

GEMstone 2.0: Detecting Evidence of Genetic Engineering

FY18 Project Report
Feb 26, 2018



An In-Q-Tel Labs Collaboration

Table of Contents

Executive Summary.....	3
Background	4
Methods: Data Sources.....	8
Methods: Model Architecture	13
Methods: Training and Testing	17
Results.....	19
Discussion.....	25
References	30
Appendices.....	32
Appendix A: Exploratory Data Analysis and Initial Experiments.....	32
Appendix B: Model Architecture and Training Details.....	35

Executive Summary

The desire and ability to genetically engineer organisms is becoming increasingly widespread, and the barriers to using the most sophisticated means of genome editing are falling rapidly. There is a corresponding risk that actors with malicious intent may decide to use these tools to create more dangerous strains of pathogenic organisms. The sophistication of the design tools to make such organisms currently far outstrips the capabilities of the tools with which genetic engineering can be detected quickly and accurately in an automated fashion.

In consultation with B.Next’s biodefense community partners, IQT Labs B.Next and Lab41 have explored how applying machine learning (ML) approaches to DNA sequence analysis may provide “triage” tools that enable users to quickly assess the likelihood that the genome of a suspect organism has been engineered. The Labs obtained a diverse dataset from both public (US and European) and private (a synthetic biology company) sources that was used to train, validate and test several ML models to detect the insertion of DNA from one organism or source into another. The performance of the trained models varied with the complexity of the underlying dataset, but was sufficient to illustrate the promise of ML-based approaches to rapid DNA sequence analysis for biodefense applications. Our findings also suggested immediate changes to model training that would likely improve performance.

The results of this project were recently briefed to the same members of the US biodefense community who helped frame the problem. The participants were uniformly optimistic about the potential for ML-based approaches, and recommended that future work include refinements in the data sources used and establishing confidence metrics for trained models.

Highlights:

- B.Next and colleagues in biodefense identified that the lack of analytical tools for detecting genetic engineering is a significant issue to enabling an effective response to a biothreat. (see Background, p. [4](#))
- ML has been applied to DNA analysis, but not extensively. (see Background, p. [6](#))
- The ML models developed by the IQT Labs team detect cloning boundaries - junctions between inserted DNA and the genome of the “destination” organism. (see Methods: Data Sources, p. [8](#))
- The team created an *in silico* method to generate an unlimited source of synthetic cloning boundaries for use in model training and validation. (see Methods: Data Sources, p. [11](#))
- The ML models we generated demonstrated classification accuracies between 93% and 74%, which correlated inversely with the complexity of the datasets used in training, validation and testing. (see Results, p. [19](#))
- Our work complements but does not duplicate other efforts across the USG and within IQT. B.Next-hosted discussions were the origin of IARPA’s FELIX program, which will generate larger-scale tools for detecting genetic engineering. B.Next staff were members of the source selection board for FELIX proposals. (see Discussion, p. [27](#))

Background

Community input on key issues. In September 2016, B.Next convened a roundtable discussion to explore whether and how a biological sample containing microorganisms could be examined quickly using available techniques and procedures to determine whether a pathogenic microorganism had been subject to genetic manipulation, and if so, whether the intended and actual functionality of the genetic manipulation could be understood.

The roundtable (called GEMstone, for Genetically Engineered Microorganisms) was attended by 26 participants, including scientists from academia, seven US Government agencies and the Lawrence Livermore National Lab, representatives from private sector companies engaged in DNA sequencing and bioinformatics, and IQT professional staff. The group's expertise included bioinformatics, genetic engineering, computer science, microbiology, and biotechnology. The discussion took place over a single day, included invited presentations from three participants, and was held on a not-for-attribution basis. The GEMstone discussion and report (In-Q-Tel, Inc. 2017) highlighted many concerns shared widely across the biological defense community, two of which formed the basis of two challenge projects for B.Next during FY18.

- **Lack of access to privately held genomic data.** An increasing proportion of DNA sequence data is being generated and held privately. Private entities may not wish to make their data publicly accessible due to concerns about privacy, trade secrets, or liability. However, the analysis of a new pathogen's DNA sequence relies substantially on the ability to compare it to already-known sequences. B.Next, in collaboration with the other three IQT Labs, is exploring the feasibility of performing encrypted queries on proprietary DNA sequence databases. Their findings are the subject of a separate B.Next challenge report for FY18.
- **Lack of robust bioinformatics tools to analyze newly acquired genomic sequences.** While algorithms for genome sequence analysis are plentiful, few are "industrial grade"; they have flaws that result from having been purpose built by scientists for use cases that are typically limited to research. The rapid pace of accumulation of raw sequence data means that tools written only a few years ago are generally unable to handle the increasingly large amount of available data, have inefficient data structures, are unable to scale performance to take full advantage of available memory or other computational advances, and have runtimes that exponentially increase compared to input data size. In addition, many bioinformatics programs are written by individual researchers who publish their algorithms; however, the software that implements them is typically not maintained long after the research studies are published.

The poor state of our tools specifically for detecting genetic engineering is also due to the lack of a market for them. No such purpose-built tools exist in academia or in commerce because the task of making them robust and keeping them current is not relevant to academic pursuits, and detecting evidence of genetic engineering is largely irrelevant for commercial applications.

Commercial synthetic biology software focuses rather on identifying and optimizing methods to engineer new microorganisms to make useful products.

In the event of a release of a genetically engineered threat today, bioinformatics scientists, using sequentially a number of tools that each search for a feature or set of features, may be able to detect genetically engineered microorganisms given substantial time (weeks), if the microorganism was engineered with “traditional” engineering techniques (such as molecular cloning using restriction enzymes). The users of these existing tools also rely on their understanding of the “rules” of genetic engineering and the underlying biology of the organisms under study. A report from Lawrence Livermore National Laboratory details one (uncompleted) example of this approach (Allen and Slezak 2010). More advanced genome engineering tools, such as CRISPR-Cas9, leave even fewer sequence-specific features as “sequence scars” in the engineered product sequence.

Features of a useful “genetic engineering detector”. Given the diversity of features in genetically engineered organisms, and the variety of methods for making them, it is unlikely that a single tool will serve to detect all genetic modifications in any subject organism. A useful genetic engineering screening pipeline would take as input data DNA, RNA, or amino acid sequences of any length, with no additional information about the source, and return a description of the sequence, the genome of origin, and a confidence estimate that any given subsample of the sequence was added synthetically.

There are many powerful and well-studied tools in the field of bioinformatics for comparing and classifying DNA sequences. These tools are usually designed with a specific biological question in mind – how similar are these two sequences, or how likely is it that they came from the same species? Such tools could be extremely useful for investigating whether a DNA sequence has been engineered – however they require human expertise to use and interpret, which presents a problem of scalability.

Perhaps more importantly, classical bioinformatics tools focus on certain features of the input data, and often make critical assumptions about that data. For example, the Basic Local Alignment Search Tool (BLAST; Altschul et al. 1990) compares an input sequence with a database of sequences. If the input sequence is not in the database, BLAST will not return a result. This is desirable behavior for a DNA alignment search tool, but it highlights a fundamental brittleness in traditional ‘rules-based’ sequence analysis tools. If a tool depends on existing knowledge, or requires a high degree of certainty about the distribution of input data, it is unlikely to generalize to new examples. This is particularly true in an adversarial setting in which the designer of an engineered pathogen may wish to hide the fact of engineering and avoid attribution. For these reasons, we decided to investigate the utility of machine learning models to identify sequences that have been genetically engineered.

Machine learning (ML): a brief introduction. Machine learning is an approach to building predictive models for some phenomenon given a relevant dataset. Classically, in bioinformatics (and science generally), a phenomenon is closely studied and an attempt is made to describe the underlying principal at work. This hypothesis is then tested and refined through iterative experimentation.

The process of applying machine learning to a problem is fundamentally different. Modern machine learning techniques do not attempt to deduce the implicit structure of whatever is being investigated. Instead, a model is exposed to large quantities of training data and driven to shape its own parameters to best predict some output of interest. This trained model can then be used to make predictions about new examples. A set of predictive models built in this way has the capacity to recognize subtle patterns, which might elude experienced human observers. Indeed, these methods have surpassed human performance in many fields.

Machine learning can supplement the traditional research tasks of careful observation and experimentation with the caveat that collecting enough representative data to train and validate a model that represents the phenomenon being investigated remains a challenge. Many scientists working in quantitative fields apply both in parallel. The process of model selection, training, and refinement, is itself a form of iterative experimentation.

Neural networks are a particular kind of machine learning model consisting of an interconnected network of nodes (neurons), typically arranged in a series of layers which perform alternating linear and non-linear transformations on a vector of numerical inputs. Each node is connected to subsequent nodes by a set of weight values. These weights can be shaped through training in order to produce some desired output (a label for example) for a given input. Neural networks are capable of representing complex highly non-linear categories of knowledge (e.g. the difference between pictures of a cat or a dog) which is why we chose to use them for this work.

Machine learning models for DNA sequence analysis. Machine learning is a potentially powerful approach to recognizing patterns in DNA sequences that indicate past genetic engineering, and may facilitate the automated triaging of unknown biological samples. Machine learning has been applied to DNA sequence data, most significantly in the prediction and identification of genetic control elements such as promoter, enhancer and terminator sequences (Libbrecht and Noble 2015). In these studies, however, workers trained models with past examples of genetic control elements that had been identified and defined by traditional “wet-lab” experimental means. Others have noted the potential for ML to accelerate the synthetic biology design-build-test cycle (Adler and Yaman 2016).

To the best of our knowledge, the direct application of ML techniques to the challenge of identifying genetically engineered sequences has only been minimally explored (C. Voigt, personal communication; Kunjapur et al. 2017). In this report, we describe a supervised method for training ML models to discern the presence of “cloning boundaries”, which we define as transitions between the inserted DNA sequence and the destination DNA into which it was cloned. We train and evaluate models using naturally-occurring (non-engineered) and

engineered DNA sequences, obtained from both public and private data repositories. We chose to train models on DNA sequences independent of defined sequence patterns; rather, we hypothesized that it would be possible to train a model to detect the insertion of DNA from one organism into another’s genome, without prior definition of “rules” (i.e., specific DNA sequences or sequence motifs), by focusing on the presence of the “cloning boundary” feature. An appendix contains a description of additional data exploration and unsupervised machine learning approaches that were performed during earlier stages of the project.

