sequence encodings in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (2018). ISBN: 9783030000295.

9. Zhang, J. X. et al. Predicting DNA hybridization kinetics from sequence. *Nature Chemistry* **10**, 91–98. ISSN: 1755-4349. doi:10.1038/nchem.2877 (2018).

10. Fornace, M. E., Porubsky, N. J. & Pierce, N. A. A Unified Dynamic Programming Framework for the Analysis of Interacting Nucleic Acid Strands: Enhanced Models, Scalability, and Speed. *ACS Synthetic Biology* **9**, PMID: 32910644, 2665–2678. eprint: https://doi.org/10.1021/acssynbio.9b00523 (2020).

11. Beliveau, B. J. et al. OligoMiner provides a rapid, flexible environment for the design of genome-scale oligonucleotide in situ hybridization probes. *Proceedings of the National Academy of Sciences*. ISSN: 0027-8424 (2018).
19. Hochreiter, S. & Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* 9, 1735–1780. issn: 0899-7667. https://doi.org/10.1162/neco.1997.9.8.1735 (Nov. 1997).

20. Vaswani, A. et al. Attention Is All You Need. *CoRR abs/1706.03762.* arXiv: 1706.03762. http://arxiv.org/abs/1706.03762 (2017).

21. Tay, Y., Delhghani, M., Bahri, D. & Metzler, D. Efficient Transformers: A Survey 2020. arXiv: 2009.06732 [cs.LG].

22. Liu, Y. et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR abs/1907.11692.* arXiv: 1907.11692. http://arxiv.org/abs/1907.11692 (2019).

23. Irie, K., Tüske, Z., Alkhouli, T., Schlüter, R. & Ney, H. LSTM, GRU, Highway and a Bit of Attention: An Empirical Overview for Language Modeling in Speech Recognition in Interspeech 2016 (ISCA, Sept. 2016). https://doi.org/10.21437/interspeech.2016-491.

24. Chung, J., Gulcehre, C., Cho, K. & Bengio, Y. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling 2014. arXiv: 2009.06732 [cs.LG].

25. Steinegger, M. & Söding, J. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology.* issn: 1087-0156. http://www.nature.com/doifinder/10.1038/nbt.3988 (Oct. 2017).

26. Daily, J. Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments. *BMC Bioinformatics* 17, 81. issn: 1471-2105. https://doi.org/10.1186/s12859-016-0930-z (2016).

27. Cock, P. J. A. et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics* 25, 1422–1423. issn: 1367-4803. eprint: https://academic.oup.com/bioinformatics/article-pdf/25/11/1422/944180/btp163.pdf. https://doi.org/10.1093/bioinformatics/btp163 (Mar. 2009).

28. Chicco, D. & Jurman, G. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics* 21, 6. issn: 1471-2164. https://doi.org/10.1186/s12864-019-6413-7 (2020).

29. Chicco, D. Ten quick tips for machine learning in computational biology. eng. *BioData mining* 10. 29234465[pmid], 35–35. issn: 1756-0381. https://pubmed.ncbi.nlm.nih.gov/29234465 (2017).

30. Min, X. et al. Predicting enhancers with deep convolutional neural networks. *BMC Bioinformatics* 18. https://doi.org/10.1186/s12859-017-1878-3 (Nov. 2017).

31. Zhang, Z. et al. Deep learning in omics: a survey and guideline. *Briefings in Functional Genomics* 18, 41–57. https://doi.org/10.1093/bfgp/ely030 (Sept. 2018).

32. Wolf, T. et al. HuggingFace’s Transformers: State-of-the-art Natural Language Processing. *CoRR abs/1910.03771.* arXiv: 1910.03771. http://arxiv.org/abs/1910.03771 (2019).

33. Pedregosa, F. et al. Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot. *Journal of Machine Learning Research.* issn: ISSN 1533-7928 (2011).

34. Akiba, T., Sano, S., Yanase, T., Ohta, T. & Koyama, M. Optuna: A Next-generation Hyperparameter Optimization Framework in Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2019).

35. Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization 2017. arXiv: 1412.6980 [cs.LG].

36. Dosovitskiy, A. et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale 2020. arXiv: 2010.11929 [cs.CV].

37. Raschka, S. MLxtend: Providing machine learning and data science utilities and extensions to Python’s scientific computing stack. *The Journal of Open Source Software* 3. http://joss.theoj.org/papers/10.21105/joss.00638 (Apr. 2018).

