# SMS Compression using Zstandard

Final review

**Under the Guidance of:** Dr. Balaji N

**Team Members:**

- 22MID0011 – Mythili Anukeerthana K S
- 22MID0105 – Varshitha V
- 21MIC0113 – Lekkala Tejaswi

**Department of Computer Science & Engineering,VIT VELLORE.**

# ALGORITHM WORKING (ZSTANDARD)

- Zstandard (Zstd) is a fast compression algorithm developed by Facebook.
- It balances speed and compression ratio efficiently.
- Uses entropy coding and dictionary-based compression internally.

# Steps with Example

1.Input Reading
The algorithm takes the input text or data as a byte stream.
It divides it into fixed-size blocks (usually 128 KB).

2.Dictionary Generation (Optional)
For small repetitive data (like SMS), Zstd can use a predefined dictionary.
This dictionary stores common words or patterns to improve compression ratio.

3.Entropy Coding (Main Compression)
Each block is processed independently.
Zstd uses a combination of:
LZ77 (Dictionary Matching): Finds repeated sequences and replaces them with references.
Finite State Entropy (FSE): Efficiently encodes the remaining symbols using probabilistic models.

**4.Literal and Match Encoding**

•The algorithm separates literals (unique bytes) and matches (repeated patterns).

•It then encodes:

- •Literal lengths
- •Match lengths
- •Offsets for repeated patterns

**5.Compression of Metadata**

•Metadata like block sizes and headers are also compressed to reduce overhead.

**6.Block Output & Framing**

•All compressed blocks and headers are assembled into a final frame format.

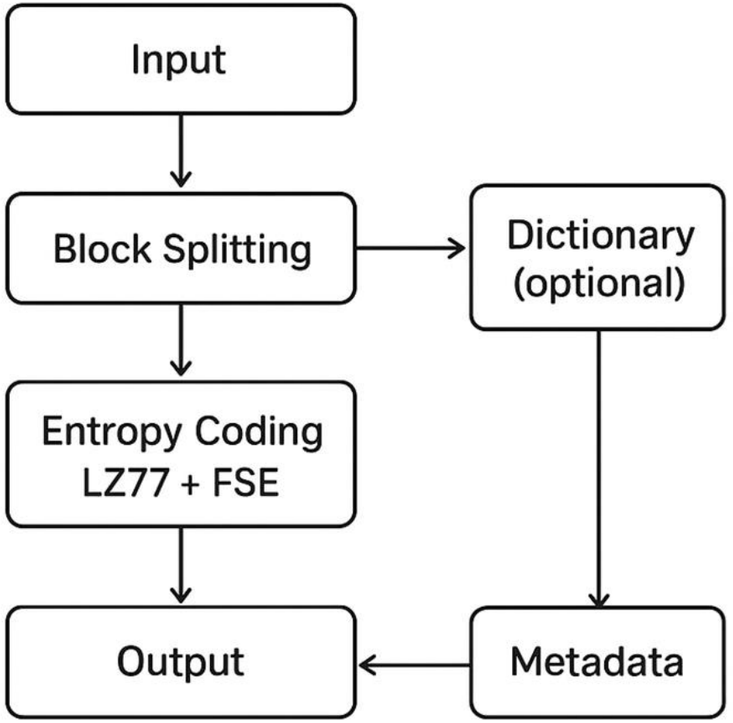
•This frame can later be decompressed independently.

**7.Decompression**

•Reverse of compression:

- •Read frame → Decode FSE → Apply LZ77 matches → Reconstruct original data.

