Ca' Foscari
University
of Venice

Master Degree
in Computer Science, Cybersecurity

Final Thesis

Encrypted Data Transmission via Audio Steganography over Messaging Applications

Supervisor

Ch.mo prof. Filippo Bergamasco

Graduand

Francesco Forcellato

Matriculation number 875290

Academic Year

2025 / 2026

1 Abstract

This thesis investigates the feasibility of covert data transmission through audio channels in instant messaging applications. We propose a system that enables secure communication by combining encryption with audio steganography, allowing data to be embedded within audio signals that resemble ordinary voice messages. The proposed pipeline converts encrypted data into audio waveforms using Differential Phase Shift Keying (DPSK) encoding. The resulting signal is further obfuscated through the addition of white noise and voice recordings, producing audio that is similar to typical user recordings. This audio is then transmitted via standard messaging functionalities, such as voice message recordings in WhatsApp Web. On the receiver side, the transmitted audio is processed to recover the embedded signal, which is subsequently decoded and decrypted. The system is designed to operate using commonly available tools, requiring only a computer and an instant messaging application. The thesis explores the challenges associated with this approach, including robustness against platform-specific transformations such as Opus compression, and evaluates the overall feasibility of the method. This work serves as a proof of concept and highlights potential implications for privacy, data security and communication control.

Contents

List of Figures

2 Statement on the Use of Artificial Intelligence

In accordance with the academic integrity guidelines established by Ca' Foscari University of Venice regarding the use of generative Artificial Intelligence (AI), this section details the specific tools, scopes, and methodologies applied during the research, development, and drafting phases of this thesis.

Software Development and Code Assistance During the implementation phase of the accompanying software project, GitHub Copilot (integrated within the Visual Studio Code environment) was utilized as an AI-powered development assistant. This tool was primarily leveraged to accelerate boilerplate code generation, explore specific library patterns, and align the codebase with industry-standard development methodologies and architectural best practices. The foundational logic, algorithmic design, and system architecture were conceived and implemented solely by the author.

Language Enhancement and Text Editing To ensure a high standard of academic writing and to refine the clarity, flow, and grammatical accuracy of the English text, large language models were employed as advanced editing tools. The specific used model are Google Gemini Flash 3.0 and Anthropic Claude Opus 4.6. The text generation process was strictly iterative: original drafts written by the author were analyzed by the models to improve stylistic coherence and rectify linguistic nuances. Additionally, these conversational models were used for exploratory research purposes, specifically to assist in querying academic citation networks and discovering relevant literature.

Academic Integrity and Human Accountability To preserve the scientific validity and originality of this master's thesis, a rigorous validation protocol was maintained throughout the entire workflow. The author confirms the following boundaries of AI involvement:

- **Ultimate Accountability:** Every code function, architectural decision, literature citation, and theoretical argument was designed by the author, critically reviewed, and manually verified for accuracy. The AI suggestions were treated strictly as preliminary recommendations subject to human double-checking.
- **No Source Material Fabrication:** AI tools were not used to synthesize data, fabricate experimental results, or generate primary text without original human drafts.
- **Exclusion of Graphic Generation:** No generative AI tools or text-to-image models were involved in the creation of figures, diagrams, or visual assets; all illustrations included in this document were crafted manually by the author.

Consequently, full responsibility for the content, interpretations, and conclusions presented in this thesis rests entirely with the author.

3 Introduction

In this thesis, we investigate the feasibility of a covert data-exchange system built upon publicly available instant messaging (IM) applications. Because popular platforms like WhatsApp (2026) are closed-source, we cannot easily verify their internal security mechanisms or claims. Furthermore, we recognize that the underlying operating system or hardware may present additional attack vectors. To address these concerns, we wanted to rely on a trusted external device over which we maintain full administrative control by building it from the ground up, while still utilizing commonly available communication services.

We created a system that encodes data into audio signals and transmits them using the standard *voice message* feature. On the receiving end, the system saves and decodes the message to retrieve the original data. By taking this approach, we leverage widely-adopted infrastructure to create an independent communication channel.

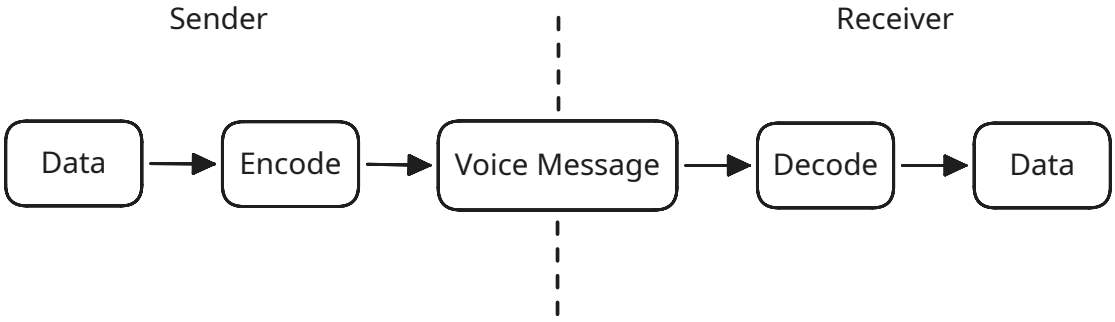


Figure 1: Simple encoding-decoding pipeline

3.1 Methodological Justification and Steganographic Constraints

To establish a robust and realistic proof of concept, the implementation must utilize native voice recording functionalities rather than generic file transfers. Transporting the generated audio as an attached file (e.g., a standard `.wav` or `.mp3` document)

bypasses the proprietary, lossy audio compression algorithms natively applied by the application.

According to Jamil (1999), while cryptography focuses on obfuscating the contents of a message to render it unreadable to unauthorized entities, steganography represents "the art and science of hiding information in plain sight". The core objective is not merely to protect the data through encryption, but to conceal the very existence of the communication channel itself. Therefore, transmitting a raw file container introduces anomalous traffic patterns that are significantly more conspicuous to anomaly detection systems than an innocuous voice note. Consequently, we developed an integration workflow that interfaces directly with standard application features, ensuring our data streams are subjected to identical operational constraints, such as aggressive codec compression, as legitimate user traffic. This optimized operational workflow is illustrated in Figure 3.

Crucially, maximizing user convenience is not a primary objective of this proof of concept. Because the architecture deliberately decouples the encoding and decoding processes from the smartphone onto an isolated, trusted hardware platform, user experience is intentionally traded off to preserve strict security isolation and cryptographic root-of-trust boundaries (Maene et al., 2018).

4 Project Architecture and Evolution

The development of this project involved iterating through several architectural approaches. In this section, we analyze the technical challenges encountered and the subsequent refinements made to the system.

4.1 Initial Approach: Virtual Headset and Smartphone Interface

To facilitate data transmission via WhatsApp (2026), our original setup was designed to leverage the standard voice messaging interface. The workflow was structured as follows:

- **The Sender Workflow:**

    1. **Data Generation:** We generate the payload on a trusted hardware platform, such as a Raspberry Pi Ltd (2026).
    2. **Encoding:** The device encodes the raw data into acoustic waveforms, which are then stored as a high-fidelity audio file.
    3. **Hardware Interfacing:** We pair the trusted device with the sender's smartphone, forcing the trusted device to emulate a Bluetooth headset (specifically utilizing the Hands-Free Profile (HFP)).
    4. **Transmission Initialization:** On the sender smartphone, we open the target chat and initiate the voice recording function.
    5. **Signal Injection:** We play the audio file generated in Step 2 directly through the virtual headset's microphone channel.
    6. **Finalization:** Once the playback is complete, we stop the recording and send the audio message to the receiver.

- **The Receiver Workflow:**

  1. **Hardware Interfacing:** We connect the receiver's smartphone to a trusted decoding device, which similarly emulates an audio headset interface.
  2. **Signal Capture:** We play back the received voice message on the smartphone, which routes the audio stream directly into the trusted device's virtual microphone interface. The trusted device records this incoming signal and saves it as an audio file.
  3. **Decoding and Extraction:** The trusted device processes the captured audio waveform, decodes the signal, and presents the original payload to the user.

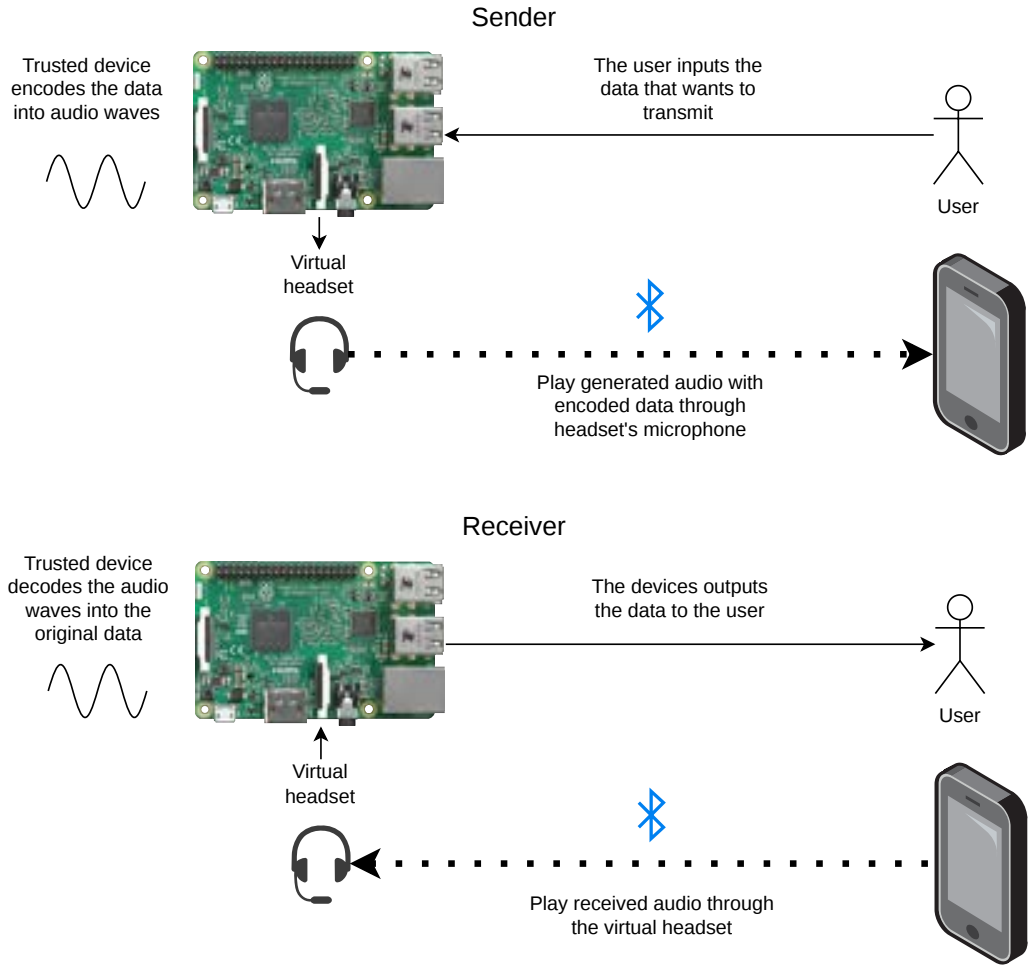


Figure 2: Data exchange workflow. Raspberry Pi image taken from wikipedia

4.1.1 Technical Bottlenecks: Bluetooth Role Asymmetry

Despite successfully establishing a connection between the smartphone and the trusted device using oFono and the BlueZ stack, we encountered a fundamental limitation regarding Bluetooth profile roles. In the Bluetooth Hands-Free Profile (HFP) architecture, roles are strictly partitioned between the Audio Gateway (AG), typically the smartphone, and the Hands-Free (HF) unit, typically the headset.

While we could register the trusted device as an HF unit, establishing a stable, bidirectional audio pipeline was obstructed by "role asymmetry" issues. Most desk-

top and embedded operating systems are optimized to act as Gateways rather than Peripherals. Specifically, we observed the following:

- **Directional and Profile Constraints:** Classic Bluetooth audio architectures dictate a strict, mutually exclusive split between high-fidelity media playback via the Advanced Audio Distribution Profile (A2DP) and bidirectional voice routing via HFP (Bluetooth Special Interest Group, 2020). When a client application attempts to access a headset's microphone, the system must terminate the active A2DP channel and negotiate a Synchronous Connection-Oriented (SCO) link under HFP (Bluetooth Special Interest Group, 2020). This transition introduces significant signal degradation and latency. Crucially, the HFP stack is typically reserved for duplex voice calls and must be explicitly invoked by the application (Android Open Source Project, 2026). Because the "voice message" feature in IM applications is treated as a high-fidelity, non-duplex media recording event, these applications prioritize the device's internal physical microphone to ensure recording quality, bypassing the HFP-linked virtual microphone entirely.

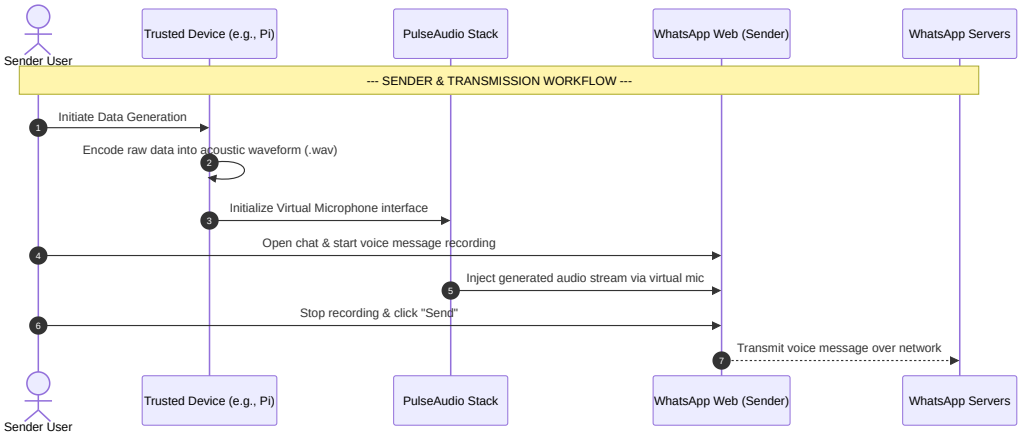
- **Peripheral Emulation Failure:** Even when the PC or Raspberry Pi Ltd (2026) successfully emulated these device classes, the lack of a "pure Bluetooth microphone" abstraction in standard Bluetooth stacks prevented the smartphone from recognizing the injected audio as a valid system-wide input source. This explains why commercially available wireless microphones often utilize proprietary physical adapters connected to the smartphone's data port (USB-C or Lightning), ensuring the device is enumerated as a standard physical audio interface rather than a Bluetooth peripheral.

Due to these architectural constraints, we concluded that a purely software-defined virtual Bluetooth headset or microphone was unfeasible for this use case.

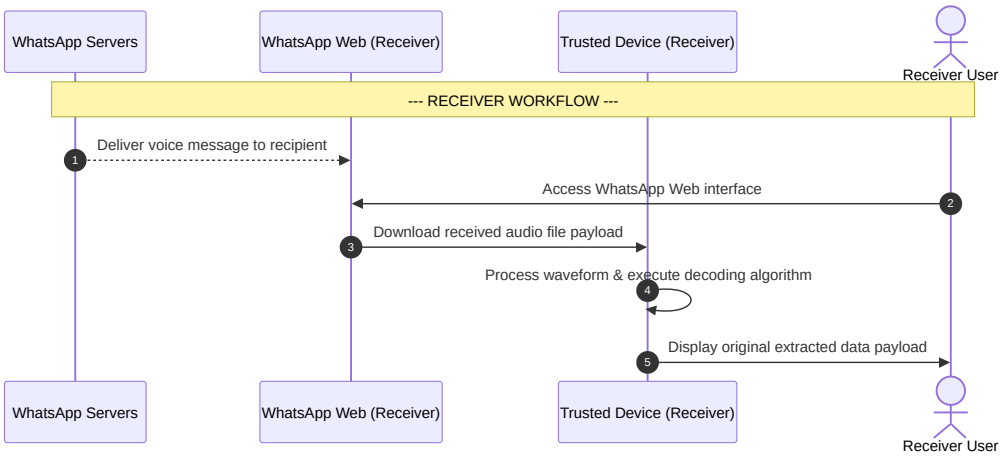
4.2 The Used Approach: Using the Web App Version of the Instant Messaging Application

Since we still want to use the WhatsApp (2026) voice message feature, we have taken a slightly different approach. Instead of using the smartphone to send the message, we directly use the web version of WhatsApp on the trusted device. The workflow is simpler than the previous one, but the core features are unchanged.

- **The Sender Workflow:**
  1. **Data Generation:** We generate the payload on a trusted hardware platform, such as a Raspberry Pi Ltd (2026).
  2. **Encoding:** The device encodes the raw data into acoustic waveforms, which are then stored as a high-fidelity audio file.
  3. **Hardware Interfacing:** On the trusted device we create a virtual microphone using PulseAudio (Poettering et al., 2026).
  4. **Transmission Initialization:** On the sender WhatsApp Web, we open the target chat and initiate the voice recording function.
  5. **Signal Injection:** We play the audio file generated through the virtual microphone created at step 3.
  6. **Finalization:** Once the playback is complete, we stop the recording and send the audio message to the receiver.
- **The Receiver Workflow:**
  1. **Hardware Interfacing:** We open WhatsApp Web on the receiver's trusted decoding device.
  2. **Signal Capture:** We download the received audio message.
  3. **Decoding and Extraction:** The trusted device processes the downloaded audio waveform, decodes the signal, and presents the original payload to the user.



(a) Encoding workflow



(b) Decoding workflow

Figure 3: Workflow pipeline using WhatsApp Web.

This framework represents the core workflow implemented throughout this thesis.

5 Modulation Strategy and Acoustic Channel Selection

In this section, we justify the selection of Phase Shift Keying (PSK) as our primary digital modulation scheme and analyze its performance boundaries within our architecture.

PSK is a robust, widely adopted modulation technique in wireless telecommunications, frequently utilized in low-power, short-range radio frequency (RF) applications such as automotive remote keyless entry (RKE) systems (Proakis & Salehi, 2008) and automated garage door transceivers.

Historically, PSK is considered poorly suited for traditional acoustic data transmission (audio-frequency over-the-air signaling). In physical air channels, acoustic waves suffer severely from multipath propagation and boundary reflections. When an acoustic wavefront collides with physical obstacles, the reflected waves experience unpredictable phase shifts and constructive/destructive interference, causing severe phase distortion and signal degradation at the receiver (Zhidkov, 2018). Consequently, conventional acoustic data networks typically favor modulation techniques less sensitive to phase anomalies, such as Frequency-Shift Keying (FSK) or Amplitude-Shift Keying (ASK).

Our architecture, however, fundamentally eliminates the physical acoustic channel and its associated environmental degradation. Because the generated waveforms are never acoustic waves propagating through free space, the system is completely immune to reflection-induced phase cancellation. As illustrated in Figure 3, the audio data is kept entirely within a digital pipeline, routed via a virtual loopback microphone stack. This virtual isolation ensures absolute channel consistency, allowing us to leverage the high data density and spectral efficiency of PSK without risking the transmission errors typically associated with acoustic environments.

Choosing PSK is ideal for steganography because it keeps the hidden data inconspicuous within a WhatsApp (2026) voice note. By maintaining both amplitude and frequency constant, the modulation introduces no sudden changes in pitch or

volume. Because human hearing is relatively insensitive to phase variations, the altered signal is easily mistaken for standard recording noise. Furthermore, BPSK offers superior spectral efficiency compared to FSK; by encoding information via phase transitions rather than frequency deviations, it achieves a significantly narrower bandwidth footprint for the same data rate.

Since the transmission medium relies on compressed voice notes, the operational frequency range must be selected carefully to survive lossy compression. Low frequencies were selected because the WhatsApp (2026) audio codec is optimized for human speech. This means that it is less likely to severely attenuate them. Consequently, the frequency range for this system is set between 150 Hz and 400 Hz.

The primary disadvantage of utilizing this lower frequency band is the resulting limitation on the achievable bitrate. The maximum transmission speed is dictated by the symbol rate formula:

$$R_s = \frac{f}{N_p}$$

where R_s is the symbol rate (in baud), f is the carrier frequency (in Hz), and N_p is the number of periods per symbol. Empirical testing revealed that each symbol must span at least two full periods ($N_p \geq 2$) to ensure receiver reliability; shorter durations prevent the decoder from properly synchronizing and resolving the phase shifts. For example, using a 200 Hz carrier frequency yields a maximum symbol rate of:

$$R_s = \frac{200 \text{ Hz}}{2 \text{ periods/symbol}} = 100 \text{ symbols/s}$$

Assuming binary modulation (where 1 symbol = 1 bit), this corresponds to a bitrate of 100 bps. At this speed, transmitting a 1 KB payload (8192 bits) would require approximately 82 s of continuous audio, highlighting the explicit trade-off between steganographic imperceptibility and data throughput.

5.1 Modulating BPSK and DPSK

We developed the PSK encoding with two different algorithms: Bit Phase Shift Keying (BPSK) and Differential Phase Shift Keying (DPSK).

Algorithm 1: Phase Modulation Waveform Generation

Input : B: bit array of size N

f: carrier frequency (Hz)

C: periods per symbol

R_s: sample rate (Hz)

Output: W: generated phase-modulated wave array

1 `BitsToPhaseWave(`B, f, C, R_s`)`

2 **begin**

3 $S \leftarrow \left\lfloor \text{round}\left(\frac{R_s \cdot C}{f}\right)\right\rfloor$ `// Samples per symbol`

4 $M \leftarrow N \cdot S$ `// Total number of samples`

`// Initialize time vector`

5 **for** $i \leftarrow 0$ **to** $M - 1$ **do**

6 $T[i] \leftarrow \frac{i}{R_s}$

`// Construct phase vector by repeating bit phases`

7 **for** $k \leftarrow 0$ **to** $N - 1$ **do**

8 **for** $j \leftarrow 0$ **to** $S - 1$ **do**

9 $P[k \cdot S + j] \leftarrow \pi \cdot B[k]$

`// Generate the modulated sine wave`

10 **for** $i \leftarrow 0$ **to** $M - 1$ **do**

11 $W[i] \leftarrow \sin(2\pi \cdot f \cdot T[i] + P[i])$

12 **return** W

BPSK modulation In order to encode data using BPSK, we just need to use Algorithm 1. Line 9 converts a bit (which is 0 or 1) into a symbol. The symbol is a phase. Since we only need two distinct phases, we use 0 or π as symbols. This explains the multiplication mapping: $\pi \cdot 0 = 0$ and $\pi \cdot 1 = \pi$. Line 11 subsequently converts these symbols into the actual discrete samples for the wave.

Algorithm 2: Complete DBPSK Modulation Pipeline

Input : B: input bit array of size N

f: carrier frequency (Hz)

C: periods per symbol

R_s: sample rate (Hz)

p: initial reference bit (defaults to 1)

Output: W: final DBPSK modulated phase wave array

```
ModulateDBPSK(B, f, C, R_s, p)
begin
    // Step 1: Perform Differential Encoding
    R[0] ← p
    v ← p
    for i ← 0 to N − 1 do
        v ← v ⊕ B[i]
        R[i + 1] ← v
    // Step 2: Generate Phase Wave from Differential Bits
    W ← BitsToPhaseWave(R, f, C, R_s)
    return W
```

DBPSK modulation The main advantage of DBPSK[1] over conventional BPSK is that it eliminates the need for a complex, synchronized coherent carrier reference at the receiver. By encoding data as the phase difference between successive symbols rather than absolute phase, it significantly reduces hardware costs and receiver tracking complexity.

According to Proakis and Salehi (2008), Phase-Shift Keying maps digital information onto discrete variations of a carrier wave's phase. For a conventional (BPSK) system, the transmitted signal during a symbol interval T_s is mathematically expressed as:

$$s(t) = \sqrt{\frac{2E_b}{T_s}} \cos\left(2\pi f_c t + \theta_n\right), \quad 0 \leq t \leq T_s \tag{1}$$

where E_b represents the energy per bit, f_c denotes the carrier frequency, and the phase parameter $\theta_n \in \{0, \pi\}$ maps directly to the binary data sequence $B[n] \in \{0, 1\}$ via the linear relationship $\theta_n = \pi \cdot B[n]$.

While BPSK achieves optimal bit-error-rate BER performance in additive white Gaussian noise (AWGN) channels, it relies on strict coherent detection. This requires the receiver to maintain perfect carrier frequency and phase synchronization; in practice, phase drift inevitably desynchronize the local oscillator from the carrier (see Figure 23 and Figure 24). To eliminate the hardware and computational overhead of carrier synchronization loops (e.g., Costas loops), DBPSK encodes information through a differential mapping sequence $R[n]$, where phase transitions are conditioned on the preceding symbol. The differential sequence is constructed via the recursive modulo-2 logic:

$$R[n] = R[n-1] \oplus B[n] \tag{2}$$

This mathematical transformation ensures that data is carried strictly by the phase transitions between successive intervals rather than an absolute coordinate system.

[1]DBPSK (Differential Binary Phase-Shift Keying) is the binary special case of the broader DPSK (Differential Phase-Shift Keying) family, in which the phase alphabet contains only two elements: $\{0, \pi\}$ Proakis and Salehi (2008).