38. Harris, C. R. et al. Array programming with NumPy. *Nature* 585, 357–362. https://doi.org/10.1038/s41586-020-2649-2 (Sept. 2020).
Supplementary information

1 Alignment scores

Notice that there are discrepancies between the parasail alignments below and the structures reported by NUPACK in Figure 3a and Figure 3b.

| Score | Yield |
|-------|-------|
| 130   | 1.0   |
| GAATACTGTCAGTGAGAGGATCTGCC | GAATACTGTCAGTGAGAGGATCTGCC |
| 77    | 0.12  |
| TTGTCATACGCTGTAAGAG     | TTGTCATCCGCTGTAAGCG    |
| 31    | 0.20  |
| ------|-------|
| AGAGCAAAACGGCGCCTTCTCTCG | AGAGCAAAACGGCGCCTTCTCTCG |
| 86    | 0.82  |
| AAATCAGGTA---TGCGGTAAAG | AAATCAGGTA---TGCGGTAAAG |

2 Hyperparameters for LDA, QDA, RF and NN

The hyperparameter optimisation for the baseline machine learning algorithms is performed using sklearn [33], more specifically with the GridSearchCV class configured to use our validation split (possible using PredefinedSplit). The monitored metric is the Matthews Correlation Coefficient, as advised in [28], [29]. The chosen search space is inspired by the default hyperparameter settings and is meant to provide reasonable coverage, keeping in mind a trade-off between exhaustiveness and time.

LDA

For LDA, the number of hyperparameters is small. We use a parameter grid with two entries:

{solver: [lsqr, eigen], shrinkage: [auto, *np.linspace(0.0, 1.0, num=11)],
 {solver: [svd]}

The found (and used) hyperparameters are: solver: eigen, shrinkage: 0.

Random forests

For random forests, the search space included:

{n_estimators: [30, 100, 200], max_depth: [None, 10, 30, 100],
 max_features = [None, auto, sqrt], min_samples_split = [2, 5],
 min_samples_leaf = [1, 2, 4], bootstrap = [True, False])

The returned hyperparameters were bootstrap: True, max_depth: 10, max_features: sqrt
min_samples_leaf: 1, min_samples_split: 2, n_estimators: 100. However, training with default parameters resulted in better performance on the test set. Thus, the reported results make use of the default hyperparameters.

Neural networks

In the case of neural networks we tuned hyperparameters using Optuna [34], an open-source optimisation framework that includes state-of-the-art algorithms. In particular, Optuna also includes a pruning capability to stop training unpromising trials. The hyperparameters included in the search are: the number of hidden dense layers n_layers ∈ {1, 2, 3, 4, 5}, a single dropout value to be applied after every
hidden layer dropout $\in [0.2, 0.5]$, the number of neurons per hidden layer $n_{\text{units}} \in \{4, 5, \ldots, 128\}$, the batch size $\text{batch} \in \{32, 64, 128, 256, 512\}$ and the learning rate $\text{lr} \in [0.0001, 0.1]$ (log domain). The neural optimisation algorithm is Adam [35].

Optuna was run with the HyperbandPruner for 50 trials with the goal of minimising the validation loss (binary cross-entropy as it is a classification task). The returned (and used) best hyperparameters were: $n_{\text{layers}}$: 4, dropout: 0.3296, $n_{\text{units}_0}$: 117, $n_{\text{units}_1}$: 18, $n_{\text{units}_2}$: 7, $n_{\text{units}_3}$: 19, batch: 1024, lr: 0.0002 (rounded to 4 decimal places).

3 Classification metrics for LDA, QDA, RF and NN

The following table provides the basis for our discussion on the classification performance of the baseline ML models.

Supplementary Table 1: Precision, recall and $F_1$ corresponding to Figure 4c.

| Metric     | LDA  | QDA  | RF    | NN    |
|------------|------|------|-------|-------|
|            | Low  | High | Low   | High  | Low   | High  | Low   | High  |
| Precision  | 0.996 | 0.881 | 0.975 | 0.901 | 0.966 | 0.927 | 0.971 | 0.914 |
| Recall     | 0.827 | 0.998 | 0.860 | 0.983 | 0.901 | 0.975 | 0.881 | 0.980 |
| $F_1$      | 0.904 | 0.936 | 0.914 | 0.940 | 0.932 | 0.950 | 0.924 | 0.946 |

4 Classification metrics for CNN, RNN and RoBERTa

Similarly to the above, we report the classification metrics for the deep learning models after applying the 0.2 threshold on the predicted yields (binarisation).

