June 28, 2023
11:00 A.M.

EURECOM
Salle 101

# ACCEL-ALIGN-RMI

A genomic sequencer powered by *learned indexing*.

**Ilaria Pilo**

Politecnico di Torino
EURECOM

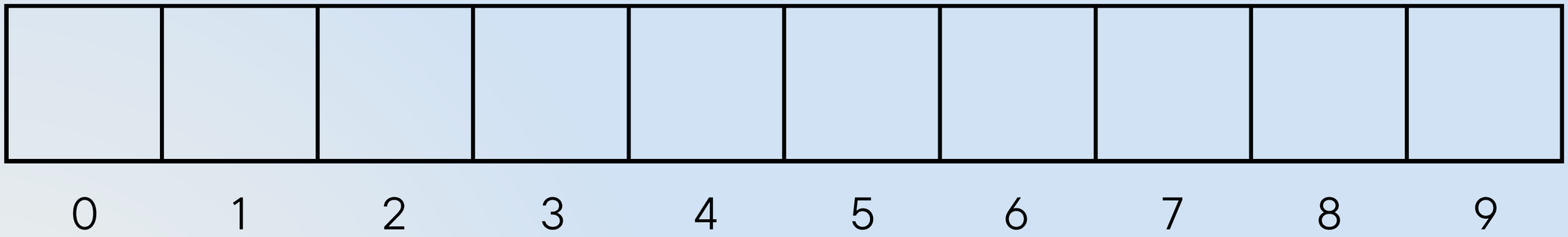
# TABLE OF CONTENTS

# WHAT IS A LEARNED INDEX?

And why do we need it?

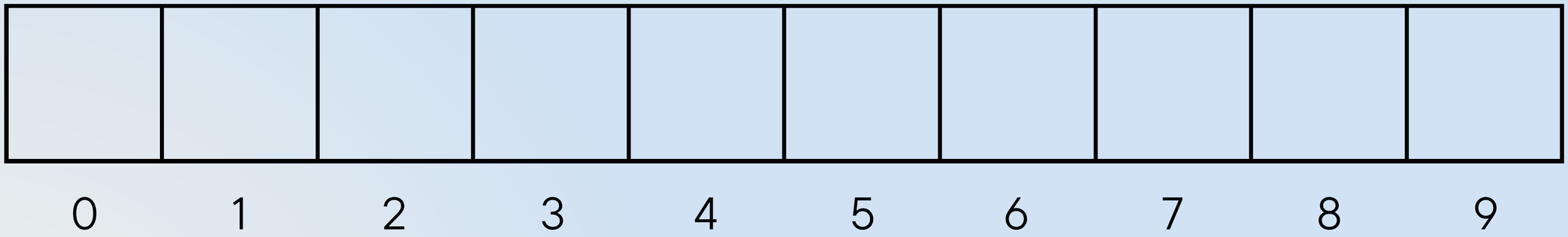
Let us suppose we have 10 keys, ranging from 0 to 99 and sorted in increasing order.

*In which position can we find key 42?*



Let us suppose we have 10 keys, ranging from 0 to 99 and sorted in increasing order.

