Hyper-*k*-mers: efficient streaming *k*-mers representation

Igor Martayan*, Lucas Robidou*,
Yoshihiro Shibuya & Antoine Limasset

*co-first authors

April 2025, Seoul

Representing a sequence by its *k*-mer set

original read: ACGTTGAACGCCATTAGAGTAGCATGCGCAG

(*k*=22)

ACGTTGAACGCCATTAGAGTAG

CGTTGAACGCCATTAGAGTAGC

GTTGAACGCCATTAGAGTAGCA

TTGAACGCCATTAGAGTAGCAT

TGAACGCCATTAGAGTAGCATG

GAACGCCATTAGAGTAGCATGC

AACGCCATTAGAGTAGCATGCG

ACGCCATTAGAGTAGCATGCGC

CGCCATTAGAGTAGCATGCGCA

GCCATTAGAGTAGCATGCGCAG

- Essential for alignment-free methods (assembly, indexing, sketching...)
- *k*-mers' frequencies let us identify errors

- Uses up to *k* times more space than the input sequence

Compacting *k*-mers into super-*k*-mers

original read: ACGTTGAACGCCATTAGAGTAGCATGCGCAG

(*k*=22, *m*=10)

ACGTTGAACGCCATTAGAGTAG
CGTTGAACGCCATTAGAGTAGC
GTTGAACGCCATTAGAGTAGCA
TTGAACGCCATTAGAGTAGCAT
TGAACGCCATTAGAGTAGCATG
GAACGCCATTAGAGTAGCATGC
AACGCCATTAGAGTAGCATGCG
ACGCCATTAGAGTAGCATGCGC
CGCCATTAGAGTAGCATGCGCA
GCCATTAGAGTAGCATGCGCAG

original read: ACGTTGAACGCCATTAGAGTAGCATGCGCAG

ACGTTGAACGCCATTAGAGTAGCATGCG
ACGCCATTAGAGTAGCATGCGC
CGCCATTAGAGTAGCATGCGCAG

220 ⟶ 73 nucleotides

Compacting *k*-mers into super-*k*-mers

- Good data locality
  (neighbors are packed together)