Supplementary Table 2: Precision, recall and $F_1$ corresponding to Figure 4d.

| Metric     | RNN  | CNN  | CNNLite | RoBERTa |
|------------|------|------|---------|---------|
|            | Low  | High | Low     | High    | Low    | High   | Low    | High   |
| Precision  | 0.984 | 0.960 | 0.989   | 0.941   | 0.990  | 0.931  | 0.985  | 0.898  |
| Recall     | 0.948 | 0.988 | 0.920   | 0.992   | 0.906  | 0.993  | 0.855  | 0.990  |
| $F_1$      | 0.965 | 0.974 | 0.953   | 0.966   | 0.946  | 0.961  | 0.915  | 0.942  |
5 Hyperparameters for CNN, RNN and RoBERTa

As the three architectures have innate differences, our approach to designing each model is different.

CNN

For our CNN model, we follow a top-down approach based on the properties of our inputs (pairs of DNA sequences represented as grids). At a high level, the network layers can be grouped in convolutional blocks, where each block has a convolutional layer, an activation (ReLU) layer and a batch normalisation layer (in this order), with dropout layers interspersed according to Supplementary Table 3. The model is trained with the Adam optimiser and a learning rate of 0.0001, a batch size of 256 (maximum that would fit in 8GB of GPU memory) with the Minimum Squared Error (MSE) loss and early stopping set to a patience of 3 epochs and no maximum number of epochs.

We also performed hyperparameters search for the CNN architecture. The search space was defined by: number and size of filters $\text{conv2d\_size} \in \{9, 10, 11, 12, 13\}$, $\text{conv1d\_size1} \in \{7, 8, 9, 10, 11\}$, $\text{conv1d\_size2, conv1d\_size3} \in \{3, 4, 5, 6, 7\}$, $\text{conv1d\_size4} \in \{1, 2, 3\}$, $\text{conv2d\_filter, conv1d\_filter1, conv1d\_filter2} \in \{256, ..., 768\}$, $\text{conv1d\_filter3} \in \{128, ..., 384\}$, $\text{conv1d\_filter4} \in \{32, ..., 128\}$, the batch size $\text{batch\_size} \in \{256, 512, 1024, 2048, 4096, 8096\}$ and the learning rate $lr \in [0.0001, 0.1]$ (log domain). Dropout can be applied after each convolutional layer, with a choice of probability in $[0, 0.5]$ and after the linear layers with probability in $[0.1, 0.5]$. The choice of size for the linear layers is based on the value selected for $\text{conv1d\_filter4}$. The values found by the hyperparameter search are available in the repository. However, they were not used as the classification performance on the test set using the reported hyperparameters was slightly worse than the architecture described in Supplementary Table 3.

Supplementary Table 3: Overview of the CNN architecture.

| Layer               | In channels | Out channels | Filter or dropout |
|---------------------|-------------|--------------|-------------------|
| 2D Convolution      | 2           | 512          | $4 \times 9$     |
| Dropout             | *           | *            | 0.2               |
| 1D Convolution      | 512         | 512          | 9                 |
| 1D Convolution      | 512         | 128          | 3                 |
| Dropout             | *           | *            | 0.2               |
| 1D Convolution      | 128         | 128          | 3                 |
| 1D Convolution      | 128         | 64           | 1                 |
| Fully connected     | 384         | 256          | *                 |
| Fully connected     | 256         | 128          | *                 |
| Dropout             | *           | *            | 0.2               |
| Fully connected     | 128         | 1            | *                 |
Supplementary Table 4: Overview of the CNNLite architecture.

| Layer            | In channels | Out channels | Filter or dropout |
|------------------|-------------|--------------|-------------------|
| 2D Convolution   | 2           | 256          | $4 \times 9$     |
| Dropout          | *           | *            | 0.2               |
| 1D Convolution   | 256         | 128          | 9                 |
| 1D Convolution   | 128         | 64           | 3                 |
| Fully connected  | 512         | 256          | *                 |
| Dropout          | *           | *            | 0.2               |
| Fully connected  | 256         | 1            | *                 |

**RNN**

The RNN architecture includes an embedding layer, a configurable multi-layer bi-directional LSTM block, a dropout layer and the fully-connected regression layer (in this order). The parameters of the LSTM-based model are not as intuitive as their CNN counterpart and we address this issue by performing hyperparameter optimisation using Optuna. The search space includes: the embedding dimension $\text{emb\_dim} \in \{16, 32, 64\}$, the number of LSTM layers $\text{n\_layers} \in \{1, 2, 3, 4\}$, the LSTM dropout $\text{lstm\_dropout} \in [0.1, 0.5]$, the number of features in the hidden state $\text{hidden} \in \{32, 64, 100\}$, the dropout preceding the regression layer $\text{lin\_dropout} \in [0.1, 0.5]$, the batch size $\text{batch} \in \{64, 128, 256\}$ and the learning rate $\text{lr} \in [0.0001, 0.1]$ (log domain). As before, the model is trained with the MSE loss and early stopping set to a patience of 3 epochs and no maximum number of epochs.

