What Compression Taught Us About Language

Craig Trim Follow 11 min read · Jan 6, 2026

105

The 30-year journey of an algorithm that accidentally learned to read

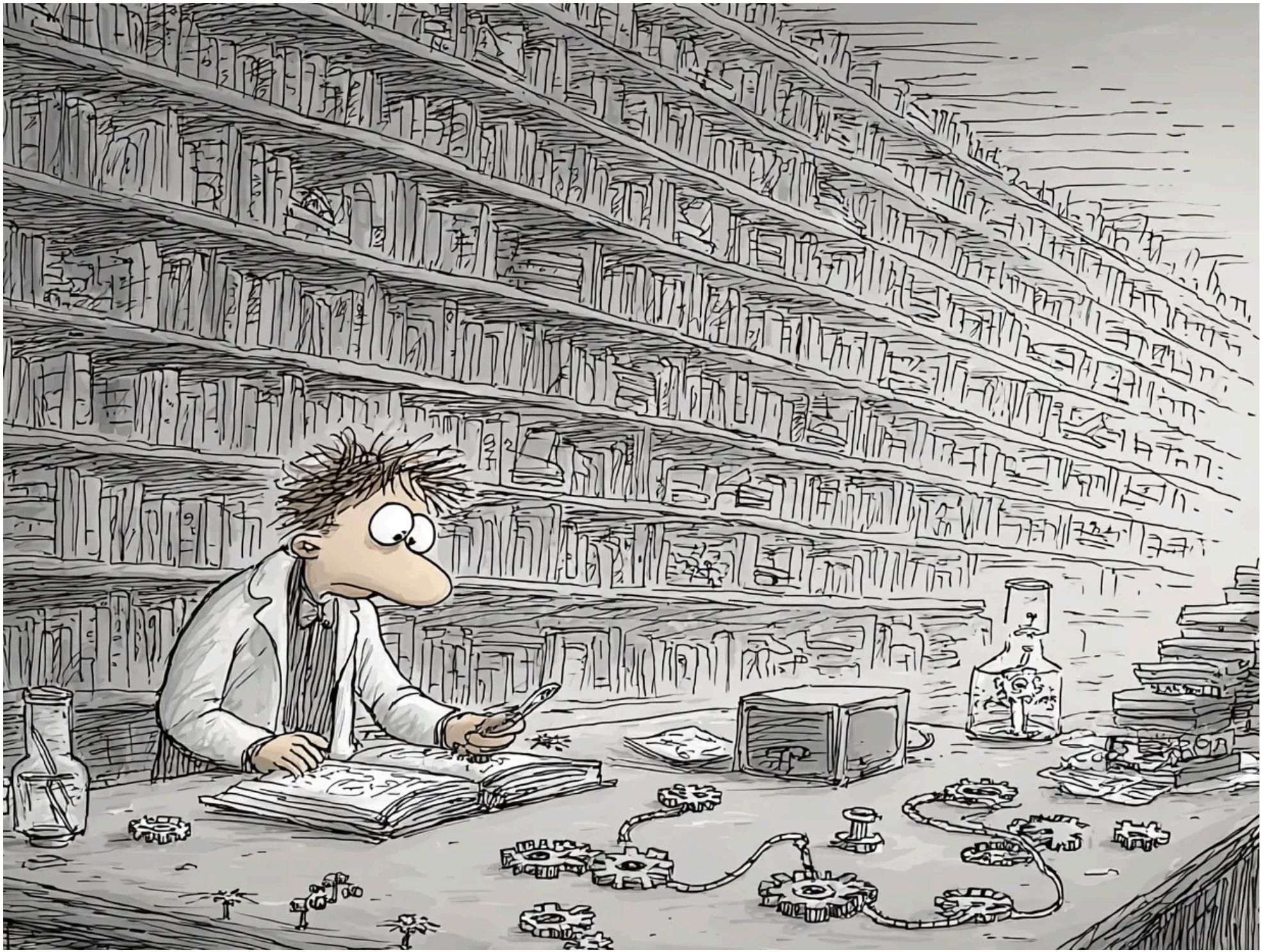
In 1994, Philip Gage published a short paper in the C Users Journal titled "A New Algorithm for Data Compression." It described a simple technique for shrinking files by replacing frequently occurring byte pairs with single bytes.

Thirty years later, that same algorithm (Byte Pair Encoding) powers every major large language model: GPT-4, Claude, LLaMA, Mistral. It's how ChatGPT reads your prompts. It's why you pay per token, not per word.

This is the story of how a compression hack became the foundation of modern AI.

Part 1: The Compression Era (1994–2015)

Philip Gage wasn't thinking about language. He was thinking about bytes.



He never once looked up.

His problem was practical: how do you compress arbitrary binary data efficiently? The dominant algorithms of the time (Huffman coding, LZW used in GIF files, and arithmetic coding) were sophisticated but complex.

Gage wanted something simpler.

