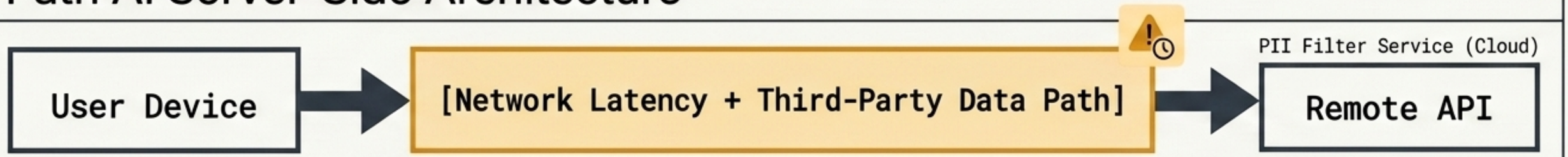
Why Client-Side PII Filtering?

The privacy and latency costs of sending raw text to a remote filter versus keeping it on the device.

Path A: Server-Side Architecture



The Constraint: Server-side PII filters require every chunk of user text to leave the device. This introduces a network round-trip, latency, and a third-party service into the raw data path.

Path B: **textsift Client-Side** Architecture

The Payoff: Zero network calls during inference. The raw text (containing emails, SSNs, credit cards) stays strictly localized to the device's memory.

The Model & The Canonical Decoder

Naive argmax fails on BIOES structures; textsift implements the required Viterbi calibration.

The Target Model

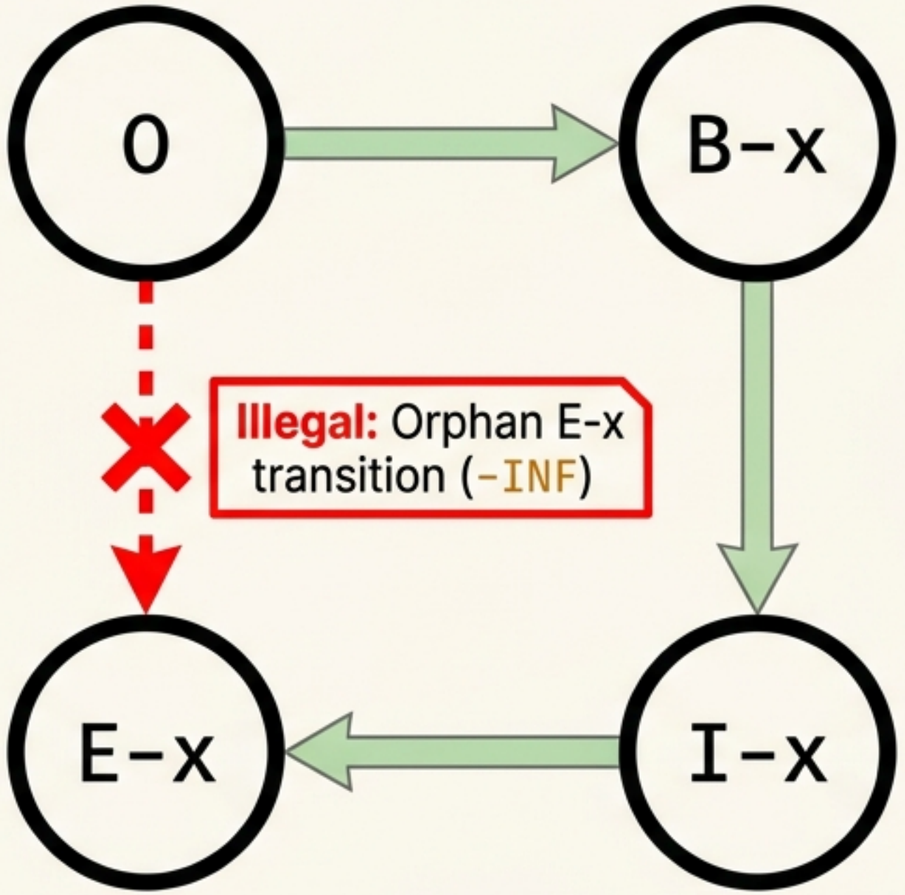
`openai/privacy-filter` (8 PII labels, 33 total output classes). We ship the `model_q4f16.onnx` artifact (`int4` weights, `fp16` activations).