Methods: Data Sources

Introduction. Broadly applied, the term “genetic engineering” refers to any direct modification of DNA sequences by any means. Frequently, these modifications include the insertion of one or more genes (which we will term “inserts” in this paper) into another piece of DNA (which we term “backbones”). Examples include the insertion of human insulin gene into the genome of the bacterium *Escherichia coli* (Johnson 1983), and the insertion of a bacterial gene for resistance to the herbicide glyphosate (found in Roundup™, made by Monsanto®) into crop plants to assist in weed control (Funke et al. 2006). Our goal was to train ML models to recognize juxtapositions between backbone and insert sequences, or cloning boundaries.

Our ML model is trained entirely on insert-free backbone sequences, for the purpose of classifying whether a query sequence contains a cloning boundary or not. We deliberately chose not to train our model on a specific kind of insert in order to avoid building a classifier that would fail to generalize to other possible inserts. Instead, we chose to build a generalized cloning boundary detector – when trained on non-engineered sequences of a certain kind, the desired model would be able to detect insertions in those sequences.

To validate the performance of the models during training, we created a set of virtual synthetic sequences (VSS; both with and without cloning boundaries) using backbone and insert sequences taken from several sources. These VSS, generated *in-silico* following a simple set of rules, allowed us to make and monitor improvements to our model without exposing our limited collection of real world experimentally modified test sequences. This sort of compartmentalization is a commonly accepted practice in machine learning (Figure 1). We describe these data sources in detail below. The model training, validation, and testing schemes are described in the following sections.

Backbone Data Sources. The backbones used here include plant and bacterial genome sequences, as well as plasmids (independent “microgenomes” that accompany many bacterial genomes both in nature and in the laboratory). Plasmids are included as backbones because genetic engineers often insert DNA that encodes a useful function into plasmids as part of the process of making genetically engineered organisms, and in some cases, the engineered plasmids are themselves the end state of the genetic engineering activity.

Table 1. Sources for backbone DNA sequence data used for model training.

Backbone Data Set	Description	Samples	Bases (Millions)
UniVec	NCBI collection of plasmid vector sequences	4,386	1.6
GenBank Plasmids	Additional plasmid sequences found on GenBank not in UniVec	564	0.2
RefSeq Bacteria	NCBI reference sequences for bacteria, only a subset was used for training	70,000	840
RefSeq Plants	NCBI reference sequences for plants, only a subset was used for training	50,000	1,105

Backbone sequence data (Table 1) were obtained from several databases of the GenBank repository of sequence data, maintained by the National Center for Biotechnology Information (NCBI), National Institutes of Health (NIH). Non-engineered bacterial and plant genomic DNA sequences were sampled from the RefSeq database of curated reference DNA sequences. Plasmid DNA sequences were sampled from two GenBank databases: 1) UniVec, a reference collection of plasmids that have been used in genetic engineering experiments reported in the scientific literature, and 2) GenBank Plasmids, a dataset that does not overlap with UniVec and contains both engineered and naturally occurring plasmid DNA sequences.

From the backbone datasets, 60 percent of the data was allotted for model training, 20 percent was allotted for the construction of VSS for model validation during training, and 20 percent was reserved for evaluation of the performance of the trained models, as illustrated in Figure 1.

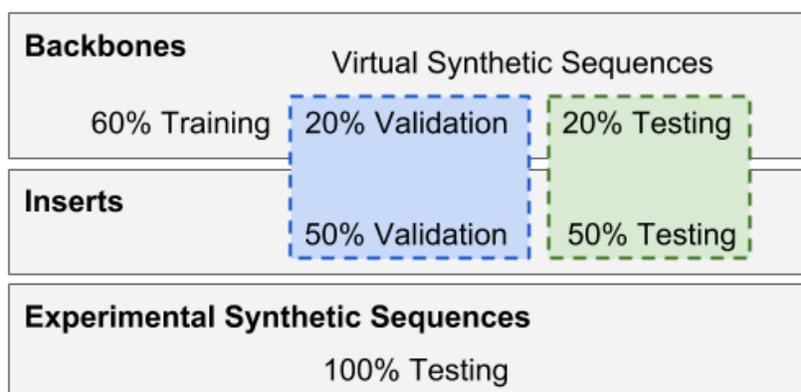


Figure 1. Data Compartmentalization Scheme. Three categories of data sources: backbones, inserts, and experimental synthetic sequences, used in three phases of ML model development: training, validation, and testing. For validation and testing, virtual synthetic sequences were created by combining backbones and inserts.

Insert Data Sources. Sample insert DNA sequences (Table 2) were obtained from three sources. BacMet (Pal et al. 2014) is a curated database of confirmed and predicted genes used by bacteria to resist being killed by antibacterial agents (such as antibiotics and heavy metals). Herbicide resistance genes were obtained from a repository maintained by the International Survey of

Herbicide Resistant Weeds (Heap 2018). Lastly, a collection of various DNA sequences was sampled from the genes of eukaryotes (higher organisms including mouse, human, worm, and fruit fly) found in the NCBI RefSeq database mentioned above.

These eukaryotic genes were included in the synthetic insert data collection in order to increase the heterogeneity of insert DNA, and potentially improve the generalizability of the final classifier. The set of model organisms: human (Lamesch et al. 2007), worm (Martin et al. 2015), mouse (Blake et al. 2017), and fruit fly (Gramates et al. 2017), were chosen because they had well-characterized whole gene sequences. Genes from these organisms were filtered to only include those with lengths below 3000 bases, and then a subset were chosen randomly -- 500 genes each from mouse and human, 250 genes each from fly and worm.

All classes of inserts were used for the construction of VSS along with backbones set aside for this purpose (see above). From the insert datasets, 50 percent was allotted for model validation during training, and 50 percent was reserved for testing the performance of the trained models.

Table 2. Sources for insert DNA sequences used for model training.

Insert Data Set	Description	Samples	Bases (Millions)
BacMet	A repository of predicted and experimentally verified antibiotic and metal resistance genes	155,512	18
Herbicide Resistance	A small collection of known herbicide resistance genes	257	0.076
RefSeq ORFs	Randomly selected open reading frames (genes) from model organisms: nematode (worm), mouse, fly, and human	1,500	2.9

Experimental Synthetic Sequences. After models were trained on sequence data prepared in the manner described above, all models were evaluated for their ability to correctly identify the presence of cloning boundaries. Experimentally modified sequences (Table 3) were obtained from both public and private sources.

A synthetic biology company provided to IQT Labs a dataset of over 7200 synthetic DNA sequences containing both backbone (plasmid) and insert gene sequences. Addgene (Herscovitch et al. 2012), a non-profit organization, offers a repository for plasmid DNA to which synthetic biologists can submit both publicly available sequence data and the corresponding physical DNA samples. Addgene stores, curates, and fulfills orders from requesting labs, saving effort on the part of depositors. The Modified Bacteria and Modified Plants datasets were hand-selected from the ENSEMBL sequence database maintained by the European Molecular Biology Laboratory (EMBL), and are genomic sequences of plants and bacteria that contain actual insertions of non-native genes (and therefore, by definition, contain cloning boundaries).

Table 3. Sources of additional DNA sequence data used to test trained ML models.

Test Data Set	Description	Samples	Bases (Millions)
A Synthetic Biology Company	Commercial DNA foundry; custom plasmids modified with unknown inserts	7,242	54
Addgene	Popular collection of experimental results – mostly common commercial plasmids with well characterized insertions	40,596	221
Modified Bacteria	Hand collected and curated set of literature reported modified bacterial sequences	143	0.6
Modified Plants	Hand collected and curated set of literature reported modified plant sequences	106	0.3

Virtual Synthetic Sequence (VSS) Construction: For model validation during training, and final model testing (these processes are described in detail below) we created a set of VSS to supplement the limited collection of experimentally modified sequences.

We developed a pipeline that creates sequences containing one or two cloning boundaries within a length of 500 bases (each base is a “letter” in the DNA code: A, G ,C or T). We selected the length of 500 bases to coincide with the longest sequence reads available from current next-generation sequencing systems, such as those made by industry leader Illumina.

To generate virtual synthetic cloning boundaries between inserts and bacterial or plant genomic backbone sequences, our pipeline randomly selected a bacterial or plant backbone sequence from RefSeq Bacteria or RefSeq Plants (Table 1). A random 500-base subsequence was extracted from the backbone sample. The pipeline then selected a random insert sample from Table 2 and performed one of the following operations.

- Select the right-most 200-400 bases (size between 200 and 400 bases selected at random) of the insert and attach them to the right side of the backbone sequence, then crop the backbone sequence to give a total combined length of 500 bases. These sequences (Figure 2, example A) contain a synthetic cloning boundary that simulates the “left most edge” of a genetically engineered insertion.
- Same as above, but attach insert DNA to the left side of backbone DNA. These sequences (Figure 2, example B) each contain a synthetic cloning boundary that simulates the “right most edge” of a genetically engineered insertion.
- Select a random subsequence of 200-400 bases from an insert sample and place within the backbone sequence, then crop the backbone sequence on both sides to a total sequence length of 500 bases (Figure 2, example C). These sequences contain two synthetic cloning boundaries.
- Select only a 500 base subsequence from within the insert sample (Figure 2, example D). These sequences represent DNA internal to an insert, at an indeterminate distance from

either (presumed) synthetic cloning boundary. These sequences do not contain a cloning boundary, although they are of different origin from the backbone sequence.