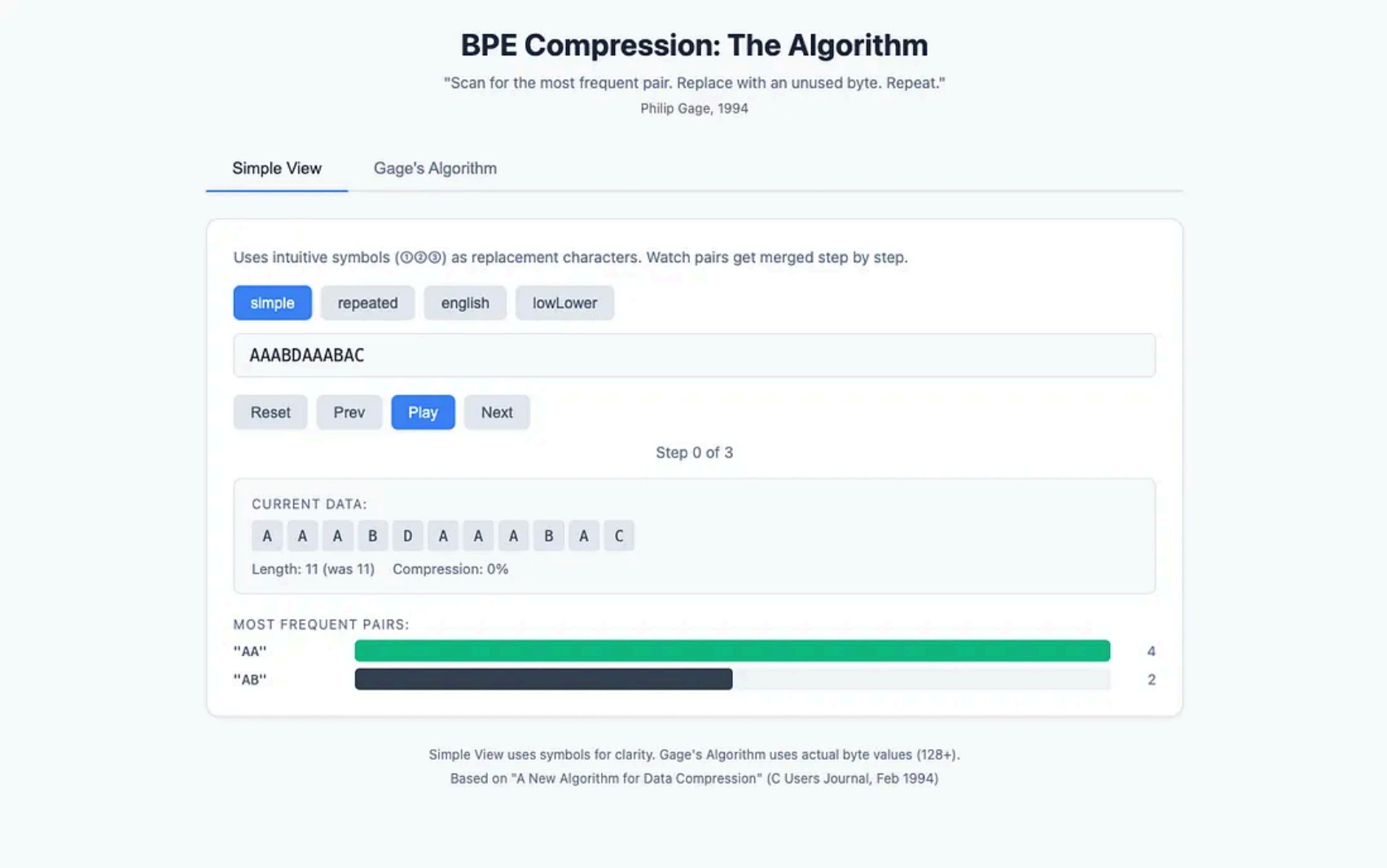
His insight: **if two bytes frequently appear next to each other, replace them with a single unused byte.**

The algorithm:

1. Scan the data for the most frequent pair of adjacent bytes
2. Replace all occurrences of that pair with a new byte not in the data
3. Record the substitution in a table
4. Repeat until no pair occurs more than once (or you run out of unused bytes)

That's it. No probability models. No complex data structures.

Just counting and replacing.



Link opens in craigtrim.com/demos/bpe-compression/

The Actual Mechanics

A byte can represent 256 values (0–255). In any given block of data, not all 256 values are present. ASCII text, for example, typically uses only 70–100 distinct byte values (letters, digits, punctuation, whitespace). The remaining 150+ byte values are “unused” and available as substitution codes.

Gage’s algorithm tracks this with two arrays:

```
unsigned char leftcode[256];
unsigned char rightcode[256];

// Initialize: every byte maps to itself (literal)
for (c = 0; c < 256; c++) {
  leftcode[c] = c;
  rightcode[c] = 0;
}
```

The sentinel value `leftcode[c] == c` means "byte c is a literal." When you create a substitution, you pick an unused byte and set `leftcode[unused]` and `rightcode[unused]` to the pair being replaced.

A Worked Example

Suppose your data block contains only the bytes: `A`, `B`, `C`, `D` (*ASCII values 65, 66, 67, 68*).

That means bytes 0–64, 69–255 are all **unused** and available as substitution codes.

```
Input: AAABDAAABAC
```

Pass 1: Most frequent pair is “AA” (appears 4 times).

- Find an unused byte. Let’s say byte 128 (0x80) isn’t in the data.
- Set: `leftcode[128] = 'A'`, `rightcode[128] = 'A'`
- Replace all “AA” with byte 128:

```
Result: [128]ABDA[128]BAC
```

Pass 2: Most frequent pair is now [128]A (*appears 2 times*).

- Pick another unused byte, say 129.
- Set: `leftcode[129] = 128`, `rightcode[129] = 'A'`
- Replace:

```
Result: [129]BD[129]BC
```

Pass 3: No pair appears more than once. Stop.

Original: 11 bytes → Compressed: 6 bytes + lookup table

To decompress, the algorithm uses a stack. When it encounters byte 129, it looks up `leftcode[129]` and `rightcode[129]`, pushes both onto a stack, and keeps expanding until it hits literals (bytes where `leftcode[c] == c`).