- Still some duplication between
  consecutive super-*k*-mers

original read: ACGTTGAACGCCATTAGAGTAGCATGCGCAG

ACGTTGAACGCCATTAGAGTAGCATGCG

ACGCCATTAGAGTAGCATGCGC

CGCCATTAGAGTAGCATGCGCAG

220 ⟶ 73 nucleotides

From super-*k*-mers to hyper-*k*-mers

TCGAAATTGCTGAACGCATCGAGAAGTATCTACGGCCTCCTTAA

AATTGCTGAACGCATCGAGAAGTATCTACGGCCTCCTTAAGTCATCCAT

ACGCATCGAGAAGTATCTACGGCCTCCTTAAGTCATCCATTGAAGTCATCC

AGTATCTACGGCCTCCTTAAGTCATCCATTGAAGTCATCCATT

ATCTACGGCCTCCTTAAGTCATCCATTGAAGTCATCCATTAGTCAA

From super-*k*-mers to hyper-*k*-mers

TCGAAATTGCTGAACGCATCGAGAAGTATCTACGGCCTCCTTAA

AATTGCTGAACGCATCGAGAAGTATCTACGGCCTCCTTAAGTCATCCAT

ACGCATCGAGAAGTATCTACGGCCTCCTTAAGTCATCCATTGAAGTCATCC

AGTATCTACGGCCTCCTTAAGTCATCCATTGAAGTCATCCATT

ATCTACGGCCTCCTTAAGTCATCCATTGAAGTCATCCATTAGTCAA

From super-*k*-mers to hyper-*k*-mers

TCGAAATTGCTGAAC ⟵ AAATTGCTGAACG ⟶ AATTGCTGAACGCATCGAGAAGTATCTACGGCCTCCTTAA

TCGAAATTGCTGAACGCATCGAGAAGTATCTACGGCCTCCTTAA

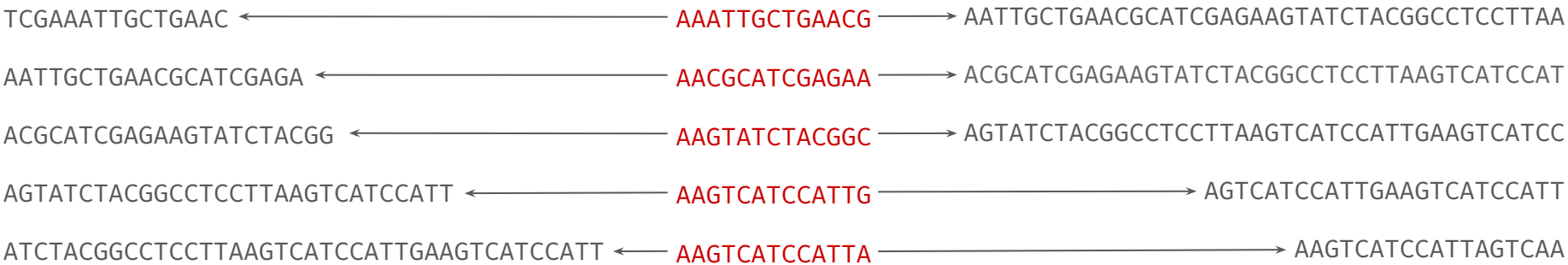
AATTGCTGAACGCATCGAGAAGTATCTACGGCCTCCTTAAGTCATCCAT

ACGCATCGAGAAGTATCTACGGCCTCCTTAAGTCATCCATTGAAGTCATCC

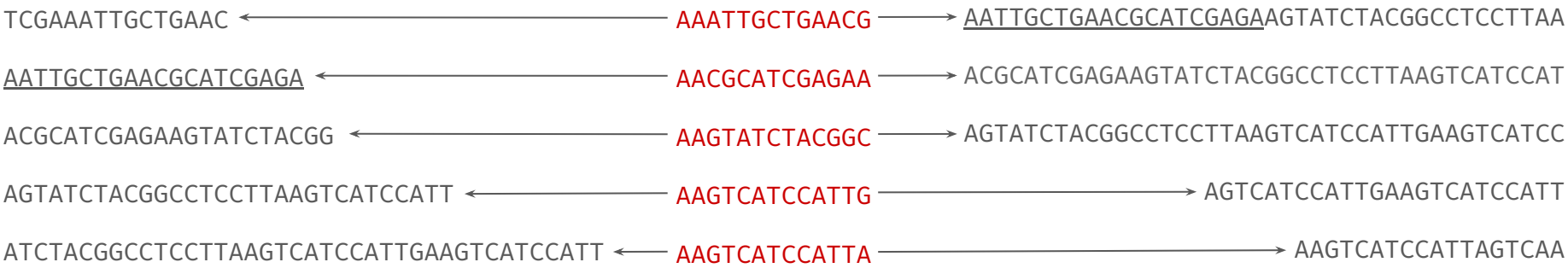
AGTATCTACGGCCTCCTTAAGTCATCCATTGAAGTCATCCATT