The compromise is that since we are using the previous symbol to encode the next one, an error in decoding can propagate to the remaining payload.

5.2 Frame Synchronization and Demodulation via Barker Sequences

With the encoding parameters for both Binary Phase Shift Keying (BPSK) and Differential Phase Shift Keying (DBPSK) established, the receiver architecture must implement robust mechanisms for signal acquisition and carrier demodulation.

Because a continuous BPSK bitstream lacks native structural boundaries, the decoder requires a deterministic method to detect the exact temporal boundary where a valid data packet begins. To achieve this frame synchronization, a 13-bit Barker sequence is prepended as a synchronization preamble to every transmitted payload:

`[+1, +1, +1, +1, +1, -1, -1, +1, +1, -1, +1, -1, +1]`.

The primary architectural advantage of utilizing Barker codes lies in their ideal mathematical autocorrelation properties (Barker, 1953). The autocorrelation function of a Barker sequence yields a highly distinct, maximum peak value of $+N$ (where $N = 13$ is the length of the code) at the exact point of perfect alignment. For all non-zero shifts (phase offsets), the sidelobes of the autocorrelation function are bounded to a maximum absolute value of ≤ 1.

By passing the incoming digital audio stream through a matched filter cross-correlated with this known 13-bit preamble, the receiver can continuously scan for this sharp spike. The presence of this peak allows the system to definitively locate the packet origin and establish correct symbol timing, even if the signal has undergone phase drift or attenuation over the virtual channel.

Algorithm 3: Cross-Correlation and Normalized Matched Filtering for Preamble Detection (Part 1)

Input: W: Input raw acoustic waveform vector

E_b: Preamble symbol array, what we expect to see

f_c: Carrier frequency (Hz)

f_s: Sampling rate (Hz)

N_c: Carrier periods per symbol

Output: k_{best}: Calculated optimal sample index offset representing packet origin

`// Step 1: Construct the expected physical reference waveform`

$X_{\text{ref}} \leftarrow \text{BitsToPhaseWave}(E_b, f_c, N_c, f_s)$ `// Generates the ideal DBPSK template signal`

$L_{\text{win}} \leftarrow |X_{\text{ref}}|$

`// Step 2: Signal zero-centering preprocessing`

$W \leftarrow \texttt{Real}(W) - \texttt{Mean}(\texttt{Real}(W))$

$X_{\text{ref}} \leftarrow \texttt{Real}(X_{\text{ref}}) - \texttt{Mean}(\texttt{Real}(X_{\text{ref}}))$

Algorithm 3: Cross-Correlation and Normalized Matched Filtering for Preamble Detection (Part 2)

// Step 3: Fast sliding matched filtering via cross-correlation

$R_{\text{raw}} \leftarrow W \star X_{\text{ref}}$ // $(\star)$ denotes cross-correlation operator

$E_{\text{tmpl}} \leftarrow \sum |X_{\text{ref}}|^2$

// Step 4: Local energy normalization calculation

$P_{\text{cand}} \leftarrow |W|^2$

$E_{\text{cand}} \leftarrow P_{\text{cand}} * \mathbf{1}_{L_{\text{win}}}$ // $(*)$ denotes convolution with moving-window vector

// Step 5: Compute normalized inner product scores and find index peak

$\gamma \leftarrow \sqrt{E_{\text{cand}} \cdot E_{\text{tmpl}}}$

$S \leftarrow \mathcal{D}\left(R_{\text{raw}}, \gamma\right)$ // Element-wise safe division avoiding zero denominators

$k_{\text{peak}} \leftarrow \texttt{ArgMax}\left(\texttt{Abs}(S)\right)$

$k_{\text{best}} \leftarrow k_{\text{peak}} - \lfloor L_{\text{win}}/2 \rfloor$

return k_{best}

Algorithm 3 describes the complete preamble detection procedure. The receiver first constructs an ideal reference waveform from the known preamble bit sequence and carrier parameters. Both the reference and the incoming audio signal are then baseline-corrected by subtracting their respective sample means, ensuring the subsequent correlation operates purely on the oscillatory content of each signal. The reference is then slid across the incoming waveform via cross-correlation, producing a raw similarity score at each sample offset. This score is normalized by the local signal energy to yield an amplitude-invariant detection statistic, whose peak position is corrected for the center-aligned output of the sliding window. This identify the sample index at which the preamble begins. The mathematical foundations underlying each of these steps are developed in Section 5.3.

5.3 Mathematical Foundations of the Matched Filter and Normalization

The preamble detection framework implemented in Algorithm 3 is theoretically grounded in classical detection theory. In digital communications, a matched filter represents the mathematically optimal linear filter for maximizing the output signal-to-noise ratio (SNR) in the presence of AWGN (Proakis & Salehi, 2008; Sklar, 2001; Turin, 1960).

The continuous-time cross-correlation between the received raw acoustic waveform $W(t)$ and the deterministic, time-reversed reference template $X_{\text{ref}}(t)$ over a finite window duration T is defined as:

$$R_{\text{raw}}(\tau) = \int_0^T W(t)\, X_{\text{ref}}(t-\tau)\, dt \tag{3}$$

In the discrete-time domain utilized by our receiver script, this operation maps directly to the sliding dot product executed via the cross-correlation operator ($\star$) (Oppenheim & Schafer, 2009):

$$R_{\text{raw}}[k] = \sum_n W[n]\, X_{\text{ref}}[n-k] \tag{4}$$

Prior to computing the cross-correlation, both the received waveform and the reference template are zero-meaned by subtracting their respective sample means. This DC-removal step ensures that the correlation is sensitive exclusively to the AC structure of the signals, suppressing any bias introduced by baseline offsets in the audio capture chain.

While a raw cross-correlation peak is sufficient for signal alignment in an unattenuated, static environment, fluctuations in the physical or virtual channel's gain (volume) can lead to amplitude-induced false positives. To ensure invariant detection thresholds independent of local volume spikes, the raw correlation scores must undergo local energy normalization to yield a bounded, amplitude-invariant decision statistic (Kay, 1998).

This normalization follows from the Cauchy-Schwarz inequality. For any two real-valued discrete vectors, the inner product is strictly bounded by the product of

their Euclidean norms:

$$\left| \sum_{n} W[n]\, X_{\text{ref}}[n-k] \right| \;\leq\; \sqrt{\sum_{n} |W[n]|^2} \cdot \sqrt{\sum_{n} |X_{\text{ref}}[n]|^2} \tag{5}$$

Unlike the template energy, which is a fixed scalar, the candidate energy must be evaluated locally over a sliding window of length L equal to the template length, so as to normalize against the instantaneous signal power at each candidate position k:

$$E_{\text{cand}}[k] = \sum_{m=0}^{L-1} |W[k+m]|^2 \tag{6}$$

The static template energy is:

$$E_{\text{tmpl}} = \sum_{n=0}^{L-1} |X_{\text{ref}}[n]|^2 \tag{7}$$

The resulting normalized decision statistic $S[k]$, commonly referred to as the normalized cross-correlation coefficient (Oppenheim & Schafer, 2009), is defined as:

$$S[k] = \frac{R_{\text{raw}}[k]}{\sqrt{E_{\text{cand}}[k] \cdot E_{\text{tmpl}}}} \tag{8}$$

By the Cauchy-Schwarz inequality, $S[k] \in [-1, 1]$, where $S[k] = 1$ indicates perfect phase alignment and $S[k] = -1$ indicates perfect phase inversion (anti-correlation). Because DBPSK detection must be invariant to absolute phase polarity, the final detection statistic is $|S[k]|$, and the true packet origin index is identified as:

$$k_{\text{best}} = \arg\max_{k} |S[k]| - \left\lfloor \tfrac{L}{2} \right\rfloor \tag{9}$$

The additive correction $-\lfloor L/2 \rfloor$ compensates for the center-aligned output of the `same`-mode cross-correlation, mapping the peak position back to the true sample offset of the preamble onset in the original waveform.

5.4 BPSK demodulation

Algorithm 4: BPSK Demodulation and Low-Pass Filtering

Function `DecodeSymbolValuesBPSK`(w, N_s, f_s, f_c)`:`

  Input: w (waveform), N_s (samples per symbol), f_s (sampling rate), f_c (carrier frequency)

  Output: Array of decoded symbols $\in \{-1, 1\}$

  $t \leftarrow [0, 1, \ldots, |w| - 1]/f_s$ `// Time vector`

  $\text{osc} \leftarrow \exp(-j \cdot 2\pi \cdot t \cdot f_c)$ `// Local oscillator`

  $\text{baseband} \leftarrow$ `LowPassFilter`$(\text{osc} \cdot w, f_s)$

  $\phi \leftarrow \angle(\text{baseband})$ `// Extract phase information`

  $I_{start} \leftarrow$ sequence from 0 to $|\phi| - N_s$ with step N_s

  $I_{mid} \leftarrow I_{start} + \lfloor N_s/2 \rfloor$

  Filter I_{mid} such that all elements $< |\phi|$

  return $\begin{cases} -1 & \text{if } \phi[I_{mid}] < 0 \\ 1 & \text{otherwise} \end{cases}$

Function `LowPassFilter`$(x, f_s, f_{cut} = 100, n = 5)$`:`

  $W_n \leftarrow \frac{f_{cut}}{f_s/2}$ `// Normalized cutoff frequency`

  $b, a \leftarrow \text{ButterworthFilterCoefficients}(n, W_n, \text{“low”})$

  $y \leftarrow \text{ZeroPhaseFilter}(b, a, x)$

  return y

The BPSK demodulation procedure (Algorithm 4) recovers the transmitted bit stream from the trimmed waveform by exploiting the phase structure of binary phase-shift keying. The algorithm proceeds in three conceptually distinct stages: frequency down-conversion, phase extraction, and symbol decision.

Frequency down-conversion. The first step multiplies the received waveform w by a complex local oscillator $\text{osc}(t) = e^{-j2\pi f_c t}$, which shifts the spectrum so that the carrier component is centred at 0 Hz yielding the baseband signal. The product $\text{osc}(t) \cdot w(t)$ is then passed through a fifth-order Butterworth low-pass filter with a

100 Hz cut-off, implemented via zero-phase forward-backward filtering (`filtfilt`) to eliminate group-delay distortion Oppenheim and Schafer (2009). After filtering, the resulting complex baseband signal retains the phase modulation imposed by the transmitter while suppressing out-of-band interference and the image component at $-2f_c$ Sklar (2001).

Phase extraction. The instantaneous phase is obtained as

$$\phi[n] = \angle\big(\text{baseband}[n]\big) \in (-\pi,\, \pi], \tag{10}$$

which maps to $\{0, \pi\}$ for ideal BPSK symbols Proakis and Salehi (2008).

Symbol decision. Rather than integrating the phase over the entire symbol interval (which would be sensitive to inter-symbol boundary effects) the algorithm samples ϕ at the midpoint of each symbol window. Formally, for symbol index k the sampling instant is

$$n_k = k\, N_s + \left\lfloor \frac{N_s}{2} \right\rfloor, \tag{11}$$

where N_s is the number of samples per symbol. Midpoint sampling maximises the distance from symbol transitions, where inter-symbol interference and phase discontinuities are most pronounced Sklar (2001). The hard decision rule

$$\hat{b}_k = \begin{cases} -1 & \text{if } \phi[n_k] < 0, \\ 1 & \text{otherwise,} \end{cases} \tag{12}$$

corresponds to the maximum-likelihood detector for BPSK under AWGN when the channel phase offset is zero Kay (1998). Because the preamble detection stage (Algorithm 3) provides coarse frequency and phase alignment before this function is invoked, the residual phase error is sufficiently small for the threshold at 0 to remain near-optimal in practice Proakis and Salehi (2008).

Role of the low-pass filter. After multiplication by the local oscillator, the spectrum of the baseband signal contains two contributions: the desired term centred

at the baseband, which carries the phase modulation, and an image term centred at $-2f_c$, which must be suppressed before phase extraction. The fifth-order Butterworth low-pass filter with cut-off $f_{cut} = 100\,\text{Hz}$ removes this image component and attenuates out-of-band noise, leaving a clean complex envelope whose argument directly yields $\phi[n]$ as in (10) Oppenheim and Schafer (2009) and Sklar (2001).

The choice of a very narrow cut-off (100 Hz relative to a carrier in the audible range) is motivated by the acoustic channel characteristics: the dominant noise sources are broadband, so a tighter filter improves the post-filter SNR at the cost of a longer transient response Zhidkov (2018).

Limitations This architecture assumes a stable carrier phase throughout the entire received waveform. In practice, WhatsApp re-encodes audio using a lossy codec (Opus Valin et al. (2012)), which introduces a slowly time-varying phase drift that accumulates over the duration of the message. Because no phase-tracking loop (e.g., Costas loop Proakis and Salehi (2008)), is employed, this residual drift is not corrected, and the effective phase error at the decision instants (11) grows with message length. As a consequence, the symbol error rate increases for longer payloads, representing a known limitation of the current implementation.

5.5 DBPSK demodulation

Algorithm 5: DBPSK Symbol Decoding via Phase-Delta Estimation

Function `DecodeSymbolValuesDBPSK`(w, N_s, f_s, f_c)**:**

Input: w (waveform), N_s (samples per symbol), f_s (sampling rate), f_c (carrier frequency)

Output: Array of decoded differential symbols $\in \{-1, 1\}$

$L_{usable} \leftarrow |w| - (|w| \pmod{N_s})$ `// Truncate to complete symbols`

$t \leftarrow [0, 1, \ldots, L_{usable} - 1]/f_s$

$\text{osc} \leftarrow \exp(-j \cdot 2\pi \cdot f_c \cdot t)$

$x_{mix} \leftarrow w[0 \ldots L_{usable} - 1] \cdot \text{osc}$ `// Downconvert to complex baseband`

$M \leftarrow L_{usable}/N_s$ `// Total number of complete symbols`

for $k \leftarrow 0$ **to** $M - 1$ **do**

  $S[k] \leftarrow \frac{1}{N_s} \sum_{m=0}^{N_s-1} x_{mix}[k \cdot N_s + m]$ `// Integrate and dump integration`

for $k \leftarrow 0$ **to** $M - 2$ **do**

  $\Delta\phi[k] \leftarrow \angle\left(\exp\left(j \cdot (\angle S[k+1] - \angle S[k])\right)\right)$ `// Phase difference between successive symbols`

return $\begin{cases} -1 & \text{if } \cos(\Delta\phi) \geq 0 \\ 1 & \text{otherwise} \end{cases}$

The DBPSK demodulation procedure (Algorithm 5) recovers the transmitted bit stream by estimating the phase difference between successive symbols, rather than the absolute phase of each symbol. This differential structure removes the need for a phase reference at the receiver, at the cost of a slight increase in symbol error rate compared to coherent BPSK Proakis and Salehi (2008). The algorithm proceeds in three stages: baseband conversion with integrate-and-dump, phase-delta estimation, and differential decision.

Baseband conversion and integrate-and-dump. As in the BPSK case, the waveform is first truncated to an integer number of symbol periods (L_{usable}) and

multiplied by the complex local oscillator $\mathrm{osc}(t) = e^{-j2\pi f_c t}$ to shift the carrier component to baseband Sklar (2001). Rather than applying a low-pass filter, the algorithm integrates the mixed signal over each symbol window of N_s samples and retains only the mean:

$$S[k] = \frac{1}{N_s} \sum_{m=0}^{N_s-1} x_{mix}[kN_s + m], \qquad k = 0, \ldots, M-1. \tag{13}$$

This *integrate-and-dump* operation is the discrete-time equivalent of a matched filter for a rectangular pulse shape Turin (1960), and it maximises the post-integration SNR under AWGN Kay (1998). Compared to the point-sampling approach used in the BPSK decoder, integration averages out within-symbol noise fluctuations, yielding a more robust phase estimate per symbol Sklar (2001).

Phase-delta estimation. The instantaneous phase of each integrated symbol is $\angle S[k]$, and the phase difference between two consecutive symbols is

$$\Delta\phi[k] = \angle\big(e^{\,j\,(\angle S[k+1] - \angle S[k])}\big), \qquad k = 0, \ldots, M-2. \tag{14}$$

Wrapping the difference inside a complex exponential before extracting the angle ensures that $\Delta\phi[k] \in (-\pi, \pi]$ regardless of any 2π ambiguity in the individual phase estimates Oppenheim and Schafer (2009).

Differential decision. In DBPSK, a bit is encoded as the *change* in carrier phase between adjacent symbols: a phase shift of π represents one logical value, while no phase shift represents the other Proakis and Salehi (2008). The decision rule

$$\hat{b}[k] = \begin{cases} -1 & \text{if } \cos(\Delta\phi[k]) \geq 0, \\ 1 & \text{otherwise,} \end{cases} \tag{15}$$

maps $|\Delta\phi| < \pi/2$ (no phase reversal) to -1 and $|\Delta\phi| \geq \pi/2$ (phase reversal) to 1 Sklar (2001). Using $\cos(\Delta\phi)$ as the decision statistic rather than thresholding $\Delta\phi$ directly is numerically preferable: it avoids any sensitivity to the sign convention of the four-quadrant arctangent at the boundary $\pm\pi$, and is equivalent to computing the real part of the complex correlation $S[k+1]\,\overline{S[k]}$, which is the standard differential detector formulation Proakis and Salehi (2008).

Limitation. As with the BPSK decoder, no phase-tracking loop is employed. However, the differential structure of DBPSK makes it intrinsically more robust to a *slowly* drifting carrier phase: a gradual drift affects both $S[k]$ and $S[k+1]$ nearly equally, so its contribution largely cancels in $\Delta\phi[k]$ Proakis and Salehi (2008). Despite this, the lossy re-encoding introduced by WhatsApp (2026) (Opus Valin et al. (2012)) can produce phase perturbations that are not negligible over longer messages, causing the symbol error rate to increase with payload length, as observed experimentally.

5.5.1 On the Multiplication with the Local Oscillator and Complex Mixing

Let $X(t) \in \mathbb{R}$ represent the real-valued received signal, temporally aligned with the onset of the preamble sequence. By Fourier analysis, $X(t)$ can be characterized as a linear combination of sinusoids spanning distinct frequencies and phases (harmonics). In our architecture, the target information is carried entirely by the phase variations of a specific harmonic component at the carrier frequency f_c (e.g., 200 Hz).

To isolate and extract the instantaneous phase of this component, the spectrum of the signal must be shifted down by $\Delta f = -f_c$. This down-conversion translates the passband signal centered at f_c into a complex baseband signal. Evaluating the phase at this zero-frequency baseline directly yields the modulated phase of the carrier. Furthermore, shifting the target band to baseband allows for straightforward isolation of the signal via a low-pass filter.

This frequency translation is accomplished by multiplying the input signal by a complex exponential oscillator:

$$Y(t) = X(t) \cdot e^{-j\omega_c t} \tag{16}$$

where $\omega_c = 2\pi f_c$ is the angular carrier frequency in radians per second. To demonstrate this operation mathematically, let us assume an idealized scenario where $X(t)$

consists exclusively of a single harmonic at frequency f_c:

$$X(t) = A\cos(\omega_c t + \phi) \tag{17}$$

Here, A represents the signal amplitude, and $\phi \in [-\pi, \pi]$ is the modulated phase parameter we seek to recover. Utilizing Euler's formula, the complex exponential is expressed as:

$$e^{-j\omega_c t} = \cos(\omega_c t) + j\sin(\omega_c t) \tag{18}$$

Consequently, the complex mixed signal $Y(t)$ becomes:

$$\begin{aligned} Y(t) &= X(t) * e^{-j\omega_c t} && (19)\\ &= (A\cos(\omega_c t + \phi)) \cdot (\cos(\omega_c t) + j\sin(\omega_c t)) && (20)\\ &= \underbrace{A\cos(\omega_c t + \phi)\cos(\omega_c t)}_{\text{Real Part}} + \underbrace{jA\cos(\omega_c t + \phi)\sin(\omega_c t)}_{\text{Imaginary Part}} && (21) \end{aligned}$$

Applying standard product-to-sum trigonometric identities:

$$\cos(A)\cos(B) = \frac{1}{2}(\cos(A-B) + \cos(A+B)) \tag{22}$$

we can expand the real component as:

$$\begin{aligned} A\cos(\omega_c t + \phi)\cos(\omega_c t) &= \frac{A}{2}\left[\cos\big((\omega_c t + \phi) - \omega_c t\big) + \cos\big((\omega_c t + \phi) + \omega_c t\big)\right] \\ &= \frac{A}{2}\cos(\phi) + \frac{A}{2}\cos(2\omega_c t + \phi) \end{aligned} \tag{23}$$

Similarly, applying the identity $\cos(\alpha)\sin(\beta) = \frac{1}{2}\big(\sin(\alpha+\beta) - \sin(\alpha-\beta)\big)$, the imaginary component simplifies to:

$$jA\cos(\omega_c t + \phi)\sin(\omega_c t) = -\frac{A}{2}\sin(\phi) + \frac{A}{2}\sin(2\omega_c t + \phi) \tag{24}$$

Combining these terms yields the complete expression for the mixed baseband signal:

$$Y(t) = \frac{A}{2}\cos(\phi) - j\frac{A}{2}\sin(\phi) + \frac{A}{2}\cos(2\omega_c t + \phi) + j\frac{A}{2}\sin(2\omega_c t + \phi) \tag{25}$$

Equation (22) reveals that the mixed signal comprises two distinct spectral components: a quasi-static basebend term carrying the target phase ϕ, and a high-frequency image component rotating at double the carrier frequency ($2\omega_c$). To

cleanly extract the phase, the high-frequency term must be suppressed using a low-pass filter, yielding the complex envelope $\hat{Y}(t)$:

$$\hat{Y}(t) \approx \frac{A}{2}\cos(\phi) - j\frac{A}{2}\sin(\phi) \tag{26}$$

From this filtered complex representation, the original phase is recovered directly via the four-quadrant argument function: $\phi = \angle\big(\hat{Y}(t)\big)$. Consequently, low-pass filtering is a fundamental mathematical necessity to achieve an accurate phase estimation in both the coherent BPSK and non-coherent DBPSK architectures.

5.5.2 Comparative Analysis of Low-Pass Filtering Strategies

While an ideal brick-wall low-pass filter would perfectly isolate the baseband signal up to the symbol rate, practical implementations must balance computational complexity, roll-off sharpness, and phase distortion. A practical and conservative choice for the filter cutoff frequency is $f_{\text{cut}} = \frac{f_c}{2}$.

The mechanical implementation of this filtering operation differs between the two receiver variants, though their operational outcomes are mathematically comparable:

- **BPSK Demodulation:** This pipeline utilizes an explicit 5th-order digital Butterworth filter. When implemented using forward-backward filtering (`filtfilt`), it provides a steep roll-off and eliminates group-delay variations, ensuring that the phase at the midpoint sampling instant remains unwarped.
- **DBPSK Demodulation:** In this variant, the explicit low-pass filter is omitted. Instead, the *integrate-and-dump* operation functions implicitly as a moving-average box filter. Integrating a continuous stream of N_s samples over a discrete symbol window corresponds to a low-pass filtering operation with a primary null located at $f_{\text{null}} = \frac{R_s}{N_s}$, where R_s represents the sampling rate.

Given our operational parameters ($R_s = 48{,}000\,\text{Hz}$, $f_c = 200\,\text{Hz}$, and $C = 2$ periods per symbol), the number of samples per symbol is derived as:

$$N_s = \frac{R_s \cdot C}{f_c} = \frac{48{,}000 \cdot 2}{200} = 480 \text{ samples} \tag{27}$$

This window configuration establishes an implicit filter cutoff at $\frac{48{,}000}{480} = 100\,\text{Hz}$. Strikingly, this matches the exact 100 Hz cutoff frequency explicitly configured for the BPSK Butterworth filter, exposing a structural symmetry between our two architectures.

To conclude, the core properties, trade-offs, and architectural constraints governing our demodulation frameworks are summarized below:

- **Ubiquitous Filtering:** Both the BPSK and DBPSK pipelines rely fundamentally on low-pass filtering configured with identical effective cutoff frequencies (100 Hz) to suppress the $2\omega_c$ modulation image.

- **Mathematical Necessity:** Regardless of the chosen keying strategy, a low-pass filtering mechanism is indispensable to separate the baseband phase component from the passband carrier product.

- **Spectral Trade-offs:** The integrate-and-dump filter used in DBPSK is fast and computationally efficient. However, because its frequency response is a sinc function, it has a shallower roll-off and prominent sidelobes. This makes it more prone to out-of-band noise leakage than the sharper Butterworth filter.

- **Open-Loop Phase Assumption:** Neither variant implements an active phase-tracking loop (e.g., a Costas loop). Both architectures operate on the baseline assumption of a stable carrier frequency and perfect synchronization derived exclusively from the initial Barker preamble.

- **Phase Drift Resilience:** The DBPSK pipeline demonstrates superior robustness against time-varying phase drift, such as the artifacts introduced by lossy WhatsApp audio compression. Because DBPSK evaluates phase transitions relative to the immediately preceding symbol—rather than an absolute historical reference coordinate anchored at the preamble origin—slow phase errors cancel out across adjacent intervals. This is particularly crucial since the Opus codec (Valin et al., 2012) inherently introduces a slowly accumulat-

ing phase and frequency distortion, as demonstrated empirically in Figures 23 and 24.