The returned (and used) RNN hyperparameters are $\text{emb\_dim}: 32$, $\text{lstm\_dropout}: 0.2412$, $\text{n\_layers}: 3$, $\text{hidden}: 64$, $\text{lin\_dropout}: 0.2166$, $\text{batch}: 128$, $\text{lr}: 0.006$ (rounded to 4 decimal places).

**RoBERTa**

Transformer models require substantially more computing power to train end-to-end (including both the pre-training and the fine-tuning stages). As such, it is not feasible to perform hyperparameter optimisation for RoBERTa. However, we base our configuration on known RoBERTa architectures such as roberta-base and on the intuition that modelling DNA sequence hybridisation is an easier task than natural language processing.

As described in the main text, the Transformer model is first pre-trained on a Masked Language Model (MLM) task after the inputs have been tokenised with RobertaTokenizerFast (the provided tokenisation class). The pre-training phase uses the following hyperparameters: vocabulary size $\text{vocab\_size} = 5000$, dimension of the encoder and pooler layers $\text{hidden\_size} = 256$, number of encoder hidden layers $\text{num\_hidden\_layers} = 6$, number of attention heads for each attention layer of the encoder $\text{num\_attention\_heads} = 8$, dimensionality of the intermediate layer $\text{intermediate\_size} = 1024$, the maximum sequence length that the model might ever use $\text{max\_position\_embeddings} = 128$, vocabulary setting $\text{type\_vocab\_size} = 1$, dropout probability for all fully connected layers in the embeddings $\text{hidden\_dropout\_prob} = 0.3$ (also in the encoder and pooler), dropout ratio of attention probabilities $\text{attention\_probs\_dropout\_prob} = 0.3$. Masked Language Model masking probability $\text{mlm\_probability} = 0.15$, number of warm-up steps $\text{warmup\_steps} = 500$, number of per-device batch size $\text{per\_device\_train\_batch\_size} = 256$ and the total number of pre-training epochs
num_train_epochs = 6.

The other parameters are left to the default values. Our higher than default dropout values are used to help combat potential overfitting in the case where the MLM training task is too easy on our sequences. We find that around 4-6 epochs are enough for the model to converge on the pre-training phase.

For the fine-tuning phase, we load the entire pre-trained RoBERTa model with trainable weights and append a dropout layer with probability 0.1 and a dense layer for regression of the same size as the encoder layers (256). The input preprocessing employs the pair encoding capabilities of the tokeniser (initially designed for sentence pair tasks) and is very similar to the RNN encoding with special characters from Figure 4b. The resulting network is trained with the MSE loss, AdamW optimiser with a learning rate of 0.00002 and weight decay set to 0.01, with early stopping set to a patience of 3 epochs and no maximum number of epochs.

6 Background on Transformers

The term Transformer usually translates to a multi-layer architecture built from composable, structurally-identical Transformer blocks, the fundamental units. The input to such a block is a d-dimensional vector, the additive composition of the input embedding vector itself with a separately-computed positional embedding vector. An important characteristic of the Transformer block is the self-attention mechanism, which computes an $N \times N$ attention matrix, with $N$ the sequence length. This procedure allows tokens to attend to other, possibly distant tokens which are deemed relevant (i.e., high attention scores). Having $k$-head attention corresponds to $k$ independent and concurrent attention computations that can be concatenated or summed. As noted in [21], Transformer blocks can be employed for different purposes. The three classic cases are encoder-only, decoder-only and encoder-decoder, where each comes with specific implementation decisions and design choices. Naturally, the above computation is quadratic, and many recent variations have been proposed to combat this lack of efficiency [21].

The original pre-training procedure includes two training objectives: Masked Language Model (MLM) and Next Sentence Prediction (NSP). MLM randomly replaces a per cent of input tokens with a special mask token and learns to predict the missing character. NSP is a classification problem where the objective is to predict if two sentences follow each other in the input text.

A possible extension is the Vision Transformer [36], a Transformer architecture designed to work on images, offering competitive performance with deep CNNs while requiring less resources. We do not investigate Vision Transformers in this work as the inference times will likely lag behind our relatively shallow CNNs, while the regression performance is also unlikely to be stellar considering our results with the sequence-based Transformer.