# STEPS IN ZSTANDARD (ZSTD) COMPRESSION ALGORITHM



# **Example**(Simple SMS Text)

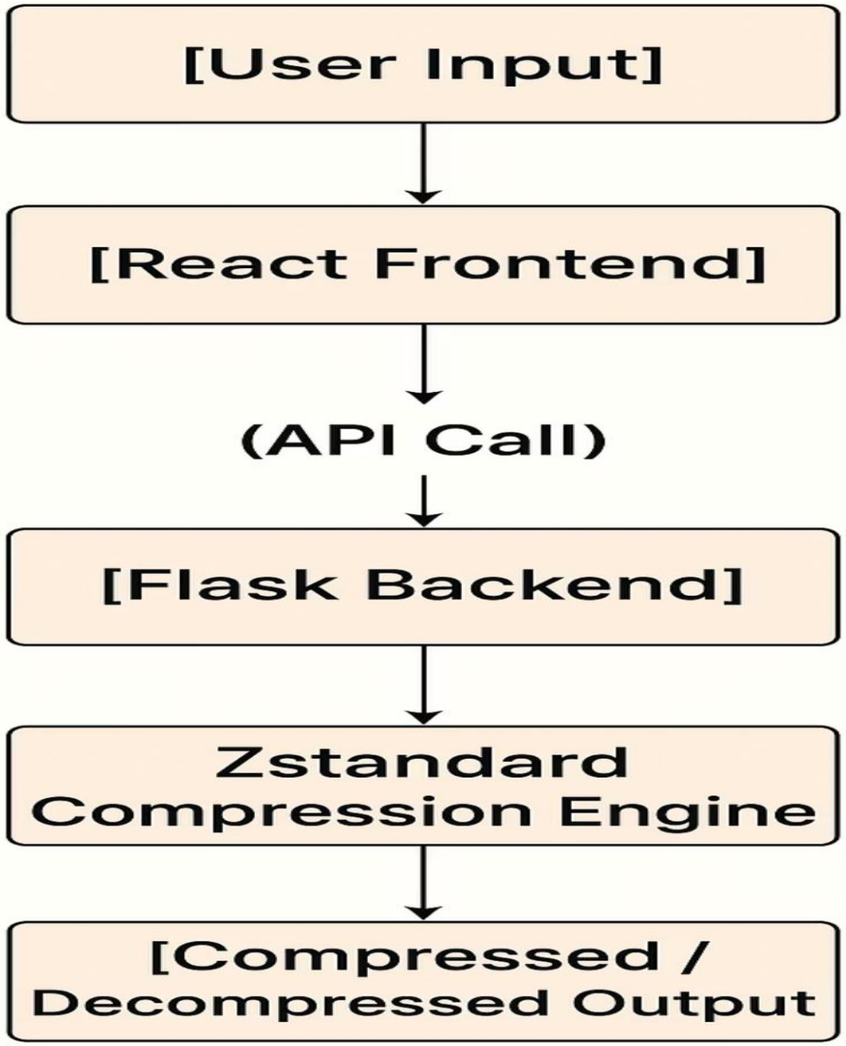
**Input:** "HELLO HELLO HELLO"
LZ77 detects “HELLO” repetition → stores once, replaces next ones with references.
FSE encodes remaining characters efficiently.
**Output:** A shorter binary (compressed) form that can be decompressed back to "HELLO HELLO HELLO" perfectly.

# System Architecture

**Description:** The system integrates a Flask backend with a React frontend, enabling real-time text compression and decompression using the **Zstandard algorithm**.

Components:

- **Frontend:** React JS (UI for user input, compression ratio display)
- **Backend:** Flask API (handles compression & decompression requests)
- **Algorithm:** Zstandard (Zstd) — for efficient, lossless compression
- **Data Flow:** User → Frontend → Backend → Algorithm → Frontend → Result


[User Input]
[React Frontend]
(API Call)
[Flask Backend]
Zstandard
Compression Engine
[Compressed /
Decompressed Output

# Workflow / Data Flow Diagram

01

## User Input

User enters SMS text on the frontend.

02

## Text to Backend

The text is sent to Flask backend via POST request.

03

## Compression

Flask processes the text and applies Zstandard compression.

04

## Metrics Sent Back

The compressed text and metrics (size, time, ratio) are sent back.

05

## Frontend Display

Frontend displays compressed text and visualization.

06

## Decompression

On clicking “Decompress,” the original text is restored.

# Technology Stack

## Frontend

- React JS
- HTML5
- CSS3
- JavaScript

## Styling Framework

- Tailwind CSS / Custom CSS

## Backend

- Python (Flask)

## Compression Library

- Zstandard (Zstd)

## Communication

- REST API (JSON)

## Visualization

- Framer Motion (animations)