*In which position can we find key 42?*

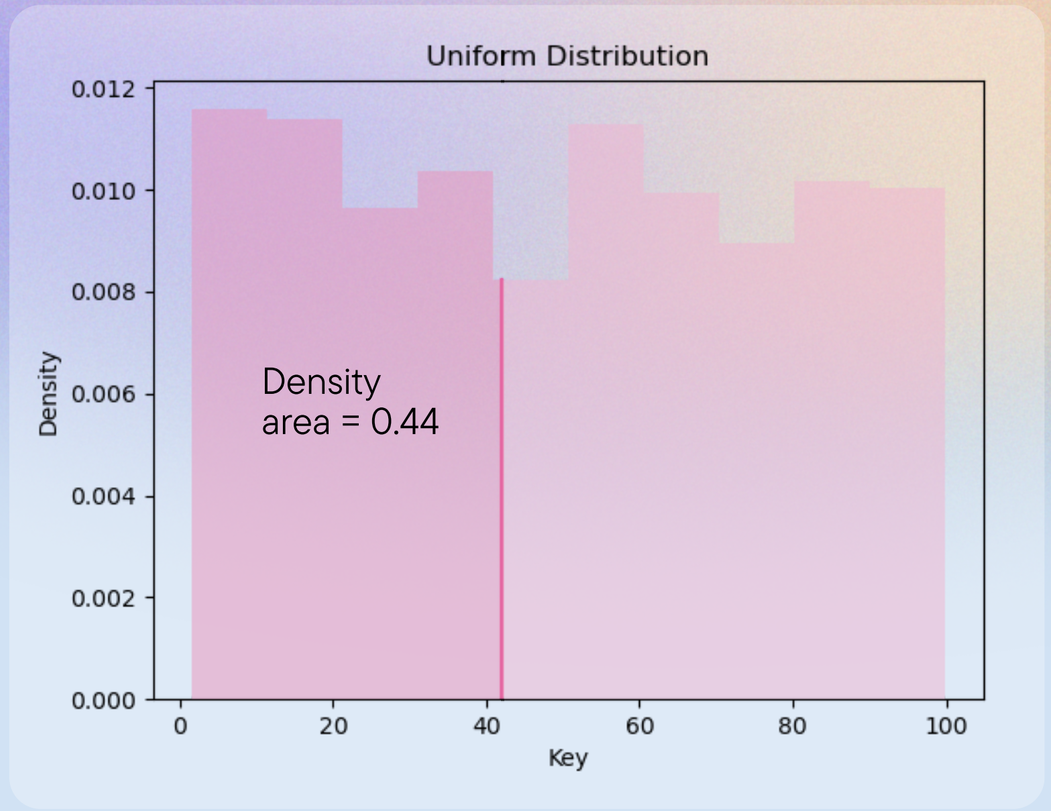


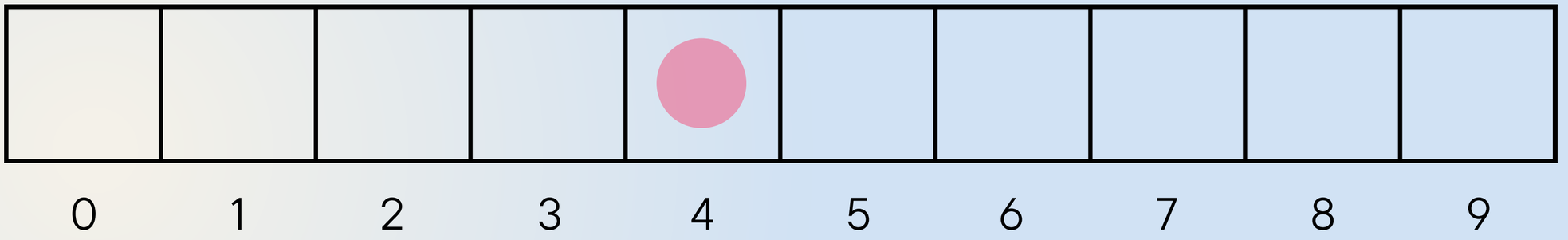

*We do not know!*

We first need to make some considerations on the keys' distribution.