The Exponential Power

Here's the clever part: pair codes can reference other pair codes.

Byte 129 expands to `[128, A]`, and 128 expands to `[A, A]`. So 129 represents `AAA` in a single byte.

Gage noted: *"1024 identical bytes can be reduced to a single byte after only ten pair substitutions."*

That's `2^10 = 1024`.

What Happens When All Bytes Are Used?

This is the natural limit of Gage's original BPE.

From the paper:

> "The algorithm repeats this process until no further compression is possible, either because there are no more frequently occurring pairs or **there are no more unused bytes to represent pairs**."

and:

> "If no compression can be performed, BPE passes the data through unchanged except for the addition of a few header bytes to each block of data."

This is why Gage processes data in **blocks** (default 5000 bytes). Smaller blocks are statistically likely to have many unused byte values. A block of

ASCII text might use only 80 of 256 possible byte values, leaving 176 available for substitution codes.

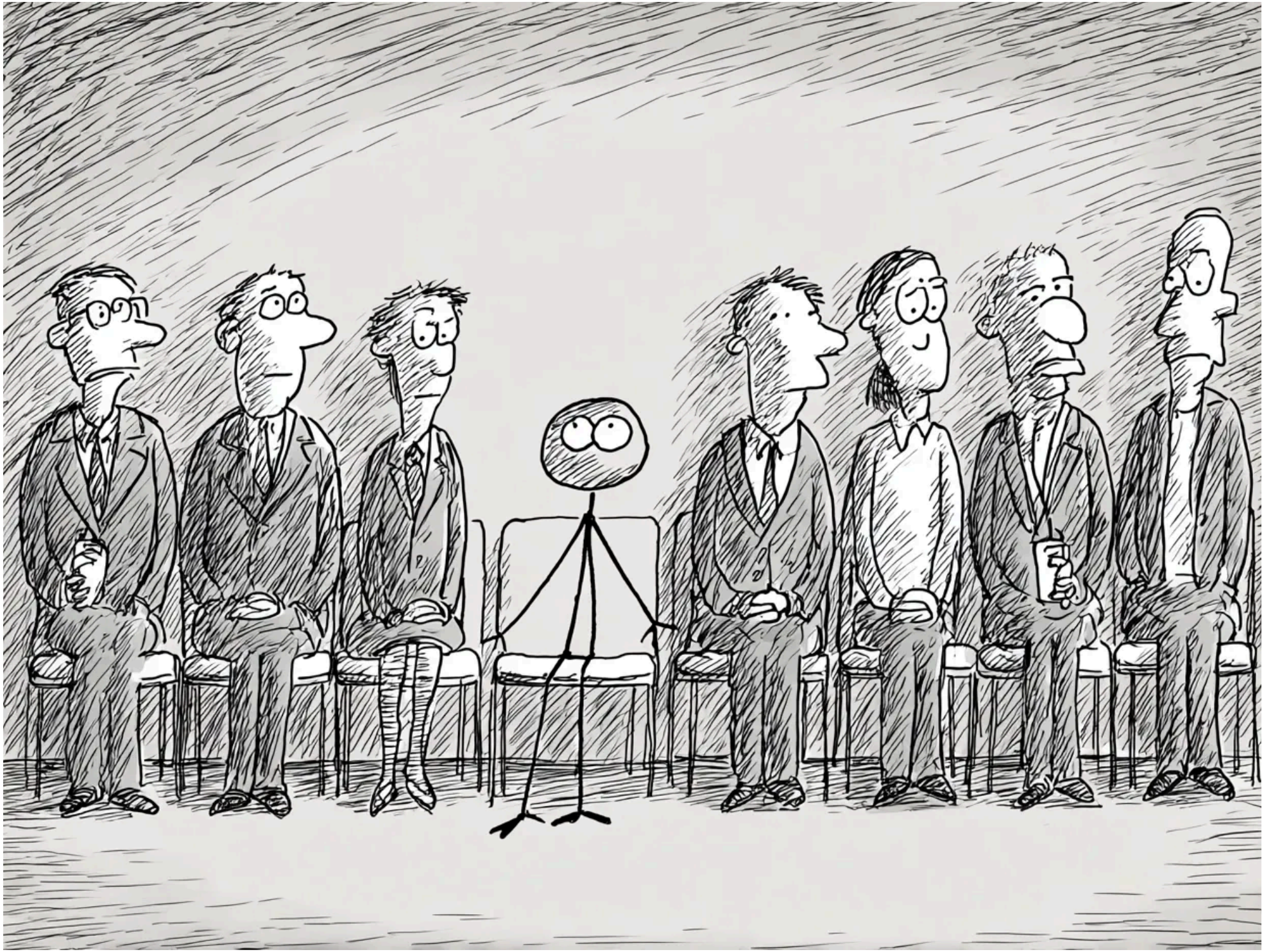
Why It Worked

BPE had several advantages over more sophisticated algorithms:

1. **Simplicity**: The code fit on a single page
2. **Speed**: Linear time complexity per pass
3. **Adaptivity**: No need to pre-specify patterns; it finds them
4. **Reversibility**: Perfect reconstruction guaranteed

It wasn't the best compression algorithm. LZW and arithmetic coding could compress further.

But BPE was *good enough* and *easy to implement*.



Twenty years later, guess who's running GPT-4?

Life on the Fringes (1994–2015)

BPE never gained mainstream adoption in compression tools.

7-Zip uses LZMA. WinZip and gzip use DEFLATE. bzip2 uses Burrows-Wheeler. These achieved better compression ratios; BPE's advantage was simplicity and a tiny memory footprint, not raw power.

Where did BPE actually live?