7 Clustering with MMseqs2

A small sample of the 39,432,713 output pairs is reproduced in Supplementary Table 5. Notice that for each ID in the first column, there are multiple sequences which are deemed similar by MMSeqs2. For example, the sequence denoted by ID seq43840 has 8 other similar sequences associated with it. This is however just a subset of all the sequences found to be similar with seq43840 and reproduced here for illustration purposes.

8 Experimental platform

The various experiments in this study are performed on a number of different platforms to accommodate their unique requirements. Hyperparameter optimisation, general non-GPU (Graphics Processing
Supplementary Table 5: Small sample of the MMseqs2 output.

| ID 1   | ID 2   | Sequence 1                  | Sequence 2                  |
|--------|--------|-----------------------------|-----------------------------|
| seq43840 | seq49789 | GCCGCAACGCGTATATTAGG         | AAGCTTAAATACACGCGGTGC       |
| seq43840 | seq81414 | GCCGCAACGCGTATATTAGG         | ATACCGCGGTGGATCGCTAGC       |
| seq43840 | seq92992 | GCCGCAACGCGTATATTAGG         | CATACGCGGCGGTCATGAA        |
| seq43840 | seq11505 | GCCGCAACGCGTATATTAGG         | GTTCTAATCTACCGGAGTC        |
| seq43840 | seq82545 | GCCGCAACGCGTATATTAGG         | GACCTTATATATCGCGTGGA       |
| seq43840 | seq19444 | GCCGCAACGCGTATATTAGG         | ATACGCCGTGGCGGCCGCA        |
| seq43840 | seq15232 | GCCGCAACGCGTATATTAGG         | ATAAATACCGCGGTACGTA        |
| seq43840 | seq25057 | GCCGCAACGCGTATATTAGG         | TTCTAATATACCGCTGACA       |
| seq43968 | seq59355 | AAATAGCCCTTACTATGTC         | CTCGCCATAAAAGGCGACCAA       |

... ... ...

Unit) intensive tasks and training of the simpler neural models (feed-forward neural networks) were performed on a portable computer equipped with a Core i9-8950HK processor, 32GB of DDR4 RAM, a PCI Express 3.0 solid-state drive (SSD) and an NVIDIA RTX 2070 external GPU connected through Thunderbolt 3, with 8GB VRAM.

RoBERTa pre-training was performed on the Google Cloud Platform with 8 TPUv3 (third generation) cores and fine-tuned on a system equipped with 8 vCPUs, 61GB RAM, an SSD and an NVIDIA Tesla V100 with 16GB VRAM. Some compute-intensive tasks such as NUPACK or MMseqs2 were either run on our portable platform or an Azure virtual machine (VM) with 64 vCPUs and a premium SSD, as indicated in the main text. Due to the different configurations, we do not report training times.

Experiments involving the measurement of time were performed on three platforms. NUPACK code, which runs only on the Central Processing Unit (CPU) was timed on an Azure VM equipped with the hardware described above. The rest of the timing experiments were performed, by design, on consumer hardware. The two chosen consumer platforms are (1) a computer equipped with an AMD Ryzen 5950X CPU, 32GB of DDR4 RAM, a PCI Express 4.0 SSD and an NVIDIA RTX 3090 GPU with 24GB of VRAM and (2) Google Colaboratory with 8 TPUv2 cores (second generation).

9 Software

All code is written in the Python programming language and has been confirmed as working under revisions 3.8.5 and 3.8.6. The required machine learning libraries include scikit-learn 0.24, PyTorch 1.7.1 and PyTorch Lightning 1.1.4, which were the latest versions available as of January 2021. The main development operating system on the portable platform and the RTX 3090 GPU platform is Windows 10 Insider Preview Build 21286 (available on the Dev Channel), while the other platforms use various recent distributions of Linux.

NUPACK, parasail and MMseqs2 computations were performed on the same portable computing platform as above, using their respective most recent versions as of May 2019. This translates to NUPACK
version 3.2.2 and parasail versions 2.4.1 and 2.4.2. Our original CNN implementation (not presented in this paper) employed the most up-to-date TensorFlow 1 and Keras versions as of May 2019.

10 Measuring inference time

The timing code for GPUs is based on the following snippet:

```python
start = torch.cuda.Event(enable_timing=True)
end = torch.cuda.Event(enable_timing=True)

with torch.no_grad():
    start.record()
    trainer.test(model, test_dataloader)
    end.record()
    torch.cuda.synchronize()
    elapsed_time = start.elapsed_time(end)
```

The code can be trivially extended to perform multiple repetitions of the same code block in order to report the average execution time and standard deviation.