# UNIFORMLY-DISTRIBUTED KEYS.

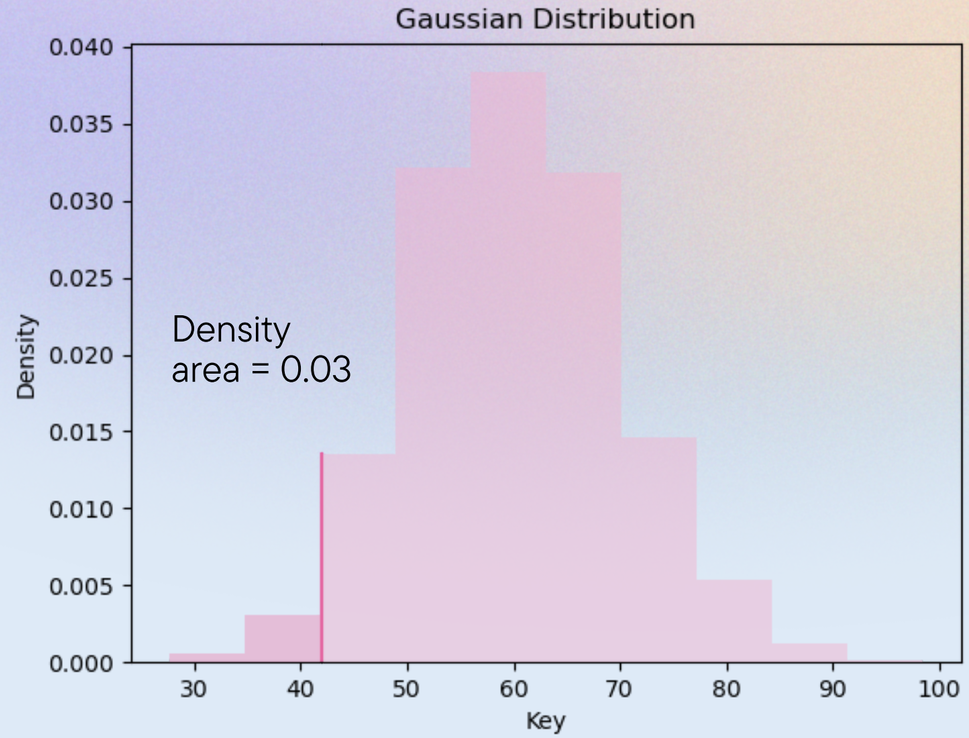
If we know that keys were drawn from a uniform distribution, then key 42 will likely be in position 4.

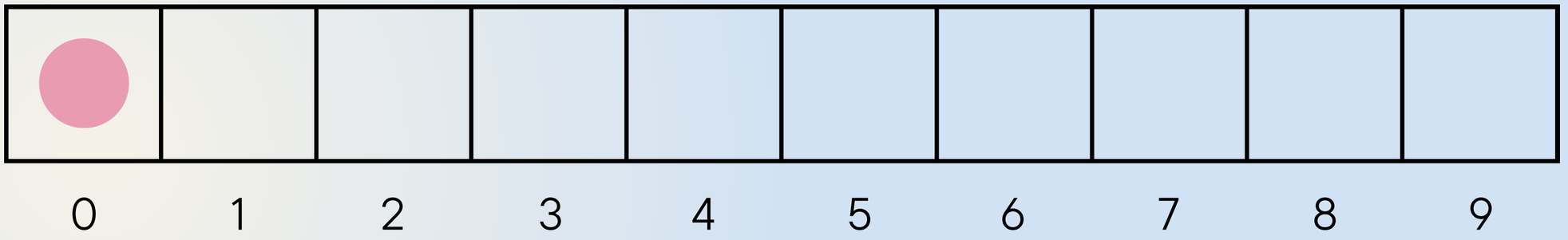


# GAUSSIANLY-DISTRIBUTED KEYS.

However, if keys were drawn from a Gaussian distribution ($\mu = 60$, $\sigma^2 = 100$), then key 42 will likely be in position 0.



What we have done up to now is use the Cumulative Distribution Function (CDF) of keys to predict a position in our index:

$$h(key) = \lfloor CDF(key) \cdot N \rfloor$$

where $N$ is the length of the index.

*But what if we do not know the CDF?*

What we have done up to now is use the Cumulative Distribution Function (CDF) of keys to predict a position in our index:

$$h(key) = \lfloor CDF(key) \cdot N \rfloor$$

where $N$ is the length of the index.

*But what if we do not know the CDF?*

***We learn it!***

# DEFINITION.

A *learned index* is an index that computes and learns the empirical CDF of keys, and then uses it to predict the position in the index.

+ The model allows the building of a tailored index that adapts to the specific distribution of keys.
- Every time a new key is inserted, the model needs to be re-trained, and the index needs to be rebuilt from scratch.