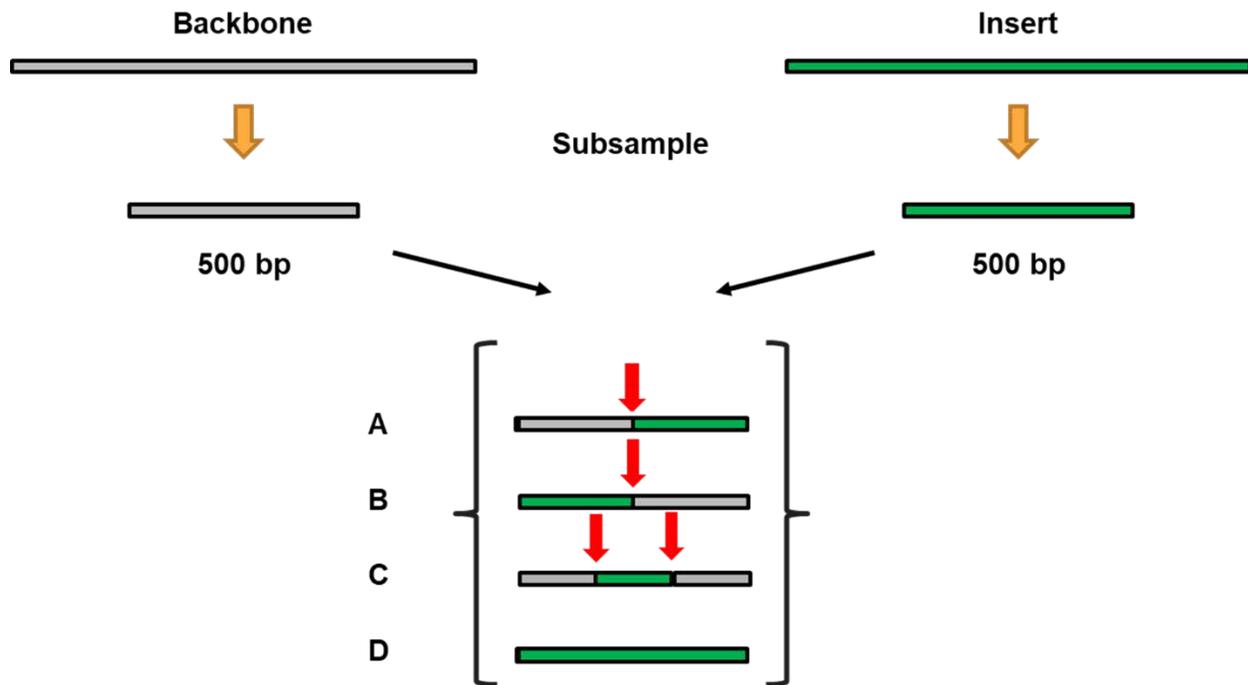


Figure 2. Virtual synthetic sequence construction: Scheme for creating synthetic cloning boundaries *in silico*. Backbone (grey) and insert (green) sample sequences were selected at random from their respective datasets and trimmed to 500 bases in length, then processed as described in the text. Red arrows indicate synthetic cloning boundaries. The positions of cloning boundaries illustrated are representative; as described in the text, they may occur at many locations in any given version of A, B, or C-type sequences.

To generate virtual synthetic cloning boundaries between inserts and backbones sampled from UniVec or NCBI Plasmids (Table 1), we made insertions based on cloning practices that rely on restriction enzymes. Restriction enzymes cut DNA at specific short (typically six base-long) sequences; two pieces of DNA that are cut with the same enzyme will have complementary “sticky ends” that allow them to be attached with another enzyme called ligase. A list of available restriction enzyme cut sites is available for most plasmids. If cut sites were available, an insert DNA sequence was sampled randomly from the datasets in Table 2, then “cut” virtually and “ligated” into the available restriction sequence in a backbone plasmid.

Methods: Model Architecture

The cloning boundary detector that we have developed is a pipeline consisting of a series of steps, described below. The pipeline takes as input DNA sequences that are 500 bases long. As discussed above, this length was chosen as representative of read-lengths commonly available from current sequencer technology.

K-merization: An input sequence is converted into an ordered set of *k*-mers. This process, known as k-merization, is a common featurization approach in bioinformatics. It entails breaking a sequence into a set of subsequences, each of length *k* bases. A stride-length must also be set, which dictates how far along to slide the k-window before collecting each subsequence (Figure 3). Our pipeline sets $k = 8$ and the stride length = 8, meaning that we convert an input sequence into a set of consecutive non-overlapping 8-mers. Appendix A contains a discussion of how these parameters were chosen.

```

k = 8, stride = 4

input sequence:
TTACAGTACGTAATGCGGAA

TTACAGTA
  AGTACGTA
    CGTAATGC
      ATGCGGAA

k-mers = {TTACAGTA, AGTACGTA, CGTAATGC, ATGCGGAA}

```

Figure 3. Sample K-merization. During bioinformatics analysis, DNA sequences are frequently broken into shorter lengths, or *k*-mers. In this illustration, the sequence is divided into sequences of length 8 ($k=8$), and a window or stride of length 4 is used to select the next 8-mer. In our analysis, we used 8-mers and a stride length of 8, meaning that the *k*-mers are end-to-end and do not overlap.

Vector Embedding: The first subunit of our pipeline is a shallow neural network that takes a single 8-mer as input and returns 40 numbers. These 40 numbers can be thought of as a vector which is associated with the given 8-mer. This process of vectorizing arbitrary inputs is known as embedding, and is a critical component of many machine-learning models (Ng 2017). Importantly, our embedding is *contextual* – it is trained in such a way that two input 8-mers which tend to appear near each other in sequences in the training dataset will map to vectors which are nearby (in a Euclidean sense) in our 40-dimensional embedding space. Dimensions in this space are akin to the components derived from a principal or independent component analysis (PCA or ICA).

Embedding is important because it exposes underlying structure in the data that may not be apparent at lower levels of abstraction. Two 8-mers, viewed either as strings of 8 bases, or as indices into a list of all 65,536 possible unique 8-mers have no meaningful relationship to each other. The literal bases may be similar, or the indices numerically close, even if the 8-mers almost

never appear in similar contexts in real-world sequences. Conversely, two 8-mers that appear to be very different in their component bases may frequently appear in proximity to the same neighboring 8-mers in the collection of input sequences. Our contextual embedding would represent the former pair of 8-mers as very different vectors, and the latter as very similar vectors. This makes it easier for a statistical learning model to detect and capitalize on patterns in the data. Potentially a single model could learn both the associations between 8-mers and the relevant patterns of cloning boundaries for a given category of backbone, but we chose to dissociate these two tasks in order to improve performance on each.

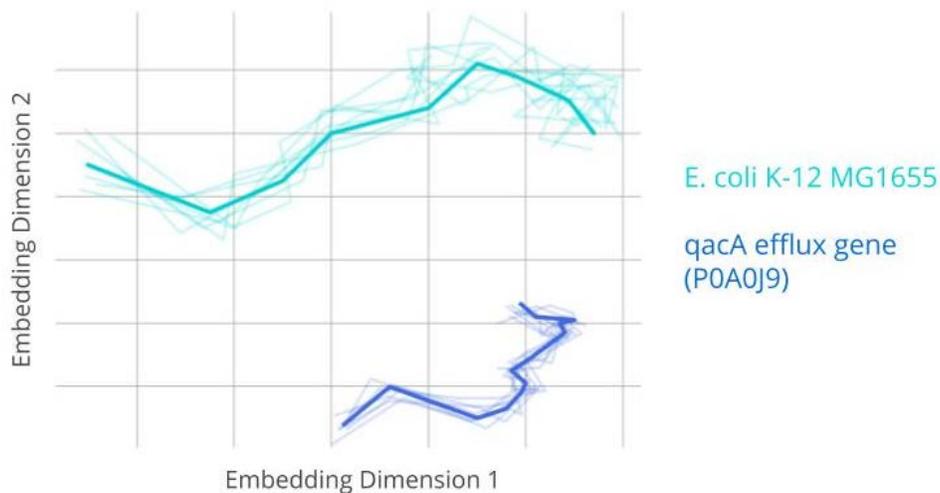


Figure 4. Sample Contextual Vector Embedding: Ten consecutive 8-mers from an *E. coli* genome (teal) and a quaternary ammonium compound resistance gene (*qacA*) from *Staphylococcus aureus* (blue) embedded in two dimensions. Individual embeddings from each sequence have been connected. Nearby lighter vectors are the same *k*-mers with random single base modifications, shown to highlight the contextual nature of the embedding.

Once the vector-embedding network has been trained, it takes a sequence of 8-mers as input and returns a corresponding sequence of 40-dimensional vectors.

Convolutional Sequence Learner: The second subunit of our pipeline is a convolutional neural network that learns to predict an element in a sequence given ten elements before and ten elements after the position to be predicted. This approach is based off work from the field of machine translation and can be thought of as similar to filling in a blank in the middle of a sentence. In our case, the words that make up the sentence are 40-dimensional vectors produced by our contextual embedding network, and the model produces an estimated 40-dimensional vector that fits best in the middle of the sequence, according to the data on which the model was trained.

Our current implementation of the convolutional subunit uses what is called ‘valid’ padding: no attempt is made to pad the boundaries in order to fully capture every element within the sliding convolution. Specifically, because the network requires ten sequence elements preceding and following its predicted location, the first element that the model can predict is the eleventh in

the sequence, and the last is the 11th from the end (the 483rd position, in our case: 493 total 8-mers). For more details on the model implementation, see Appendix B. The output of the convolutional sequence learner is a predicted vector at each location within the sequence.

Classification of Cloning Boundaries: The final step in our pipeline is to compare the sequence of predicted vectors with the actual embedding vectors present in the input sequence in order to estimate the likelihood that a cloning boundary is present. Note that this final classification step does not require any training. We use a simple distance metric – the L2 distance (more commonly known as the Euclidian norm in two dimensions: the square root of the sum of the squared differences between vector components) – to calculate the similarity between the actual vectors and the predicted vectors. A large distance between the predicted and actual vectors indicates that the k-mer in the input sequence is different from what the sequence learner expected to see at that location, given the k-mers that come before and after. In principal, there are many ways that such a discrepancy could be used to assess the likelihood of a cloning boundary. We used a simple rolling average threshold: if the average distance between the most recent five predicted and actual vectors ever exceeded a threshold (0.25 for example), we labelled the sequence as containing a cloning boundary. The threshold was chosen using a receiver operator characteristic curve (see the results section below) for each distinct pipeline. For a more detailed discussion of classification thresholds, see Appendix B.