- **Error Propagation Cost:** The resilience of DBPSK introduces a distinct vulnerability: a single phase estimation error caused by an instantaneous noise spike alters the differential reference point for the subsequent symbol, causing the error to propagate into downstream payload decoding.

6 Header Composition and Error Correction

In order to establish a robust communication framework resilient against adversarial channel manipulation and accidental noise, the protocol implements a structured forward error correction (FEC) layer at two levels: the packet header and the payload body. This section delineates the architectural layout of the packet header and describes how Hamming-based error correction is applied uniformly across the entire transmitted packet.

The primary objective of the header is to securely convey foundational packet metadata. To ensure that the receiver can parse these essential parameters even under severe channel degradation, a mandatory, fixed linear block code is applied uniformly across the header block prior to packet encapsulation.

6.1 Raw Header Architecture

Prior to forward error correction mapping, the raw header comprises a total of HEADER_DATA_BITS = 20 bits. The allocation of these bits is strictly partitioned into three fields designed to balance protocol agility with minimal data overhead:

- **Header Version (2 bits):** Specifies the structural revision of the protocol wrapper, preventing version mismatch exploits and accommodating up to 4 distinct protocol iterations.
- **ECC Scheme Identifier (2 bits):** Signals the dynamic error correction methodology applied to the subsequent *payload*. This field permits runtime agility, enabling transitions between unencoded payload transmissions and protected payloads utilizing a systematic Hamming (7,4) scheme. Crucially, this field governs only the payload encoding; the header itself is *always* protected by Hamming (7,4) regardless of its value.
- **Payload Length (16 bits):** A 16-bit unsigned integer defining the absolute size of the trailing payload in bytes. This field bounds the maximum payload

capacity to:

$$M_{\text{payload}} = 2^{16} - 1 = 65{,}535 \text{ bytes} \tag{28}$$

6.2 Forward Error Correction via Hamming (7,4)

The Hamming (7,4) code was originally introduced by Hamming (1950) as one of the first systematic constructions of a single-error-correcting linear block code. Its properties are well established in the coding theory literature (Lin & Costello, 2004; Moon, 2005): it maps $k = 4$ data bits into an $n = 7$ bit codeword by appending $m = n - k = 3$ parity bits, achieving a minimum Hamming distance of $d_{\min} = 3$, which is sufficient for single-bit error correction (SEC) within each independent codeword.

6.2.1 Header Encoding

To shield the header metadata against single-bit errors, the protocol passes the 20-bit raw header through a systematic Hamming (7,4) block code. The encoding requires partitioning the header into discrete 4-bit nibbles ($k = 4$), each of which is independently transformed into a 7-bit codeword ($n = 7$). Given the 20-bit raw header, the number of independent encoding iterations is:

$$N_{\text{blocks}} = \frac{\text{HEADER\_DATA\_BITS}}{\text{HEADER\_CODEWORD\_DATA\_BITS}} = \frac{20}{4} = 5 \text{ blocks} \tag{29}$$

Consequently, the transmitted, hardened header expands to:

$$L_{\text{encoded}} = N_{\text{blocks}} \times \text{HEADER\_CODEWORD\_BITS} = 5 \times 7 = \mathbf{35} \text{ bits} \tag{30}$$

This represents a fixed structural overhead of 75% on the header profile. The scheme guarantees correction of up to one corrupted bit per 7-bit codeword block, i.e. up to 5 independent single-bit errors across the full header, provided no two errors fall within the same codeword. The resulting header structure is illustrated in Figure 4.

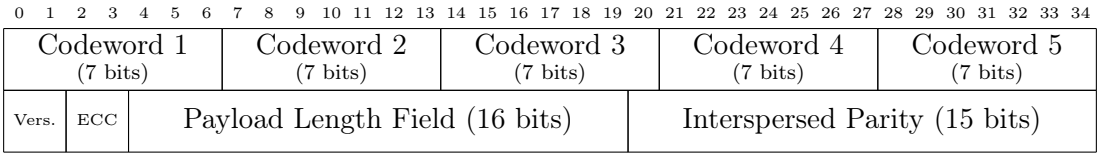


Figure 4: 35-bit header structure after Hamming (7,4) encoding.

6.2.2 Payload Encoding

When the ECC Scheme Identifier field is set to `ECC_SCHEME_HAMMING_7_4`, the same Hamming (7,4) code is applied to the entire payload body prior to transmission. Because the payload length in bytes is not guaranteed to be divisible by 4 bits, a zero-padding step is performed first: the raw payload bit stream is padded with trailing zero bits to the nearest 4-bit boundary before encoding. The padding length is discarded after transmission, since the true byte count is recovered from the Payload Length field in the decoded header.

The encoded payload length in bits for a payload of B bytes is therefore:

$$L_{\text{payload}} = \left\lceil \frac{8B}{4} \right\rceil \times 7 = \lceil 2B \rceil \times 7 = 14B \text{ bits} \tag{31}$$

where the $\lceil \cdot \rceil$ accounts for the padding step (the expression simplifies to $14B$ because $8B$ is always divisible by 4).

6.3 Full Packet Structure

The complete transmitted bit stream is assembled by concatenating three components in order:

$$\mathbf{p} = \left[\mathbf{s}_{\text{preamble}} \parallel \mathbf{s}^{\text{enc}}_{\text{header}} \parallel \mathbf{s}^{\text{enc}}_{\text{payload}} \right] \tag{32}$$

where $\mathbf{s}_{\text{preamble}}$ is the synchronisation preamble used for frame detection and timing recovery (see Section 5.2), $\mathbf{s}^{\text{enc}}_{\text{header}}$ is the 35-bit Hamming-encoded header, and $\mathbf{s}^{\text{enc}}_{\text{payload}}$ is the Hamming-encoded payload body. This two-level protection strategy ensures that the most critical metadata — packet length and codec selection —

benefits from unconditional error correction, while the payload error correction can be configured or disabled at runtime depending on channel conditions.

7 Cryptography

The integration of a cryptographic layer has not been pursued within the scope of this project, primarily owing to time constraints inherent to the breadth of the implementation work. Nevertheless, this section analyses the viability of two candidate approaches and discusses the practical limitations each presents in the context of the proposed system.

7.1 Signal Protocol

Given the instant-messaging nature of the underlying medium, the Signal Protocol (Cohn-Gordon et al., 2020; Marlinspike & Perrin, 2016a) was considered as a potential cryptographic foundation. The protocol combines the Extended Triple Diffie–Hellman (X3DH) key agreement (Marlinspike & Perrin, 2016b) with the Double Ratchet Algorithm to provide end-to-end encryption with forward secrecy and break-in recovery. Each session begins with an asynchronous multi-message handshake in which the communicating parties exchange ephemeral and identity key material to derive a shared session key without any prior shared secret.

Two principal obstacles preclude the adoption of this protocol in the current system.

- **Absence of a standalone reference implementation.** Signal Research LLC has deliberately refrained from releasing an independent, audited library implementing the protocol. The rationale is partly legal: by requiring adopters to produce their own implementation, Signal limits its liability for security deficiencies arising in third-party deployments. Consequently, any integration would require a custom made implementation of non-trivial cryptographic primitives, introducing both engineering overhead and the risk of subtle implementation errors.

- **Incompatibility with the audio-transport constraint.** The handshake phase of the Signal Protocol requires the asynchronous exchange of several key-

establishment messages prior to any application data being transmitted. In the system described here, every logical message must be modulated, encoded into an audio carrier, and delivered via WhatsApp voice notes. The latency and round-trip overhead entailed by the handshake would render the channel entirely impractical, even before any payload is communicated.

7.2 AES-GCM with Out-of-Band Key Exchange

A more tractable alternative is symmetric encryption via the Advanced Encryption Standard in Galois/Counter Mode (AES-GCM) (Dworkin, 2007). AES-GCM is an authenticated encryption with associated data (AEAD) scheme: it combines CTR-mode [2] stream encryption with a GHASH-based [3] authentication tag, providing simultaneously confidentiality and integrity protection for the ciphertext. With a 256-bit key, it offers a security level considered computationally infeasible to attack under current cryptanalytic knowledge (Dworkin, 2007).

Because AES-GCM is a symmetric scheme, both parties must possess an identical key prior to communication. While asymmetric key-transport protocols such as Diffie-Hellman or RSA key encapsulation would typically address this requirement, the audio-channel overhead discussed above makes any in-band key agreement equally impractical. A viable alternative is an out-of-band, physical key exchange: the shared key, together with any associated parameters (e.g. an initialisation vector seed or a key identifier), can be encoded in a QR code and exchanged between parties in person. The receiving device scans the code, stores the key material in a trusted local store, and subsequently uses it to decrypt incoming audio payloads without any further network interaction.

This approach sacrifices the convenience of automated key management in favour

[2] CTR mode (Counter mode) is a cryptographic mode of operation that turns a traditional block cipher (like AES) into a stream cipher.

[3] GHASH is a universal hash function operating over GF(2^{128}); it authenticates the ciphertext and any associated data by evaluating a polynomial whose coefficients are the input blocks at a secret hash key derived from the cipher, producing a 128-bit authentication tag.[4]

of operational simplicity and the elimination of any cryptographic handshake over the covert channel. While the requirement for physical proximity is admittedly a significant practical constraint, it is consistent with the overall design philosophy of the system: as a proof of concept for a covert audio steganography channel, usability has been deliberately subordinated to demonstrating technical feasibility.

8 Signal Obfuscation and Steganographic Masking

In the previous section we described how data is encoded into an audible waveform. In order to obfuscate this signal and minimise the PoD, the modulated waveform must be made perceptually indistinguishable from a genuine voice note transmitted over WhatsApp (2026). To achieve this, the encoded signal is mixed with a set of concatenated voice recordings drawn from a curated speech database, supplemented by additive white Gaussian noise (AWGN). Since the data is encoded using PSK, the carrier exhibits no amplitude modulation and no frequency variation; the perceptual character of the composite signal is therefore dominated by the voice component, with the steganographic carrier buried beneath it.

8.1 Voice Database: OpenSLR

To produce a realistic audio background, voice recordings are sourced from OpenSLR (Open Speech and Language Resources, Povey (2012))[5], a repository *"devoted to hosting speech and language resources, such as training corpora for speech recognition"*. The recordings provide a realistic acoustic foreground that masks the periodic structure of the PSK waveform.

After data modulation, the total number of samples N_{data} of the encoded waveform is known. The corresponding duration $d = N_{\text{data}}/f_s$, where f_s is the sample rate, determines how much voice material must be assembled. Voice recordings are concatenated using the greedy cover-selection procedure described in Algorithm 6: the algorithm greedily fills the required duration with the longest available clips first, appending the shortest available clip once no clip fits in the remaining gap, and terminates as soon as the accumulated duration meets or exceeds d. The resulting playlist is then shuffled to avoid a recognisable ordering of clips across repeated transmissions.

[5]The voice recording dataset used in this work is the LibriSpeech `dev-clean` subset, freely available at https://openslr.trmal.net/resources/12/dev-clean.tar.gz.

Algorithm 6: Greedy Voice Cover Selection

Input: $d \in \mathbb{R}_{>0}$: required cover duration (seconds);
$V = \{v_1, v_2, \ldots, v_n\}$: set of available clip durations, sorted in descending order

Output: A: ordered playlist of clip durations whose total $\geq d$

$A \leftarrow [\,]$;
while $d > 0$ **do**
    `// Greedy pass: take every clip that still fits`
    progress $\leftarrow$ `false`;
    for $v \in V$ **do**
        if $d - v > 0$ **then**
            $A \leftarrow A \parallel [v]$;
            $d \leftarrow d - v$;
            progress $\leftarrow$ `true`;
    `// No clip fits: close the gap with the shortest available clip`
    if not progress **then**
        $v_{\min} \leftarrow \min(V)$;
        $A \leftarrow A \parallel [v_{\min}]$;
        $d \leftarrow d - v_{\min}$;
Shuffle A uniformly at random;
return A

8.2 White Noise Generation

Additive white Gaussian noise is superimposed on the mix to further flatten the spectral signature of the composite signal. The noise samples are drawn from a zero-mean normal distribution $\mathcal{N}(0, \sigma^2)$, as formalised in Algorithm 7.

Algorithm 7: White Noise Generation

Input: $N \in \mathbb{Z}_{\geq 0}$: number of samples;

$\mu \in \mathbb{R}$: distribution mean (default 0.0);

$\sigma \in \mathbb{R}_{\geq 0}$: standard deviation (default 1.0);

$s \in \mathbb{Z} \cup \{\texttt{None}\}$: optional random seed

Output: $\mathbf{w} \in \mathbb{R}^N$: white noise waveform

`// Input validation`

if $N < 0$ **then return** raise `ValueError`;

if $\mu \notin \mathbb{R}$ (non-finite) **then return** raise `ValueError`;

if $\sigma < 0$ or $\sigma \notin \mathbb{R}$ (non-finite) **then return** raise `ValueError`;

`// Edge case`

if $N = 0$ **then return** $[\,]$;

`// Noise generation`

Initialise PRNG $\mathcal{G}$ with seed s;

$\mathbf{w} \leftarrow \mathcal{G}\big(\mathcal{N}(\mu,\, \sigma^2),\; N\big)$;

return $\mathbf{w}$

8.3 Signal Mixing and Gain Structure

The three components voice cover $\mathbf{v}$, encoded data signal $\mathbf{x}$, and white noise $\mathbf{w}$ are combined into the final composite waveform $\mathbf{y}$ according to:

$$\mathbf{y} = \mathbf{v} + g_x\,\mathbf{x} + g_n\,\mathbf{w} \tag{33}$$

where $g_x \in [0, 1]$ and $g_n \in [0, 1]$ are scalar gain factors applied to the data signal and noise respectively, normalised so that a value of 1.0 corresponds to the full-scale amplitude of the source signal. The voice component is kept at unity gain ($g_v = 1.0$) and constitutes the dominant perceptual element of the transmission. The noise gain is fixed at $g_n = 0.01$ (i.e. 1% of full scale), a level sufficient to perturb the spectral baseline without introducing audible hiss. The data signal gain g_x was

determined empirically: the minimum value at which the receiver can still reliably demodulate and decode the payload was identified through systematic testing, and that value is used for transmission in order to minimise the acoustic footprint of the steganographic carrier.

9 Implementation and Experimental Methodology

This section describes the prototype implementation of the proposed system and the experimental methodology used to evaluate its performance.

Programming Language The implementation language is Python (Van Rossum & Drake, 2009), chosen for its rapid prototyping capabilities and the breadth of its scientific computing ecosystem. The iterative nature of the development process (i.e., exploring multiple parameter configurations and signal processing strategies) benefited greatly from Python's concise syntax and interactive tooling. The following third-party libraries were employed throughout the implementation:

- **NumPy** Harris et al. (2020): used for vectorised array arithmetic and as the primary numerical backend for most algorithms.
- **SciPy** Virtanen et al. (2020): used for digital signal filtering, specifically the Butterworth filter design routine (`butter`) and the zero-phase filtering functions `filtfilt` and `sosfiltfilt`.
- **Matplotlib** Hunter (2007): used for generating diagnostic plots and visualising signal waveforms and error rate distributions.
- **Pandas** McKinney (2010): used for result aggregation, data filtering, and writing experimental output to CSV files for subsequent analysis.
- **CuPy** Okuta et al. (2017): used as a drop-in GPU accelerated replacement for NumPy in the preamble detection algorithm (Algorithm 3). By offloading vector operations to the GPU via the CUDA toolkit, CuPy substantially reduces execution time when testing configurations with large numbers of periods per symbol, where the cross-correlation search space is widest.
- **Playwright** Microsoft Corporation (2020): used to automate interaction with WhatsApp Web, including voice note recording and audio file download, as detailed in Section 9.1.1.

Tools FFmpeg (FFmpeg Developers, n.d.) was used for all audio format conversions. The voice recording database was stored in FLAC format, whilst audio files downloaded from WhatsApp (2026) are encoded in the OGG Opus (Valin et al., 2012) format. FFmpeg was used to convert both formats to WAV, which can be read directly by Python's standard audio libraries.

Jupyter notebooks were used during exploratory development to iterate rapidly over plots and result filtering, providing an interactive environment well suited to parameter tuning and visualisation tasks.

Implementation Guidelines A key design principle throughout development was the maintenance of strict modularity. The system encompasses several largely independent concerns: encoding, decoding, modulation, demodulation, noise generation, and voice mixing. Keeping each as a self-contained module made it straightforward to test, replace, or extend individual components without affecting the rest of the pipeline. Multiple entry-point scripts were written to compose these modules into complete workflows for different experimental scenarios.

A functional programming style was adopted for the majority of the codebase, with stateless functions operating over NumPy arrays forming the core of the signal processing pipeline. Object-oriented patterns were reserved for configuration management and testing infrastructure, where encapsulation improved reusability and gave the test harness a more framework-like structure. Python's type annotation syntax was used consistently throughout to facilitate static analysis and reduce type-related defects.

9.1 Experimental Methodology

In order to evaluate the proposed system, two experimental scenarios were defined: a *best-case scenario*, which serves as a controlled baseline, and a *WhatsApp scenario*, which reflects the realistic deployment conditions of the system.

Best-case scenario This scenario follows these steps:

1. Take input data and generate a composite audio file as described in Section 8.
2. Demodulate and decode the generated file directly, without any intermediate transmission step.

Because no lossy compression or transmission artifacts are introduced, this scenario establishes an upper bound on achievable performance. It is particularly useful for isolating the influence of the individual signal mixing parameters (i.e., carrier frequency, voice volume gain, and noise volume gain) on the error rate, independently of the degradation introduced by WhatsApp. The simplicity of the pipeline also makes it straightforward to parallelise across a large parameter grid.

WhatsApp scenario This scenario follows these steps:

1. Configure a virtual microphone as the default audio input device in the browser.
2. Take input data and generate a composite audio file as described in Section 8.
3. Open WhatsApp Web and navigate to the target recipient's chat.
4. Begin recording a voice note.
5. Play the audio file generated at Step 2 through the virtual microphone.
6. Once playback concludes, stop the voice note recording on WhatsApp Web and allow it to be sent.
7. Download the transmitted audio file.
8. Demodulate and decode the downloaded file.

This is the primary evaluation scenario, reflecting the actual intended use of the system. Since the experimental parameter grid required sending in excess of 500 voice notes per test run, manual execution was not feasible, and the entire WhatsApp interaction pipeline was automated. Because only a single WhatsApp account was available, the sending and downloading stages cannot be parallelised;

however, the decoding and demodulation stage is independent of WhatsApp and was parallelised to reduce total evaluation time.

9.1.1 WhatsApp Automation

WhatsApp Web interaction was automated using Playwright (Microsoft Corporation, 2020), a browser automation library that drives a modified Chromium instance to simulate user actions programmatically.

A critical aspect of the automation is the suppression of audio processing features that modern browsers apply by default to microphone input. These include echo cancellation, automatic gain control, noise suppression, and high-pass filtering. All of these features are intended to improve the quality of voice and video calls, but they distort the waveform of the injected signal and thereby degrade demodulation performance. Two complementary mechanisms were used to disable them.

First, Chromium-level flags were passed at launch time:

```
context = p.chromium.launch_persistent_context(
    user_data_dir=str(config.profile_dir),
    headless=config.headless,
    args=[
        "--start-maximized",
        "--goog-echo-cancellation=false",
        "--goog-auto-gain-control=false",
        "--goog-noise-suppression=false",
        "--goog-highpass-filter=false",
    ],
    viewport={"width": 1270, "height": 720},
)
```

Second, a script injected into every page via `add_init_script` intercepts calls to `navigator.mediaDevices.getUserMedia` and enforces the disabling of WebRTC-level audio constraints before the stream is acquired by WhatsApp:

```
context.add_init_script("""
    const origGUM = navigator.mediaDevices.getUserMedia.bind(
        navigator.mediaDevices
```

```
    );
    navigator.mediaDevices.getUserMedia = function(constraints) {
        if (constraints && constraints.audio) {
            if (typeof constraints.audio === 'boolean') {
                constraints.audio = {};
            }
            constraints.audio.autoGainControl = false;
            constraints.audio.noiseSuppression = false;
            constraints.audio.echoCancellation = false;
        }
        return origGUM(constraints);
    };
""")
```

The injected script is necessary because WhatsApp Web may re-enable gain control at the WebRTC layer (Jennings et al., 2025) regardless of the browser-level flags. While amplitude distortion is of secondary concern for a PSK-based system, which encodes information in phase rather than amplitude, maintaining a consistent signal level throughout transmission reduces a potential source of variability in the experimental results.

9.2 On the Use of Real WhatsApp Transmission

Opus (Valin et al., 2012) is an open, royalty-free audio codec for which multiple free and open-source implementations exist. It might therefore seem feasible to compress audio locally using one such implementation, bypassing WhatsApp entirely, and thereby avoiding the constraints of a single-account sequential pipeline.

However, Opus exposes a large number of tunable parameters (e.g., bitrate, frame duration, complexity, and packet loss resilience) and determining the exact configuration employed by WhatsApp is not trivial. Even if locally compressed audio were perceptually indistinguishable from a WhatsApp voice note, any parametric mismatch between the two would contaminate the experimental results, since the measured error rates would reflect a compression profile that does not correspond

to the actual deployment scenario. Validating such an approach would itself require a separate verification system to certify that locally generated files are sufficiently faithful reproductions of authentic WhatsApp output.

For these reasons, the experimental pipeline was designed to use real WhatsApp transmission throughout, ensuring that all results are obtained under conditions that are as representative of the actual user interaction model as possible.

10 Experimental Evaluation and Results

This section presents the experiments conducted throughout the project and their results. Multiple experiments were carried out iteratively during development, allowing each design choice to be evaluated and refined progressively. This thesis reports the results of the final configuration.

Fixed Variables Several system parameters were held constant across all experiments, either due to hardware constraints or deliberate design decisions. These are described below.

- **Sample rate**: fixed at 48,000 Hz, determined by the audio interface of the machine used for all experiments.
- **Noise gain**: fixed at 0.01, as discussed in Section 8.3. This value was established empirically: maintaining the AWGN gain below the data signal gain was found to preserve a low bit error rate. A low value was deliberately chosen to avoid audible degradation of the carrier voice recording, which is essential to the steganographic concealment requirement of the system.
- **Seed**: fixed at 12345, applied uniformly to all stochastic processes in the system. Using a fixed seed ensures reproducible audio outputs across independent experiment runs, making comparisons between parameter configurations valid. Setting the seed to `None` switches the system to a fully non-deterministic mode.
- **Preamble**: fixed as the 13 symbol sequence as discussed in Section 5.2.

```
[+1, +1, +1, +1, +1, -1, -1, +1, +1, -1, +1, -1, +1]
```

10.1 BPSK and DBPSK Comparison

The experiment grid was defined over the following four parameters:

$$\begin{aligned} f &= \{200, 222, 244, 266, 288, 311, 333, 355, 377, 400\}\,\text{Hz} \\ v &= \{0.25,\ 0.4125,\ 0.575,\ 0.7375,\ 0.9\} \\ m &= \{m_1,\ m_2,\ m_3,\ m_4,\ m_5\} \\ T &= \{2,\ 5,\ 7\} \end{aligned} \tag{34}$$

where f is the carrier frequency, v is the volume gain applied to the data signal, m is the test message payload, and T is the number of periods per symbol. The five test payloads are:

- m_1: `Hi!`
- m_2: `Hello world, this is a 123456 message!`
- m_3:

  ```
  Special symbols: @#$%^&*+=_|~`<>/.,;:!?-()[]{}\
  ```

- m_4:

  ```
  Line breaks test:\nFirst line\nSecond line\nThird line\nTabs and
  ↪ spacing: col1\tcol2\tcol3
  ```

- m_5:

  ```
  Long message: This is a significantly longer test payload
  ↪ intended to simulate realistic secret content with multiple
  ↪ sentence-like fragments, numbers like 2026, and enough
  ↪ characters to stress encoding/decoding across different
  ↪ audio samples.
  ```

The full experiment set E is the Cartesian product of all parameter sets:

$$E = f \times v \times m \times T, \tag{35}$$

yielding $|E| = 10 \times 5 \times 5 \times 3 = 750$ distinct experimental configurations, each corresponding to a uniquely generated audio file. Since both BPSK and DBPSK are evaluated, the total number of generated audio files is $2|E| = 1500$.

10.1.1 BPSK Results

We begin the analysis of BPSK performance by examining the 3D plots in Figures 5, 6, and 7, each corresponding to a different number of cycles per symbol (period 2, 5, and 7, respectively). Each plot displays the parameter space over carrier frequency (x-axis) and volume gain (y-axis), with the z-axis representing the used message and the color encoding the BER. Across all three figures, a consistent trend is visible: longer messages incur higher error rates, regardless of the chosen frequency or gain.