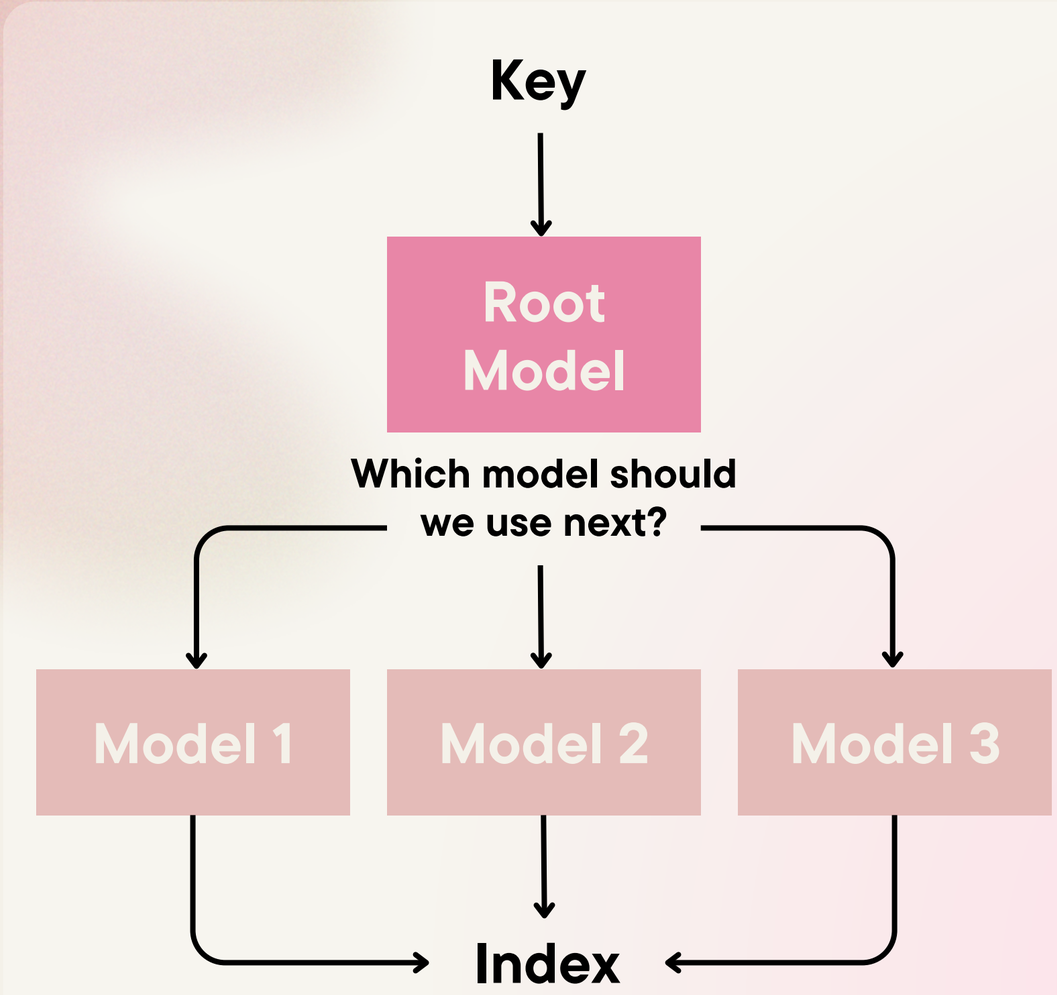
# RECURSIVE MODEL INDEX.

A hierarchical approach to CDF modelling.

# DEFINITION.

The *Recursive Model Index* is a hierarchical learned index that employs multiple layers of staged models to achieve high accuracy at a micro-level.

The model definition includes determining the number of stages, as well as the number and the kind of models for each stage.


Key
Root Model
Which model should we use next?
Model 1
Model 2
Model 3
Index

# THE ACCEL-ALIGN FRAMEWORK.

Building the index keys.

# ACCEL-ALIGN AND GENOMIC ALIGNERS.

*Accel-Align*[1] is a fast genomic aligner based on the seed–embed–extend method.

More in general, a *genomic aligner* is a tool used to align DNA sequences to a reference genome. Its purpose is to identify the position and orientation of short DNA fragments, called *reads* or sequences, within a larger reference genome.