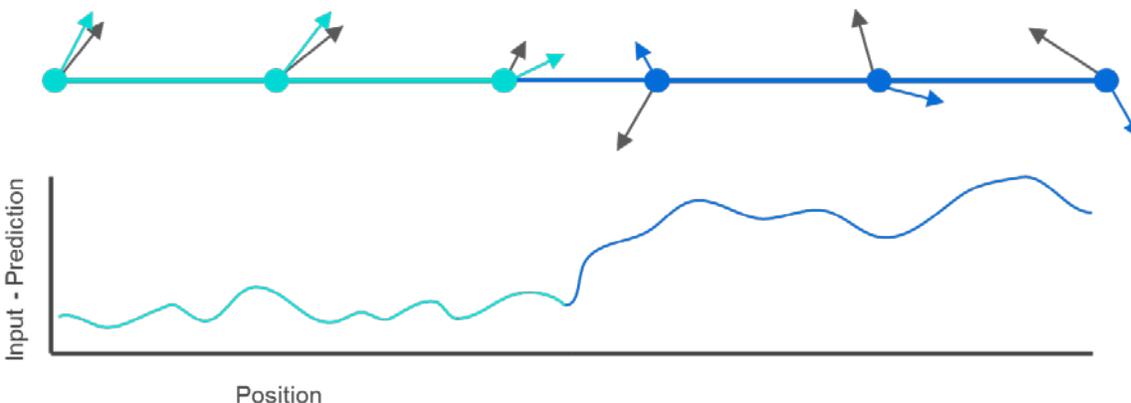


Figure 5. Cloning Boundary Classifier. Upper panel: A hypothetical representation of the comparison between predicted vectors (in black) and actual embedding vectors (in color). Note that before the cloning boundary, the predicted and actual vectors are similar (teal), whereas after the cloning boundary the predictions are not as good (blue). Lower panel: A rolling average of the distance between input and predicted vectors. Before the cloning boundary, the predictions are good, and the distance metric remains low. After the cloning boundary, the convolutional sequence learner is unable to accurately predict the vectors that are present in the input sequence and the distance metric increases. This increase is indicative of a deviation from the sequences expected by the pipeline, i.e. a likely modification.

In summary, our pipeline consisted of the following steps:

1. Take a 500 base sequence as input, which may or may not contain a cloning boundary.
2. K-merize the sequence with $k = 8$ and stride = 8.
3. Embed the sequence of k-mers as 40-dimensional vectors using an embedding network trained on backbone sequences.
4. Use a convolutional sequence learner, trained on backbone sequences, to assign a predicted vector to positions 11 – 483 of the input sequence (offset due to the valid padding).
5. Calculate the L2 distance between each predicted vector and the actual embedding vector present in the input.
6. Compute a rolling average of the L2 distances computed in step 5, with a rolling window of 5 positions. If this rolling average is ever greater than 0.25, label the input sequence as *likely to contain a cloning boundary*. Otherwise, label the sequence as *unlikely to contain a cloning boundary*.

Methods: Training and Testing

The two major subunits of our classification pipeline – the contextual vector embedding and the convolutional sequence learner – both require training. As discussed above, we elected to train these components of our model only on backbone sequences with the goal of creating a tool capable of identifying differences from those backbones containing DNA inserted from another source organism.

We randomly selected 60% of the backbone sequences from each backbone category (plasmids, bacteria, and plants) and trained three separate pipelines on these sequences. The decision to create three separate pipelines was based on the expected capacity of the constituent models to represent genetic sequences from different organisms: would a model that learned how to predict plant sequences also be able to make reasonable predictions about plasmid sequences? This question is addressed in the Discussion section below.

We used an additional 20% of the backbone sequences as a validation set during training. First as a traditional validation set, we simply observed the model performance on these held-out sequences and stopped the training process once the model no longer improved on the validation set. This is important because, in general, a machine-learning model will continue to improve until its performance on training data approaches 100 percent; however, its performance on evaluation data will cease to improve or decrease, a phenomenon known as overfitting.

The validation backbone sequences were also combined with insert sequences (50% of the total sequences from category of insert were selected randomly) to generate VSS as described above. These VSS were then used to validate the pipeline as a whole. Specifically, as the convolutional sequence learner was being trained, after each epoch (an epoch is one pass over the entire training data set), a set of VSS was pushed through the current model and the rolling average L2 distance between input and predicted vectors was computed for each sequence. A range of threshold values were used to compute classifier accuracies for the current pipeline, and training was continued as long as those classifier accuracies continued to improve. The validation VSS were also used to generate the receiver operator characteristic curves that were used to select the optimal classifier threshold value for each pipeline.

We chose to optimize the classifier for overall accuracy, providing the user with high sensitivity and specificity. In practice, a screening tool for detecting genetically engineered sequences will need to prioritize sensitivity over specificity: i.e. the cost of false negatives (missing engineered sequences) is greater than the cost of false positives (flagging non-engineered sequences for further investigation). To prioritize sensitivity while not ignoring the false-positive rate, we evaluate our performance using the F2 score. Our approach is aligned with IARPA's FELIX program, which has a sensitivity goal of at least 90 percent because of the operational risk of false-negative results.

The remaining 20% of the backbone sequences and 50% of the inserts were used to construct VSS to be used as test sequences to evaluate the performance of the final model. In addition to these test VSS, we had several sources of experimentally modified sequences: a synthetic biology company synthetic DNA constructs, Addgene plasmids, and modified bacterial and plant sequences collected from the literature (as previously described in Table 3).

When developing a machine-learning model, the goal is to create a tool that will generalize well to new examples. To this end, we elected not only to exclude the experimental synthetic sequences from the training set – the examples to which the model parameters are adjusted – but also from the validation set. This is important because even when a model is not trained on a particular data subset, if the experimenter makes adjustments to model architecture or hyperparameters in pursuit of improved validation performance (the entire point of a validation set), it is still possible to over fit held-out data. A true holdout set, a test set that is exposed to your model only once, is the best way to avoid this problem and develop robust and generalizable models.

Additional sequence randomization. An additional layer of data augmentation was used while training the vector embedding network. Each base within a 500-bp sequence was given a 1 in 100 chance of being changed to one of the other three bases, with an equal likelihood of being flipped to each of the three. This was intended to prevent the model from simply memorizing the input sequences and learning embeddings only for the specific 8-mers within the training set. When presented with a k-mer that was underrepresented during training (or not present at all), the model will rely on its random initialization to perform the embedding. Injecting noise into the training data increases the variation in k-mer contexts to which the training model is exposed, and helps it develop a more generalizable embedding.

Within the field of machine learning, a model that has learned to simply mirror its training is said to have high variance, and high variance models do not generalize well. Challenging a model by adding random noise (very much like what might arise due to DNA sequencer errors) forces it to develop more robust representations of cloning boundaries, thus improving its generalizability.

Results

Following the training procedure described above, we produced three models, each trained on a set of backbones from one of three data sources: UniVec + GenBank Plasmids, RefSeq Bacteria, or RefSeq Plants (Table 1). Once trained, each model was tested using sequences extracted from the datasets in Table 3. In each test, subsequences were extracted from samples in Table 3 that contained either a known cloning boundary or not (subsequences entirely within either backbone or insert regions). We also tested the models using VSS, constructed by the method described above, using data withheld from the training and validation process (Figure 1). For our metrics (Table 4), we chose to report accuracy in order to give a sense of a pipeline’s overall strength, and the F2 score as an indication of performance against false negatives, as we deemed it more consequential to allow an engineered sequence to go undetected by our proposed triage tool as discussed above.

Table 4. Performance of ML models trained to detect cloning boundaries in three sets of backbone sequences.

Training Dataset	Testing Dataset	Accuracy	F2
UniVec +GenBank Plasmids	UniVec + GenBank Plasmids VSS ^a	0.92	0.922
	Addgene	0.93	0.926
	A Synthetic Biology Company	0.87	0.889
RefSeq Bacteria	RefSeq Bacteria VSS	0.87	0.856
	Modified Bacteria	0.79	0.792
RefSeq Plants	RefSeq Plants VSS	0.83	0.846
	Modified Plants	0.74	0.740

^aVSS, virtual synthetic sequences.

Pipeline trained on plasmid backbones: Plasmids have sequences that range from thousands to a few hundred thousand bases, in contrast to microbial genomes, which typically consist of over a million bases. Plasmids used in genetic engineering are typically 1,000 to 20,000 bases in length. There is a group of plasmids that have been used widely in molecular biology for a few decades. There is significant sequence overlap among commonly used commercial plasmids, so it seemed likely that a model trained on known plasmid data would recognize boundaries between plasmid sequences and insert sequences regardless of origin.

To train a detector for cloning boundaries within plasmid DNA backbones, the vector embedding and convolutional sequence learning models were trained on UniVec and GenBank plasmid backbones. The model was validated during training with VSS constructed from plasmid backbones, and inserts from all categories (Table 2). These VSS were also used to determine the optimal classifier threshold for the rolling average distance metric, which was 0.25. The model was tested on VSS constructed from reserved plasmid backbones with inserts from all categories,

and on sequences obtained from a synthetic biology company and Addgene. This model achieved 92% accuracy on VSS generated from held-out plasmid backbones and inserts from all categories, 93% accuracy on sequences from the Addgene dataset, and 87% accuracy on the synthetic biology company sequences. The corresponding F2 scores were 0.922, 0.926, and 0.889 respectively.