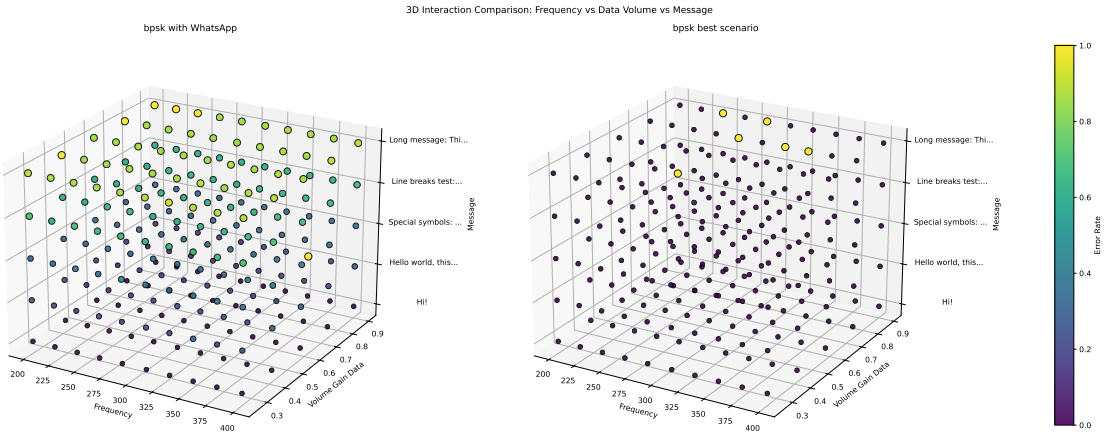


Figure 5: 3D plot for BPSK algorithm with period 2

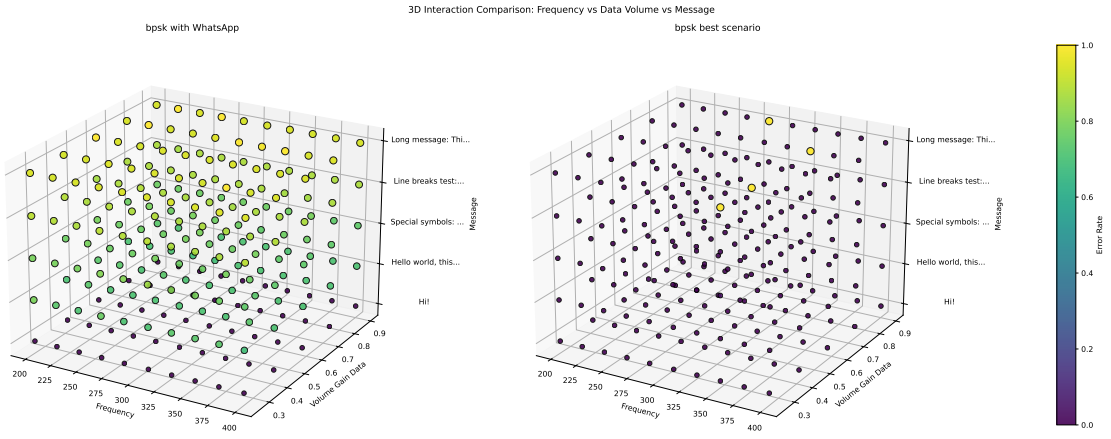


Figure 6: 3D plot for BPSK algorithm with period 5

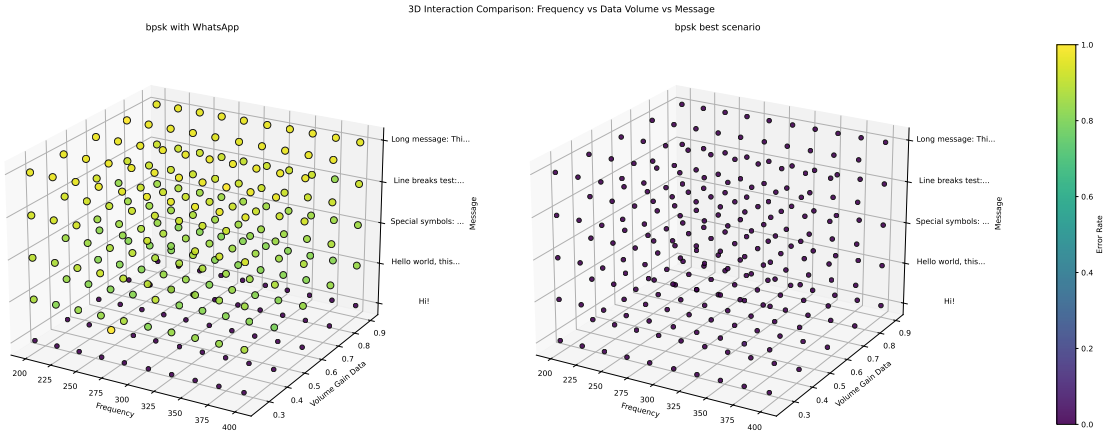


Figure 7: 3D plot for BPSK algorithm with period 7

A clear degradation in performance is also observed as the period increases under the WhatsApp channel, as confirmed by the mean BER values in Table 1: period 2 yields a mean BER of 0.4121, rising to 0.6618 and 0.7159 for periods 5 and 7, respectively. This behaviour is consistent with the slow phase drift introduced by the Opus codec (Valin et al., 2012): longer symbols occupy more time and therefore accumulate greater phase error. In the best scenario, however, the absence of codec-induced drift reverses this trend, and longer periods achieve lower error rates due to the increased energy per symbol.

Given these results, the remainder of the BPSK analysis focuses on period 2, as

it yields the most robust performance under the WhatsApp channel. All plots for periods 5 and 7 are reported in Appendix A for completeness.

Period	WhatsApp	Best Scenario
2	0.4121	0.0243
5	0.6618	0.0164
7	0.7159	0.0003

Table 1: Mean BPSK error.

Period	WhatsApp	Best Scenario
2	0.1015	0.0235
5	0.1164	0.0158
7	0.1279	1e-05

Table 2: Variance in BPSK error.

Figures 8–12 present a series of heatmaps, each corresponding to a message of increasing length. In each heatmap, the x-axis represents the carrier frequency, the y-axis the volume gain, and the cell colour encodes the BER for that parameter combination. This view allows the joint effect of frequency and gain to be assessed at a glance for a fixed payload, and the progression across figures illustrates how the error landscape evolves as message length grows.

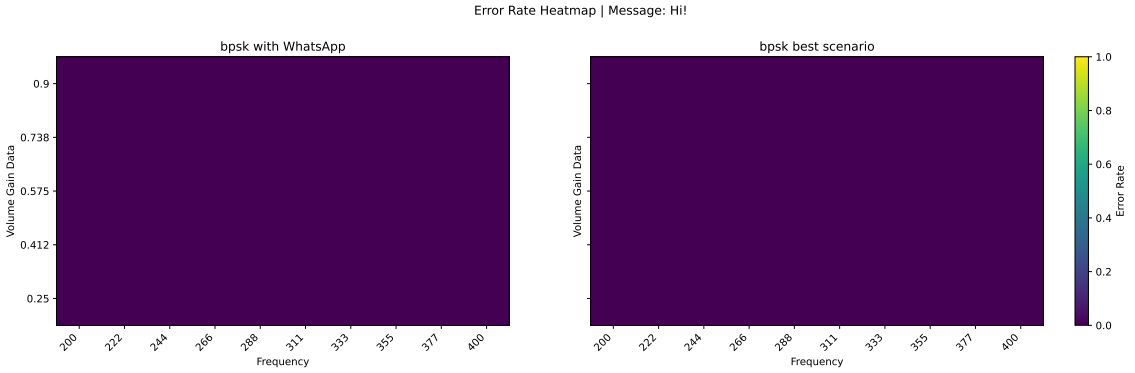


Figure 8: Heatmap for BPSK algorithm with period 2

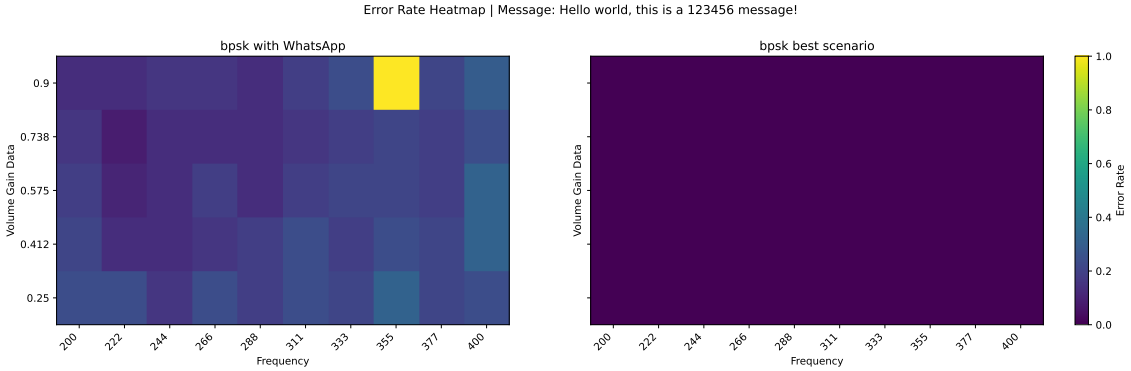


Figure 9: Heatmap for BPSK algorithm with period 2

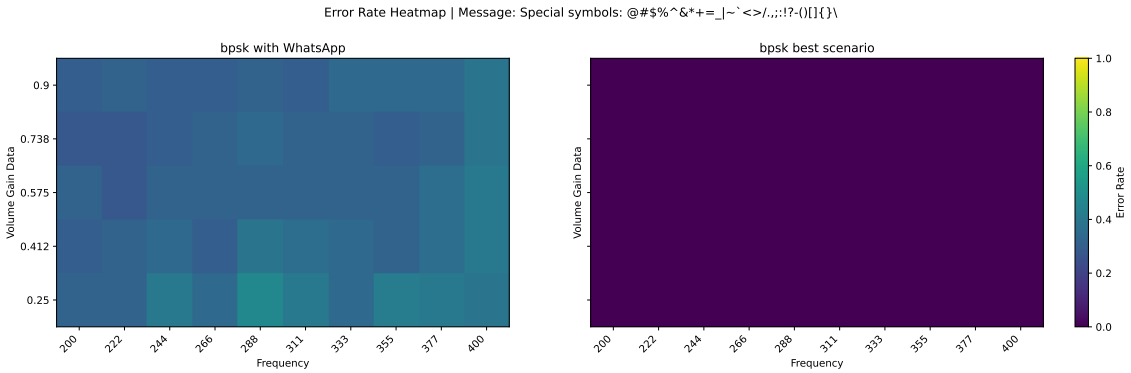


Figure 10: Heatmap for BPSK algorithm with period 2

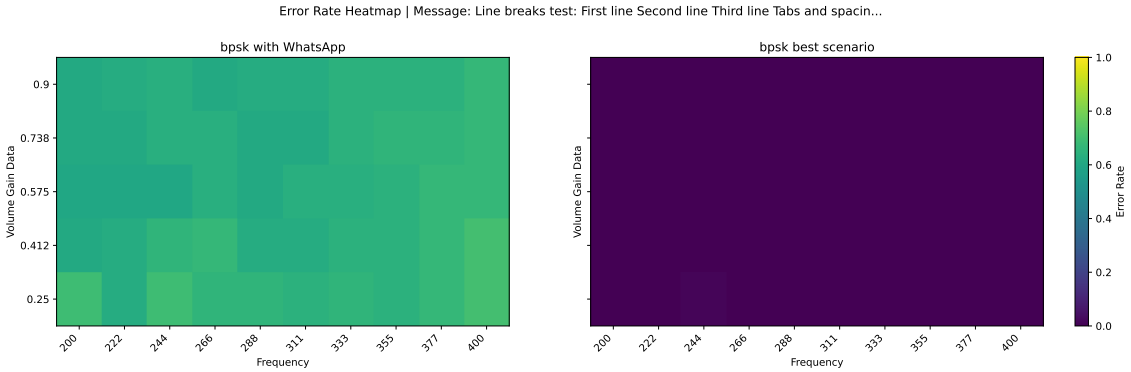


Figure 11: Heatmap for BPSK algorithm with period 2

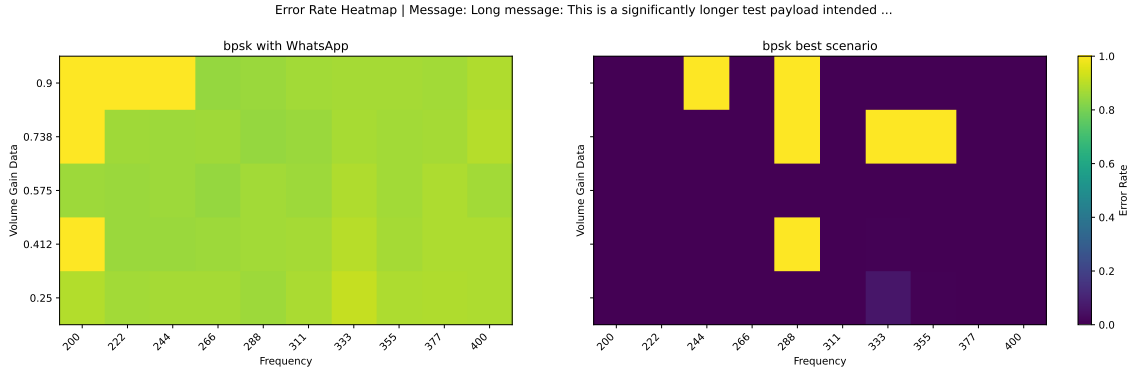


Figure 12: Heatmap for BPSK algorithm with period 2

Figures 13–17 show the corresponding line plots, one per message length, where each line represents a distinct carrier frequency and the x-axis reports the volume gain. The vertical position of each line indicates the BER at that gain level. Notably, the lines do not exhibit a consistent ordering by frequency: no carrier frequency systematically outperforms or underperforms the others across the tested gain range. This suggests that, under the WhatsApp channel with BPSK modulation, carrier frequency is not a significant determinant of error rate, and that volume gain is the more influential parameter to optimise.

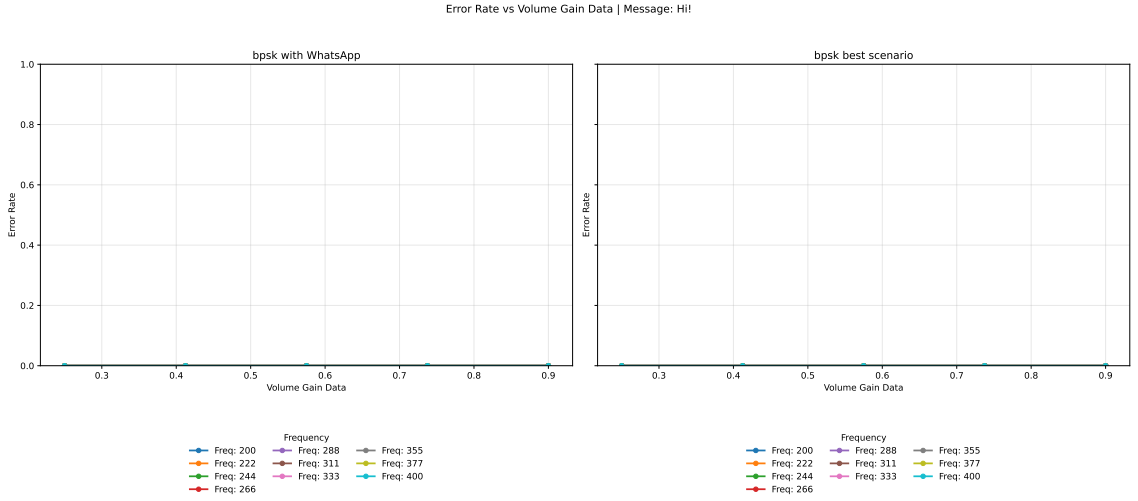


Figure 13: Line plot for BPSK algorithm with period 2

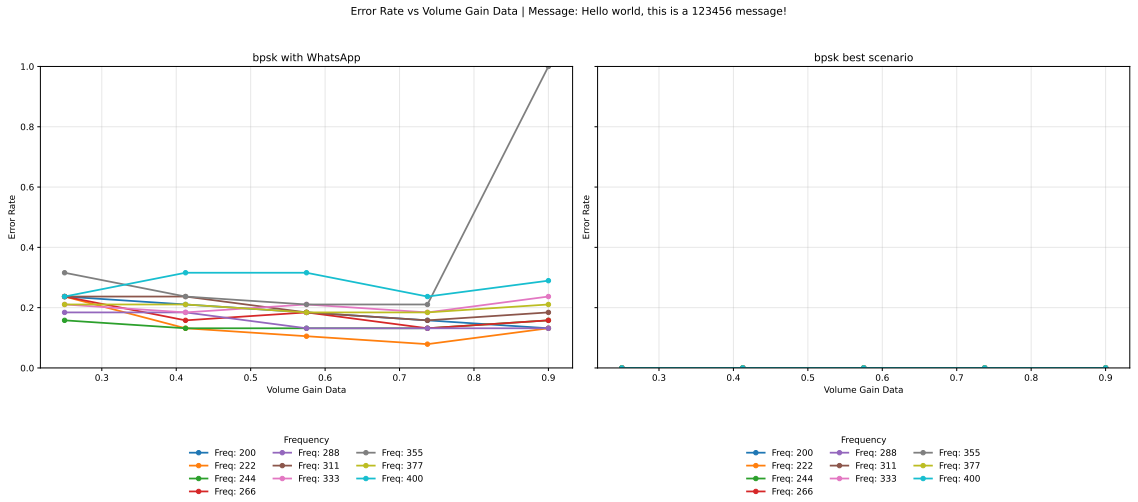


Figure 14: Line plot for BPSK algorithm with period 2

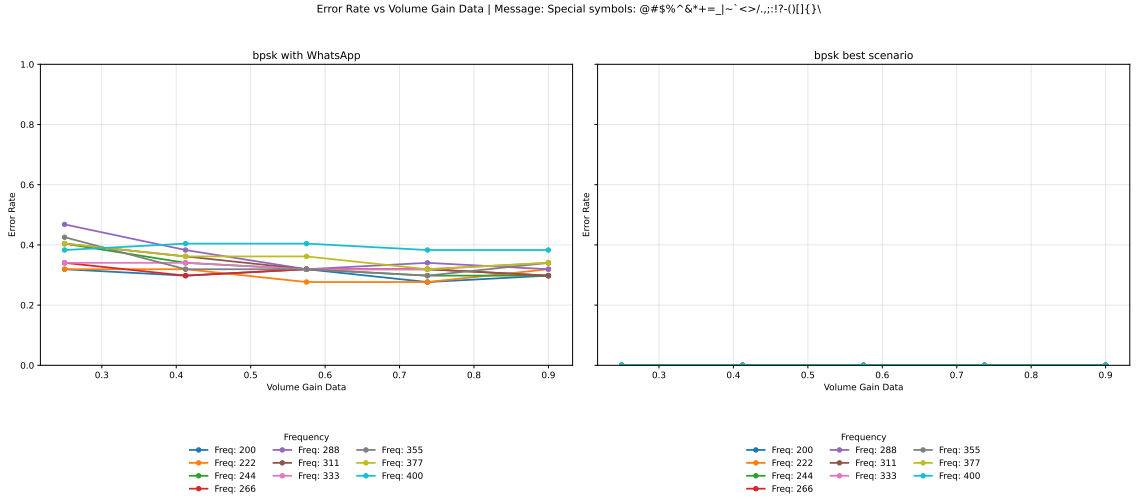


Figure 15: Line plot for BPSK algorithm with period 2

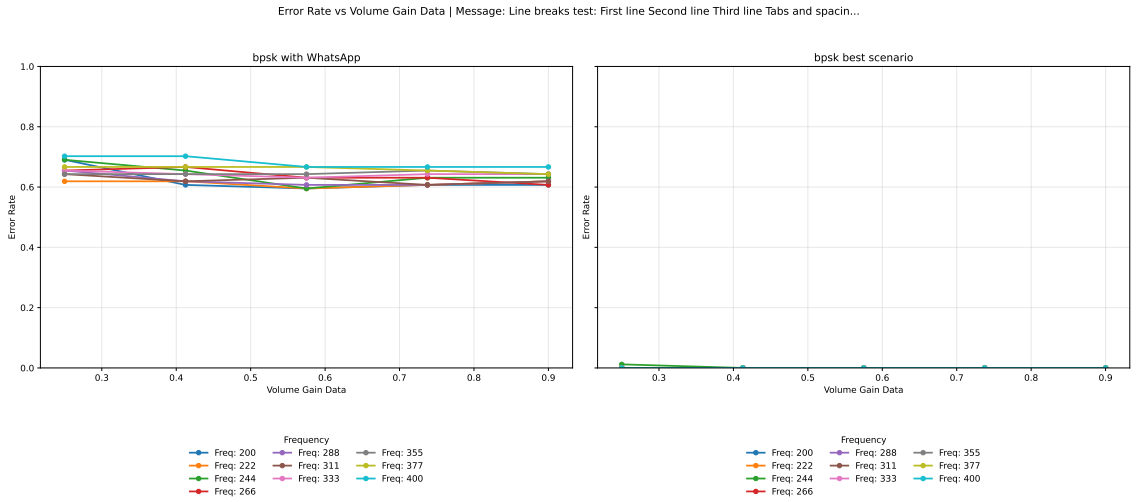


Figure 16: Line plot for BPSK algorithm with period 2

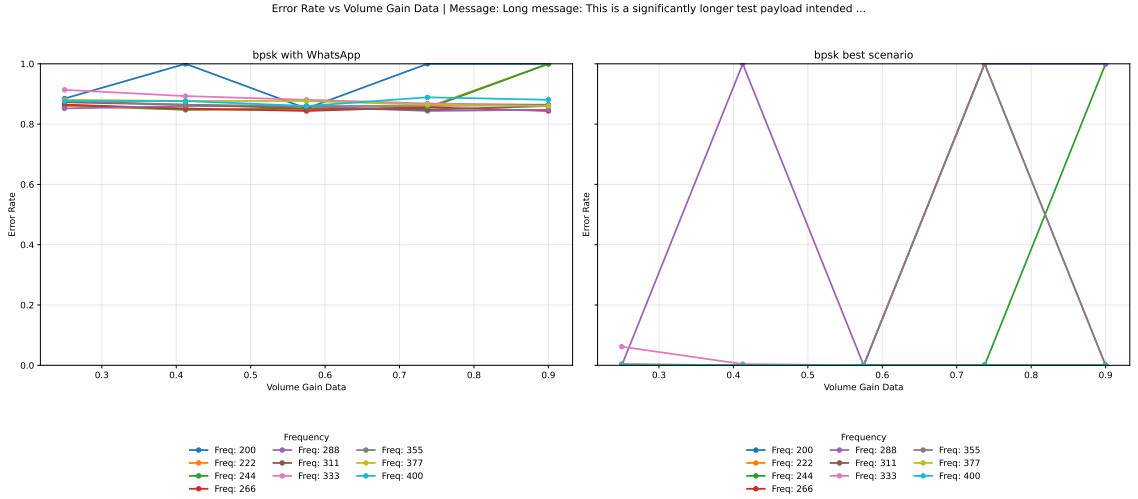


Figure 17: Line plot for BPSK algorithm with period 2

Figures 18–22 present violin plots of the BER distribution, one per message. Each violin corresponds to a carrier frequency and aggregates the error rates observed across all tested volume gain levels, thus summarizing the BER spread attributable to gain variation at that frequency. The width of each violin at a given BER value reflects the density of observations at that error level. This representation complements the heatmaps by making the spread and skewness of the error distribution visible, rather than reporting only a single aggregate statistic.