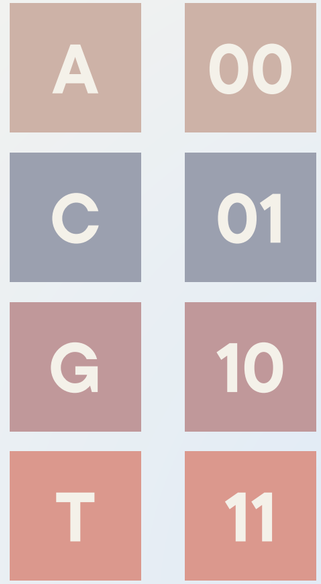
[1] github.com/yanlolo/accel-align-release

# REFERENCE STRING.

The *reference string* is a sequence of nitrogenous bases consisting of 3 billion characters.

The aligner's goal is to determine the position within the reference where a read was obtained: the reference string is the object that we aim to index!

To do so, we first need to extract all possible substrings (*k-mers*) of size 16. Then, we convert them into keys by expressing each base with a 2-bit value.

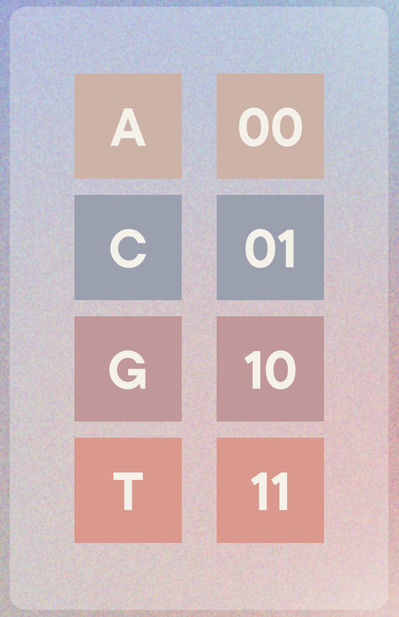


| | |
|---|---|
| A | 00 |
| C | 01 |
| G | 10 |
| T | 11 |

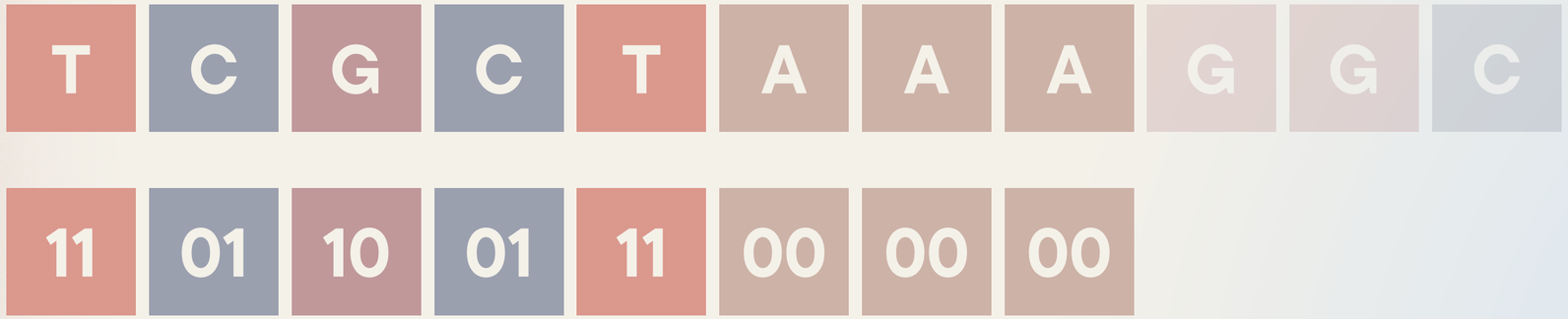
# EXAMPLE.