The model's lower performance on the synthetic biology company sequences relative to performance on other plasmid data may be due to any of several factors. False negative results may have occurred because a synthetic biology company designs their DNA for synthesis in a sophisticated in-house foundry, both insert and backbone sequences alike. Therefore, the backbone portions of the synthetic biology company sequences contain modifications that are shared with insert sequences (that facilitate their chemical synthesis in the foundry). Any modifications to insert and backbone sequences that make them more similar may obscure the differences between backbone and insert sequences that are observed by the model.

The fact that the F2 score on the synthetic biology company sequences was slightly higher than the accuracy suggests that a small majority of misclassifications were false positives (examining the confusion matrix confirms that this was the case). The model somewhat more frequently labeled sequences without explicitly defined cloning boundaries as likely to be engineered. A synthetic biology company uses a more limited set of plasmids than those used in the training data, and which contain engineered features that may have been misidentified by the model as containing cloning boundaries when none (as defined by us) were present. A reexamination of the synthetic biology company dataset may reveal the need to reclassify certain portions of the evaluation data to ensure that sequences were correctly categorized and the ML evaluation correctly scored.

Pipeline trained on bacterial backbones: The bacterial cloning boundary detector was trained on bacterial backbone sequences taken from RefSeq, validated on VSS generated by combining held-out RefSeq bacterial sequences and inserts from all categories (Table 2), and tested on additional held-out VSS, and on a small collection of modified bacterial sequences collected from the literature. The validation VSS were also used to determine the optimal classifier threshold for the distance metric: 0.29.

This model achieved 87% accuracy on the testing VSS and 79% accuracy on the literature-reported modified bacterial sequences. The corresponding F2 scores were 0.856 and 0.792 respectively. The lower performance of the bacterial pipeline relative to the plasmid pipeline was anticipated. The sequence space from which bacterial genomic sequence training data were drawn was much greater than the sequence set available for plasmid DNA (Table 1). Simply put, there is more bacterial genomic sequence data available than plasmid sequence data, and the variability in the bacterial genomic sequence data is greater, reflecting the vast diversity of bacterial life on earth. A model trained on a more highly varied dataset will likely have a more difficult time discerning "backbone" from "insert" DNA sequences, as the backbone training set may in fact have contained sequences that resemble (to the model) some features of insert

sequences. In light of this observation, it is somewhat remarkable that this model performed as well as it did.

Indeed, most of the antibiotic and metal resistance genes (as well as many of the herbicide resistance genes) are bacterial in origin. Although our pipeline attempts to learn explicit sequential information, it seems likely that sequences with similar evolutionary origins would simply tend to be closer together in the embedding space, even if they were not sequential neighbors. In traditional bioinformatics, genome sequence similarity is commonly used as a method of species differentiation, and close evolutionary relationships could potentially account for decreased distance metrics in our classifier. However, our F2 score results do not suggest a strong preference for false negatives, and this question bears further empirical examination.

The bacterial pipeline showed a larger drop in performance between the VSS and the literature-reported modified bacterial sequences than the drop between the plasmid pipeline and the bacterial pipeline. This seems intuitive, given that the VSS, while assembled from held-out backbones and inserts, are still very similar in construction to the sequences on which the pipeline was trained. The modified bacterial sequences (Table 3) are derived from actual experimentation, and represent an intentional, non-uniform sampling (by bacterial geneticists) of the bacterial backbone sequence space. This fact, coupled with the size of the dataset, make it difficult to draw strong conclusions about model strength and generalizability from this result.

Pipeline trained on plant backbones: The plant cloning boundary detector was trained on plant backbone sequences taken from RefSeq, and validated on VSS generated by combining held-out RefSeq plant sequences and inserts from only the herbicide resistance insert category (Table 2). This model was tested on additional held-out VSS, and on a small collection of modified plant sequences collected from the literature. The validation VSS were also used to determine the optimal classifier threshold for the distance metric: 0.44.

This pipeline achieved 85% accuracy on the testing VSS and 74% accuracy on the literature reported modified plant sequences. The corresponding F2 scores were 0.866 and 0.740 respectively.

Note that optimal classifier performance for the plant pipeline was obtained at a higher classifier threshold than for the plasmid and bacterial pipelines (0.44 vs 0.25 and 0.29, respectively). This indicates that the distance between the predicted vectors and the actual embedding vectors tends to be higher, which suggests that the predictions made by the convolutional sequence learner are less accurate.

Plant genomes are generally an order of magnitude larger than bacterial genomes (billions of bases vs microbial genomes with millions of bases), reflecting their much greater complexity. Perhaps more importantly, plant DNA is considerably more variable than microbial DNA with regard to k-mer context space – both within a given genome and between two different plant genomes. This may just be primarily a function of length, as a k-mer will tend to appear near a wider variety of neighbors as a sequence grows. Again, as above, an increase in the complexity

of the data in the training set seemed to correlate with a drop in the performance of the trained model, although again, not as great a drop as we might have imagined.

Overall Performance: The correlation of decreasing performance with increasing training set complexity may be a consequence of the choice of vector embedding as part of the model-training pipeline. One potential problem known to arise with vector embedding methods is that the vector space can become ‘overcrowded’. In essence, the embedding model is being asked to carve out a region of 40-dimensional space for each small neighborhood of k-mers. If k-mers always appeared near the same neighbors, i.e. the distribution of k-mer contexts was very small, then the corresponding embedding vectors could be very tightly packed: k-mers will be very close to their neighbors and very far from non-neighbors. However, if there is significant variability in k-mer context - i.e., k-mers tend to appear near a large number of different neighbors (as is the case in a large and varied dataset like the plant backbone sequences) - the corresponding embeddings will be more diffuse. The embedding vectors will be spread out more in the space since they need to be ‘nearby’ a larger number of neighbors, which themselves have a large collection of associated neighboring k-mers. In this latter case, the embedding space can be overwhelmed and forced to embed k-mers near vectors that are not contextually similar. This would wreak havoc on an attempt to make sequential predictions in such an embedding space, but there is no evidence that it occurred in this case. In fact, the relatively small decrease in performance seems to suggest that it did not. However, this problem is important to keep in mind as our model is scaled up and applied to larger datasets.

The best performance of each model is represented as a Receiver Operating Characteristic (ROC) curve in Figure 6. Each curve reflects the relative level of performance as shown in Table 4.

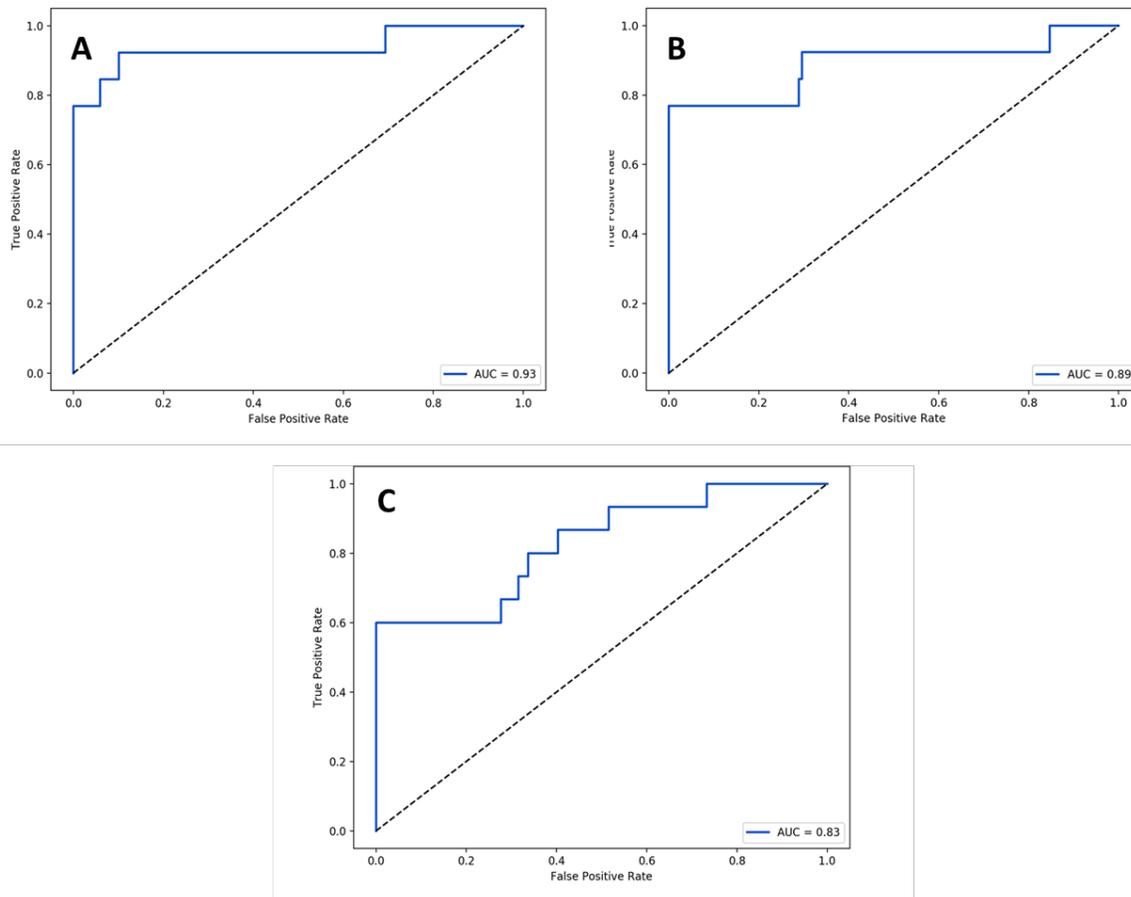


Figure 6. Receiver Operator Characteristic (ROC) curves for each ML model trained to detect cloning boundaries using backbone data, validated using VSS data and evaluated with VSS data constructed from sequences withheld from validation data. All models were trained on k -merized sequence data with $k=8$ and stride length of 8. Training, validation, and evaluation data were 500 bp sequences as described in the text. (A) ML model trained on UniVec and NCBI Plasmids sequences and evaluated on withheld sequences drawn from UniVec, NCBI Plasmids and insert DNA samples; AUC=0.93. (B) Model trained on RefSeq Bacteria sequences and evaluated on withheld sequences drawn from RefSeq Bacteria and insert DNA samples; AUC=0.87. (C) Model trained on RefSeq Plant sequences and evaluated on withheld sequences drawn from RefSeq Plant and Herbicides Resistance DNA samples; AUC=0.83. See Tables 1-3 for more details.