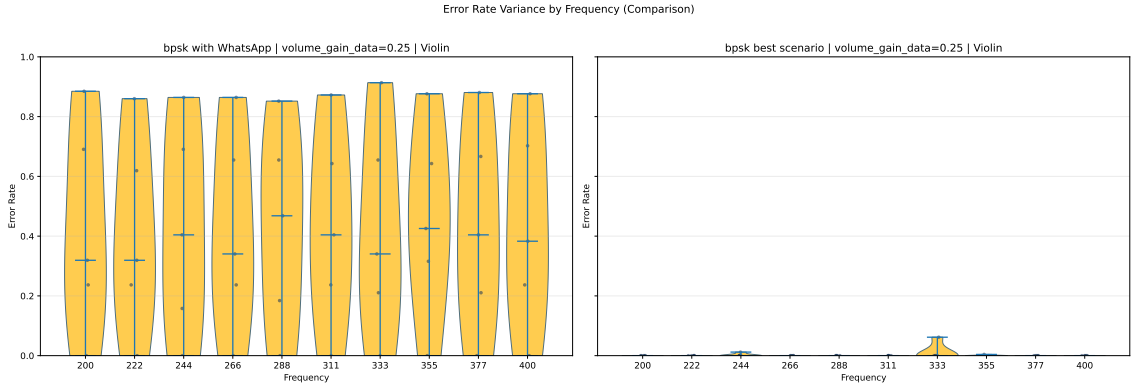


Figure 18: Violin plot for BPSK algorithm with period 2

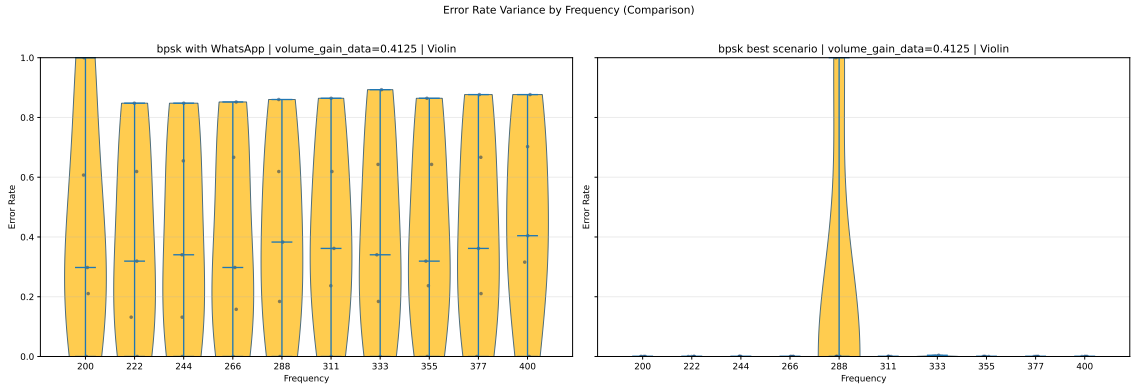


Figure 19: Violin plot for BPSK algorithm with period 2

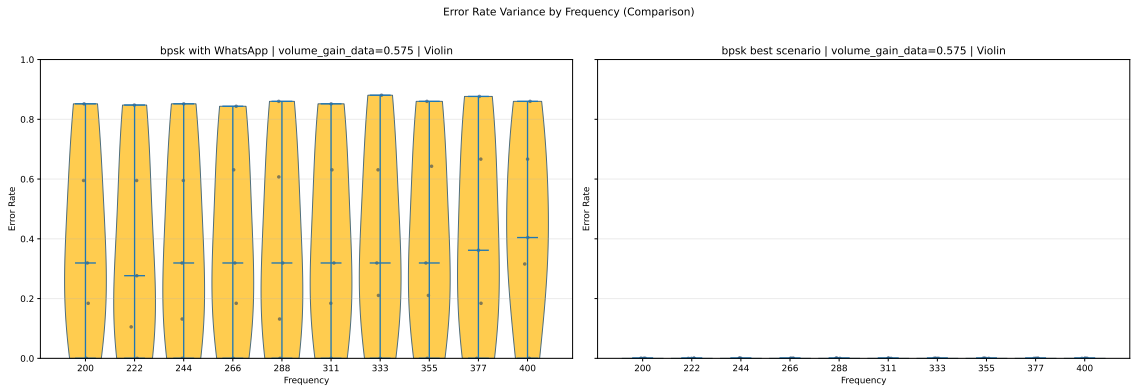


Figure 20: Violin plot for BPSK algorithm with period 2

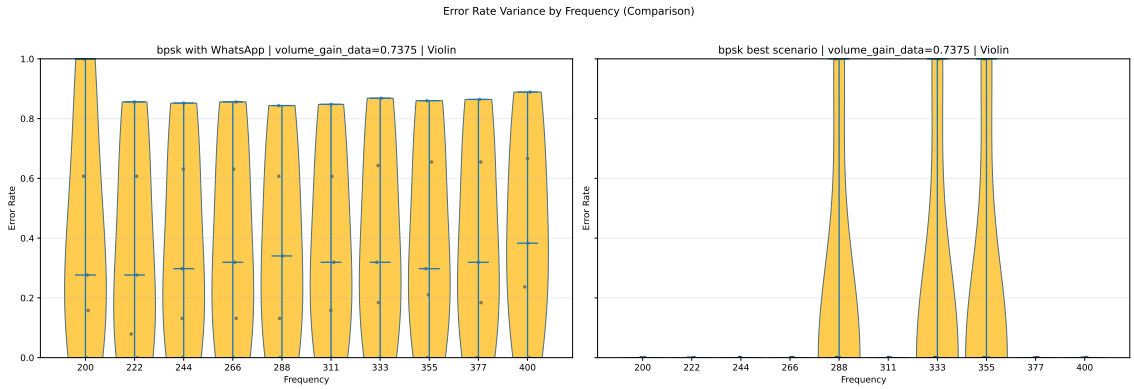


Figure 21: Violin plot for BPSK algorithm with period 2

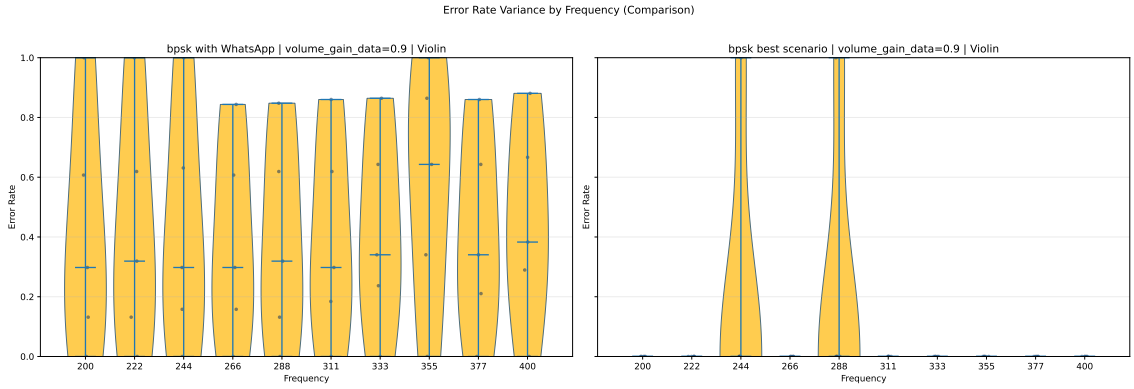


Figure 22: Violin plot for BPSK algorithm with period 2

To confirm that phase drift is the primary driver of performance degradation, debugging waveform data extracted during the decoding process was analyzed. Both the WhatsApp and "Best Scenario" waveforms were aligned starting from the beginning of the preamble, with all preceding data truncated. Message m_5 with a period of 7 was selected to isolate and highlight the maximum cumulative drift.

Figure 23 illustrates the initial segment of the transmission, demonstrating that the WhatsApp waveform aligns closely with the ideal reference, exhibiting negligible phase variance. Conversely, Figure 24 captures the terminal segment of the same transmission, revealing a stark phase divergence in the WhatsApp signal relative to the best-case scenario. This visible misalignment over time confirms that the time-varying phase drift introduced by the Opus codec (Valin et al., 2012) progressively

corrupts symbol synchronization, directly leading to the elevated BER observed in longer messages and periods.

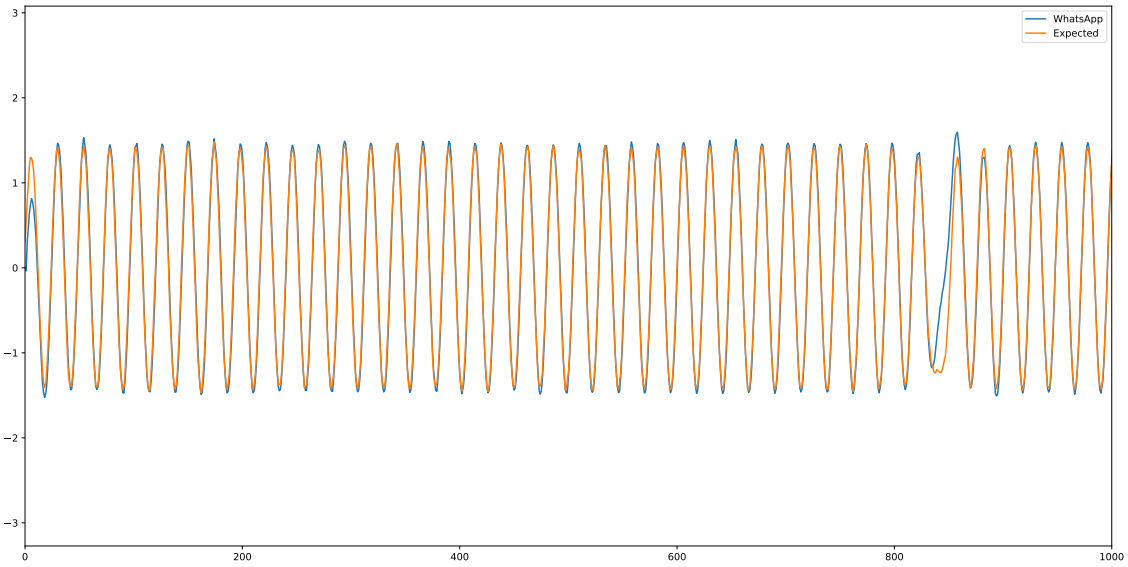


Figure 23: Waveform alignment at the transmission onset for message m_5 (period 7), showing tight phase coherence between WhatsApp and the Best Scenario.

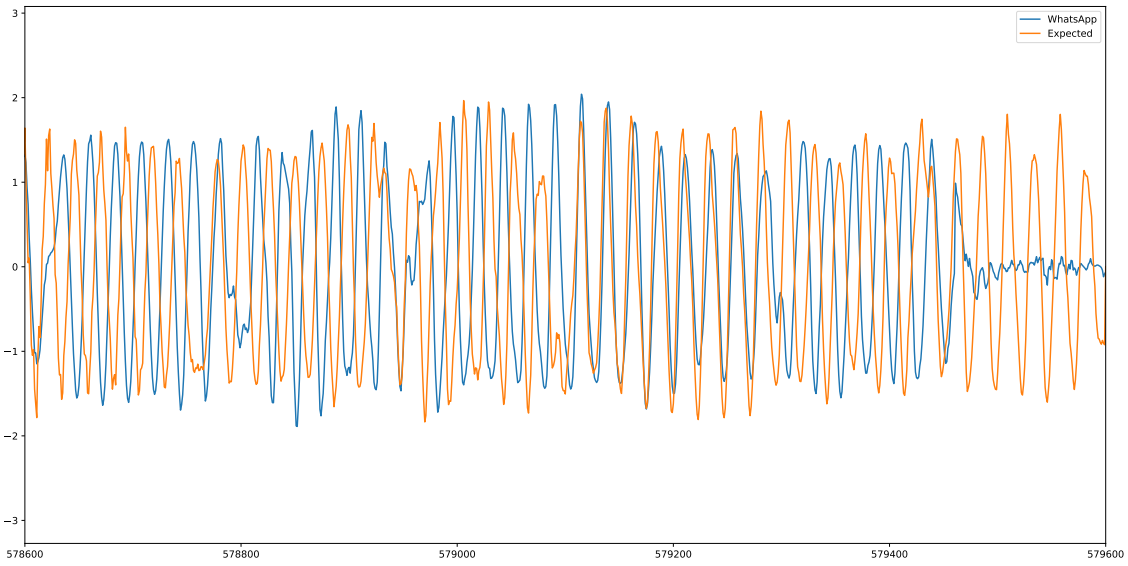


Figure 24: Waveform alignment at the transmission termination for message m_5 (period 7), illustrating severe cumulative phase drift in the WhatsApp channel compared to the Best Scenario.

10.1.2 DBPSK Results

Figures 25, 26, and 27 present the 3D error landscapes for DBPSK modulation across periods 2, 5, and 7, respectively, using the same axis conventions as the BPSK plots. The most striking feature visible across all three figures is a sharp degradation in performance for carrier frequencies above 250 Hz: in that region, the BER approaches its maximum regardless of volume gain or message length. This behaviour is consistent with the frequency-dependent attenuation introduced by the Opus codec (Valin et al., 2012), which progressively suppresses signal energy at higher frequencies. As a result, the preamble and symbol energy at those frequencies are effectively destroyed before the receiver can process them.

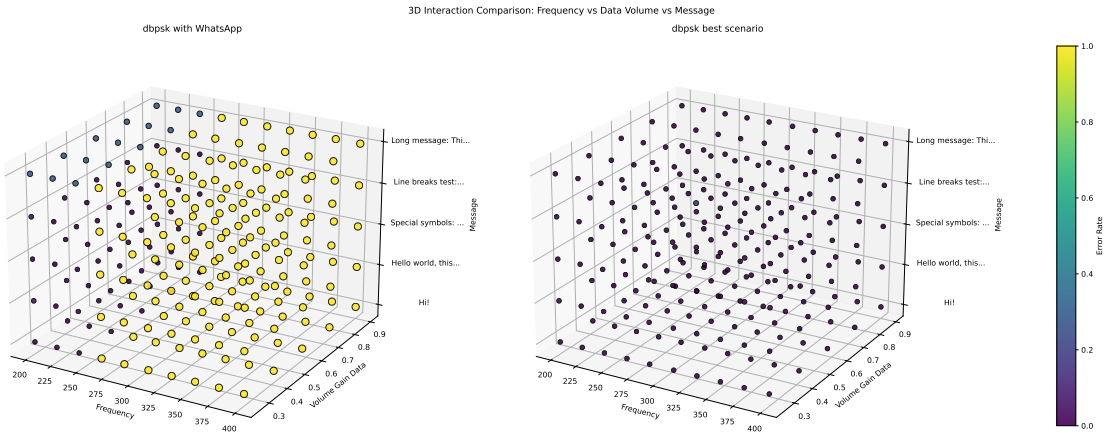


Figure 25: 3D plot for DBPSK algorithm with period 2

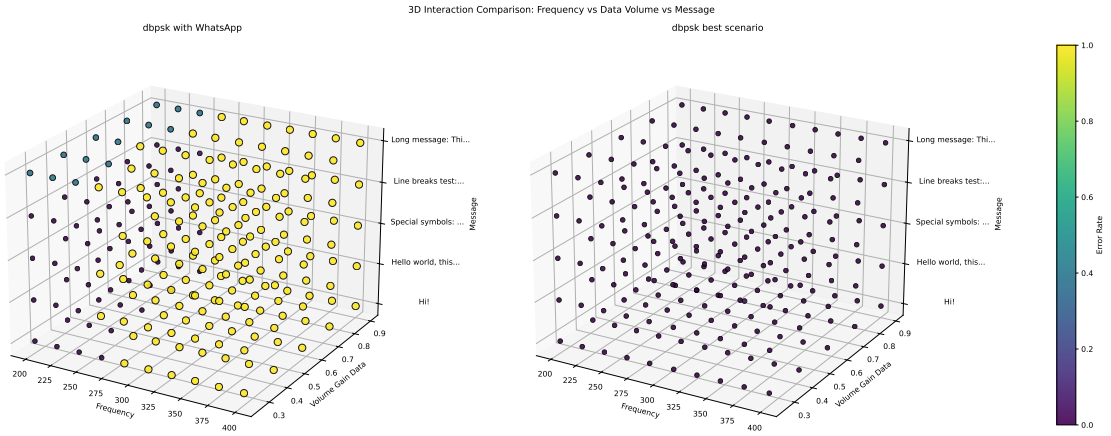


Figure 26: 3D plot for DBPSK algorithm with period 5

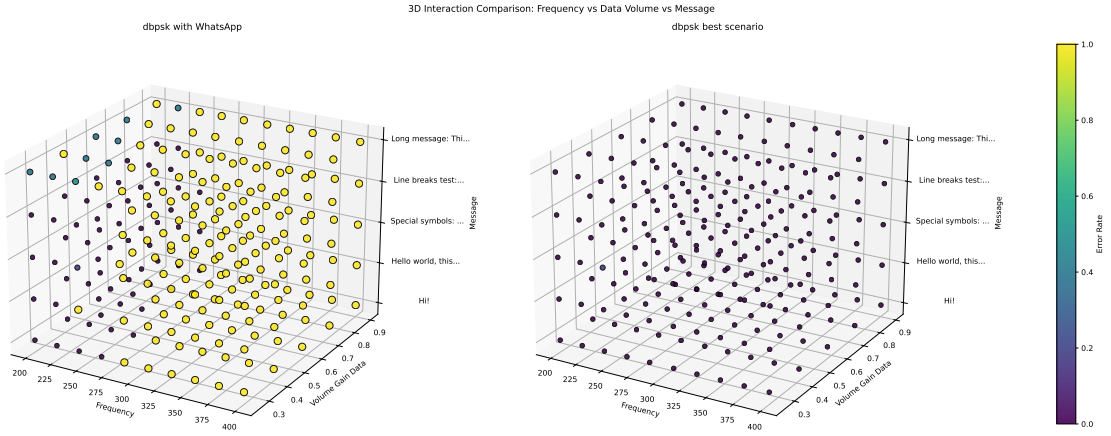


Figure 27: 3D plot for DBPSK algorithm with period 7

The aggregate statistics in Tables 3 and 4 reflect this: the mean BER under WhatsApp hovers around 0.72 for all three periods, with relatively high variance. These figures are substantially worse than the corresponding BPSK results and, taken in isolation, would suggest DBPSK is an inferior modulation choice. However, this conclusion is misleading, as the aggregate statistics are dominated by the failed high-frequency trials.

Period	WhatsApp	Best Scenario
2	0.7223	0.0018
5	0.7257	0.0003
7	0.7312	0.0009

Table 3: Mean DBPSK error.

Period	WhatsApp	Best Scenario
2	0.1884	0.0001
5	0.1863	0.0
7	0.1881	0.0001

Table 4: Variance in DBPSK error.

A clearer picture emerges from the heatmaps in Figures 28–32, which display the BER as a function of frequency and volume gain for messages of increasing length. The frequency cutoff at 250 Hz is clearly visible as a boundary separating near-total failure above it from substantially lower error rates below it. This motivates the exclusion of frequencies above 250 Hz from the subsequent analysis, as those operating points are not viable under the WhatsApp channel.

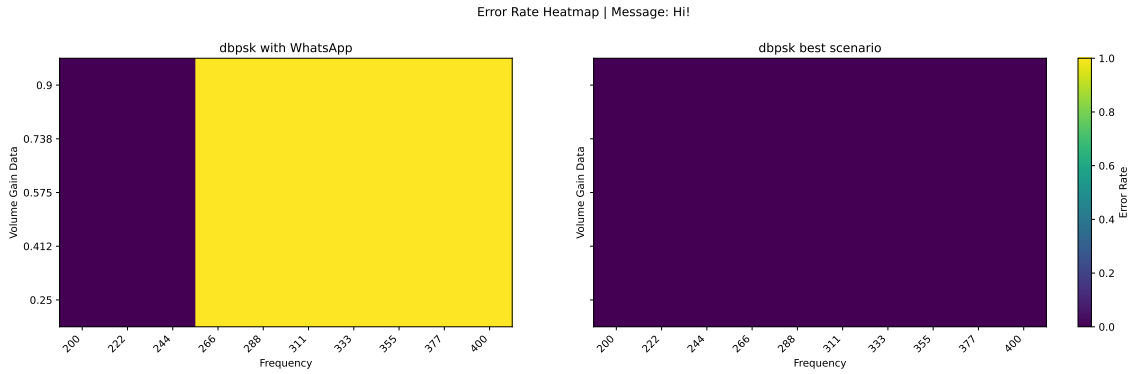


Figure 28: Heatmap plot for DBPSK algorithm with period 2

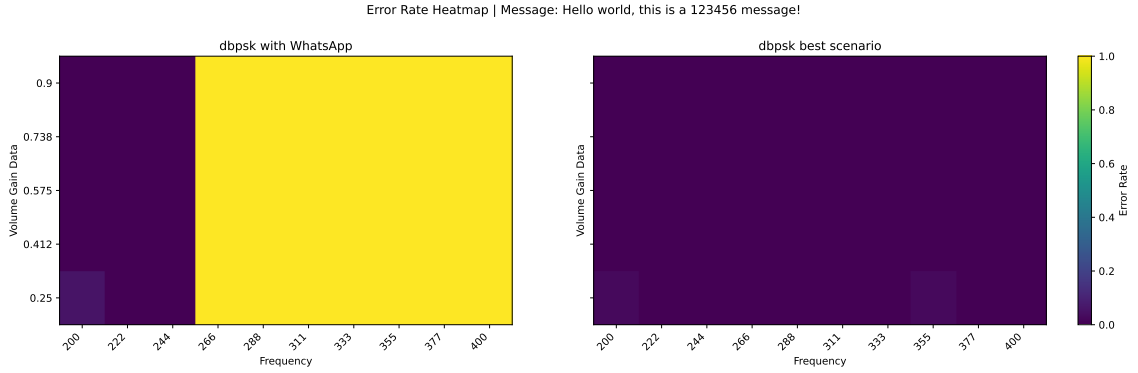


Figure 29: Heatmap plot for DBPSK algorithm with period 2

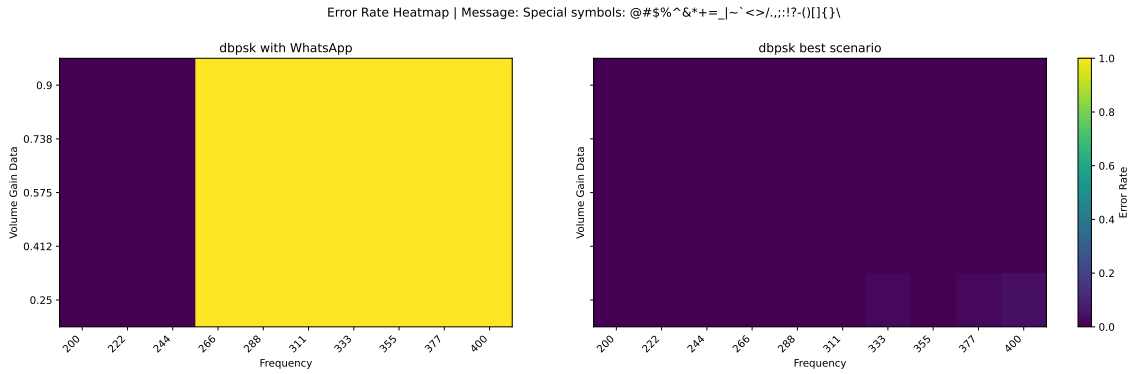


Figure 30: Heatmap plot for DBPSK algorithm with period 2

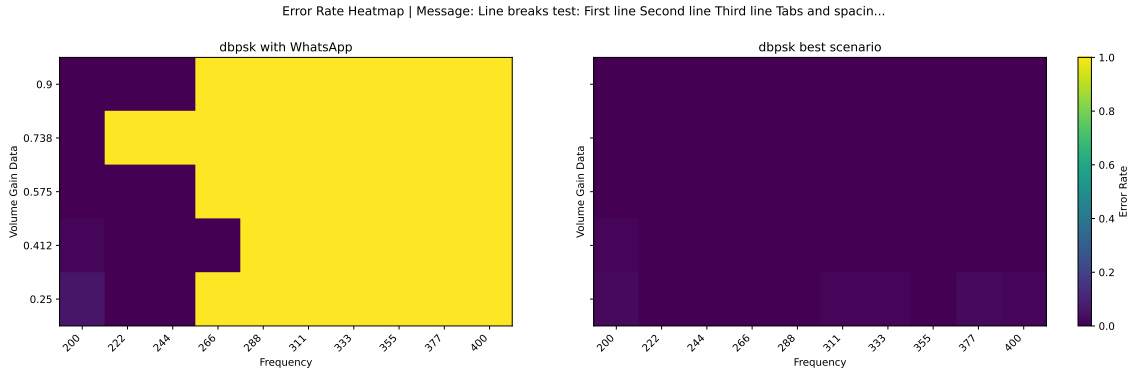


Figure 31: Heatmap plot for DBPSK algorithm with period 2

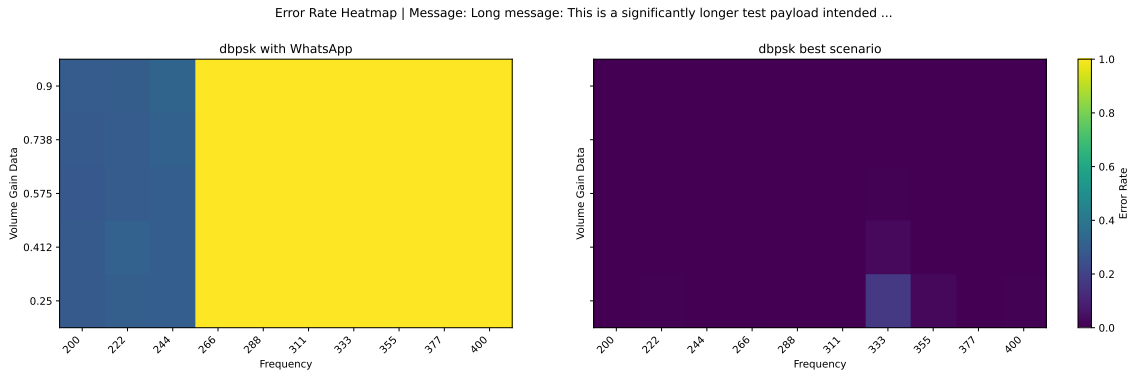


Figure 32: Heatmap plot for DBPSK algorithm with period 2

Tables 5 and 6 report the mean and variance of the BER restricted to frequencies at or below 250 Hz. The results are markedly different: the mean BER for period 2 drops to 0.0876, compared to 0.4121 for BPSK at the same period, demonstrating the advantage of differential encoding in mitigating the phase drift introduced by the Opus codec. The period-dependent degradation observed in BPSK is also present here, though less severe: the mean BER rises from 0.0876 to 0.1841 as the period increases from 2 to 7, a smaller relative increase than in the BPSK case. The low variance values, particularly for periods 2 and 5, further indicate that DBPSK produces more consistent results within the viable frequency range. A more detailed analysis of the frequency-dependent behaviour below 250 Hz is presented in Section 10.1.3, where it is shown that lower carrier frequencies yield systematically better performance.