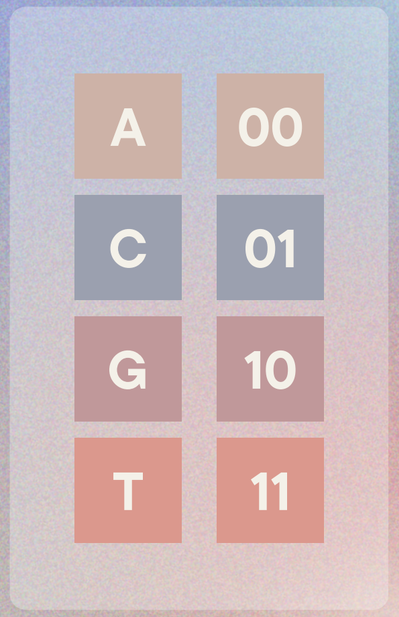
We want to compute all keys in this reference string.
For the sake of simplicity, suppose k-mer length = 8.
As a consequence, keys will be 16-bit long.

T C G C T A A A G G C

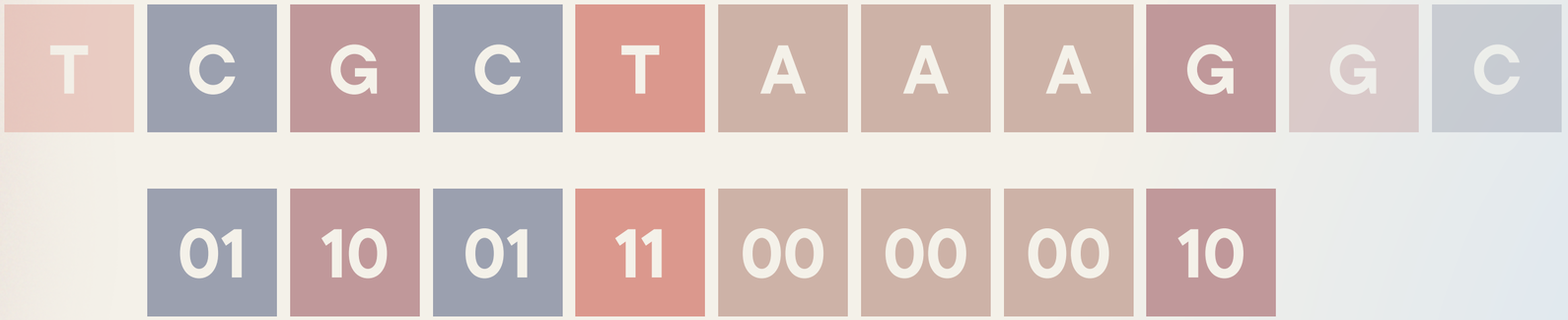


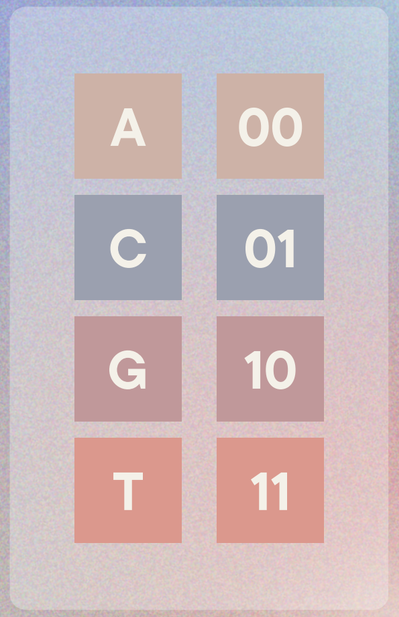

```
===============
      Keys   Pos
 0:  55744    00
 1:  -----    --
 2:  -----    --
 3:  -----    --
===============
```