- **Embedded systems** with severe memory constraints (550 bytes for decompression!)
- **Educational implementations** in algorithms courses
- **Academic papers** comparing compression techniques
- **Niche utilities** where "good enough" mattered more than optimal

In 1999, Japanese researchers at Kyushu University and Kyushu Institute of Technology published "Byte Pair Encoding: A Text Compression Scheme

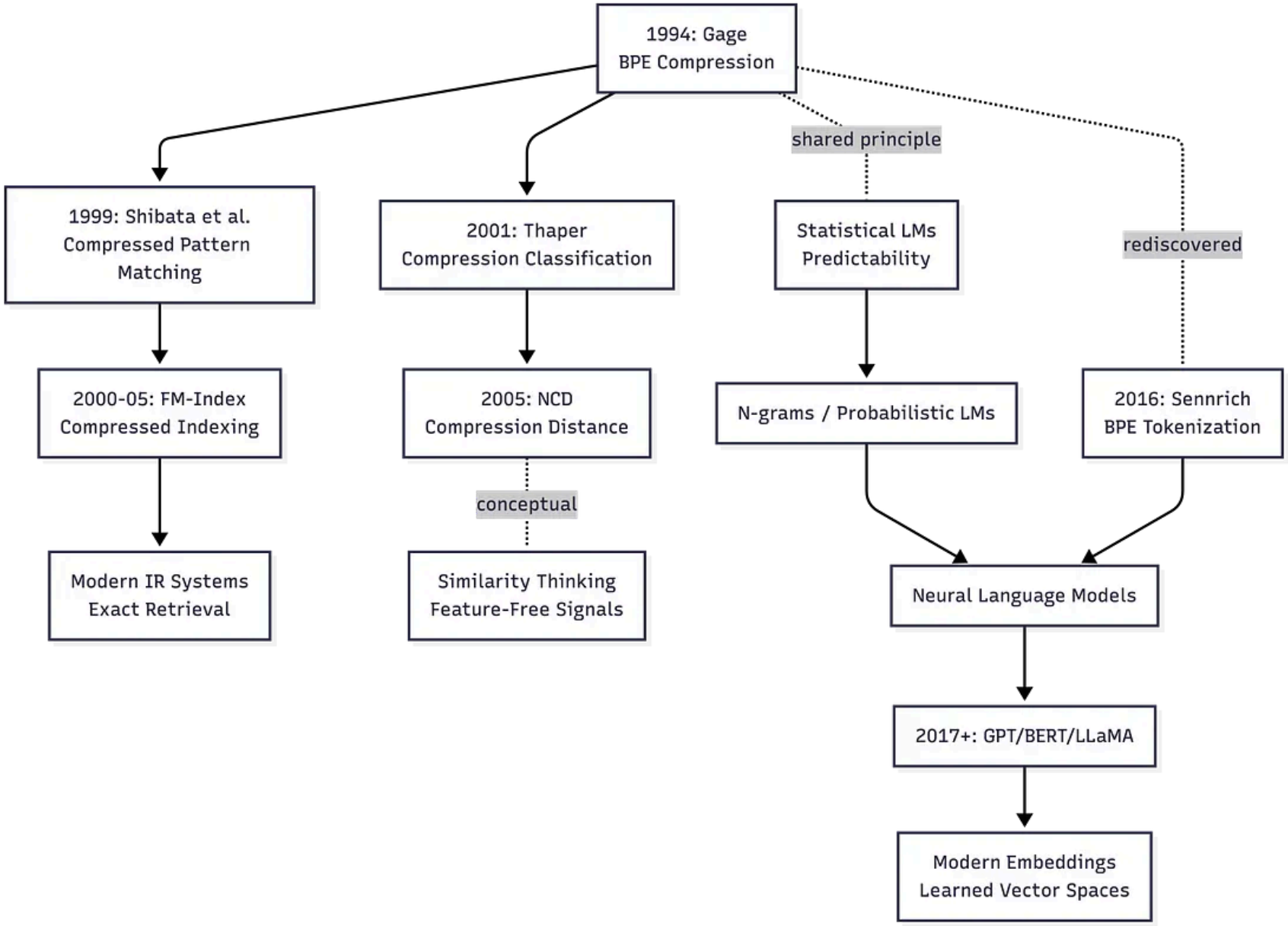
That Accelerates Pattern Matching" (*Shibata, Kida, Fukamachi, Takeda, Shinohara, Shinohara, and Arikawa*).

They discovered something surprising: searching BPE-compressed text was 1.6–1.9x *faster* than searching the original. The compression reduced data size; the structured byte codes accelerated pattern matching.

However, even this was still about compression and search, rather than language understanding.

The Parallel Bridges (1999–2005)

BPE wasn't the only compression technique finding new applications. During this period, two distinct "bridges" emerged from compression research into adjacent fields:



Innovation doesn't always travel the obvious path.

Figure: The two bridges from compression research (1994–2005) and their relationship to modern AI. Solid arrows show direct lineage; dotted arrows show conceptual influence or independent rediscovery. Note that NCD and embeddings share a philosophical goal (representation without manual features) but have no technical ancestry. The missing spine is language modeling: the insight that compression and prediction are mathematically equivalent.

Part 2: The NLP Problem (2000–2015)

While BPE was compressing files, natural language processing was hitting a wall.

The Vocabulary Explosion

Statistical NLP models require vocabularies, which are lists of known words. But how big should the vocabulary be?

Too small: Common words only. “The”, “a”, “is” work fine. But “iPhone”? “COVID-19”? “Beyoncé”? All become `<UNK>` (unknown).