Period	WhatsApp	Best Scenario
2	0.0876	0.0009
5	0.1258	0.0003
7	0.1841	0.0023

Table 5: Mean DBPSK error.

Period	WhatsApp	Best Scenario
2	0.0372	0.0
5	0.066	0.0
7	0.121	0.0004

Table 6: Variance in DBPSK error.

10.1.3 DBPSK Investigation

This section presents a focused investigation of DBPSK performance restricted to the viable frequency range identified in the previous section. The experiment parameters are:

$$\begin{aligned} f &= \{150, 200, 250\}\,\text{Hz} \\ v &= \{0.25, 0.5, 0.75, 1.0\} \\ m &= \{m_1,\, m_2,\, m_3,\, m_4,\, m_5\}^{10} \\ T &= \{2\} \end{aligned} \tag{36}$$

The full experiment set is defined as the Cartesian product described in Equation 35, yielding $|E| = 3 \times 4 \times 5 \times 10 \times 1 = 600$ trials. Compared to the previous grid, each of the five messages is repeated 10 times at every parameter combination. This repetition is motivated by the stochastic nature of certain system parameters. Averaging over multiple repetitions therefore produces more reliable BER estimates.

The 3D plot in Figure 33 already confirms this: the error landscape is noticeably more uniform than in the previous experiment, with less trial-to-trial variation visible across the frequency-gain plane.

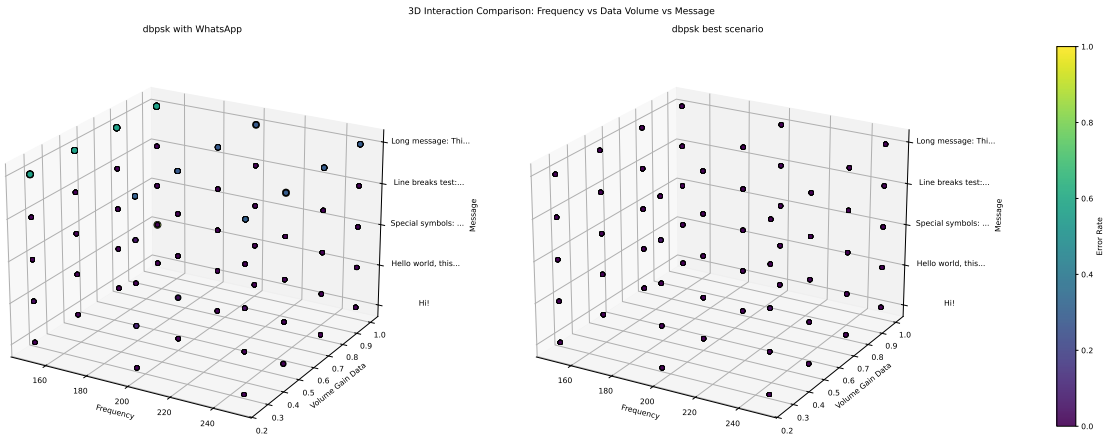


Figure 33: 3D plot for DBPSK investigation with period 2

Tables 7 and 8 confirm this quantitatively. The mean BER of 0.0856 under WhatsApp is consistent with the filtered results from Table 5, while the variance of

0.031 is notably lower, reflecting the improved statistical stability afforded by the repeated trials.

Period	WhatsApp	Best Scenario
2	0.0856	0.0011

Table 7: Mean DBPSK BER for the focused investigation.

Period	WhatsApp	Best Scenario
2	0.031	0.0

Table 8: Variance in DBPSK BER for the focused investigation.

Figures 34–38 show heatmaps of the BER as a function of frequency and volume gain, each averaged over the 10 repetitions of a given message. The five figures correspond to the five messages of increasing length. Averaging across repetitions smooths out the stochastic variation, making the underlying parameter trends more legible than in the single-trial heatmaps of the previous section.

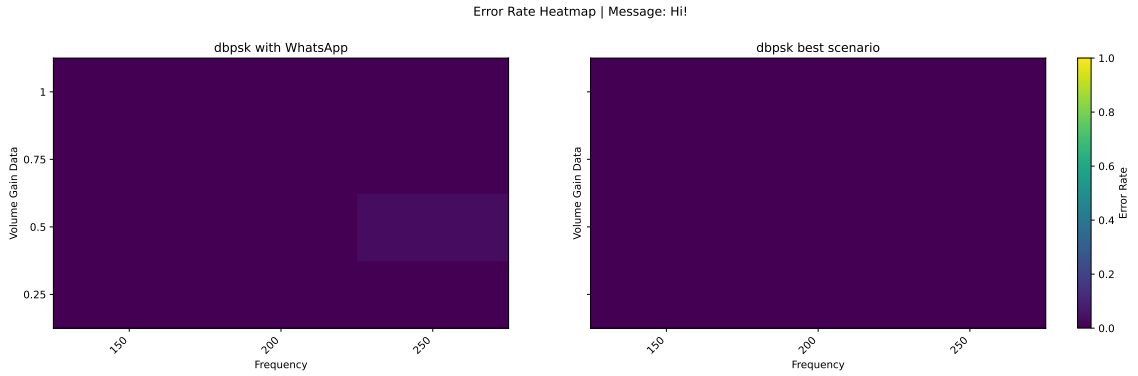


Figure 34: Heatmap for DBPSK investigation, m_1

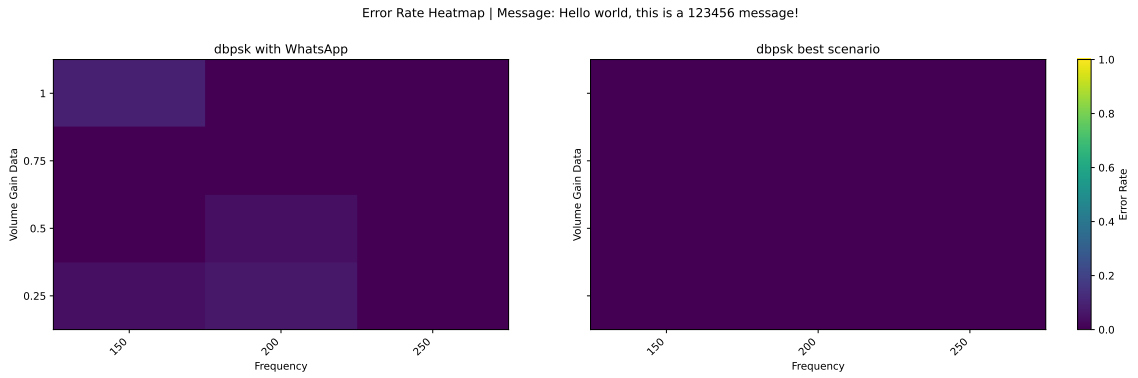


Figure 35: Heatmap for DBPSK investigation, m_2

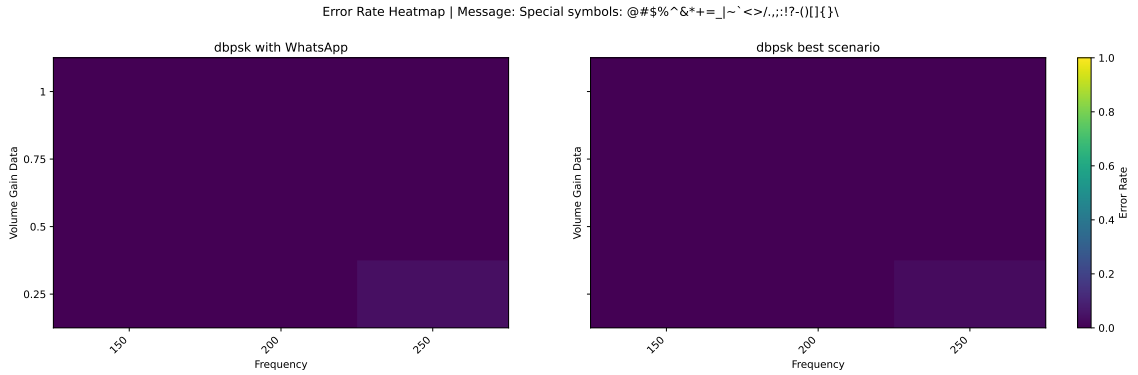


Figure 36: Heatmap for DBPSK investigation, m_3

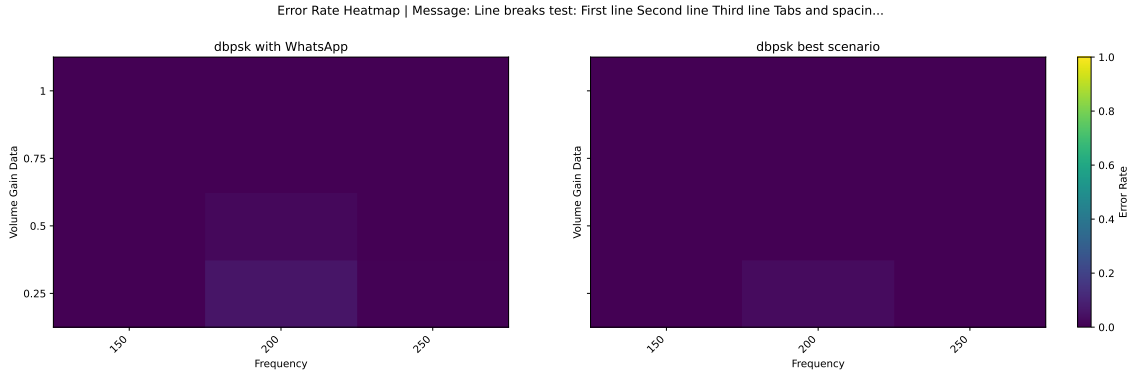


Figure 37: Heatmap for DBPSK investigation, m_4

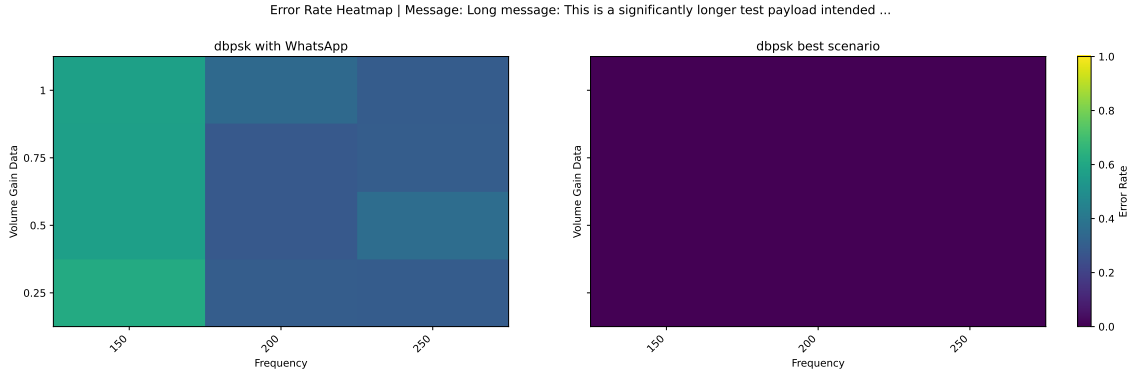


Figure 38: Heatmap for DBPSK investigation, m_5

Figures 39–43 show the corresponding line plots, again averaged over repetitions. Each line represents a distinct carrier frequency, and the x-axis reports the volume gain. As observed in the BPSK analysis, no consistent ordering by frequency is apparent: the three tested frequencies (150, 200, and 250 Hz) do not exhibit a systematic difference in BER across the gain range, suggesting that within this restricted band, carrier frequency remains a secondary factor compared to volume gain.

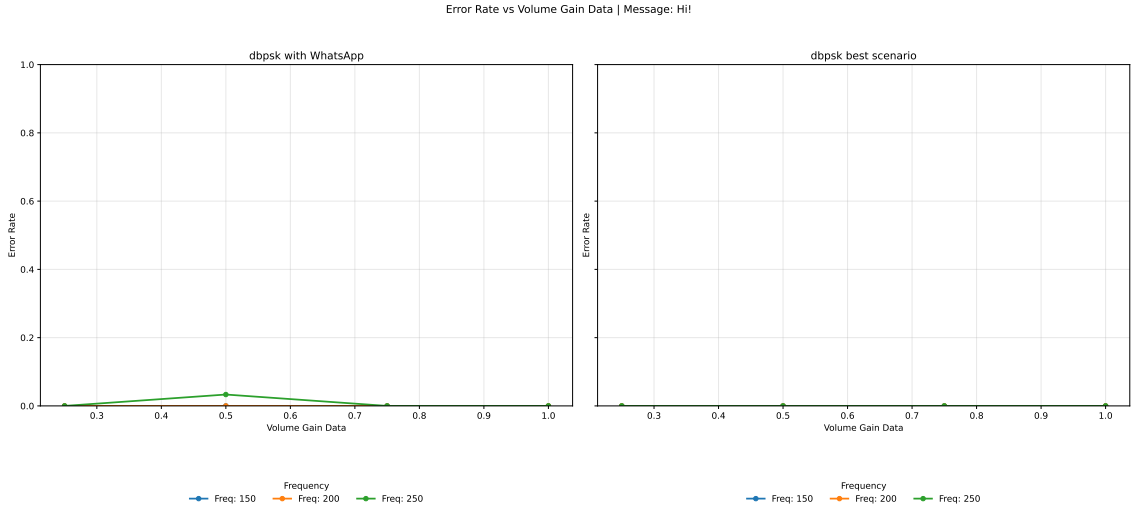


Figure 39: Line plot for DBPSK investigation, m_1

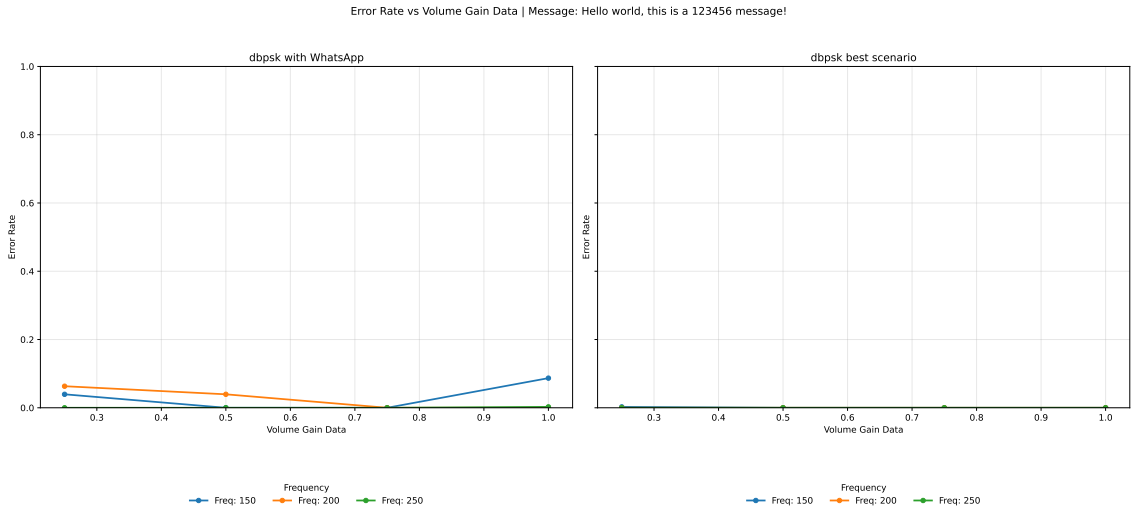


Figure 40: Line plot for DBPSK investigation, m_2

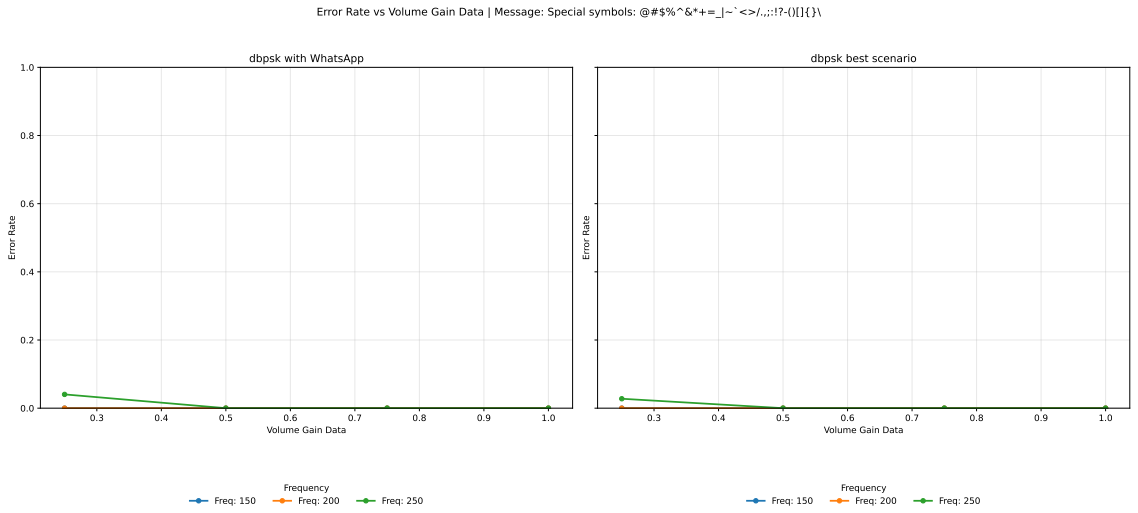


Figure 41: Line plot for DBPSK investigation, m_3

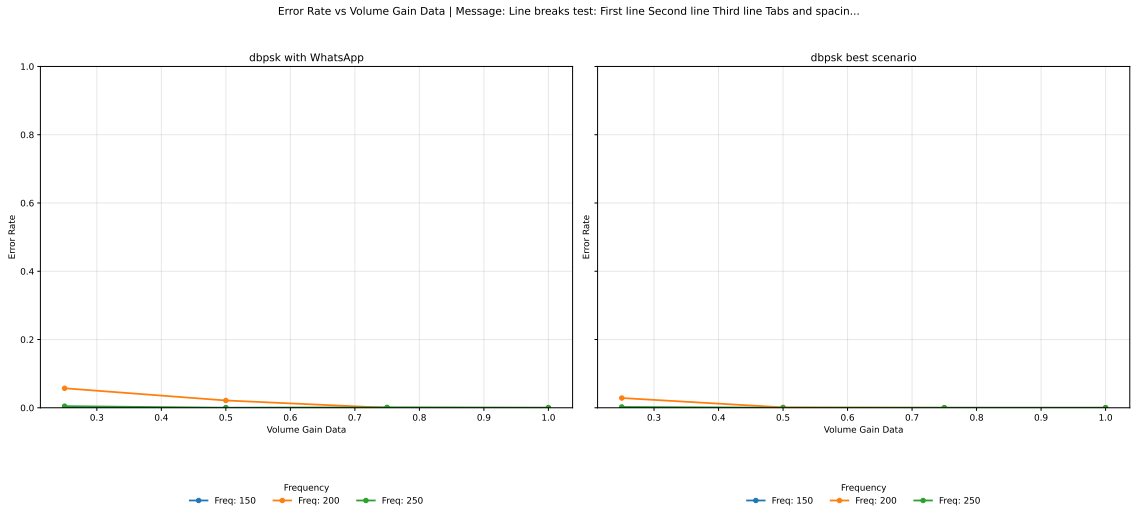


Figure 42: Line plot for DBPSK investigation, m_4

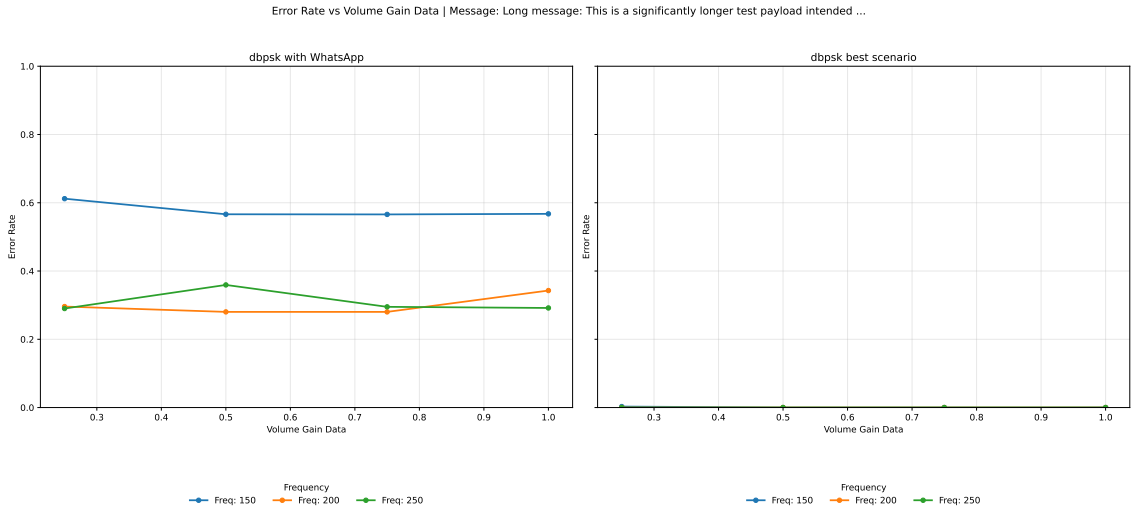


Figure 43: Line plot for DBPSK investigation, m_5

Figures 44–47 present violin plots of the raw BER distribution across all messages and repetitions, without any averaging. Here each violin plot corresponds to one of the four volume gain levels, making it possible to assess both the central tendency and the spread of BER as a function of gain. Unlike the heatmaps and line plots, which suppress trial-to-trial variation through averaging, the violin plots reveal the full distribution and highlight whether errors are concentrated or scattered across repeated trials.

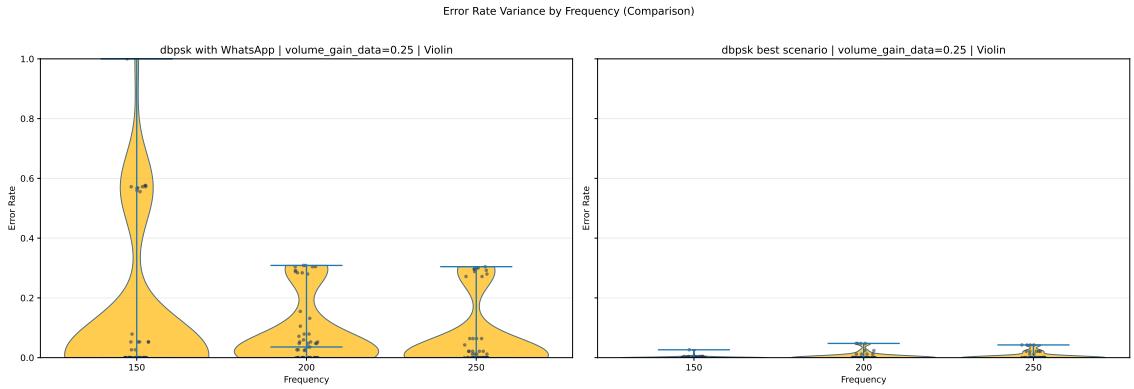


Figure 44: Violin plot for DBPSK investigation, gain level 0

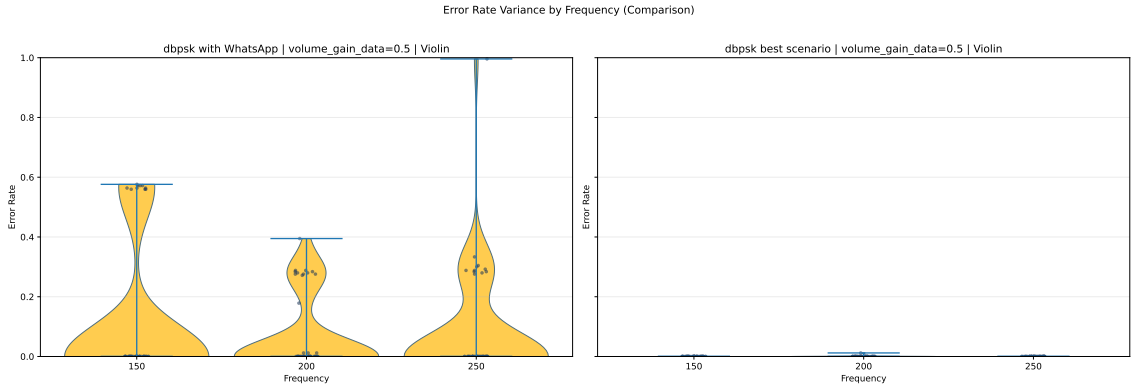


Figure 45: Violin plot for DBPSK investigation, gain level 1

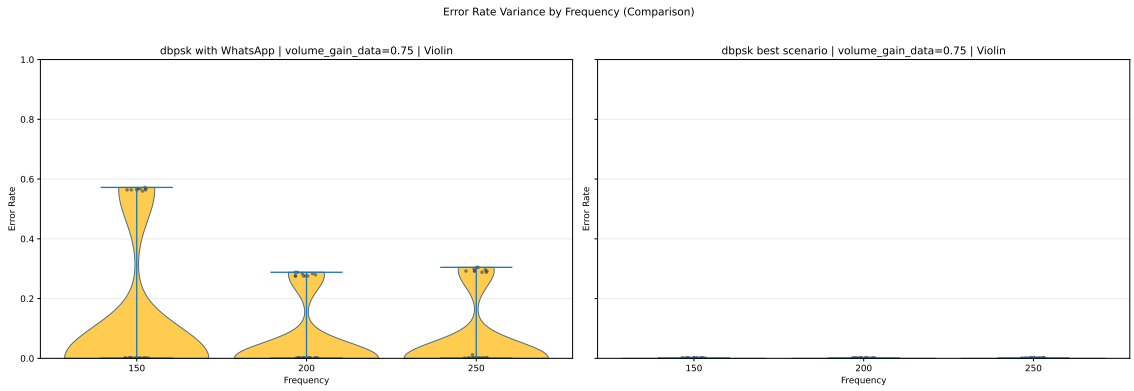


Figure 46: Violin plot for DBPSK investigation, gain level 2

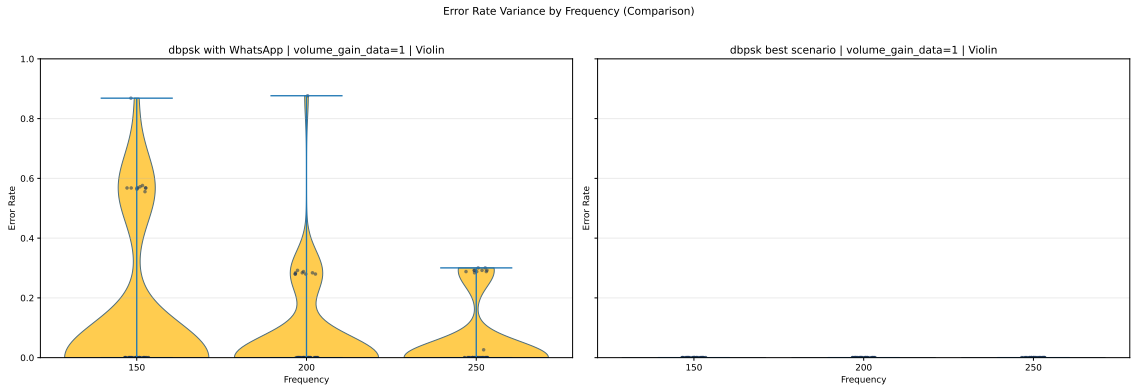


Figure 47: Violin plot for DBPSK investigation, gain level 3

Taken together, these results confirm that DBPSK with period 2 operates reliably within the 150-250 Hz band under the WhatsApp channel, achieving a mean BER of 0.0856 which is substantially lower than the 0.4121 recorded for BPSK at the same period. The repetition-based experiment design also validates the consistency of this result: the low variance of 0.031 indicates that the observed BER is a stable property of the system rather than an artefact of a favourable random draw. Volume gain remains the dominant controllable parameter, with higher gain values consistently associated with lower error rates, while carrier frequency (within the tested sub-250 Hz range) does not produce a statistically distinguishable effect on performance.