The Mathematical Gap

Standard `argmax` is the wrong decoder. It might select a standalone `E-x` (End of span) token without a preceding `B-x` (Begin) token, breaking structural BIOES rules.

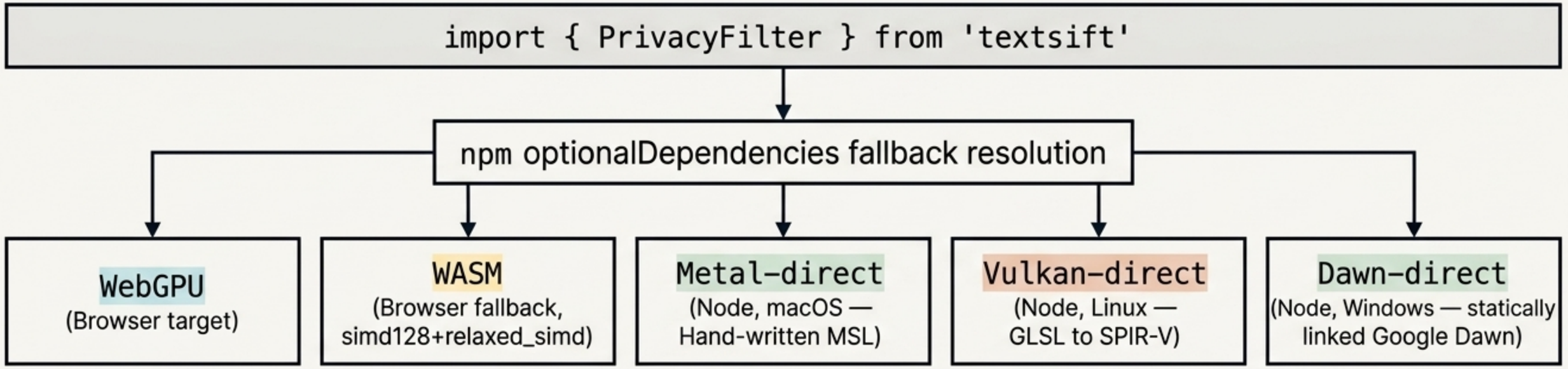
The textsift Engine

The upstream HuggingFace model ships `viterbi_calibration.json` containing six transition biases. The engine mirrors this exact calibration, executing standard forward-DP Viterbi over the BIOES tag space, mathematically setting illegal transitions to `-INF`.



Five Backends, One API

The same TypeScript surface routes to the appropriate compute layer at runtime.



Single Target

Re-implements the exact same model graph against the same `model_q4f16.onnx` file across all compute paths.

Compute Core

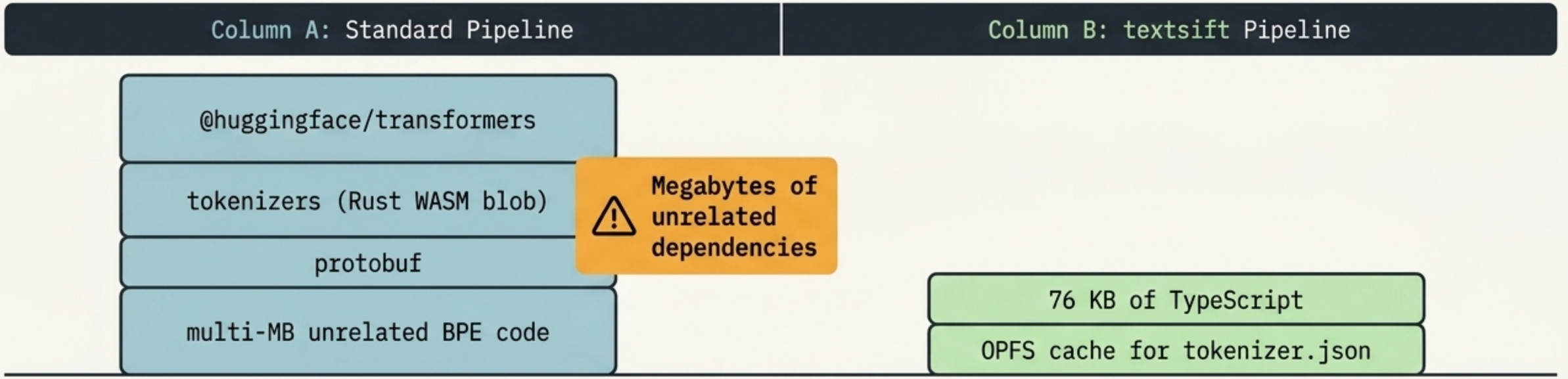
Features custom kernels (e.g., `matmul_int4_fp16`) tuned for specific graphics layers. `WGSL` acts as the canonical source translated to native OS targets.