Omitting the non-timing code, the timing code structure for TPUs is as follows:

```python
class Module(pl.LightningModule):
    def __init__(self, ...):
        super(Module, self).__init__()
        self.forward_flag = 0

    def forward(self, x):
        if not self.forward_flag:
            self.forward_flag = 1
            self.start = time.time()
        x = ...

    def on_test_epoch_end(self):
        end = time.time()
        elapsed_time = end - self.start
```

This introduced probe effect will alter the execution time, but we assume the impact is negligible. Importantly, the above code is run on all of the 8 available TPU cores, meaning that each core reports its own time. As the cores operate concurrently, we take the mean of the 8 elapsed times and assign the result to the corresponding trial.

It is also important to note that raw GPU performance roughly doubles with each generation, a trend also observed by Tensor Processing Units. Thus, the deep learning models we have developed will continue to improve in terms of inference time without any additional effort. A further optimisation would be the use of 16-bit precision. For a minor accuracy penalty, 16-bit implementations can both reduce video memory usage and improve execution times by several factors in the range of $\times 1 - 5$.

Furthermore, at the time of writing, the latest NVIDIA GPU architecture, Ampere, is still not fully exploited under CUDA Toolkit 11.0, the official version supported by PyTorch 1.7.1. For this reason, we compiled from source a nightly version of PyTorch 1.8.0 with CUDA 11.1 and this setup was used for all GPU timing experiments. However, it is likely that further optimisations will be possible on the Ampere architecture.
11 Detailed performance metrics on other temperatures

Supplementary Table 6 presents the MSE, MCC and AUROC for the five evaluated temperatures and Supplementary Table 7 lists per-class precision, recall and F\(_1\) scores.

Supplementary Table 6: Summary of the evaluation metrics for the deep learning models at the selected temperatures.

| Model     | Metric | 37.0C | 42.0C | 47.0C | 52.0C | 62.0C |
|-----------|--------|-------|-------|-------|-------|-------|
| RNN       | MSE    | 808.126 | 655.260 | 433.125 | 190.772 | 333.637 |
|           | MCC    | 0.862 | 0.872 | 0.888 | 0.916 | 0.833 |
|           | AUROC  | 0.946 | 0.949 | 0.954 | 0.963 | 0.911 |
| CNN       | MSE    | 719.616 | 576.134 | 374.768 | 170.865 | 398.881 |
|           | MCC    | 0.884 | 0.893 | 0.908 | 0.929 | 0.811 |
|           | AUROC  | 0.955 | 0.958 | 0.962 | 0.967 | 0.899 |
| CNNLite   | MSE    | 712.598 | 574.295 | 381.676 | 188.943 | 418.637 |
|           | MCC    | 0.893 | 0.901 | 0.914 | 0.929 | 0.801 |
|           | AUROC  | 0.958 | 0.961 | 0.964 | 0.966 | 0.892 |
| RoBERTa   | MSE    | 546.058 | 452.876 | 336.600 | 249.726 | 674.329 |
|           | MCC    | 0.925 | 0.930 | 0.930 | 0.914 | 0.762 |
|           | AUROC  | 0.971 | 0.971 | 0.968 | 0.954 | 0.870 |
Supplementary Table 7: Precision, recall and $F_1$ scores for the Low and High classes resulting from the deep learning models at the selected temperatures.

| Metric | 37.0C | 42.0C | 47.0C | 52.0C | 62.0C |
|--------|-------|-------|-------|-------|-------|
| CNNLite |       |       |       |       |       |
| Precision | 0.872 | 0.998 | 0.885 | 0.996 | 0.908 | 0.992 | 0.947 | 0.979 | 0.999 | 0.817 |
| Recall   | 0.996 | 0.921 | 0.993 | 0.929 | 0.987 | 0.942 | 0.967 | 0.965 | 0.785 | 0.999 |
| $F_1$    | 0.930 | 0.958 | 0.936 | 0.961 | 0.946 | 0.966 | 0.957 | 0.972 | 0.879 | 0.899 |
| CNN      |       |       |       |       |       |
| Precision | 0.860 | 0.999 | 0.873 | 0.997 | 0.897 | 0.995 | 0.940 | 0.984 | 0.999 | 0.826 |
| Recall   | 0.998 | 0.912 | 0.996 | 0.920 | 0.991 | 0.934 | 0.975 | 0.960 | 0.798 | 0.999 |
| $F_1$    | 0.924 | 0.953 | 0.931 | 0.957 | 0.942 | 0.963 | 0.957 | 0.971 | 0.887 | 0.905 |
| RNN      |       |       |       |       |       |
| Precision | 0.835 | 0.999 | 0.848 | 0.998 | 0.872 | 0.996 | 0.918 | 0.988 | 0.999 | 0.844 |
| Recall   | 0.998 | 0.893 | 0.997 | 0.901 | 0.993 | 0.915 | 0.982 | 0.944 | 0.822 | 1.000 |
| $F_1$    | 0.909 | 0.943 | 0.917 | 0.947 | 0.929 | 0.954 | 0.949 | 0.965 | 0.902 | 0.915 |
| RoBERTa  |       |       |       |       |       |
| Precision | 0.915 | 0.995 | 0.926 | 0.992 | 0.942 | 0.983 | 0.962 | 0.957 | 0.996 | 0.788 |
| Recall   | 0.991 | 0.950 | 0.986 | 0.956 | 0.971 | 0.965 | 0.932 | 0.976 | 0.743 | 0.997 |
| $F_1$    | 0.951 | 0.972 | 0.955 | 0.974 | 0.956 | 0.974 | 0.947 | 0.967 | 0.851 | 0.881 |
To study the effectiveness of the different extracted features, we perform an ablation study examining the classification performance of various configurations of interest. The results are presented in Supplementary Table 8, Supplementary Table 9, Supplementary Table 10, Supplementary Table 11. There are less entries for NN due to extended training time.