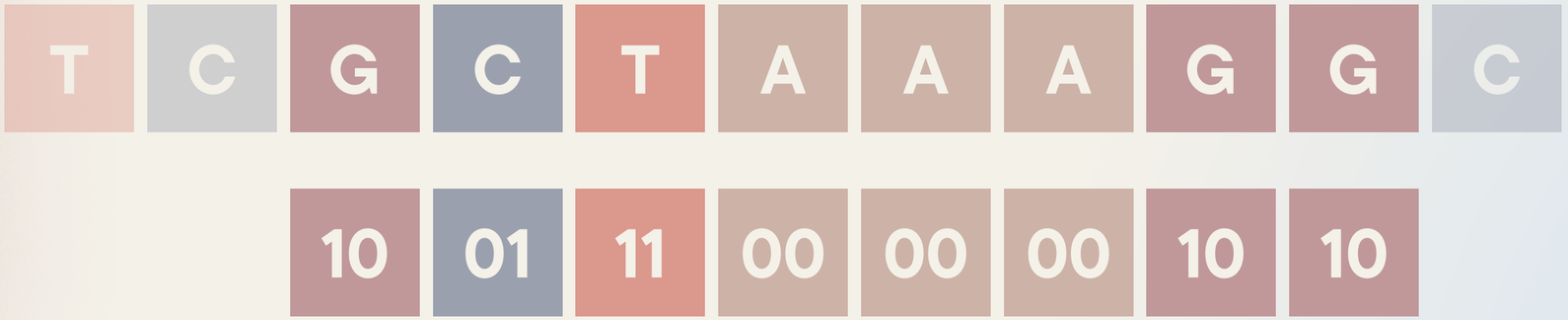



```
===============
      Keys   Pos
 0:  55744    00
 1:  26370    01
 2:  -----    --
 3:  -----    --
===============
```






```
===============
      Keys   Pos
 0:  55744    00
 1:  26370    01
 2:  39946    02
 3:  -----    --
===============
```



| | |
|---|---|
| A | 00 |
| C | 01 |
| G | 10 |
| T | 11 |

```
===============
      Keys   Pos
 0:  55744    00
 1:  26370    01
 2:  39946    02
 3:  28713    03
===============
```

| T | C | G | C | T | A | A | A | G | G | C |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | 01 | 11 | 00 | 00 | 00 | 10 | 10 | 01 |

# KEYS & POSITIONS GENERATION.

Once all keys have been extracted, we proceed with the *keys_uint32* and *pos_uint32* files generation.

***!*** given a reference string, the content of the generated files will never change

```
sort keys in ascending order
========== keys_uint32 ==========
write(number_different_keys, 8)
cumulative_pos = 0
prev_k = -1
for each k in keys:
     if k != prev_k:
         write(k, 4)
         write(cumulative_pos, 4)
     cumulative_pos++
=========== pos_uint32 ==========
write(number_positions, 8)
for each p in positions:
      write(p, 4)
```

| key | first position |
|---|---|
| 0 | 0 |
| 1 | 4 |
| 2 | 5 |
| 4 | 7 |

| positions |
|---|
| 43,796 |
| 77,174 |
| 77,175 |
| 77,176 |
| 12,543 |
| 33,421 |
| 33,422 |
| 92,393 |

# DYNAMIC LIBRARY IMPLEMENTATION.

Coding the RMI index.

# RMI.

*RMI*[2] is the most popular implementation of the Recursive Model Index, written in Rust and fully parallelizable.

The tool can work in two different modes:

*train* - trains the parameters of a given RMI architecture, and generates the C++ code to use it [default].

*optimize* - automatically tunes the hyperparameters, returning a small set of solutions covering the Pareto front.

[2] github.com/learnedsystems/RMI

# TAILORED FORK.

The biggest drawback of the original repository implementation is the *code generation* aspect.

***!*** change index → change code → recompile the entire program

To overcome such an issue, we wrap up the generated code into a *shared object*. The object is dynamically loaded at run time, and automatically managed by a C++ wrapper.

All these changes are included in *my tailored fork*[3].

[3] github.com/IlariaPilo/RMI

# TAILORED FORK.

Other extra features of the *fork*[3] are:

- remove <*filesystem*> library requirement in the generated code.
- add support for *(key,value)* structured files, where the value is ignored when training the index [this is the kind of file generated from the reference string].
- enhanced optimization output.

[3] github.com/IlariaPilo/RMI

# INTEGRATION.

Time to integrate the RMI index into Accel-Align!

- Use the library wrapper to load and interact with the index
- Modify the file-loading procedure to comply with the new *keys_uint32* and *pos_uint32* files
- Implement a helper function to perform a bounded binary search, using the output of the index lookup function
- Replace every direct index access with a call to the binary search function

# DEMO.

# RESULTS.

It's benchmarking time!