Too large: Include every word ever written. The vocabulary explodes. English has 500,000+ dictionary words. Add proper nouns, technical terms, misspellings, and you're at millions. Each word needs an embedding vector (hundreds or thousands of parameters). Memory and computation blow up.

The field settled on an uncomfortable compromise: fixed vocabularies of 30,000–50,000 words, with everything else mapped to `<UNK>`.

This worked… poorly.

Imagine a sentiment analyzer encountering: "The new iPhone is amazign!"

Result: "The new `<UNK>` is `<UNK>`!"

The model has no idea what the user is talking about.

The Morphology Blindness

It got worse. Word-level vocabularies couldn't capture morphological structure.

"run", "runs", "running", "runner" each got their own vocabulary slot, with no shared representation. The model couldn't learn that they're related.

For agglutinative languages like Turkish or Finnish, where words compound endlessly, the problem was catastrophic. A single Turkish word might encode an entire English sentence. No reasonable vocabulary could cover the combinatorial space.

The Character-Level Detour

One radical solution: forget words entirely. Use characters.

Vocabulary size drops to ~100–200 (letters, digits, punctuation). No OOV problem ever. Morphology becomes learnable. “running” is just r-u-n-n-i-n-g, and the model can see “run” as a prefix.

But character-level models struggled:

1. **Sequence explosion**: A 500-word paragraph becomes 2,500+ characters
2. **Long-range dependencies**: The model must learn that c-a-t spells “cat” across huge distances
3. **Computational cost**: Attention is $O(n^2)$ in sequence length

The models worked for some tasks but couldn’t match word-level performance on semantic understanding. Learning spelling, morphology, syntax, and semantics all from raw characters was too much to ask.

The field needed a middle ground.

Part 3: The Breakthrough (2015–2016)

The Lost Bridge

Here's what we don't know: how did Rico Sennrich find BPE?

The 2016 paper cites "Gage (1994)" without providing any narrative. No "*I was reading old C Users Journal issues.*" No "*a colleague mentioned this obscure algorithm.*"

The bridge between a 1994 compression paper in a C programming magazine and 2015 neural machine translation research at Edinburgh is undocumented.

Was it systematic literature mining? Institutional knowledge passed through a department? A chance encounter in an old textbook? We don't know. The discovery story itself is lost to history.

This matters for more than historical curiosity.

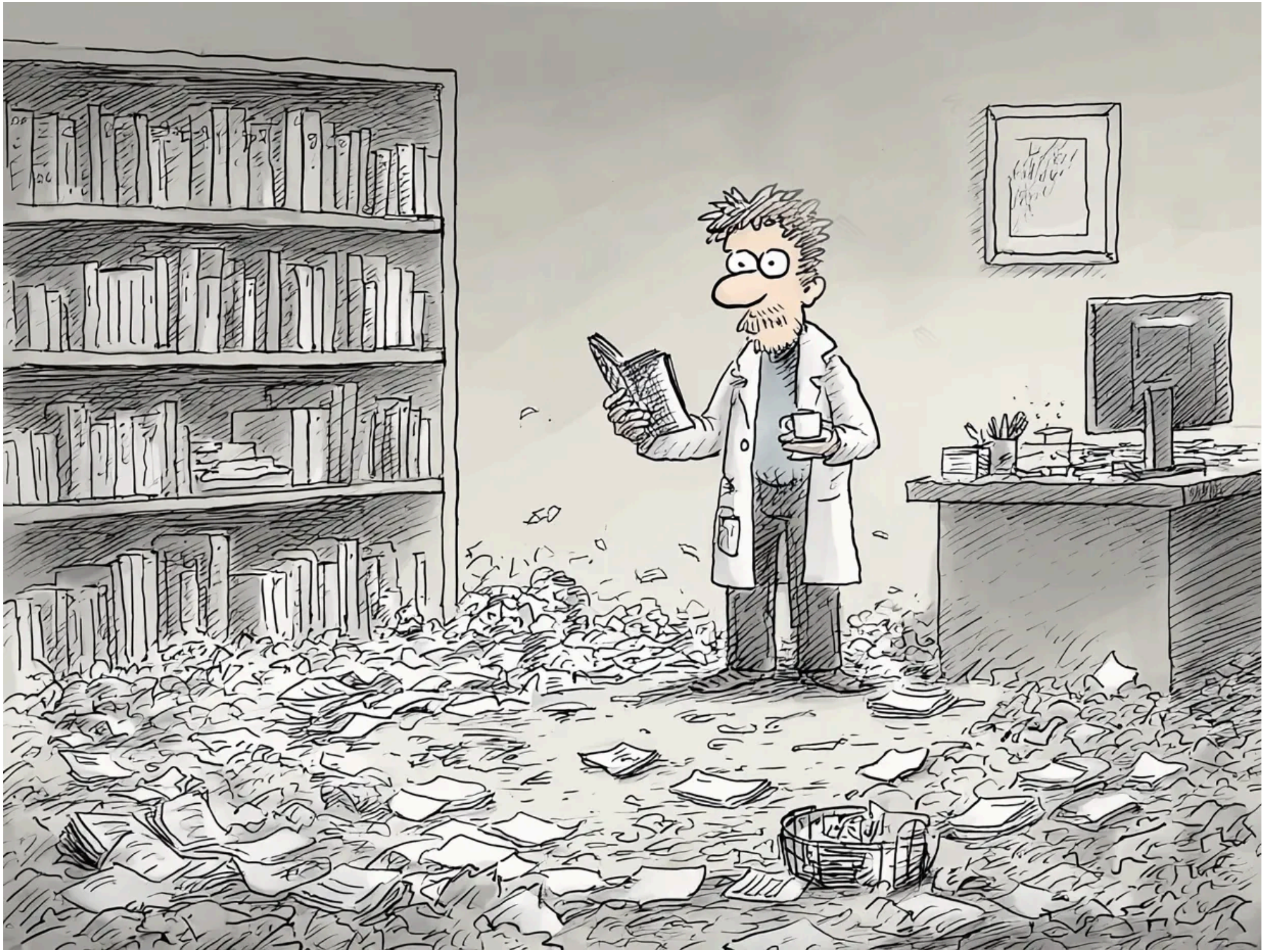
It's a reminder about academic contribution:

Philip Gage finished his work. He didn't abandon a half-baked idea. He wrote complete C code, published detailed explanations, and made his algorithm reproducible.