11 Future Improvements

Using a Physical Connection Approach Rather than relying on a web-based client interface, a compelling direction for future iteration involves a hardware-centric approach that establishes a direct, analog interface between the trusted device and the smartphone. This implementation could be achieved by connecting the two systems via an auxiliary (AUX) cable engineered to exploit the phone's microphone input lines, typically utilizing a Tip-Ring-Ring-Sleeve (TRRS) connector or a dedicated USB-C/Lightning audio bridge.

The primary objective of this architecture is direct analog signal injection: the trusted device outputs the modulated acoustic waveforms directly into the smartphone's microphone pins, causing the host operating system to perceive the connection as a standard, physically attached external headset.

However, this methodology introduces significant hardware constraints, particularly regarding voltage levels and impedance matching. To prevent catastrophic electrical overstress (EOS) that could damage the smartphone's audio codec or input port, the trusted device's output voltage must be carefully calibrated to match standard microphone-level inputs (typically in the millivolt range). While modern mobile hardware incorporates robust overvoltage protection circuits, empirical tinkering is still required to optimize the output gain. If the injected signal's amplitude is too low, the SNR degrades significantly; conversely, excessive voltage will saturate the input stage, causing severe clipping and harmonic distortion that would render the acoustic payload un-decodable by the receiver.

Costas Loop A Costas loop is a phase-locked loop variant designed for carrier phase recovery in coherent demodulation schemes. Its integration would critically improve the decoding reliability of both BPSK and DBPSK, particularly under conditions of non-trivial phase drift.

The initial assumption was that, since signal injection occurs via a virtual microphone rather than over a physical air channel, the carrier phase would remain

consistent across transmission. It was subsequently discovered, however, that Opus compressor (Valin et al., 2012) introduces not only a frequencies cutoffs but also a degree of inter-packet phase drift. As a result, a Costas loop was not implemented in the current system, and its integration is left as a direction for future work.

Bit Stuffing The current implementation does not check for occurrences of the Barker code sequence within the payload. While this did not produce any errors during testing, it represents a potential failure mode: sufficiently long or high-entropy payloads may contain a bit pattern that matches the preamble sequence, causing the decoder to misidentify a frame boundary mid-payload.

A standard mitigation for this class of problem is *bit stuffing* (Tanenbaum & Wetherall, 2011). During encoding, the algorithm scans the payload for any occurrence of the reserved preamble pattern and inserts a designated escape sequence at each match site. During decoding, the inverse operation is applied: the escape sequences are detected and removed before the payload is passed to the demodulator. This ensures that the preamble pattern remains unambiguous as a unique frame delimiter, regardless of payload content.

Cryptography As discussed in Section 7, the current implementation transmits payload data without encryption. Integrating an authenticated encryption scheme, would protect the confidentiality and integrity of the covert channel against an adversary capable of intercepting and manipulating the host audio stream.

12 Conclusions

12.1 Summary

This thesis has presented the design, implementation, and evaluation of a steganographic communication system that encodes covert binary messages into audio signals transmitted through WhatsApp voice notes. The system employs BPSK and DBPSK modulation schemes, supplemented by Hamming(7,4) forward error correction, a matched-filter preamble detector based on normalised cross-correlation, and signal obfuscation through mixing with voice recordings and AWGN. The modulated signal is injected into the WhatsApp pipeline via a PulseAudio/PipeWire virtual microphone, with the recording and sending process automated using Playwright (Microsoft Corporation, 2020). The complete pipeline (i.e., from message encoding to decoded output) operates without any modification to the WhatsApp client or the underlying Opus codec (Valin et al., 2012).

The experimental evaluation was conducted over a grid of 750 parameter combinations per modulation scheme, varying carrier frequency, volume gain, message length, and number of cycles per symbol. Performance was measured as BER under two conditions: transmission through WhatsApp (the target channel) and a direct file-based baseline (the best-case scenario).

The results demonstrate that the Opus codec introduces two principal impairments. First, a time-varying phase drift that progressively corrupts symbol synchronisation, increasing the BER of BPSK with symbol duration; period 2 achieved a mean BER of 0.4121 under WhatsApp, rising to 0.7159 at period 7. Second, a frequency-dependent attenuation with a practical cutoff near 250 Hz, which renders DBPSK transmission above that threshold non-viable.

Within the sub-250 Hz band, however, DBPSK demonstrates a significant advantage: its differential encoding eliminates the absolute phase reference requirement, making it inherently resilient to the codec's phase drift. Restricted to carrier frequencies of 150–250 Hz and period 2, DBPSK achieves a mean BER of 0.0856, a

reduction of approximately 79% relative to the BPSK baseline at the same period. The focused investigation, in which each parameter combination was repeated ten times, confirmed the stability of this result, with a variance of 0.031. Volume gain emerged as the dominant controllable parameter across both modulation schemes, while carrier frequency within the viable sub-250 Hz range did not produce a statistically distinguishable effect on performance.

12.2 Limitations

Several limitations of the present work warrant acknowledgement. The experimental evaluation was conducted on a single hardware configuration and a fixed network environment; performance under varying network conditions, device hardware, or WhatsApp server routing may differ.

Furthermore, the BER figures reported here reflect the raw channel performance prior to any application-layer error correction beyond the Hamming(7,4) code. The residual error rates, particularly at lower volume gains, may be insufficient for applications requiring reliable delivery of longer payloads. The mean BER of 0.0856 for DBPSK, while substantially lower than the BPSK baseline, still corresponds to a non-negligible symbol error probability over messages of practical length.

Finally, while the system satisfies a functional steganographic concealment requirement by mixing the modulated signal with a natural voice recording, no formal perceptual evaluation or steganalysis was performed. The imperceptibility of the hidden signal to a human listener, and its resistance to automated detection, remain open empirical questions.

12.3 Closing Remarks

This thesis has demonstrated that reliable covert communication through *voice notes* is achievable with differential phase-shift keying, provided that the carrier frequency is constrained to the codec's low-frequency passband. The WhatsApp channel, rather than being simply a transparent pipe, imposes a specific and characterisable

set of distortions (phase drift and frequency attenuation) that directly determine which modulation strategies are viable. Understanding these channel properties is a prerequisite for any practical acoustic steganography system operating over modern instant messaging platforms, and the results presented here provide a concrete empirical characterisation of those constraints.

List of Abbreviations

A2DP Audio Distribution Profile. 14

AEAD Authenticated Encryption with Associated Data. 43

AES-GCM Advanced Encryption Standard-Galois/Counter Mode. 43

AG Audio Gateway. 13

AI Artificial Intelligence. 7

ASK Amplitude Shift Keying. 18

AUX Auxiliary. 81

AWGN Additive white Gaussian noise. 22, 26, 29, 32, 45, 55, 83

BER Byte Error Rate. 22, 57–59, 61, 63, 66–69, 71, 73, 74, 76, 78, 80, 83, 84

BPSK Bit Phase Shift Keying. 19, 20, 22, 23, 29, 31–33, 35, 36, 57, 58, 61, 67, 68, 71, 76, 80, 81, 83, 84

CSV Comma Separated Values. 49

CTR Counter. 43

CUDA Compute Unified Device Architecture, NVIDIA GPU programming language. 49

DBPSK Differential Binary Phase Shift Keying, it is a subset of DPSK. 6, 22, 23, 27, 31–33, 35–37, 57, 67, 68, 71, 73–81, 83, 84

DPSK Differential Phase Shift Keying. 19

EOS Electrical Overstress. 81

FEC Forward Error Correction. 38

FLAC Free Lossless Audio Codec. 50

FSK Frequency Shift Keying. 18, 19

GPU Graphic Processing Unit. 49

HF Hands Free. 13

HFP Hands-Free Profile. 11, 13, 14

IM Instant Messaging. 9, 14

PoD Probability of Detection. 45

PRNG Pseudorandom Number Generator. 47

PSK Phase Shift Keying. 18, 19, 45, 53

QR Quick Response. 43

RF Radio Frequency. 18

RKE Remote Keyless Entry. 18

RSA Rivest-Shamir-Adleman is a foundational asymmetric (public-key) encryption algorithm. 43

SCO Synchronous Connection-Oriented. 14

SEC Single-bit Error Correction. 39

SNR Signal to Noise Ratio. 26, 32, 81

TRRS Tip Ring Ring Sleeve. 81

References

Android Open Source Project. (2026). *Audiomanager api reference* [Documentation on `startBluetoothSco()` and `setCommunicationDevice()`]. Google Developer Network. https://developer.android.com/reference/android/media/AudioManager

Barker, C. H. (1953). Group synchronizing of binary digital systems. In W. Jackson (Ed.), *Communication theory* (pp. 273–287). Academic Press.

Bluetooth Special Interest Group. (2020, April). *Hands-free profile (hfp) specification, version 1.8* (Specification). Bluetooth SIG. https://www.bluetooth.com/specifications/specs/hands-free-profile-1-8/

Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., & Stebila, D. (2020). A formal security analysis of the Signal messaging protocol. *Journal of Cryptology*, *33*, 1914–1983. https://doi.org/10.1007/s00145-020-09360-1

Dworkin, M. J. (2007). *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC* (tech. rep. No. SP 800-38D). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-38D

FFmpeg Developers. (n.d.). *FFmpeg* [Available from https://ffmpeg.org/]. https://ffmpeg.org/

Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell System Technical Journal*, *29*(2), 147–160. https://doi.org/10.1002/j.1538-7305.1950.tb00463.x

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Jamil, T. (1999). Steganography: The art of hiding information in plain sight. *IEEE Potentials*, *18*(1), 19–23. https://doi.org/10.1109/45.747237

Jennings, C., Castelli, F., Boström, H., & Bruaroey, J.-I. (2025, March). WebRTC: Real-time communication in browsers [Latest version available at https://www.w3.org/TR/webrtc/]. https://www.w3.org/TR/2025/REC-webrtc-20250313/

Kay, S. M. (1998). *Fundamentals of statistical signal processing, Volume II: Detection theory*. Prentice Hall.

Lin, S., & Costello, D. J. (2004). *Error control coding* (2nd). Pearson Prentice Hall.

Maene, P., Götzfried, J., de Clercq, R., Müller, T., & Verbauwhede, I. (2018). Hardware-based trusted computing architectures for isolation and attestation. *IEEE Transactions on Computers*, *67*(3), 361–374. https://doi.org/10.1109/TC.2017.2647955

Marlinspike, M., & Perrin, T. (2016a). *The Double Ratchet algorithm* (tech. rep.). Signal Messenger LLC. Retrieved January 1, 2024, from https://signal.org/docs/specifications/doubleratchet/

Marlinspike, M., & Perrin, T. (2016b). *The X3DH key agreement protocol* (tech. rep.). Signal Messenger LLC. Retrieved January 1, 2024, from https://signal.org/docs/specifications/x3dh/

McKinney, W. (2010). Data structures for statistical computing in Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in science conference* (pp. 51–56). https://doi.org/10.25080/Majora-92bf1922-00a

Microsoft Corporation. (2020). *Playwright* [Version used: see https://github.com/microsoft/playwright/releases]. https://github.com/microsoft/playwright

Moon, T. K. (2005). *Error correction coding: Mathematical methods and algorithms*. Wiley-Interscience.

Okuta, R., Unno, Y., Nishino, D., Hido, S., & Loomis, C. (2017). CuPy: A NumPy-compatible library for NVIDIA GPU calculations. *Proceedings of Workshop*

on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS). http://learningsys.org/nips17/assets/papers/paper_16.pdf

Oppenheim, A. V., & Schafer, R. W. (2009). *Discrete-time signal processing* (3rd). Prentice Hall.

Poettering, L., Shinn, P.-L., & The PulseAudio Developers. (2026). *PulseAudio: A networked sound server* [Available at https://www.freedesktop.org/wiki/Software/PulseAudio/]. Freedesktop.org.

Povey, D. (2012). OpenSLR: Open speech and language resources [Accessed: 2026]. https://www.openslr.org

Proakis, J. G., & Salehi, M. (2008). *Digital communications* (5th). McGraw-Hill.

Raspberry Pi Ltd. (2026). *Raspberry pi hardware documentation* [Available at https://www.raspberrypi.com/documentation/]. Raspberry Pi Foundation.

Sklar, B. (2001). *Digital communications: Fundamentals and applications* (2nd). Prentice Hall.

Tanenbaum, A. S., & Wetherall, D. (2011). *Computer networks* (5th ed.). Pearson.

Turin, G. L. (1960). An introduction to matched filters. *IRE Transactions on Information Theory*, *6*(3), 311–329. https://doi.org/10.1109/TIT.1960.1057571

Valin, J.-M., Vos, K., & Terriberry, T. B. (2012, September). *Definition of the Opus audio codec* (RFC No. 6716). Internet Engineering Task Force. https://doi.org/10.17487/RFC6716

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. CreateSpace.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

WhatsApp. (2026). *Whatsapp* [Mobile application/Web Client]. Meta Platforms, Inc. https://www.whatsapp.com/

Zhidkov, S. V. (2018). Statistical characterization and modeling of noise effects in near-ultrasound aerial acoustic communications. *The Journal of the Acoustical Society of America*, *144*(4), 2605–2612. https://doi.org/10.1121/1.5063988