# HYPERPARAMETERS OPTIMIZATION.

| Models | Branch | AvgLg2 | MaxLg2 | Size (B) | Build time (ns) |
|---|---|---|---|---|---|
| radix,linear | 16777216 | 4.22363 | 8.00000 | 402653200 | 124443169288 |
| radix,linear | 8388608 | 5.02224 | 9.00000 | 201326608 | 112133104229 |
| radix,linear | 4194304 | 5.60928 | 10.00000 | 100663312 | 104003204641 |
| radix22,linear | 1048576 | 7.20767 | 10.98299 | 41943040 | 86061496663 |
| radix,linear | 524288 | 8.43821 | 12.91700 | 12582928 | 89637122826 |
| radix,linear | 262144 | 9.19067 | 13.94471 | 6291472 | 85086825486 |
| radix18,linear | 32768 | 12.08808 | 15.36150 | 1835008 | 107038194455 |
| radix,linear | 32768 | 12.25469 | 16.53535 | 786448 | 92663812165 |
| radix,linear | 1024 | 16.89976 | 21.13954 | 24592 | 93477981081 |
| radix,linear | 128 | 19.98812 | 23.62716 | 3088 | 110412216339 |

# HYPERPARAMETERS OPTIMIZATION.

| Models | Branch | AvgLg2 | MaxLg2 | Size (B) | Build time (ns) |
|---|---|---|---|---|---|
| radix,linear | 16777216 | 4.22363 | 8.00000 | 402653200 | 124443169288 |
| radix,linear | 8388608 | 5.02224 | 9.00000 | 201326608 | 112133104229 |
| radix,linear | 4194304 | 5.60928 | 10.00000 | 100663312 | 104003204641 |
| **radix22,linear** | **1048576** | **7.20767** | **10.98299** | **41943040** | **86061496663** |
| radix,linear | 524288 | 8.43821 | 12.91700 | 12582928 | 89637122826 |
| radix,linear | 262144 | 9.19067 | 13.94471 | 6291472 | 85086825486 |
| radix18,linear | 32768 | 12.08808 | 15.36150 | 1835008 | 107038194455 |
| radix,linear | 32768 | 12.25469 | 16.53535 | 786448 | 92663812165 |
| radix,linear | 1024 | 16.89976 | 21.13954 | 24592 | 93477981081 |
| radix,linear | 128 | 19.98812 | 23.62716 | 3088 | 110412216339 |

# ALIGNER.

**===== accuracy =====**

As the index does not affect the strategy of the aligner, we get the same accuracy with and without the RMI index.

```
-------------------------
Total strands:  10000000
Not aligned(%): 0.0
Correctly aligned(%): 97.20971
Exact aligned(%): 97.13467
-------------------------
```

**====== memory ======**

The *position* portion of the indices remains the same.
The *key* portion is compressed with respect to the original one (we keep only existing kmers).

```
-------------------------
Original size (keyv): 4 GB
     RMI size (keyv): 2 GB
-------------------------
```

**======= time =======**

As the RMI index is smaller than the original one, the total time of execution is lower. However, the original index uses direct access, meaning that it is 10x faster.

```
-------------------------
Original TOT time: 20:13 min
     RMI TOT time: 17:03 min
Original lookup time: 7.79 s
     RMI lookup time: 89.2 s
-------------------------
```

Time benchmarks were computed with 12 cores and averaged over 10 runs.

# THANK YOU FOR LISTENING!

# REFERENCES.


Yan, Y., Chaturvedi, N. & Appuswamy, R. *Accel-Align: a fast sequence mapper and aligner based on the seed–embed–extend method*. BMC Bioinformatics 22, 257 (2021).

Kraska, T., Beutel, A., Chi, E. H., Dean, J., & Polyzotis, N. (2017). *The Case for Learned Index Structures*. ArXiv. /abs/1712.01208

Maltry, M., & Dittrich, J. (2021). *A Critical Analysis of Recursive Model Indexes*. ArXiv. /abs/2106.16166

Sabek, I., Vaidya, K., Horn, D., Kipf, A., Mitzenmacher, M., & Kraska, T. (2022). *Can Learned Models Replace Hash Functions?* Proceedings of the VLDB Endowment, 16(3), 532-545.


Source code available at
*github.com/llariaPilo/accel-align-rmi*