ATCTACGGCCTCCTTAAGTCATCCATTGAAGTCATCCATTAGTCAA

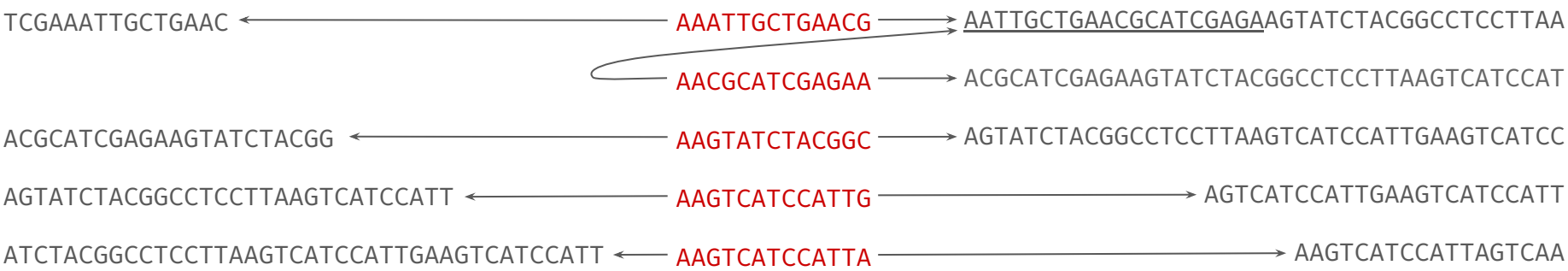
From super-*k*-mers to hyper-*k*-mers



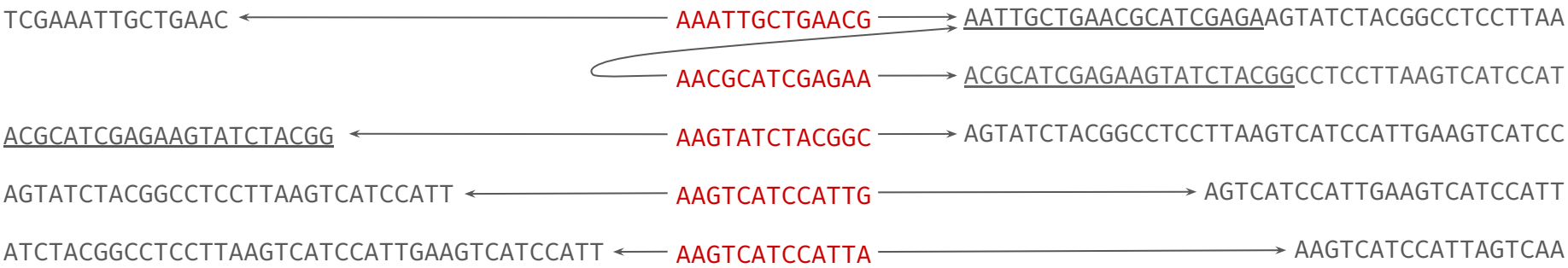
From super-*k*-mers to hyper-*k*-mers



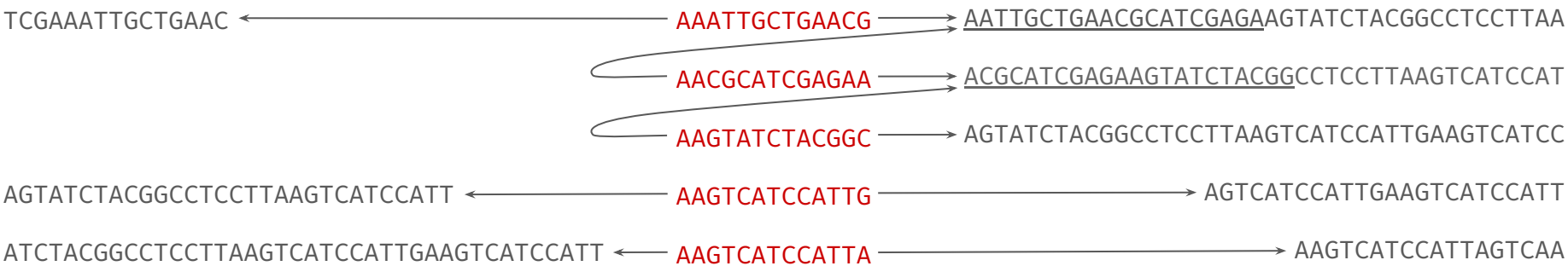
From super-*k*-mers to hyper-*k*-mers



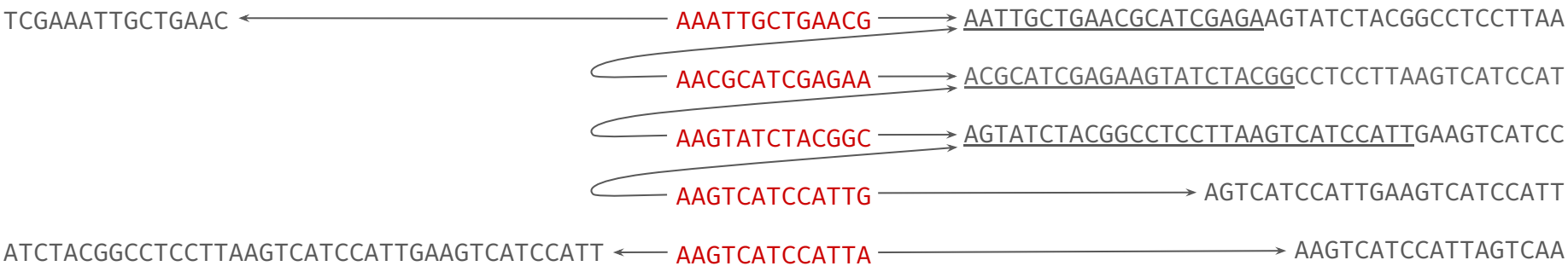
From super-*k*-mers to hyper-*k*-mers



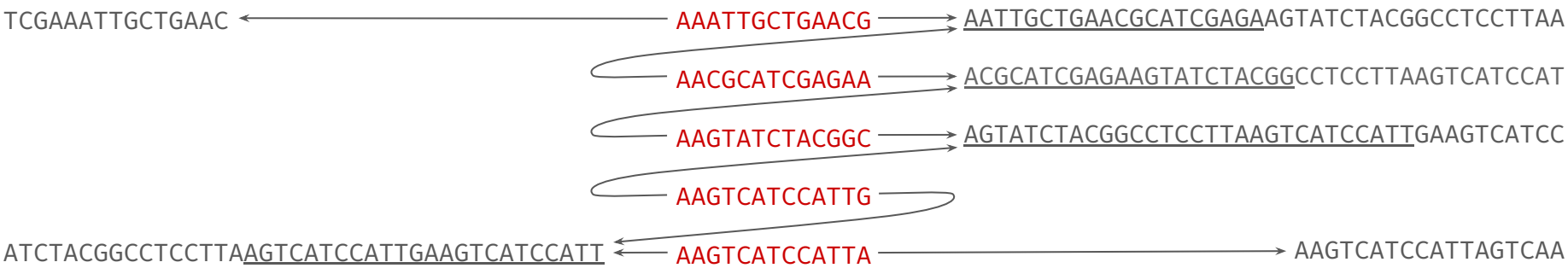
From super-*k*-mers to hyper-*k*-mers



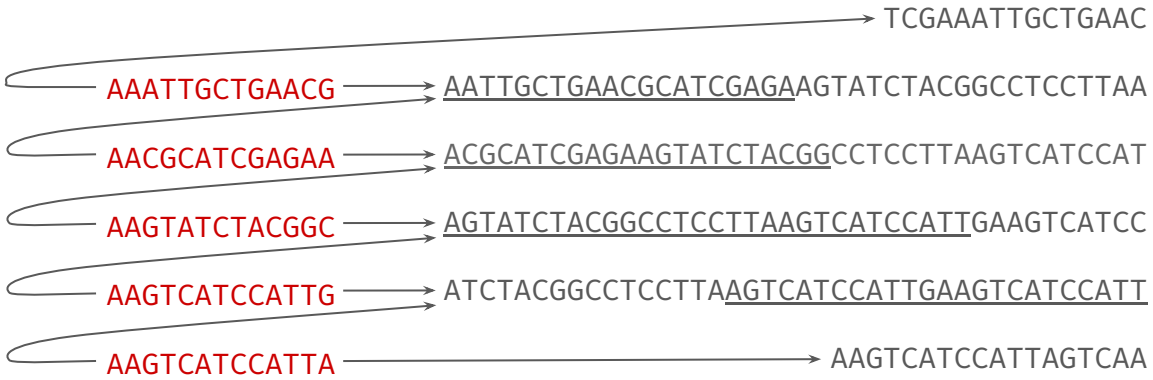
From super-*k*-mers to hyper-*k*-mers