**Supplementary Table 8: Ablation study summary for LDA.**

| Aln. | GC | S.C. | P.C. | S. MFE | MCC | AUROC |
|------|----|------|------|--------|-----|--------|
| ✓    |    |      |      |        | 0.840 | 0.906  |
| ✓    | ✓  |      |      |        | 0.846 | 0.909  |
| ✓    | ✓  | ✓    |      |        | 0.846 | 0.909  |
| ✓    | ✓  | ✓    | ✓    |        | 0.846 | 0.909  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.850 | 0.912  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.850 | 0.912  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.850 | 0.912  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.851 | 0.912  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.000 | 0.500  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.000 | 0.500  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.000 | 0.500  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.033 | 0.502  |

**Supplementary Table 9: Ablation study summary for QDA.**

| Aln. | GC | S.C. | P.C. | S. MFE | MCC | AUROC |
|------|----|------|------|--------|-----|--------|
| ✓    |    |      |      |        | 0.845 | 0.910  |
| ✓    | ✓  |      |      |        | 0.853 | 0.916  |
| ✓    | ✓  | ✓    |      |        | 0.852 | 0.918  |
| ✓    | ✓  | ✓    | ✓    |        | 0.852 | 0.918  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.858 | 0.920  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.859 | 0.922  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.859 | 0.922  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.859 | 0.922  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.859 | 0.922  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.023 | 0.504  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.023 | 0.504  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.003 | 0.501  |
| ✓    | ✓  | ✓    | ✓    | ✓     | 0.096 | 0.548  |
The table headers refer to the concepts introduced in *Designing a diverse hybridisation dataset* (Aln., Alignment, GC, GC content, S.C., Single Concentration, P.C. Pair Concentration, S. MFE, Single MFE, MCC, Matthews Correlation Coefficient, AUROC, Area Under the Receiver Operating Characteristics).

Supplementary Table 10: Ablation study summary for RF.

| Aln. | GC  | S.C. | P.C. | S. MFE | MCC | AUROC |
|------|-----|------|------|--------|-----|-------|
| ✓    | ✓   |      |      |        |     | 0.848 |
| ✓    | ✓   | ✓    |      |        |     | 0.884 |
| ✓    | ✓   | ✓    | ✓    |        |     | 0.875 |
| ✓    | ✓   | ✓    | ✓    | ✓      |     | 0.875 |
| ✓    | ✓   | ✓    | ✓    | ✓      | ✓   | 0.884 |
| ✓    | ✓   | ✓    | ✓    | ✓      | ✓   | 0.879 |
| ✓    | ✓   | ✓    | ✓    | ✓      | ✓   | 0.880 |
| ✓    | ✓   | ✓    | ✓    | ✓      | ✓   | 0.879 |
| ✓    | ✓   | ✓    | ✓    | ✓      | ✓   | 0.141 |
| ✓    | ✓   | ✓    | ✓    | ✓      | ✓   | 0.151 |
| ✓    | ✓   | ✓    | ✓    | ✓      | ✓   | 0.190 |
| ✓    | ✓   | ✓    | ✓    | ✓      | ✓   | 0.559 |

Supplementary Table 11: Ablation study summary for NN.

| Aln. | GC  | S.C. | P.C. | S. MFE | MCC | AUROC |
|------|-----|------|------|--------|-----|-------|
| ✓    | ✓   | ✓    |      |        |     | 0.867 |
| ✓    | ✓   | ✓    | ✓    |        |     | 0.872 |
| ✓    | ✓   | ✓    | ✓    | ✓      |     | 0.873 |