Graceful Degradation

If a native `.node` binary fails to load (missing GPU or Vulkan loader), execution silently falls through to the pure WASM backend.

The Tokenizer Is Its Own Win

A pure-TypeScript implementation of the o200k BPE avoids the massive dependency tree of standard pipelines.



The Ecosystem Bloat

For developers running purely in the browser, dragging in standard tokenizer wrappers pulls down megabytes of dependencies and native/WASM bridges entirely unrelated to basic tokenization. Standard tools hit the Cache API (which silently rejects 770 MB payloads with QuotaExceededError in some browsers).

The textsift Approach

A hand-rolled pure TypeScript implementation of the o200k BPE. It is strictly 76 KB gzipped over the wire, utilizing zero native code and zero protobuf dependencies. It bypasses the Cache API by using OPFS directly to persist model weights and tokenizer.json.

The Linux Story: Filling the Hardware Gap

A hand-tuned Vulkan backend brings functional inference to non-NVIDIA Linux machines.

Measured on Intel Iris Xe at T=32

The Hardware Gap

PyTorch CPU is slow, ONNX Runtime Node falls back to CPU, and Mesa Vulkan typically sits entirely unused on Intel iGPU / AMD APU architectures.

The Measured Reality

On an Intel Iris Xe at T=32, execution via ONNX Runtime Node CPU takes ~785 ms. The identical task via textsift Vulkan-direct measures ~25 ms.

Ceiling Analysis

The textsift execution achieves ~50% of the theoretical memory-bandwidth ceiling for this specific hardware, driven entirely by the custom `matmul_int4_fp16` compute kernels.

What I Learned: The Windows Port Slog

Making the Windows native binding work required ten commits of purely diagnostic troubleshooting.

Terminal Error	Engineered Fix
`ERR: missing headers`	`Fetched node-v<ver>-headers.tar.gz`
`ERR: duplicate symbol`	`Targeted MSVC ABI explicitly`
`ERR: undefined NAPI_AUTO_LENGTH`	`Bypassed Zig translate-c SIZE_MAX bug`
`ERR: undefined napi_*`	`Fetched node.lib Windows import library`
`ERR: exit 139`	`Explicitly linked UCRT (-lucrt)`

The Reality of Porting

Writing the compute code was straightforward; navigating C++ toolchain idiosyncrasies was the actual engineering tax. Node distributions on Windows don't ship **C headers** by default, and Zig's default linker paths required manual parsing of the MSVC `LIB` environment variable.

Silent Failures

CI cycles took ~25 minutes each due to Dawn cold builds. Diagnosing issues like `PowerShell` **silently** swallowing child process `stderr` **during** `dlopen` failures severely compounded the iteration time.

Artifacts & Conformance

Verified byte-for-byte span matching with the reference implementation and available via standard registries.

10/10 Conformance (v0.1.0)

*Span-equivalent to the canonical ONNX reference.

Rigorous Verification: CI runs exact byte-for-byte span matching against the canonical ONNX reference (matching model_q4f16.onnx and Viterbi+biases decoding). At v0.1.0, the test reports exactly 10/10 inputs span-equivalent.

License:	Apache 2.0
npm:	textsift
Source	https://github.com/teamchong/textsift
Docs:	https://teamchong.github.io/textsift/
Playground	https://teamchong.github.io/textsift/playground/
Faker mode demo	https://teamchong.github.io/textsift/playground-faker/
Model	https://huggingface.co/openai/privacy-filter