He cited his sources. Gage's paper carefully compared BPE to LZW and other algorithms, situating his work in context.

And he made it accessible. The C Users Journal wasn't Nature, but it was indexed and findable.

Twenty-two years later, someone found it. Research that might have seemed "merely practical" (a niche compression trick for a niche audience) became the foundation of modern AI.



Wait. This one's from 1994.

You never know who will need your work, or when.

Sennrich's Insight

In 2015, Rico Sennrich, Barry Haddow, and Alexandra Birch at the University of Edinburgh were working on neural machine translation. They faced the OOV problem acutely: technical terms, names, and rare words mangled their translations.

However they found BPE, they recognized something: what if you applied it to text?

Not bytes. Characters and character sequences.

The algorithm stayed identical:

```
1. Start with a vocabulary of individual characters
2. Count all adjacent pairs in the training corpus
3. Merge the most frequent pair into a new token
4. Repeat until you reach your target vocabulary size
```

But the output was something new: a **subword vocabulary**.

Common words like “the” would get merged all the way to single tokens. Rare words would stay fragmented, assembled from smaller pieces:

Word	Tokenization
"the"	["the"]
"running"	["run", "ning"]
"unhappiness"	["un", "happi", "ness"]
"Pneumonoultramicroscopicsilicovolcanoconiosis"	["P", "ne", "um", "ono", "ultra

The Paper

Their 2016 paper, “Neural Machine Translation of Rare Words with Subword Units,” demonstrated dramatic improvements on translation tasks. Rare words that previously became `<UNK>` were represented, even if the model had never seen them.

The key insight: **BPE automatically identifies morpheme-like units.**

The algorithm doesn’t know linguistics. It doesn’t know that “un-” is a negation prefix or “-ing” marks present participles. It just notices that these

character sequences appear frequently across many different words.

Frequency, in language, correlates with meaning.

This accidental alignment between compression and linguistics made BPE perfect for NLP.

Part 4: The Scaling Era (2017–2020)

Transformers Enter

In 2017, "Attention Is All You Need" introduced the Transformer architecture. Suddenly, sequence models could be parallelized. Training scaled to unprecedented sizes.

But Transformers still needed tokenization. And BPE (*simple, effective, scalable*) became the default choice.

GPT (2018): OpenAI's first GPT used BPE with a 40,000-token vocabulary.

BERT (2018): Google chose WordPiece (a BPE variant using PMI instead of raw frequency), but the core logic was the same.

GPT-2 (2019): Introduced Byte-Level BPE, starting from raw bytes (0–255) instead of characters. This guaranteed any Unicode text could be tokenized, including emoji, code, and corrupted data.

GPT-3 (2020): 175 billion parameters, still using BPE. The vocabulary grew to ~50,000 tokens, but the algorithm remained Philip Gage's 1994 invention.

Part 6: The Variants and Evolutions

Byte-Level BPE (GPT-2+)

Original BPE started with characters. Byte-level BPE starts with raw bytes (0–255).

Why? Characters are encoding-dependent. UTF-8 represents "é" differently than Latin-1. Byte-level BPE sidesteps this: it sees the raw byte sequence, whatever the encoding.

Benefits:

- Truly universal: any byte sequence is tokenizable
- No special handling for Unicode, emoji, or binary data
- Consistent behavior across encodings

The GPT-2/3/4 tokenizer uses byte-level BPE with ~50,000 merge operations.

WordPiece (BERT)

Google's variant changes the merge criterion. Instead of most frequent pair, merge the pair that maximizes:

```
score(a, b) = frequency(ab) / (frequency(a) × frequency(b))
```

This is Pointwise Mutual Information (PMI). It asks: "Do these tokens appear together more than chance predicts?"

High PMI = "These belong together" Low PMI = "They're just both common"

WordPiece uses `##` to mark continuation tokens:

```
"tokenization" → ["token", "##ization"]
```

Unigram (SentencePiece)

Unigram inverts the BPE approach:

1. Start with a huge vocabulary (all substrings up to some length)
2. Train a unigram language model
3. Remove tokens that hurt model likelihood least
4. Repeat until target vocabulary size

Unlike BPE, Unigram is **probabilistic**. The same text might have multiple valid tokenizations; the algorithm picks the most probable.

Part 7: Where BPE Lives Today

BPE (or close variants) powers everything.

```
Model        Tokenizer                      Vocabulary Size
.....................................................................
GPT-4        Byte-level BPE (tiktoken)      ~100,000
GPT-3.5      Byte-level BPE (tiktoken)      ~100,000
Claude 3     Byte-level BPE                 ~100,000
LLaMA 2      SentencePiece (BPE mode)       32,000
Mistral      SentencePiece (BPE mode)       32,000
BERT         WordPiece                      30,522
T5           SentencePiece (Unigram)        32,000
```

tiktoken: BPE at Scale

OpenAI's tiktoken is BPE implemented in Rust with Python bindings. Same algorithm, 3–6x faster than alternatives.