It is clear that alignment scores alone are a good predictor of hybridisation yield. A modest, but visible increase in classification performance is further provided by the quick-to-compute GC content. Depending on the machine learning model, the benefit of adding thermodynamic information is more or less subtle. In Supplementary Table 8, Supplementary Table 9, Supplementary Table 10, Supplementary Table 11 the best scores are highlighted in bold: first by MCC; in case of equality also by AUROC; in case of equality of AUROC the most complete model (most features) is highlighted. For LDA, QDA and NN the best performing models use all features. For RF, the Aln-GC model has nearly identical performance to the Aln-GC-MFE model.

However, the differences are statistically significant: for LDA, between the complete model and the Aln-GC variant ($P = .034$, permutation test with 5000 iterations, thus significant at the 95% confidence interval); for QDA, between the complete model and the Aln-GC variant ($P < .001$, permutation test with 5000 iterations, thus significant at the 99% confidence interval); for RF, between the Aln-GC-MFE
model and the Aln-GC variant \( (P < .001, \) permutation test with 5000 iterations, thus significant at the 99% confidence interval) and for NN between the complete model and the Aln-GC variant \( (P < .001, \) permutation test with 5000 iterations, thus significant at the 99% confidence interval). The null hypothesis that the two groups come from the same distribution is rejected in all cases at the mentioned confidence interval. This two-sided permutation test is provided by the permutation_test function of the mlxtend [37] Python library. The statistical difference in otherwise close scores can be explained by trade-offs made by the model: some models might favour reducing false positives over false negatives and vice-versa.

### 13 Inference times for different dataset sizes

The inference time on second generation TPUs was measured using the methodology described in Supplementary Information 10, for subsets of size: 250,000, 500,000, 1,000,000, 2,000,000 and 2,556,976 (full dataset). These datasets can be entirely loaded in memory, hence a PyTorch TensorDataset (without shuffling) was used to hold the data, initially loaded with NumPy [38]. We further measured inference time on a dataset of size 5,113,952 (double the full dataset length), by concatenating two TensorDataset objects holding the 2.5 million data points using PyTorch’s ConcatDataset. The batch size was set to 8,192 as it enables better scalability on the larger datasets.

Supplementary Figure 1 illustrates the measured inference time as well as the extrapolated values based on two reference points: 500,000 and 1,000,000, called "500K projection" and respectively "1M projection", for the three most efficient algorithms: RNN, CNN and CNN\(_{\text{Lite}}\).

All evaluated models see a spike in inference time for the 1 million dataset, however this trend improves for the RNN, whose times for the > 1 million datasets stay below the 1M projected line (only \( \times 1.84 \) increase from 2.5M to 5M). The CNN and CNN\(_{\text{Lite}}\) follow the 1M projected line closely until around 2.5M in dataset size. Afterwards, the difference compared to the 1M projected line increases; however the actual measured time increases by a factor of about \( \times 2.27 \) for CNN\(_{\text{Lite}}\) and \( \times 2.30 \) for the CNN when transitioning from the 2.5M dataset to the 5M dataset. Thus, both the CNN and CNN\(_{\text{Lite}}\) models are close the the ideal \( \times 2 \) increase in inference time as the dataset size doubles; in addition, the times are better than the RNN, which scales better than linearly relative to its own performance.

Supplementary Figure 1: Scatter plots of the inference times for the three most efficient models: RNN, CNN and CNN\(_{\text{Lite}}\) as the dataset size increases, together with projections based on the times for the 500K and 1M datasets. The batch size is set to 8,192. In the graphs, M is used for million(s).
14 Similarity between yields at different temperatures

An intuition for the similarity between the ground truth yields as computed by NUPACK at different temperatures can be attained by pairwise scoring the values using a criterion such as mean absolute error (MAE) or mean squared error (MSE). For this comparison, all yields are in the range \([0, 1]\]. Supplementary Figure 2 captures these metrics and indicates that the yields at 62°C are the most different, even when compared to their closest neighbour (57°C). On the other hand, yields in the range 37°C to 52 °C are relatively close to each other (in all cases MAE lower than 0.1, MSE lower than 0.04).

The choice of 57°C for training and evaluation is motivated by being close to the standard melting and/or annealing temperatures of PCR primers (more details in the main text) and not being too far away from the behaviours at 62°C and < 57°C as illustrated in Supplementary Figure 2.

Supplementary Figure 2: The ground truth yields at the six different temperatures are compared in a pairwise manner to assess their relatedness, using the MAE and MSE as the scoring functions.