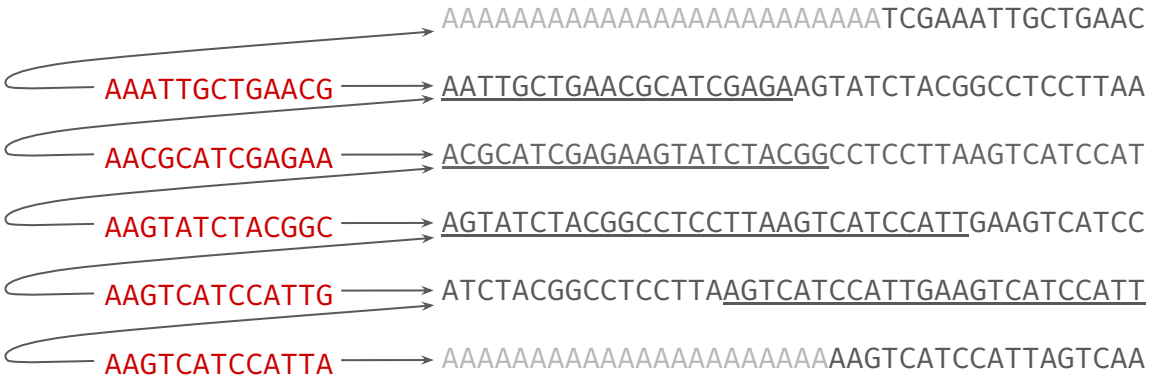
From super-*k*-mers to hyper-*k*-mers



From super-*k*-mers to hyper-*k*-mers



From super-*k*-mers to hyper-*k*-mers

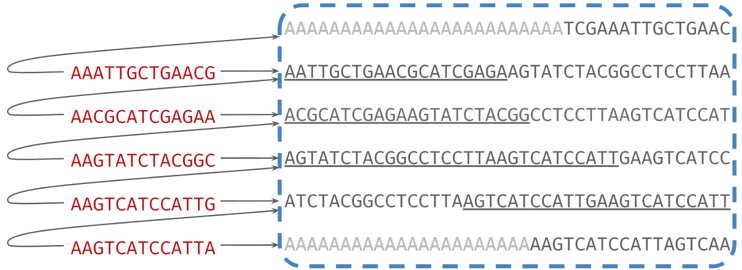


From super-*k*-mers to hyper-*k*-mers

Hyper-*k*-mers space analysis

Notations:

- random string of length $|S|$
- d = proportion of m-mers that are minimizers (a.k.a. the *density*)



# hyper-*k*-mer parts	⊠	part space usage
$d \cdot \|S\|$		$2\,(k-1)$ bits

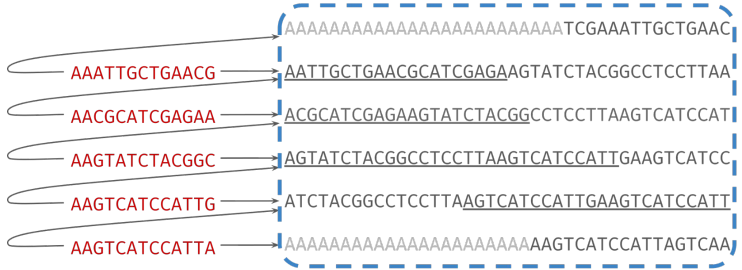
Expected space usage:

$$2d \cdot |S| \cdot (k-1) \text{ bits}$$

Hyper-*k*-mers space analysis

Notations:

- random string of length $|S|$
- d = proportion of m-mers that are minimizers (a.k.a. the *density*)



# hyper-*k*-mer parts	⊠	part space usage
$d \cdot \|S\|$		$2\,(k - 1)$ bits

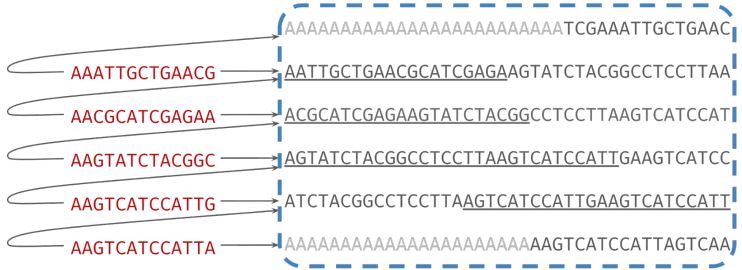
Expected space usage, including pointers:

$$2d \cdot |S| \cdot [k - 1 + \log_2(d \cdot |S|)] \text{ bits}$$

Hyper-*k*-mers space analysis

Notations:

- random string of length $|S|$
- d = proportion of m-mers that are minimizers (a.k.a. the *density*)