When we examine the performance of our pipelines on the experimentally modified sequences (Table 4 and Figure 7), we see decreased overall accuracy with an approximate maintenance of symmetry (i.e. the models maintain similar balances between sensitivity and specificity.) The plant pipeline illustrates this clearly: the AUC drops from 0.83 (Figure 6, panel C) to 0.76 (Figure 7, panel D) between the plant-derived VSS test set and the modified plant sequences sourced from the published literature.

The shape of the plant-derived ROC curve (Figure 7, panel D) was unexpected. The sudden jump in the false positive rate near the bottom left of the curve is likely due to the presence of true negative examples with closely clustered scores, interspersed with no or few true positive examples. These examples may represent backbones in the evaluation data that differed sufficiently from those in the training data to earn a high distance metric from the convolutional sequence learner.

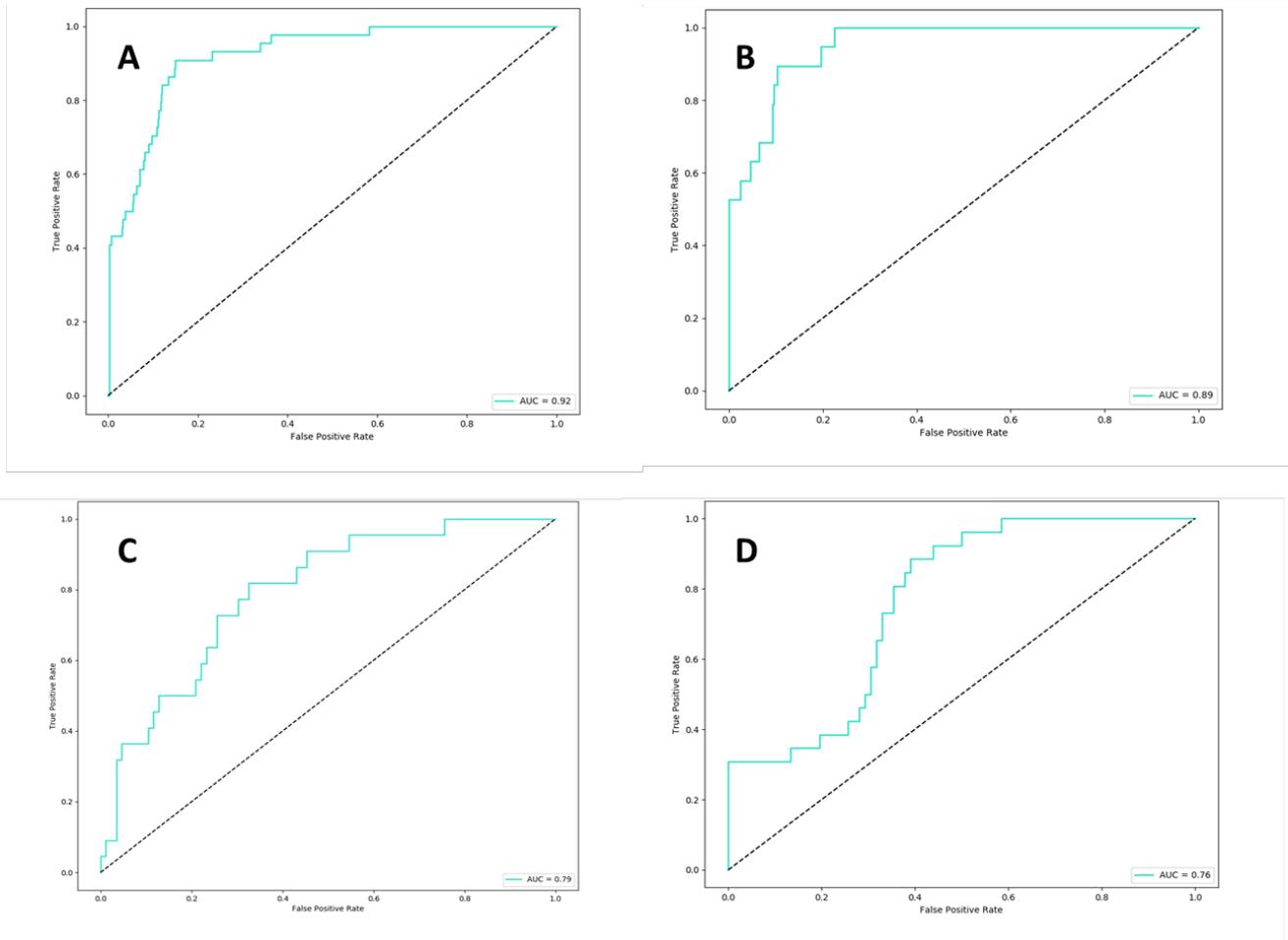


Figure 7. Receiver Operator Characteristic (ROC) curves for each ML model trained to detect cloning boundaries using backbone data, validated using VSS data and evaluated with sequences from engineered organisms. All models were trained on k -merized sequence data with $k=8$ and stride length of 8. Training, validation, and evaluation data were 500 bp sequences as described in the text. (A) ML model trained on UniVec and NCBI Plasmids sequences and evaluated on sequences drawn from Addgene plasmids; AUC=0.92. (B) ML model trained on UniVec and NCBI Plasmids sequences and evaluated on sequences drawn from a synthetic biology company data; AUC=0.89. (C) Model trained on RefSeq Bacteria sequences and evaluated on sequences drawn from modified bacterial DNA published in the literature; AUC=0.79. (D) Model trained on RefSeq Plant sequences and evaluated on sequences drawn modified plant DNA published in the literature; AUC=0.76. (See Tables 1-3 for more details).

Discussion

Our objective in this project was to investigate the utility of machine learning based tools for identifying engineered DNA. We have built and tested one such tool and found that it is capable of achieving 74-93% accuracy as a cloning boundary classifier, depending on the choice of training and evaluation datasets. Unsurprisingly, performance of the trained models seemed to correlate inversely with the size and complexity of the dataset available for training. Our goal for this project was to obtain an accuracy of 90% as a threshold for concluding that ML approaches show promise as a rapid triage tool for detecting the insertion of non-native DNA sequences into plasmids or the genome of an organism, and we approached this level of performance in this work. The goal for an operational triage tool would be accuracy over 99% (to reduce the burden of manual verification), with a very low false-negative rate (false positive results being more tolerable than missing a genetically engineered modification of significant importance). Our results suggest that, with further development, ML models are likely to be incorporated in such analyses in the future. We use “models” in the plural because, as our results show, training models on different classes of DNA sequence data lead to varying results, so models will likely need to be trained to detect cloning boundaries in a similar variety of inputs.

Reaction to this work by B.Next’s expert community. On February 6, 2018, B.Next held a roundtable discussion event that was attended by many of the same individuals present at the original September 2016 meeting that launched the B.Next GEMstone project and IARPA’s FELIX program. The work presented in this report was discussed in detail during the roundtable and the reception was strongly positive. The participants expressed explicit interest in our methodology and in further developing machine learning synthetic DNA detection tools. During this discussion, several important questions were raised surrounding operationalization of such a tool:

How well does the model work on a collection of sequence reads? Modern DNA sequencing is performed on millions of short fragments from a longer strand of DNA. We built our model to work within this paradigm by assuming that input sequences would be within the length range produced by current sequencer reads. However, our model has no ability to treat a set of sequences as a collection from a given longer sequence. The model can be applied to sequential subsequences taken from a longer sequence (such as an entire bacterial genome), but this is possible only after the individual reads have been assembled into the longer source sequence. Sequence assembly is a time consuming and complex process, and is an interesting machine learning challenge in its own right. If our model is intended to work as a rapid triage tool, it should be tailored to take a set of un-assembled reads as input and return an estimation that the collection as a whole has been engineered in some way. It is likely that a move in this direction would ultimately improve model performance, as there is more information about sequence origin in a collection of related sequences than within any one individual sequence. However, a test of this hypothesis question was deemed outside the scope of our initial approach.

What level of expertise is required by the user? One of the key motivations in exploring a machine learning approach was to make detecting evidence for genetic engineering scalable. Even after prolonged model development, it is likely that for any particular input sequence, an expert human biologist will be able to provide a more accurate assessment of the sequence source. Such an analysis requires time and resources that restrict its usefulness as a triage tool in the field. Ideally, a genetic engineering detector would be useable by clinicians or operatives with no specific domain knowledge. Of course, such users will likely require an interface other than the command line, so a productized version of a triage tool will need a graphical user interface.

Our current model does not require any domain knowledge of machine learning or biology to apply, but it does require some level of knowledge to train. A potential user has to decide what sorts of sequences a given model will be used to screen and train a pipeline accordingly – a non-trivial task. The goal would be to have a set of pre-trained models for the most common classes of sequences, available as packages and applicable without additional training.

How many pipelines or models will ultimately be required? As discussed above, we trained three separate pipelines – one each on a set of plasmid, bacterial, and plant backbone sequences. Was this necessary? Could we have trained a single model on backbone sequences from all three sources without sacrificing performance? A series of simplified experiments was performed early in the development process to investigate this question, and the answer was unequivocal: the wider variety of backbone sequences that a model of a given size is driven to learn, the worse it will perform at identifying deviations from those sequences. It is possible to increase the model size in several ways which has the potential to increase its representational capacity (allowing it to learn more), but this increases the likelihood that the model will overfit its training data and dramatically increases computational demands.

This is a common problem in machine learning applications. During research projects, it is useful to build and test single models with as large of a representational capacity as possible. However, for practical purposes it is better to build a set of smaller models, each focused on one narrow aspect of the problem. Ultimately, this would be our strong recommendation: establish a set of interlocking tools, say one for screening viral sequences and one for screening bacterial sequences, rather than overburdening a single model for the sake of elegance.