Why does speed matter? Every API call tokenizes. Every prompt. Every response. Millions per second globally. A 3x speedup saves enormous compute.

```python
import tiktoken

enc = tiktoken.encoding_for_model("gpt-4")
tokens = enc.encode("Hello, world!")
```

```
print(tokens)       # [9906, 11, 1917, 0]
print(len(tokens))  # 4
```

Timeline: The BPE Journey

Byte Pair Encoding is a compression algorithm.

It doesn't understand language. It understands frequency distributions. It doesn't know grammar. It knows which character sequences appear together often.

And yet: it works.

GPT-4 writes poetry, explains quantum mechanics, and debugs code, all while processing text through an algorithm designed to shrink files in 1994.

The gap between "frequency-based substring merging" and "understanding" turns out to be narrower than anyone expected.

Or perhaps understanding was never what we thought it was.

Philip Gage probably didn't anticipate that his little compression trick would become the reading apparatus of artificial intelligence. He was trying to

reduce the file size.

From compression curiosity to the foundation of modern AI.

```
Year   Event
.....................................................................
1994   Philip Gage publishes BPE for data compression
2015   Sennrich et al. apply BPE to NLP
2016   "Subword Units" paper establishes BPE for neural MT
2017   Transformers paper; BPE becomes standard tokenization
2018   GPT-1 uses BPE with 40K vocabulary
2018   BERT uses WordPiece (BPE variant)
2019   GPT-2 introduces byte-level BPE
2020   GPT-3: 175B parameters, same tokenization
2022   ChatGPT brings BPE to mainstream awareness
2023   tiktoken released; BPE optimized for production scale
2024   GPT-4, Claude, LLaMA all use BPE variants
```

Thirty years. Same algorithm.

Sometimes the most important innovations are accidents.

References

Core BPE Papers

1. Gage, P. (1994). “A New Algorithm for Data Compression.” The C Users Journal, Vol. 12, №2.

2. Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., & Arikawa, S. (1999). “Byte Pair Encoding: A Text Compression Scheme That Accelerates Pattern Matching.” Technical Report DOI-TR-CS-161, Kyushu University.

3. Sennrich, R., Haddow, B., & Birch, A. (2016). “Neural Machine Translation of Rare Words with Subword Units.” Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), Berlin.

Bridge A: Compression → Queryability

1. Ferragina, P. & Manzini, G. (2000). “Opportunistic Data Structures with Applications.” FOCS 2000. (FM-Index introduction)

2. Navarro, G. & Mäkinen, V. (2007). “Compressed Full-Text Indexes.” ACM Computing Surveys.

Bridge B: Compression → Similarity

Master’s Thesis.

2. Cilibrasi, R. & Vitányi, P. (2005). “Clustering by Compression.” IEEE Transactions on Information Theory.

3. Marton, Y., Wu, N., & Hellerstein, L. (2005). “On Compression-Based Text Classification.” ECIR 2005.

Transformers and Modern AI

1. Vaswani, A., et al. (2017). “Attention Is All You Need.” NeurIPS.

2. Radford, A., et al. (2018). “Improving Language Understanding by Generative Pre-Training.” OpenAI.

3. Radford, A., et al. (2019). “Language Models are Unsupervised Multitask Learners.” OpenAI.

4. OpenAI. (2023). tiktoken. GitHub.

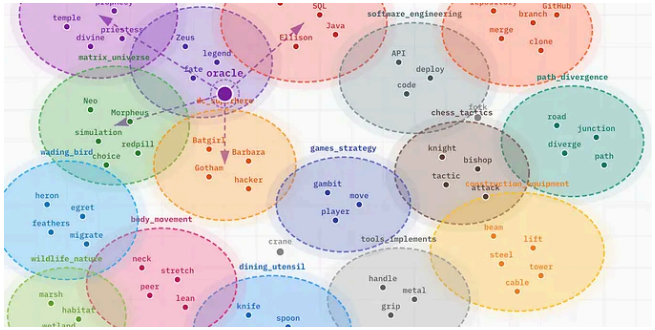
Artificial Intelligence | Naturallanguageprocessing | Transformers | Data Compression | Tokenization

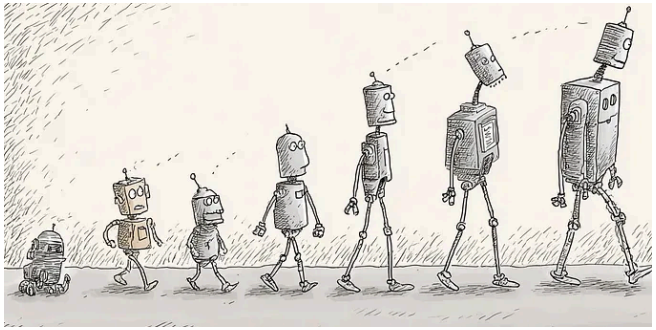
Written by Craig Trim

26 followers · 6 following

Follow

More from Craig Trim





Craig Trim

Words Learning the Company They Keep

Firth said you know a word by the company it keeps. Early embeddings disagreed, giving...