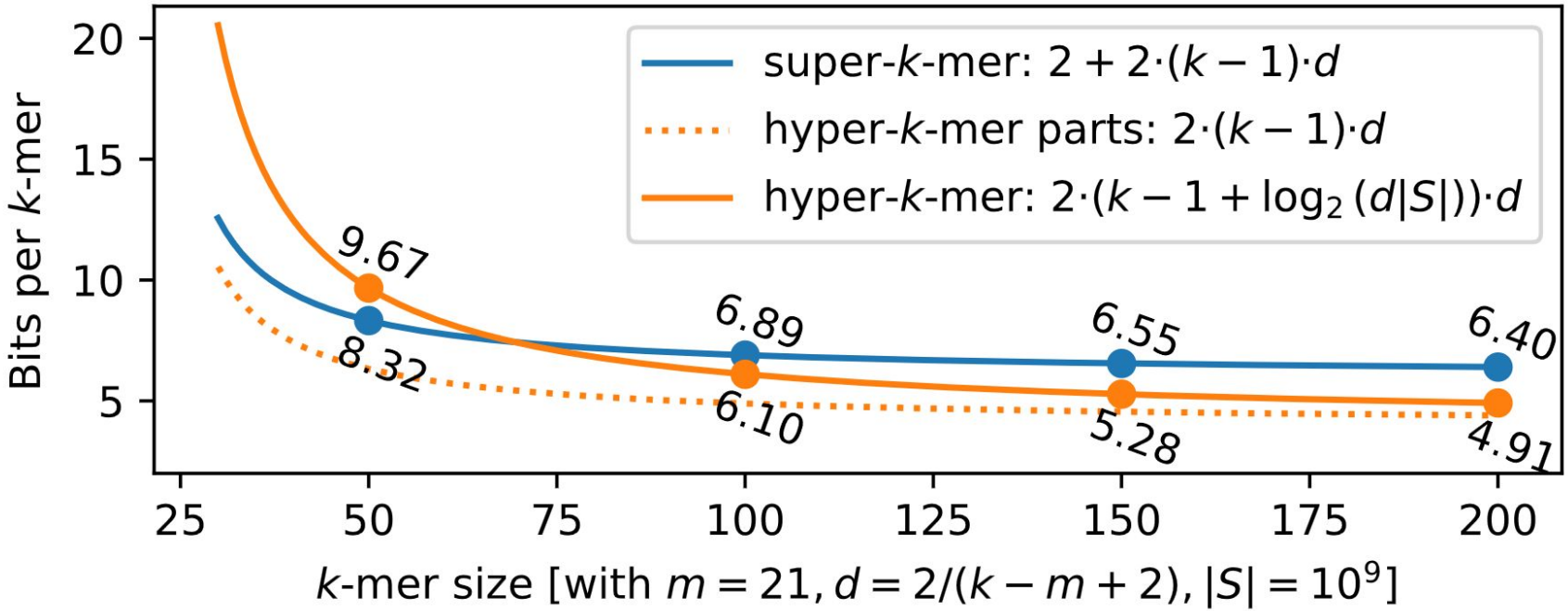
# hyper-*k*-mer parts	⊠	part space usage
$d \cdot \|S\|$		$2\,(k - 1)$ bits

Expected space usage, including pointers:

$$2d \cdot |S| \cdot [k - 1 + \log_2(d \cdot |S|)] \text{ bits}$$

$$\longrightarrow_{k \to \infty} \text{4 bits / } k\text{-mer}$$

Hyper-*k*-mers are more compact than super-*k*-mers



Use case: *k*-mer count data structure

minimizer ⟶ hyper-*k*-mer + count

minimizer	left	right	count
AAATTGCTGAACG	<0>	<1>	1
AACGCATCGAGAA	<1>	<2>	1
AAGTATCTACGGC	<2>	<3>	1
AAGTCATCCATTG	<3>	<4>	1
AAGTCATCCATTA	<4>	<5>	1

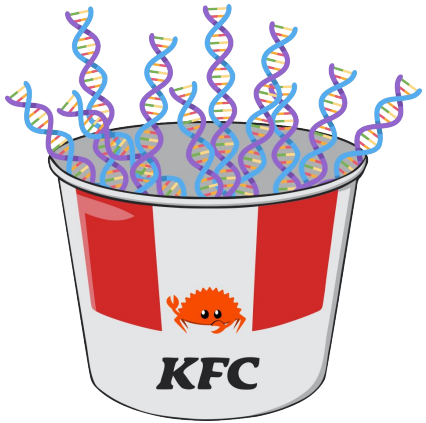
hyper-*k*-mer parts

<0>	AAAAAAAAAAAAAAAAAAAAAAAAATCGAAATTGCTGAAC
<1>	AATTGCTGAACGCATCGAGAAGTATCTACGGCCTCCTTAA
<2>	ACGCATCGAGAAGTATCTACGGCCTCCTTAAGTCATCCAT
<3>	AGTATCTACGGCCTCCTTAAGTCATCCATTGAAGTCATCC
<4>	ATCTACGGCCTCCTTAAGTCATCCATTGAAGTCATCCATT
<5>	AAGTCATCCATTAGTCAAAAAAAAAAAAAAAAAAAAAAA

K-mer **F**ast **C**ounter: a new *k*-mer counter

2 main goals:

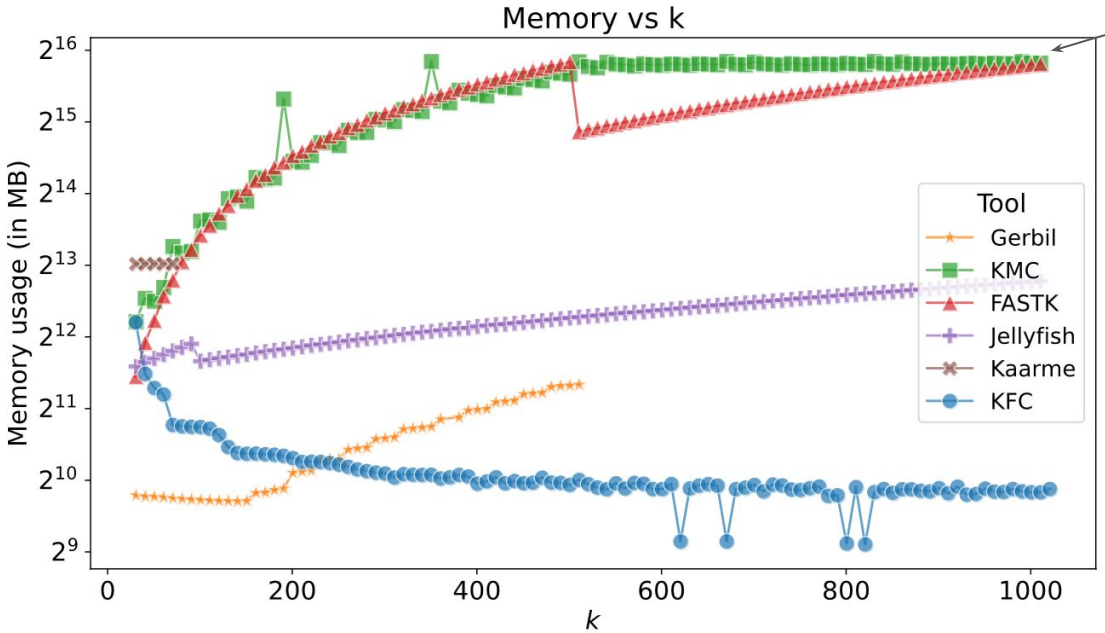
- **Easy on memory** (no disk required)
- **Fast** (multithreaded & SIMD instructions)



- Written in Rust
- Supports text and KFF outputs

- Available at `github.com/lrobidou/KFC` (AGPL)
- Experiments reproducible at `github.com/imartayan/KFC_experiments`

KFC uses less memory than its competitors

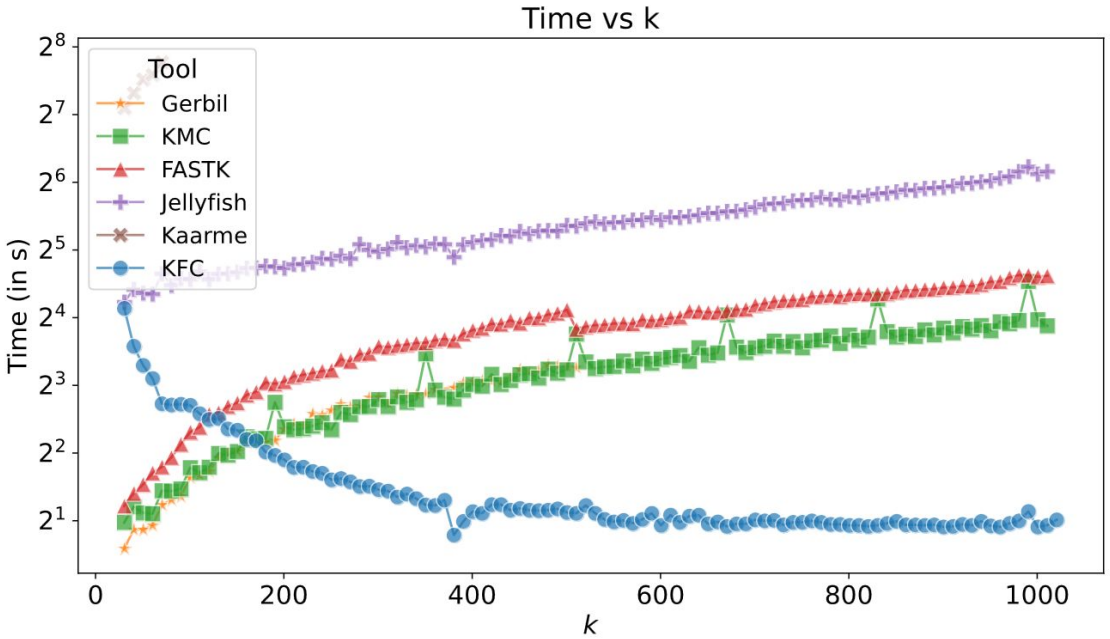


overflows on disk

k-mer counting benchmark on ONT Simplex E. coli dataset (SRR28370668)