Training/Test Bleed: It is likely, particularly for the plasmid pipeline, that sequences in the training set were highly similar to sequences in the testing set. Care was taken to carefully separate sequences between training, validation, and testing sets as described above, and a simple filter was applied to remove any identical subsequences from the dataset. However, it would likely be appropriate to apply a more sophisticated screen of sequence similarity in order to prevent very similar sequences from being present in both the training and testing sets. If present, such sequences could artificially inflate our final performance metrics. Ultimately, the goal of the machine learning practitioner is to select a training set that is representative of the actual real-world data to which the model will be applied.

What about ‘natural’ boundaries? We chose to focus on a narrow indicator of what makes a sequence engineered: according to our model, a sequence is engineered if it contains a juxtaposition of two sequences that do not normally appear in nature (i.e. a cloning boundary). However, there are many natural situations in which just such a juxtaposition could occur. Two of the most common are horizontal gene transfer between bacterial species and transposon relocation, which occurs in virtually every known organism (transposable elements make up an estimated 2% of the human genome). As it currently exists, our pipelines, even perfectly trained, would incorrectly label either of these phenomena as clear examples of genetic engineering. This could be addressed by augmenting the training set with examples of ‘natural’ boundaries, but it seems likely that this would require far too many examples for the model to develop a meaningful representation of such a highly stochastic process. An alternative approach, which seems more promising, is to develop a primary screening layer that could be applied before or after the machine learning pipeline and would directly check for known natural boundaries – there are well established tools for identifying horizontal gene transfer in certain bacterial species.

What is the smallest modification our model can detect? To what degree is the model sensitive to the size of the insert? Current DNA engineering techniques are capable of modifying single bases within a sequence. Is our model capable of detecting such small changes? In its current instantiation, probably not. Within a given training set of limited size, a k-mer is likely to appear in nearly identical contexts as the same k-mer with a single base modified. In fact, certain k-mers simply will not appear in the training set at all (barring our additional layer of data augmentation in which we randomly flip bases) – and the embedding vector will tend to place them near the k-mers that they most resemble. Thus, a single base pair modification will register as at most a small deviation in the embedding space and the classifier will generally fail to register this deviation as an indicator of engineering. In fact, errors in the sequence learner’s predictions are likely to overwhelm the relatively small differences between the embedding vectors of two highly similar k-mers.

This problem extends beyond single base pair modifications to any insertion that fits entirely within one convolution of the convolutional sequence learner. A well-trained sequence learner might still produce accurate predictions if only a portion of its input resembles the training data. In this case, an insertion that was small enough not to disrupt the predictions of the sequence learner would very likely go undetected. This generalization of the problem also suggests a solution: modify the pipeline to operate at multiple scales in parallel. For example, a separate sequence learner could be trained and applied directly to smaller sequence subunits, say individuals bases or 2-mers. This introduces a new set of problems during training, but ultimately it seems clear that a different tool will be required to detect very small modifications.

Will an optimized machine learning tool replace human experts? No. We envision our pipeline as one more tool in the investigative toolbox of bioinformaticists. As in many other areas, the goal is not supplant human expertise, but to supplement it.

What is the relationship between this study and efforts at other organizations? This effort complements but does not duplicate new initiatives at IARPA and DARPA:

- *Finding Engineering-Linked Indicators (FELIX)*, is a new program (currently in the contract negotiation phase) seeking new experimental and computational tools to detect evidence for engineering in biological systems. The program was started by a senior IC scientist who attended the B.Next roundtable in 2016, and soon after obtained a leave of absence to create the program at IARPA. A significant amount of FELIX’s effort will be devoted to signatures other than those found in DNA sequences, and the requirements for tools being developed are not focused on rapid assessment. The work featured in this study was focused on methods that can generate results quickly upon obtaining a genomic sequence from a pathogen, and as such could be considered a “triage” tool or method to be used while the user employs more complex tools developed under FELIX. B.Next staff served as subject matter experts on the source selection board for proposals submitted to the FELIX program.
- *Functional Genomic and Computational Assessment of Threats (Fun-GCAT)*, an IARPA program that is developing new ways to predict “dangerous” functions encoded by DNA or RNA when the query sequences are short (50-200 bases). The eventual users include commercial DNA foundries that receive orders for synthetic DNA from academic and commercial customers. The sequences in question could be either engineered or natural; distinguishing between these is not a Fun-GCAT objective.
- *Safe Genes*, a DARPA program, proposes to understand the functional limits of genome editing technology. The program also is attempting to develop countermeasures to the misuse of genome editing technology, with the potential goal of “resetting” an edited genome to its natural state. The program assumes that performers already understand the engineered nature of a subject sequence.

Future Directions

We propose to continue exploring the utility of machine learning tools for the detection and classification of engineered DNA. We have identified two promising areas of interest on which we may focus during the coming fiscal year, pending decisions on FY19 programming:

Functional Classification: Given a purported modification, can we determine the intended functional change? Was the engineer trying to confer resistance to a common antibiotic or enable a microorganism to evade the immune system in some way? Predicting biological function from DNA sequences is a long-standing problem in biology – recent work suggests that machine learning may be a powerful tool for addressing it. This goes beyond the simple binary case, as the multiple classes of modification can quickly balloon.

Technical Identification and Source Attribution: Can we identify what sort of technical processes were employed to produce a given strand of engineered DNA? Can we infer what sequencing methodology was most likely used to generate the actual sequence data that our model takes as input? In conjunction with these questions, can we identify the most likely organizational sources

of a given piece of engineered DNA – a particular academic group or company, a national laboratory, or an unaffiliated individual biologist? Admittedly, this seems like an extremely difficult problem, requiring a quantity and quality of training data that will simply not be available – and yet even a modest contribution towards this end would be a significant step towards the implicit goal of the national security community: combatting the misuse of biotechnology.

References

- In-Q-Tel, Inc. 2017. Capabilities and problems associated with detecting engineered microorganisms and deducing function. Available at <https://www.bnext.org/article/roundtable-interrogation-of-suspect-biological-samples/>. Accessed 2/21/2018.
- Allen J. and T. Slezak. 2010. Genetic engineering workshop report. US Department of Energy technical report LLNL-TR-463112.
- Altschul S.F., Gish W., Miller W., Myers E.W., and D.J. Lipman. 1990. Basic local alignment search tool. *J. Mol. Bio.* 215:403-410.
- Libbrecht M.W., and W.S. Noble. 2015. Machine learning applications in genetics and genomics. *Nat. Rev. Genet.* 16: 321-332.
- Kunjapur A.M., Pfungstag, P. and N.C. Thompson. 2017. Gene synthesis allows biologists to source genes from farther away in the tree of life. *bioRxiv preprint 190868*. Available at <https://www.biorxiv.org/content/early/2017/09/19/190868>. Accessed 2/21/2018.
- Johnson I. S. 1983. Human insulin from recombinant DNA technology. *Science* 219: 632-637.
- Funke T., et al. 2006. Molecular basis for the herbicide resistance of Roundup Ready crops. *Proc. Nat. Acad. Sci. USA* 103:13010-13015.
- Adler A. and F. Yaman. 2016. AI for Synthetic Biology, IJCAI Proceedings. Presentation slides available at http://synthetic-biology.bbn.com/ijcai_workshop/content/ai_for_synbio_web.pdf. Accessed 2/21/2018.
- Pal C., Bengtsson-Palme J., Rensing C., Kristiansson E., and D.G.J. Larsson. 2014. BacMet: antibacterial biocide and metal resistance genes database, *Nucl. Acids Res.* 42:D737-D743.
- Heap I. The International Survey of Herbicide Resistant Weeds. Available at <http://www.weedscience.org/Sequence/sequence.aspx>. Accessed 2/21/2018.
- Lamesch P., et al. 2007. hORFeome v3.1: a resource of human open reading frames representing over 10,000 human genes. *Genomics* 89: 307-315.
- Martin J., Rosa B.A., Ozersky P., et al. 2015. Helminth.net: expansions to Nematode.net and an introduction to Trematode.net. *Nucl. Acids Res.* 43:D698-706.

Blake J.A., Eppig J.T., Kadin J.A., Richardson J.E., Smith C.L., Bult C.J., and the Mouse Genome Database Group. 2017. Mouse Genome Database (MGD)-2017: community knowledge resource for the laboratory mouse. *Nucl. Acids Res.* 45: D723-D729.

Gramates L.S., et al. and the FlyBase Consortium. 2017. FlyBase at 25: looking to the future. *Nucl. Acids Res.* 45(D1):D663-D671

Herscovitch M., Perkins E., Baltus A., and M. Fan. 2012. Addgene provides an open forum for plasmid sharing. *Nat. Biotechnol.* 30:316-317.

Brownlee J. 2017. Why one-hot encode data in machine learning? *Machine Learning Mastery*. Available at: <https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/>. Accessed 2/25/2018.

Ng, P. 2017. dna2vec: Consistent vector representations of variable-length k-mers." arXiv preprint. arXiv:1701.06279. Available at <https://arxiv.org/pdf/1701.06279v1.pdf>. Accessed 2/21/2018.

Rehurek R. and P. Sojka. 2010. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 workshop on New Challenges for NLP Frameworks*. Available at: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.695.4595>. Accessed 2/22/2018.

Conneau, A. et al. 2016. Very deep convolutional networks for natural language processing. arXiv preprint. arXiv:1606.01781. Available at <https://arxiv.org/pdf/1606.01781.pdf>. Accessed 2/21/2018.

Gehring, J. et al. 2017. Convolutional sequence to sequence learning. arXiv preprint. arXiv:1705.03122. Available at <https://arxiv.org/pdf/1705.03122.pdf>. Accessed 2/21/2018.

Appendices

Appendix A: Exploratory Data Analysis and Initial Experiments

In developing machine-learning algorithms for this particular task, we used a number of exploratory tools in order to build our model's complexity. We first worked to determine whether the necessary structure was present in the training data to utilize statistical approaches. For this, we used a clustering algorithm to extract classes or categories from a dataset that were not included in the training dataset but were known to contain cloning boundaries. The success of the unsupervised approach in clustering the blind dataset suggested that inherent statistical structure exists and there was potential for more complex modeling. Next we utilized some classical machine learning techniques. These models were used to build features that could define baseline inferencing accuracy. From these features, we considered a neural network as a classifying algorithm. Included below is a description of our efforts using unsupervised clustering, supervised non-neural-network methods, and featurization as applied to plant genomic sequence data.