## Environment

- Node JS, npm

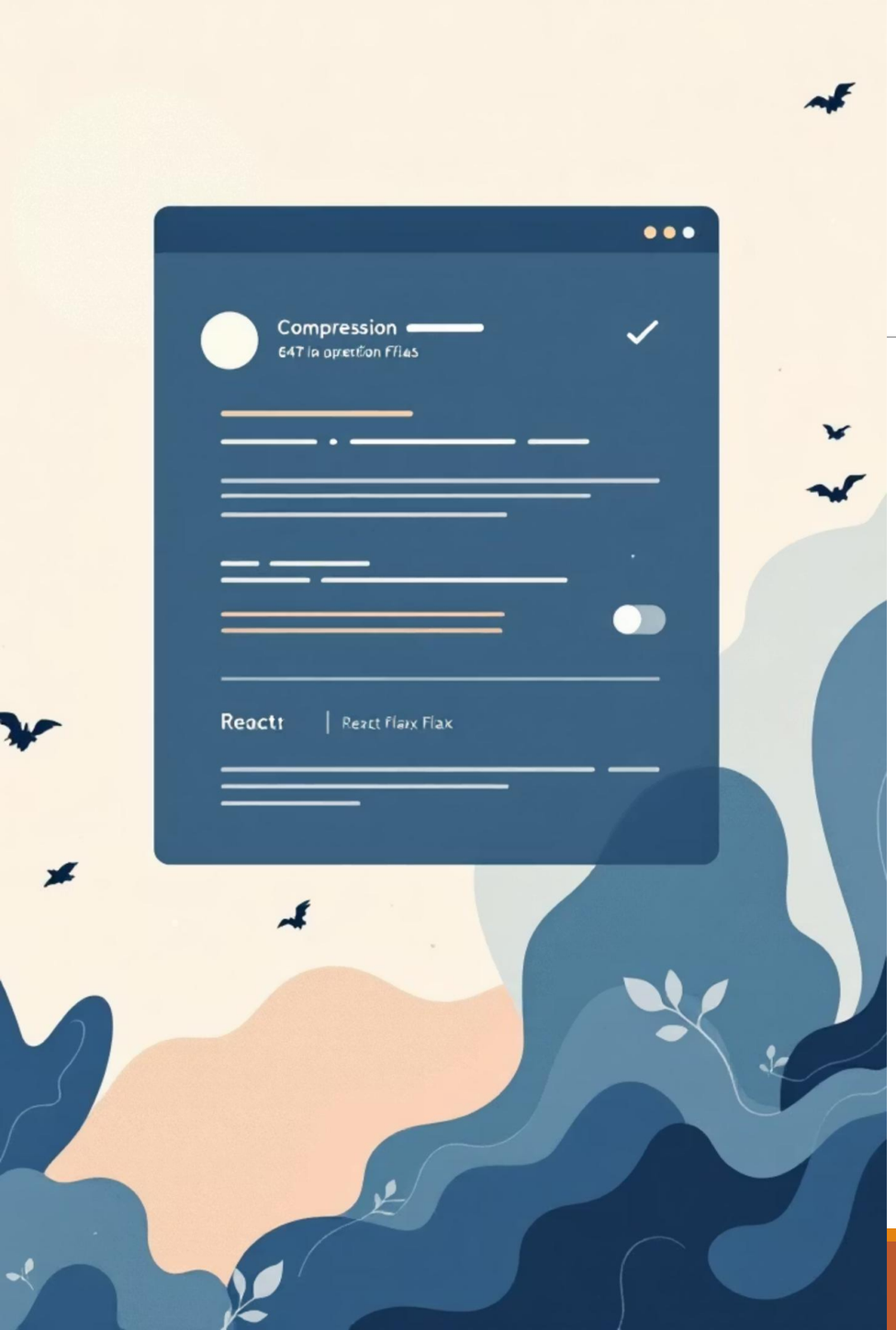
# Implementation (Backend + Frontend)

## Backend (Flask):

- Receives POST requests /compress & /decompress
- Uses Zstd for compression/decompression
- Calculates metrics (original size, compressed size, time)
- Returns JSON responses to frontend

## Frontend (React):

- Provides textboxes for input and output
- Displays compression ratio and visual indicator
- Handles API calls to Flask and renders results




# Results

## Observations:

- Compression Ratio ≈ 60 – 80 % (depending on text size)
- High speed (< 5 ms per operation for short SMS text)
- Accurate lossless decompression

# Future Scope

- Integrate with mobile SMS gateways for real deployment.
- Implement batch compression for multiple messages.
- Compare Zstandard with other algorithms (LZMA, Brotli).
- Add data analytics on compression performance.
- Extend for multilingual and emoji support.

# Conclusion

Efficient **lossless compression** achieved for SMS text.

Optimized transmission and storage through Zstandard.

Successfully integrated **Flask + React** for real-time demo.

Demonstrated practical implementation of Zstd algorithm in a modern web app.

# THANK YOU