downsampled at 100X coverage, with unique *k*-mer filtering

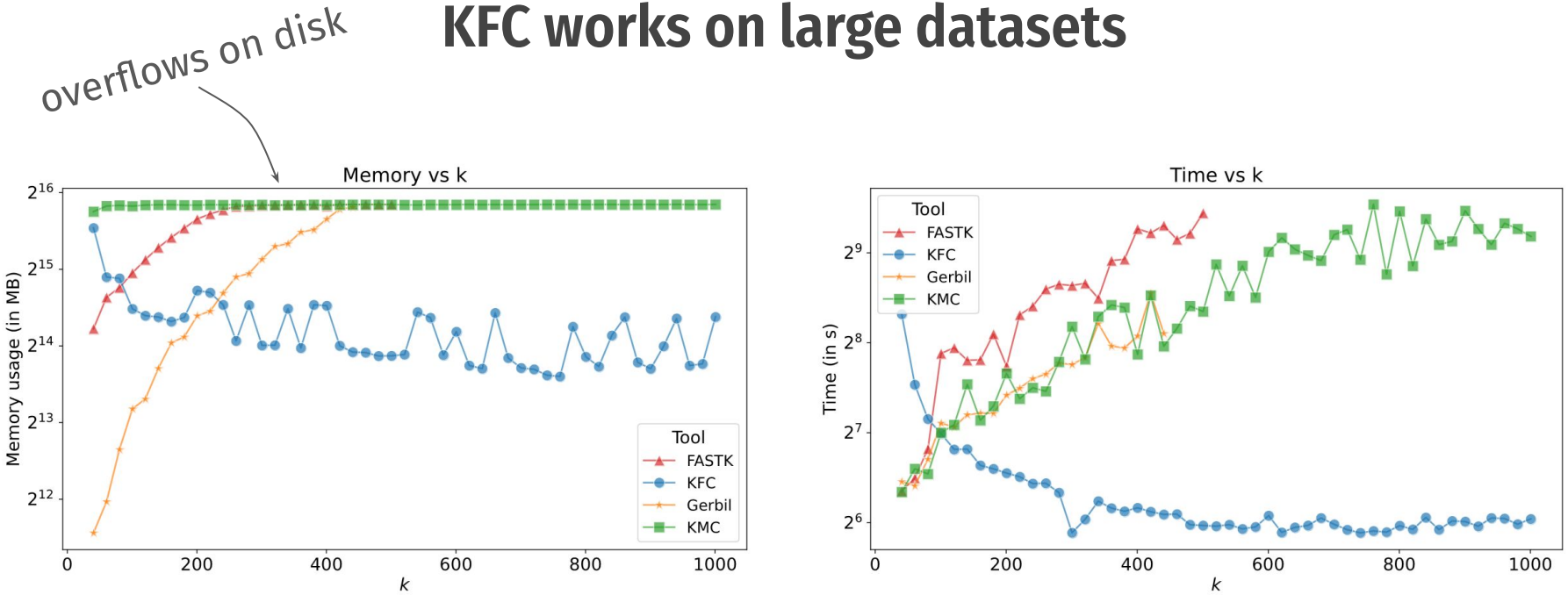
KFC is faster than its competitors



k-mer counting benchmark on ONT Simplex E. coli dataset (SRR28370668)

downsampled at 100X coverage, with unique *k*-mer filtering

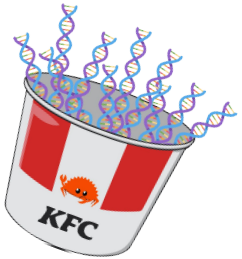
KFC works on large datasets



k-mer counting benchmark on HIFI human gut datasets

downsampled at 15Gbp, with unique *k*-mer filtering

Summary & what's next



	super-*k*-mers	hyper-*k*-mers
bits / *k*-mers	6	4
good data locality	yes	yes
searchable	yes	yes
fixed size	no	yes
"grepable" (SPSS)	yes	no

Hyper-*k*-mers:
- well-suited for large *k* values
- fewer bases while keeping locality
- adaptable to other minimizer schemes

Future work:
- optimize for $k < 63$
- improve for very large datasets
- further reduce duplication?

Preprint

Try KFC at `github.com/lrobidou/KFC`

Thank you!