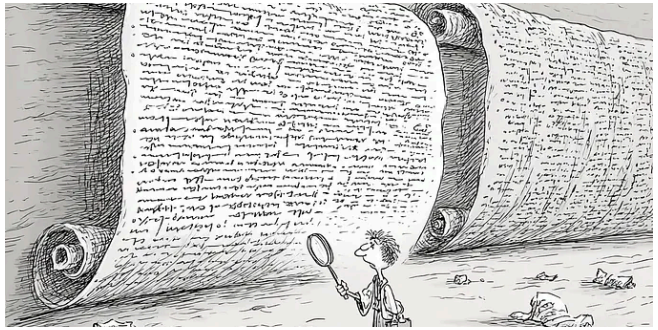
Jan 14 107 3

Craig Trim

A Brief History of Text Generation

From Shannon's hand-picked letters to modern LLMs. The real outputs from ELIZA,...

Jan 10 193 3

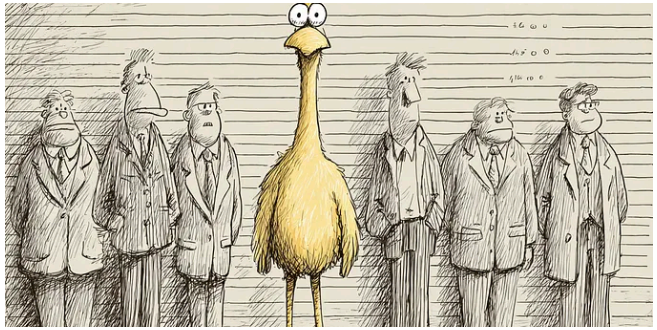


Craig Trim

The Invisible Boundaries of AI Conversation

Every LLM operates within a fixed-size window of attention. What happens at the...

Jan 15 5



Craig Trim

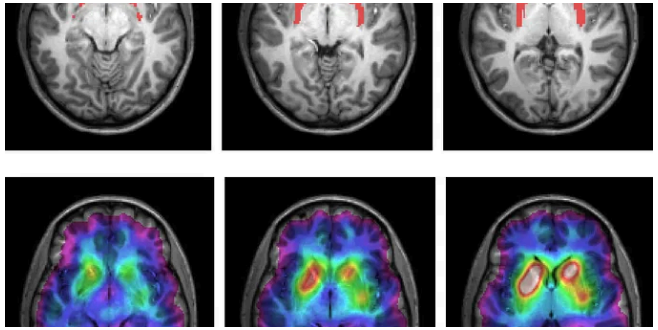
The Hidden Geography of Language

How words turn into coordinates, and why “king minus man plus woman” equals...

Jan 13 6

See all from Craig Trim

Recommended from Medium

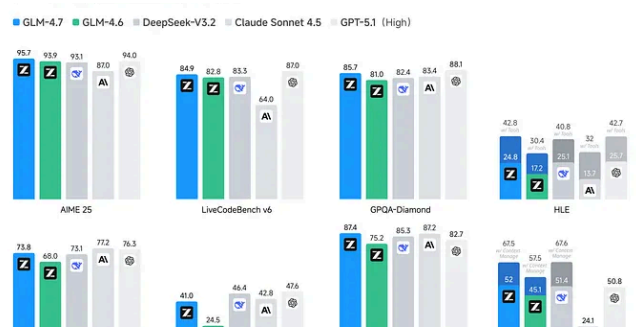


In Write A Catalyst by Dr. Patricia Schmidt

As a Neuroscientist, I Quit These 5 Morning Habits That Destroy You...

Most people do #1 within 10 minutes of waking (and it sabotages your entire day)

Jan 14 20K 345

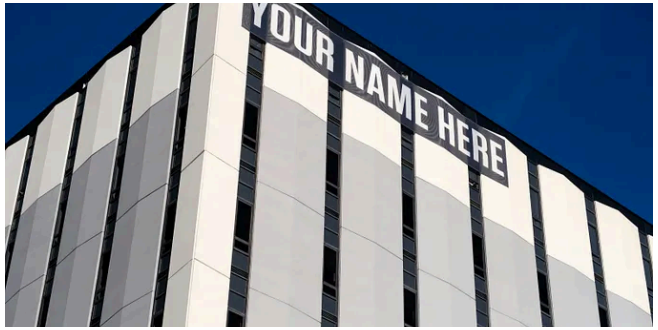


Agent Native

Local LLMs That Can Replace Claude Code

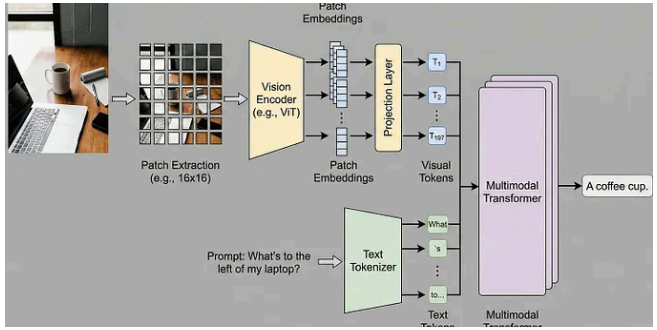
Small team of engineers can easily burn >$2K/mo on Anthropic's Claude Code...

Jan 20 209 7



DamenC

Fine-Tuning BERT for Named Entity Recognition: A Step-by-Step Guide



Togo AI Labs

Understanding Visual Tokenization and the Gap Between Pixels and...

How Vision-Language Models Actually 'See'

Named Entity Recognition (NER) is a fundamental task in Natural Language...

Oct 3, 2025 20

6d ago 137 2

Sebastian Buzdugan

A 30B Qwen model runs in real time on a Raspberry Pi, here's wh...

Click here to read this article for free...

Jan 7 642 9

Barack Obama

A Wake-Up Call for Every American

The killing of Alex Pretti is a heartbreaking tragedy. It should also be a wake-up call to...

3d ago 54K 743

See more recommendations