Unsupervised clustering of a synthetic biology company's synthetic sequences. Sequences from a synthetic biology company dataset were broken into k-mers, with $k = 4, 8,$ and $16,$ and with strides of $1, 2, 4,$ and $8.$ K-merized sequences for each k-stride pair were then used to fit a Principal Component Analysis (PCA) matrix with 20 components. The sequences, reduced to 20 component vectors by PCA, were then clustered using k-means, where k was increased from 5 to 20. The optimal k was found to be 8. Upon investigation of the specific cluster members, they were found to match closely with 7 or 8 different synthetic processes used by a synthetic biology company. A similar set of experiments was conducted using t-SNE for dimensionality reduction instead of PCA, although these results were used primarily for visualization purposes. The results of both approaches suggested that the synthetic biology company dataset is suitable for supervised classification models.

These experiments also helped us to understand the significance of stride length during k-merization. Smaller strides produce more k-mers in a linear fashion (stride = 2 yields twice as many k-mers as stride = 4), and this yields a superlinear increase in total computation time during model application (training and testing). For PCA and gradient boosted trees, this increase is tolerable, but for more complicated network-based models, it quickly becomes unmanageable. This concern ultimately led us to adopt a stride length equal to our k-mer size, eliminating overlap.

Supervised non-network approaches. In order to establish an appropriate featurization of DNA sequences – a way of representing the sequence to a learning model that compactly retains the relevant information – we performed a set of experiments using a gradient boosted tree classifier (xgboost. XGB). We tested the XGB model against three different classifications: a synthetic biology company vs Addgene, *E. coli* vs a synthetic biology company and viral families.

- **A synthetic biology company vs Addgene:** Plasmid subsequences of length 1000 from the Addgene dataset and the synthetic biology company sequences were k-merized as above, with $k = 4$ and 8 only, and with strides of 2 and 8 only. An XGB classifier was trained on the k-mers without a representation of order – a given sequence was treated as an unordered ‘bag of kmers’. Another XGB Classifier, with the same hyperparameters, was trained on the same sequences represented as ordered lists of k-mers. Unsurprisingly, the classifier trained on ordered k-merizations performed significantly better. Another important consideration is the impact of k-mer overlap. Consider that two adjacent k-mers with stride = 1 will differ at only a single base. In this experiment, the performance of the XGB Classifier was barely affected by varying stride length. This was not surprising, given that working directly with k-mers provides the tree-based classifier with no notion of context. However, it seemed likely that as we moved towards a more representational featurization, k-mer overlap had the potential to act as a confounder for a context based model. If two k-mers show up in ‘the same context’ simply because they were produced by sliding the k-window over one base, that really isn’t what we want a contextual model (like our ultimate vector embedding) to learn. We’d rather the model represent which subsequences appear near each other in sequences without overlapping. This confirmed our decision to adopt a stride length equal to our k-mer size, eliminating overlap. We chose to set these values equal to 8 for computational reasons: longer k-mers would have delayed model development. However it is possible that increasing k-mer length could also provide more contextual information and this is a promising future direction.
- ***E. coli* vs a synthetic biology company:** An *E. coli* genome was obtained from RefSeq. This genome and the synthetic biology company dataset were broken into subsequences of lengths of 100, 200, 500, 1000, 1500, and 2000 bases. Sequences of each length were separately k-merized as above and used to train a gradient boosted classifier. Performance was approximately uniform across input sequence length, although slightly worse for 100 base inputs. Models trained with 1500 and 2000 base sequences exhibited long training times: 6 hours for 1500 and 18 hours for 2000.
- **Viral Family Classification:** This experiment was conducted to compare featurization methods – k-merization schemes as described above and a base pair ‘one-hot encoding’ approach (Brownlee 2017). One-hot encoding is a representation of categorical variables as binary vectors. It first requires that the categorical values be mapped to integer values. Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1. Each base location is treated as a list of 4 numbers, all zeros except for a 1 corresponding to the base that is present at the given location (or a fraction for ambiguous bases). Genomic sequences from eight families of virus were featurized in one of these two ways and then used to train a corresponding gradient boosted tree classifier. Data for testing the classifier was held out from the training dataset. The classifier trained on k-mers obtained 97% accuracy (for inclusion of test cases in a viral family or an “other” category) compared to 91% accuracy with the one-hot-encoding scheme. This result was interpreted as compelling evidence that k-mers were a promising preprocessing method for DNA sequence classification. We did

not test this classifier on any other viral family sets to see if the method would generalize. This is an interesting direction for future work, but we did not pursue it in this work because it did not further our goal of building a cloning boundary detector.

Exploring featurization: detecting open reading frame insertions in plant genomes. This was the first real attempt at the stated problem – we collected a small set of natural plant genomes from RefSeq and herbicide resistance genes and used a VSS pipeline similar to the one in our final approach to generate a set of modified plant sequences. We trained two classifiers using this dataset, the first of which used a BLAST featurization scheme. We created a BLAST database from a large collection of RefSeq plant genomes, overlapping with but not totally containing the sequences in our VSS training and test sets. We then searched against this database using established BLAST tools and used the output homology results to train a gradient boosted tree classifier. This approach achieved 98% accuracy on sequences that were in the BLAST database (it was able to perfectly identify them and any alterations were seen as segments of decreased homology), but only 51% accuracy on sequences outside of the database (the test set was balanced to have equal numbers of examples of both modified and non-modified plant sequences, meaning that purely guessing at random would have an accuracy of 50%).

Our second classifier used a sequential k-merization similar to what was used in our final approach, but instead of embedding these k-mers and predicting their sequence, we used them to directly train a gradient boosted tree classifier. This approach achieved 86% accuracy on held-out test sequences. This early result was a clear indication that BLAST based approaches would require significantly more work to be useful as generalizable screening tools. Unless we knew in advance exactly what sorts of sequences we'd be looking at, it seemed more promising to pursue an unstructured, sequence based approach to engineering detection.

Appendix B: Model Architecture and Training Details

Vector Embedding Model: This model is based on the model of Ng (2017) called dna2vec, which itself is an extension to DNA sequences of the popular natural language processing tool word2vec (Rehurek and Sojka 2010). The model consists of a two layer fully connected neural network. Each layer consists of 600 units with ReLU nonlinearities. We implemented a skip-gram training scheme because in general skip-gram methods are preferred for large data sets with smooth distributional information (i.e., any given k-mer is about as likely as any other k-mer to show up). Certain K-mers, such as 8 T's in a row, may appear infrequently, but no small set of k-mers dominate the distribution (i.e., its reasonably flat - the perfect setting for skip-gram training).

Within each pipeline, the embedding model is trained on 60% of the corresponding backbone sequences. The training, validation, and testing partitions were chosen at random on the sequence level, without regard for specific sequence lengths. The plasmid embedding model was trained for 20 epochs, and the bacterial and plant embedding models were trained for 35 epochs each due to the larger sizes of these backbone datasets.

Convolutional Sequence Learner: This model is loosely based off of a fully convolutional sequence-to-sequence machine translation tool built by Facebook® AI in 2017 (Conneau et al. 2016, Gehring et al. 2017). It consists of four convolutional layers followed by two fully connected layers, each with alternating non-linearities. Each convolutional layer contains 256 kernels of width five and full depth (each kernel samples all 40 dimensions of 5 consecutive vectors) with ReLU activations. The fully connected layers each contain 512 units with ReLU activations. The model takes twenty 40-dimensional vectors as input, or 800 total input features. The final output layer is a single 40-unit fully connected layer with a linear activation.

During training, ten vectors before and after a given position within a sequence are used as input and the linear output layer is driven to reproduce the vector that is actually at the given position. Specifically, the loss function for the model is the L2 distance between the predicted and actual vector (the same function which will ultimately produce the classifier distance metric).

This model was trained with an adam optimizer and an initial learning rate of $2e-3$. Weights were initialized randomly and an early stopping threshold of $1e-5$ was used. No dropout was used in this model during training.

Classification Threshold: In order to use the output of the convolutional sequence learner as a classifier, our pipeline compares these predictions with the actual embedding vector present in the input sequence. This comparison consists of computing the L2 distance between these vectors – which still leaves the question of how to tell if a given distance is sufficient to suspect a cloning boundary. We elected to use a simple threshold value applied to a rolling average of the distance metric. A rolling average was used because the distance metric varies considerably between k-mer locations and occasionally jumps to high values within a non-engineered backbone region. We chose a fixed threshold purely for simplicity, even though it seems likely

that some kind of rolling area-under-the-curve, or adaptive threshold approach could improve model performance.

Ideally there would be one more layer of statistical learning – a final model, possibly a gradient boosted tree classifier, that takes the distance metrics from an input sequence and predicts the sequence label. Such a capstone model would potentially allow for “end-to-end” training, a relatively recent advancement in machine learning in which a set of models arranged serially are all trained together so that the output of one model isn’t fit to whatever arbitrary performance metric has been established – but instead to be the optimal input for the next model.

The specific threshold values that we chose corresponded to the optimal point on each pipeline’s ROC curve: the point at which the sum of sensitivity and specificity was maximized. We then evaluated and reported on both accuracy and F2 score as seen in Table 4.

Technical considerations:

All of this work was done in Python®. The xgboost and scikit-learn libraries were used during the exploratory data analysis phase, and the keras and TensorFlow frameworks were used to train neural networks on Nvidia® GPUs. All of this software is open source.

Training the vector embedding used in this work takes approximately two hours per epoch. The convolutional sequence learner takes approximately 30 minutes per epoch and requires around 60 epochs to converge, roughly 30 hours per pipeline. This training time is not dependent on the initial size of the dataset as the embedded vectors are all of fixed size.