Compression Levels

`typst-bake` compresses all embedded resources using **zstd** to minimize binary size.

Compression Caching

High compression levels produce smaller binaries with little impact on decompression speed, but they take significantly longer to compress. Recompressing unchanged files on every build would be wasteful, so `typst-bake` **caches compressed outputs on disk**. The cache key is derived from:

- The **BLAKE3 hash** of the file contents
- The **compression level** in use

If both match, the previously compressed result is reused — no recompression occurs.

```
target/typst-bake-cache/{package-name}/
├── a1b2c3d4e5f6...zst   (cached compressed blob)
├── f7e8d9c0b1a2...zst
└── ...
```

This makes **incremental builds fast** even at high compression levels.

Custom Compression Level

The default compression level is **19** (zstd supports levels 1 through 22).

You can set a custom level in `Cargo.toml` under `[package.metadata.typst-bake]`:

```toml
[package.metadata.typst-bake]
compression-level = 22
```

Alternatively, override the level at build time with an environment variable:

```Shell
TYPST_BAKE_COMPRESSION_LEVEL=3 cargo build
```

The environment variable takes precedence over the `Cargo.toml` setting.

Higher levels produce smaller binaries but compress more slowly.

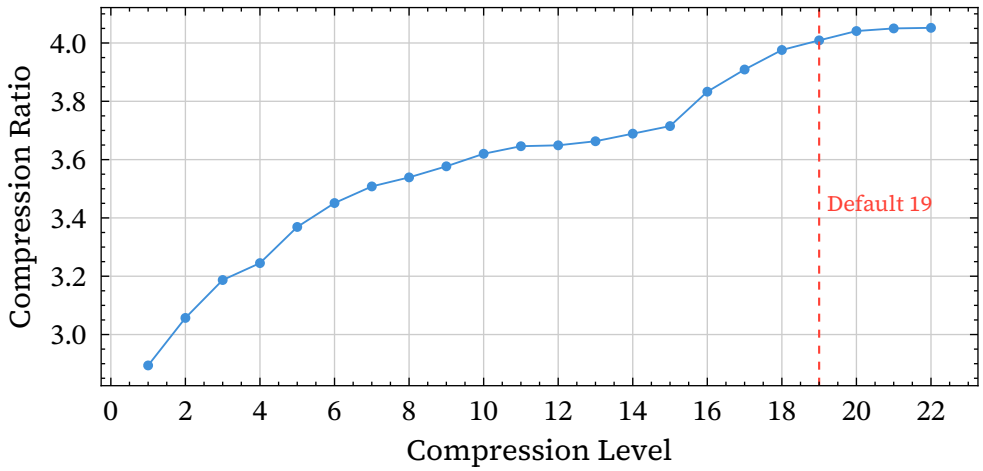
Running Benchmarks

The next page shows benchmark results. To run them and generate this PDF on your own machine:

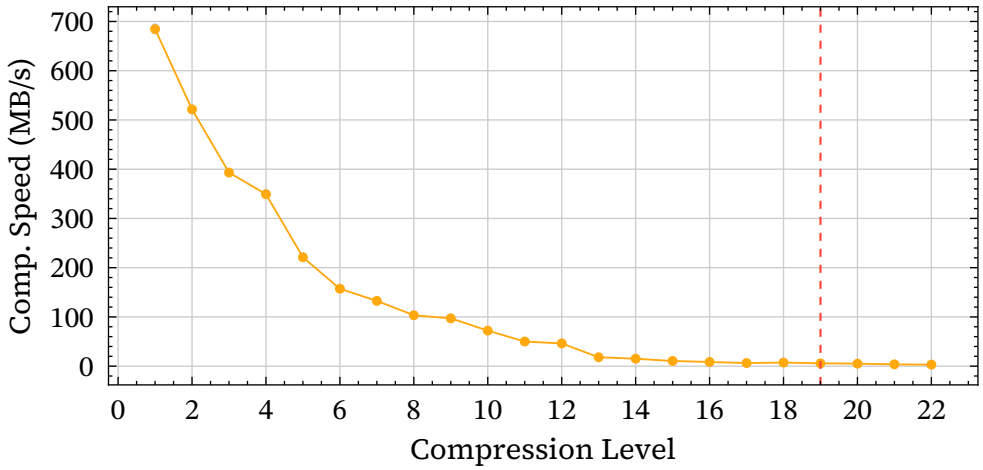
```Shell
cargo test -p example-compression-levels --release --test benchmark -- --ignored --nocapture --test-threads=1
cargo run -p example-compression-levels
```

Benchmark: zstd Compression Levels

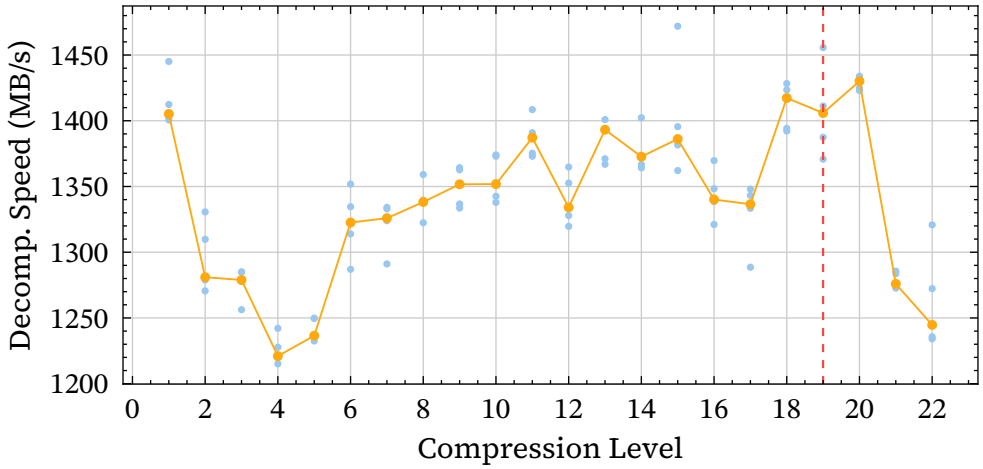
Compression Ratio vs Level



Compression Speed vs Level



Decompression Speed vs Level



Measured on Apple M4 Max --- Silesia corpus (202.1 MB), 5 iterations, zstd 0.13