A Supplementary BPSK Plots

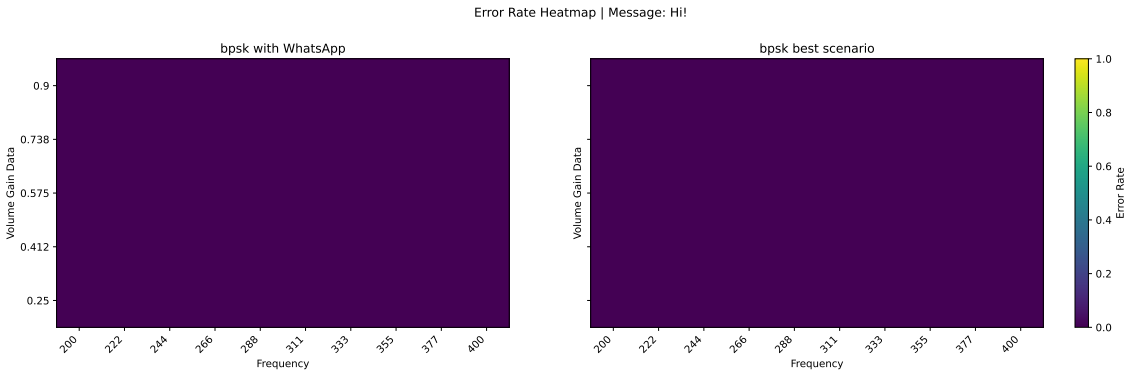


Figure 48: Heatmap plot for BPSK algorithm with period 5

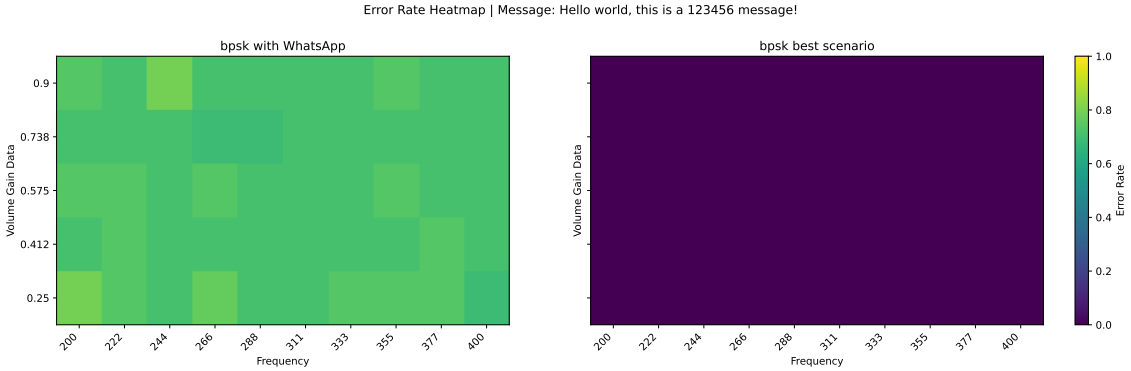


Figure 49: Heatmap plot for BPSK algorithm with period 5

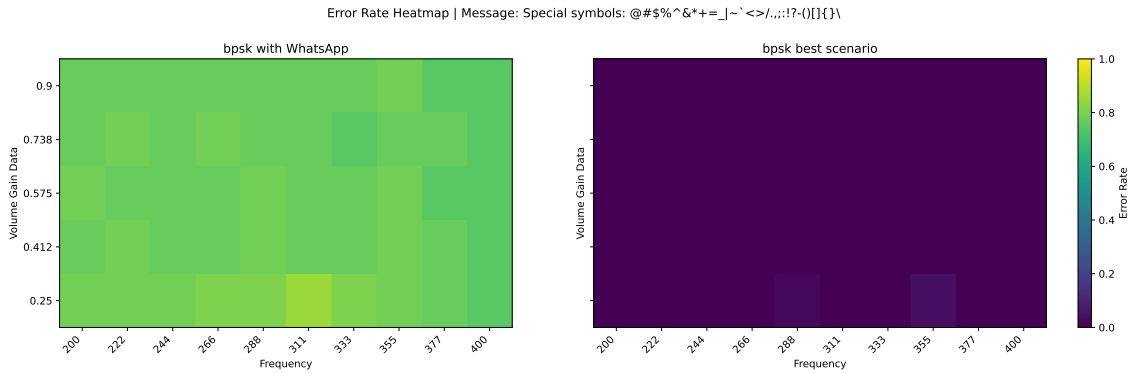


Figure 50: Heatmap plot for BPSK algorithm with period 5

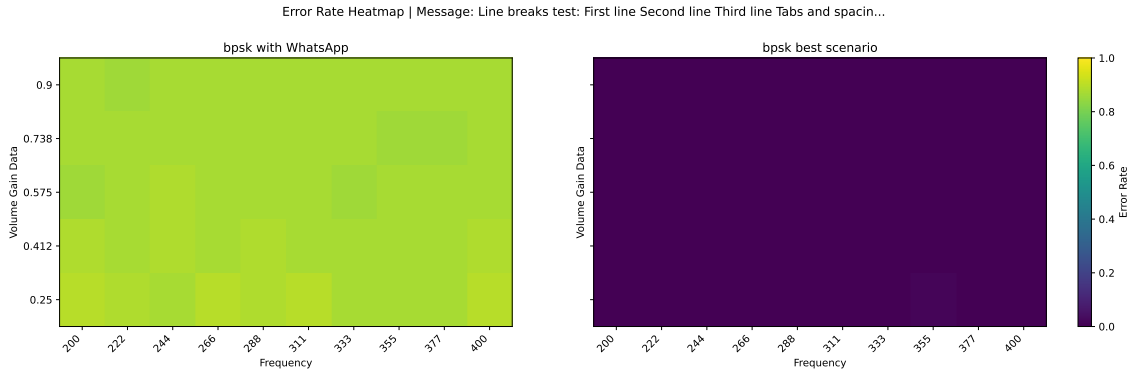


Figure 51: Heatmap plot for BPSK algorithm with period 5

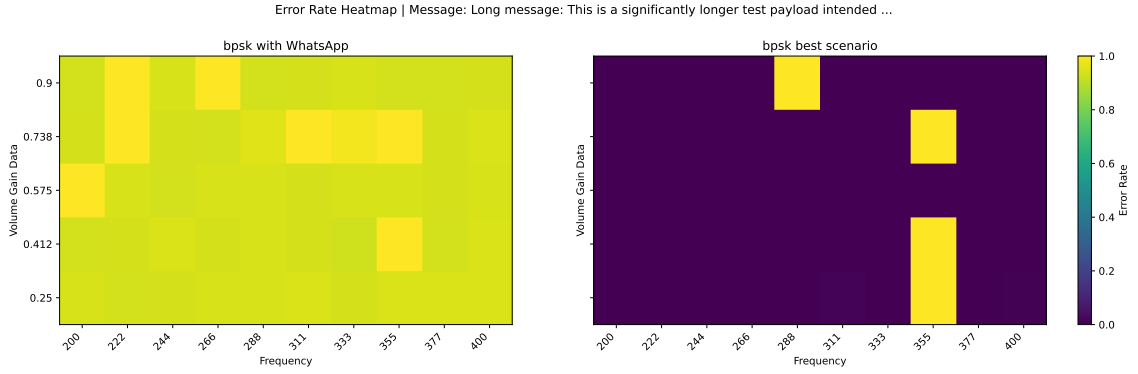


Figure 52: Heatmap plot for BPSK algorithm with period 5

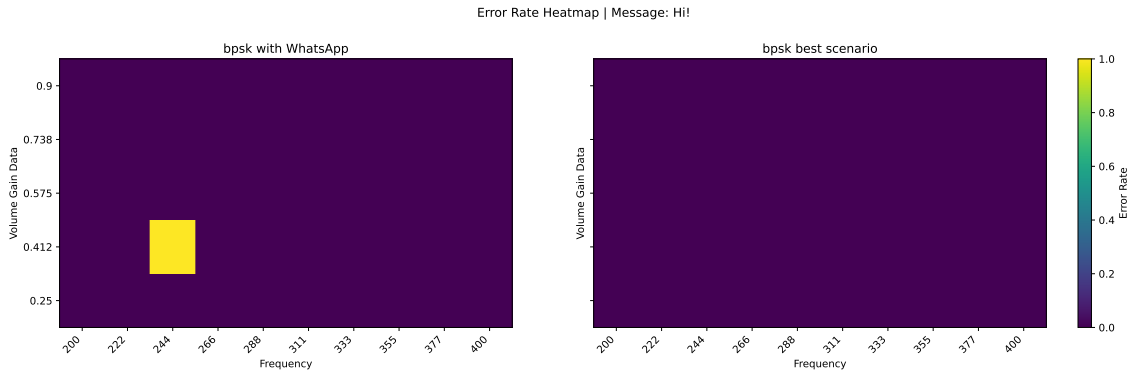


Figure 53: Heatmap plot for BPSK algorithm with period 7

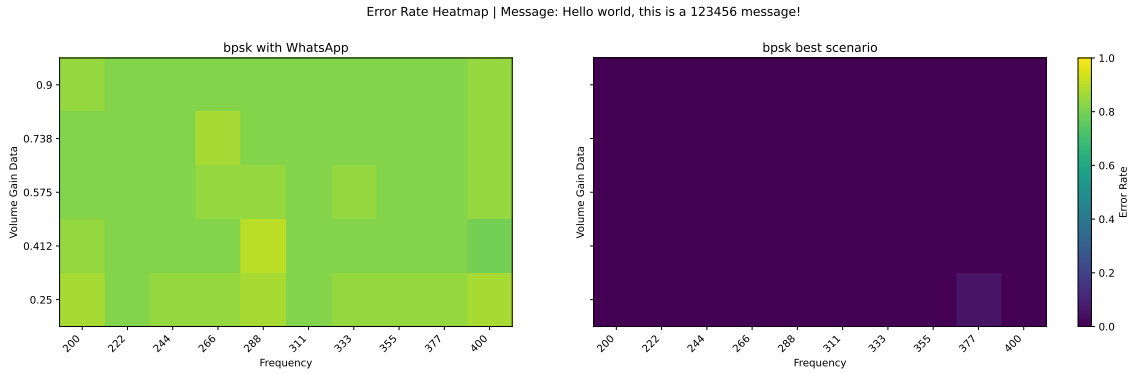


Figure 54: Heatmap plot for BPSK algorithm with period 7

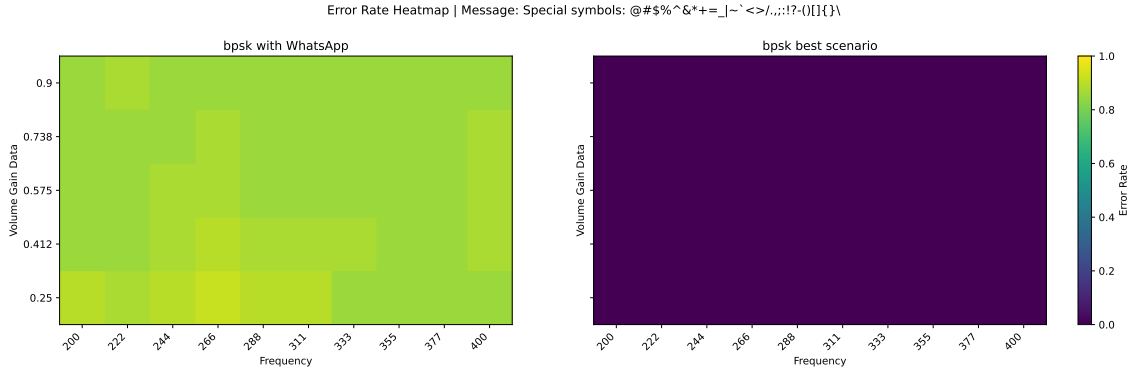


Figure 55: Heatmap plot for BPSK algorithm with period 7

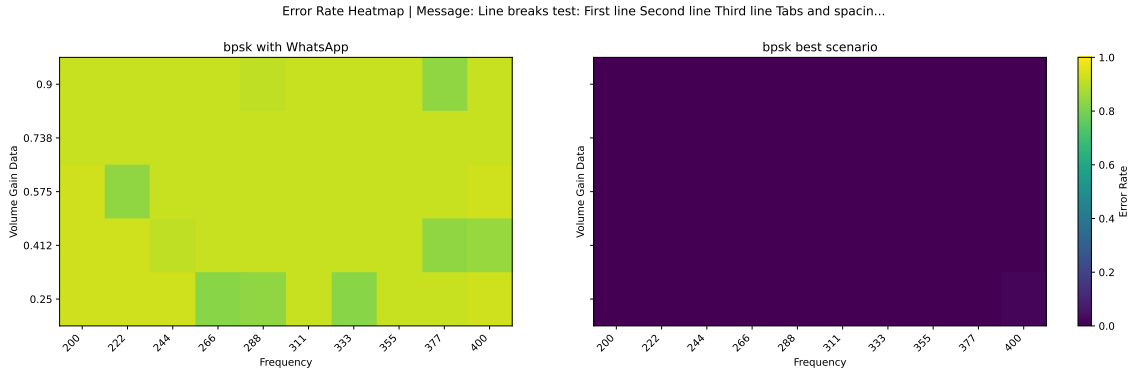


Figure 56: Heatmap plot for BPSK algorithm with period 7

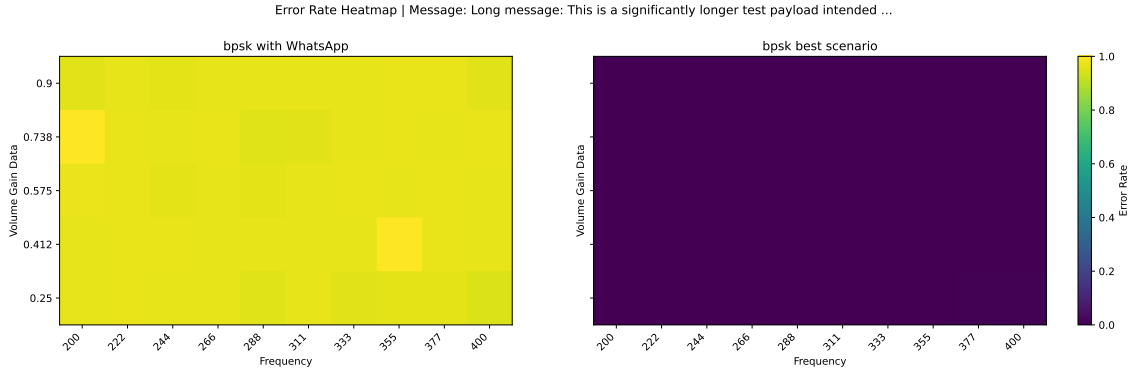


Figure 57: Heatmap plot for BPSK algorithm with period 7

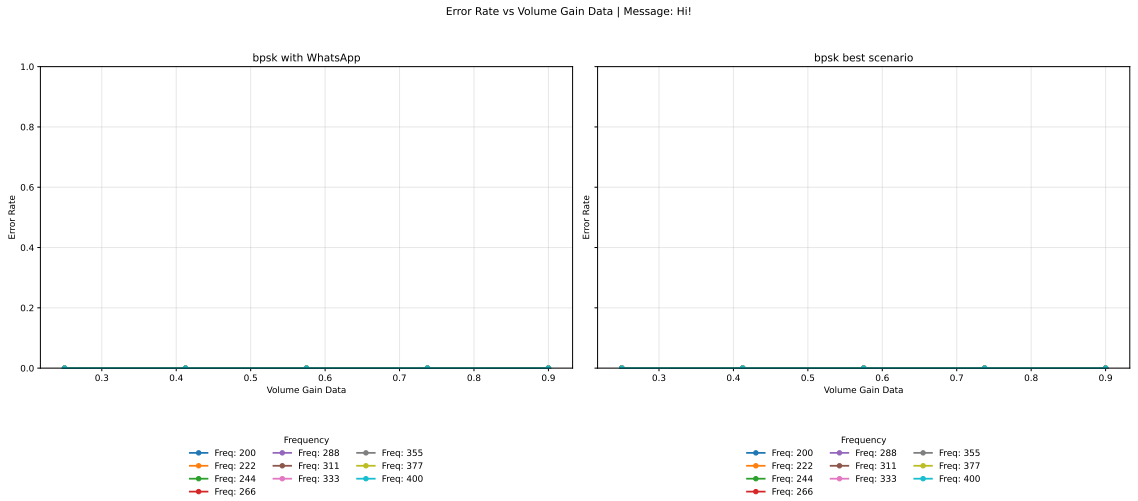


Figure 58: Line plot for BPSK algorithm with period 5

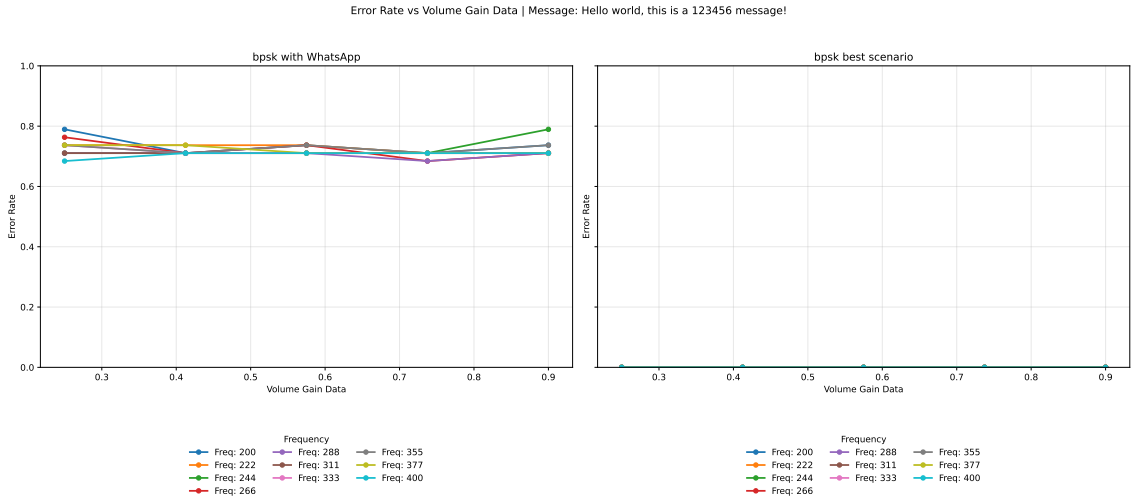


Figure 59: Line plot for BPSK algorithm with period 5

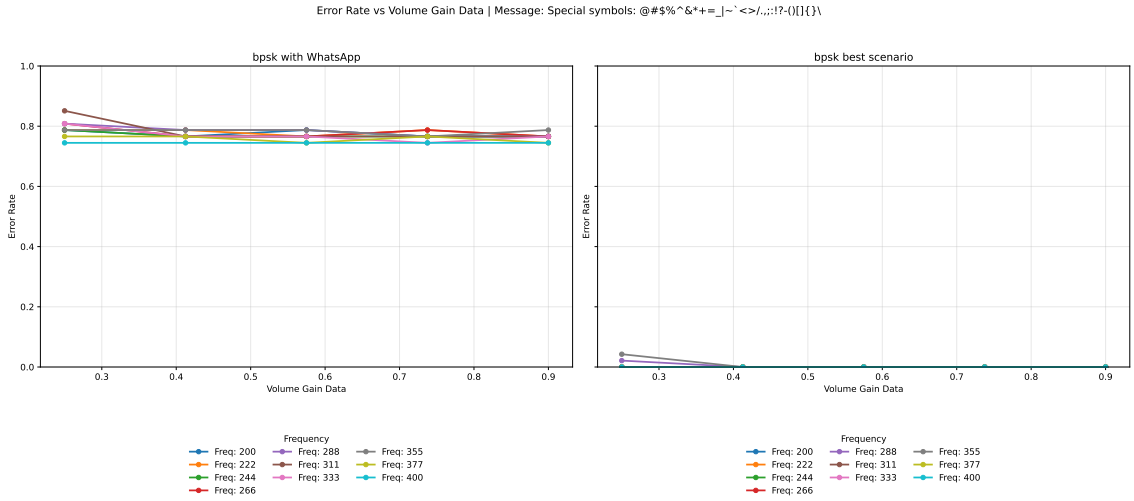


Figure 60: Line plot for BPSK algorithm with period 5

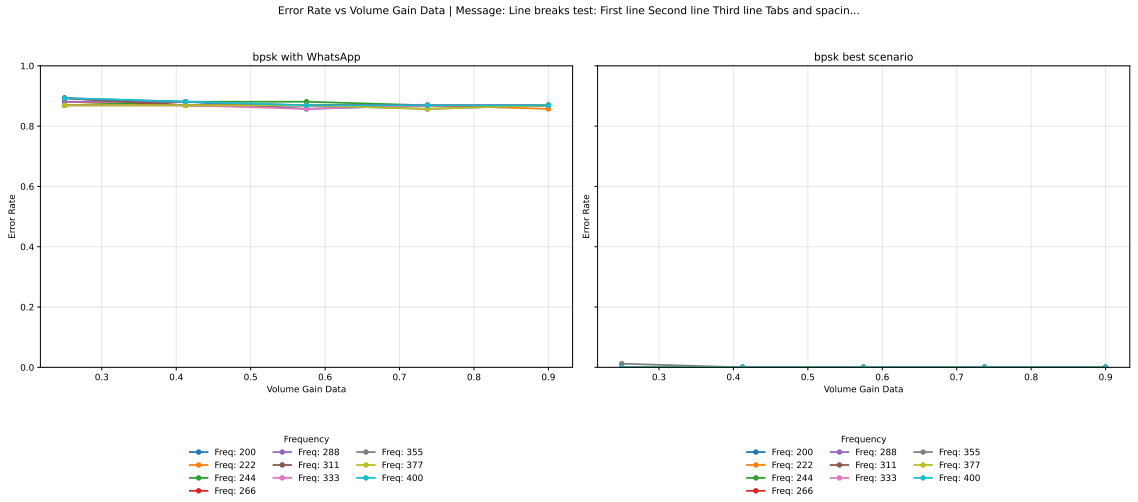


Figure 61: Line plot for BPSK algorithm with period 5

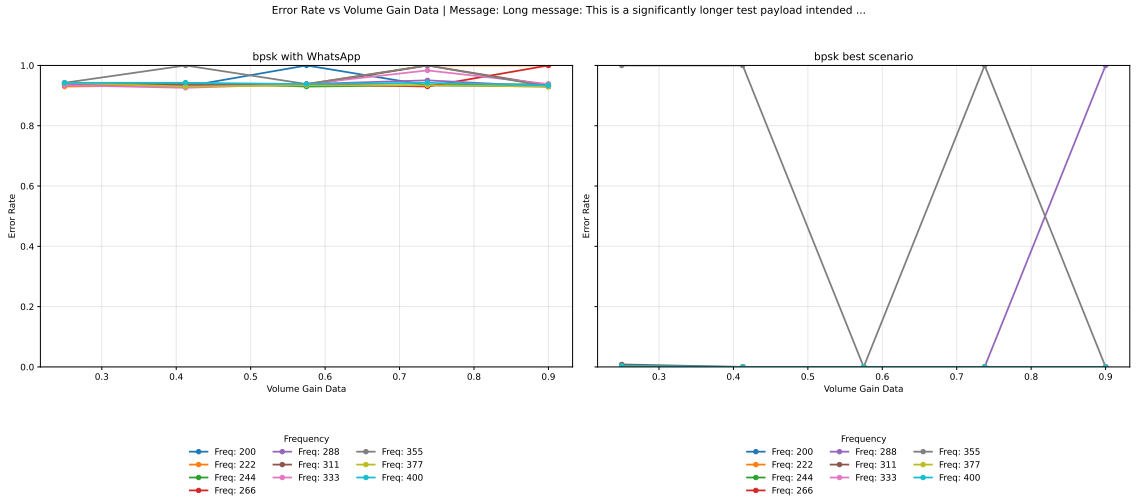


Figure 62: Line plot for BPSK algorithm with period 5

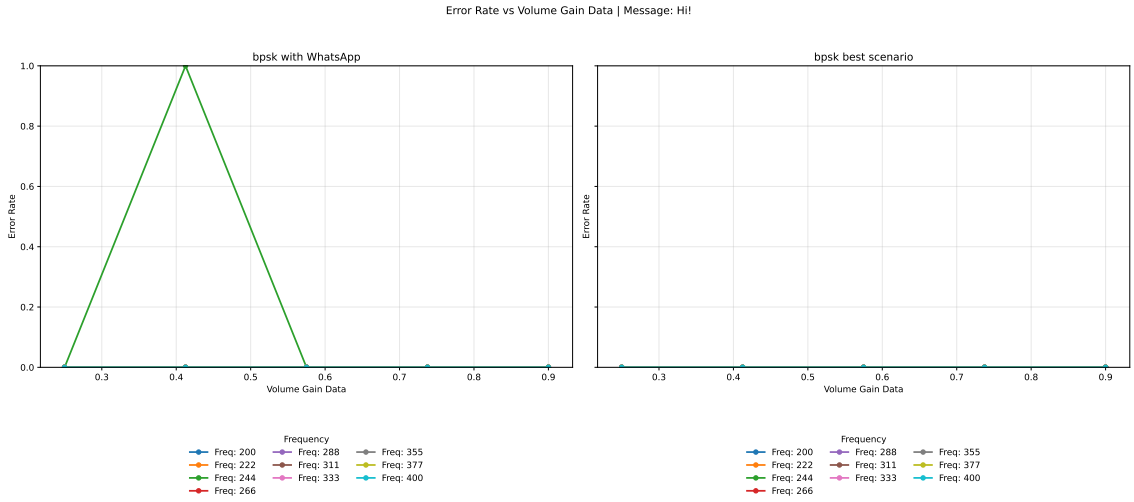


Figure 63: Line plot for BPSK algorithm with period 7

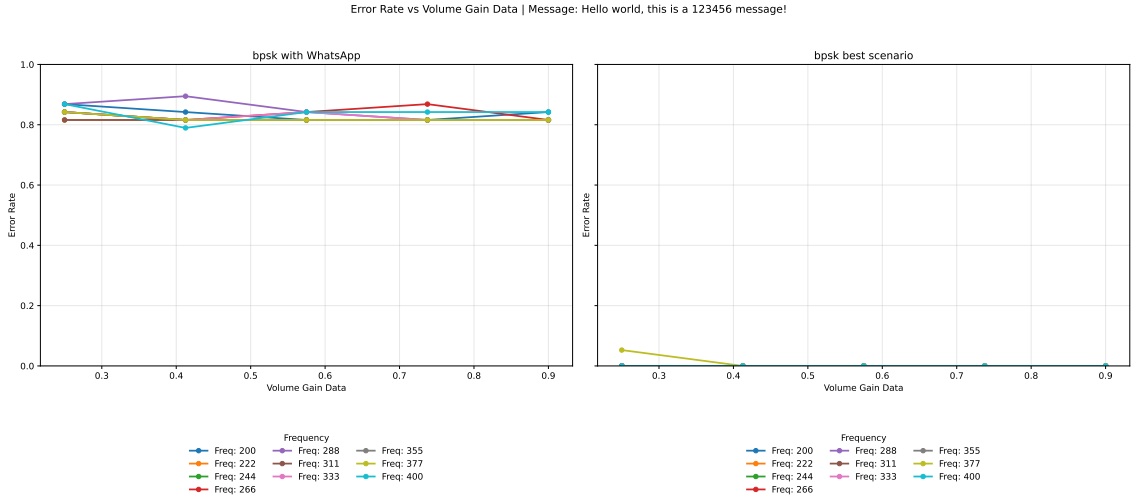


Figure 64: Line plot for BPSK algorithm with period 7

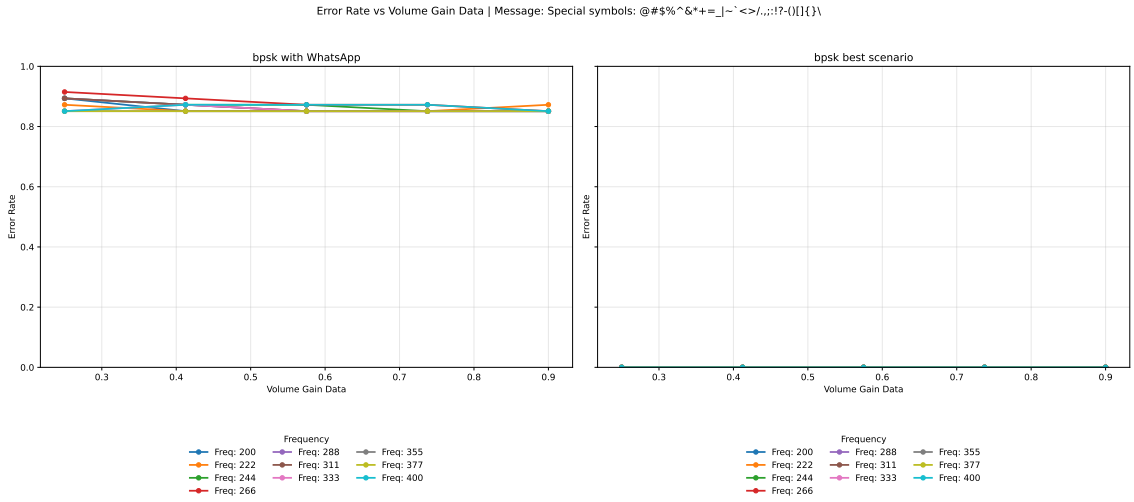


Figure 65: Line plot for BPSK algorithm with period 7

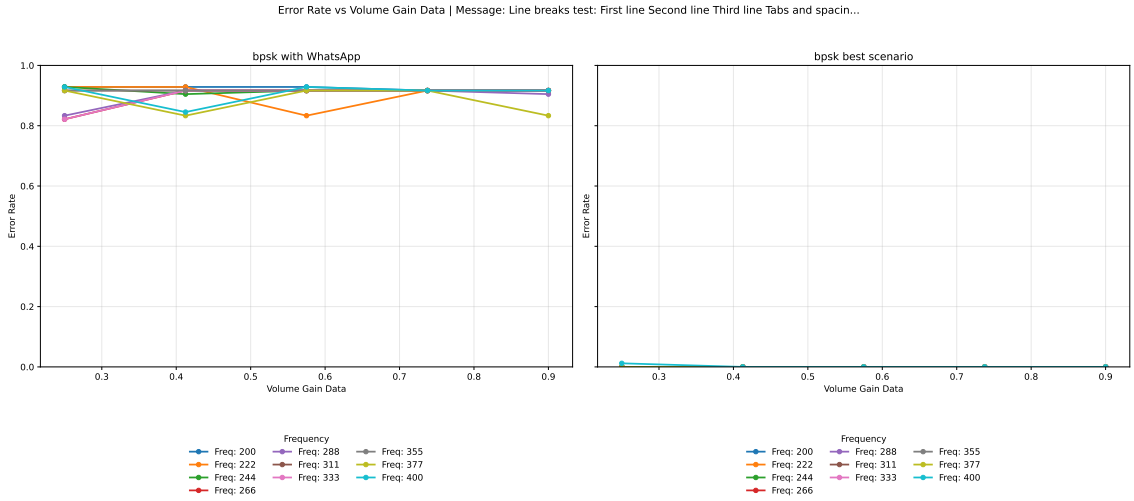


Figure 66: Line plot for BPSK algorithm with period 7

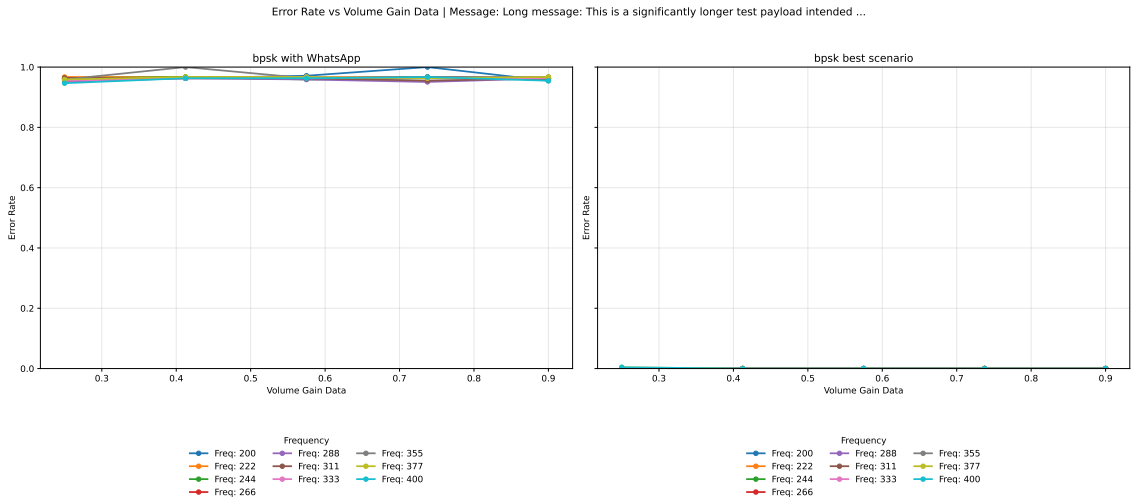


Figure 67: Line plot for BPSK algorithm with period 7

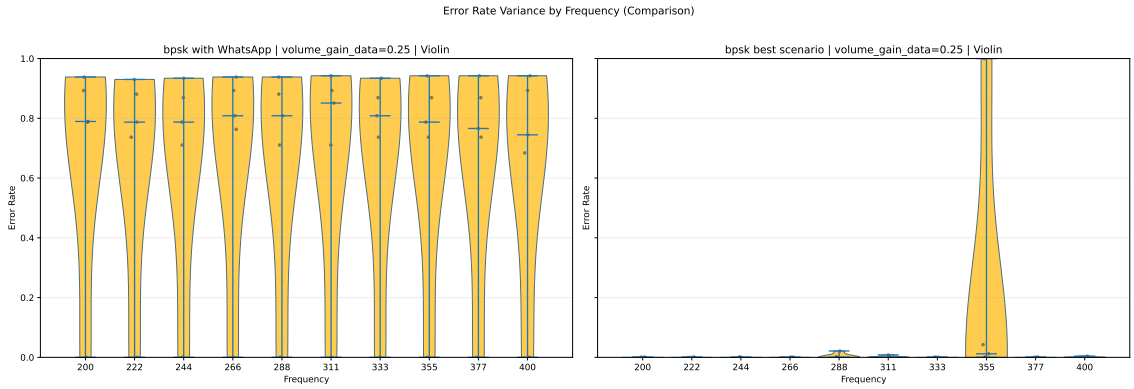


Figure 68: Violin plot for BPSK algorithm with period 5

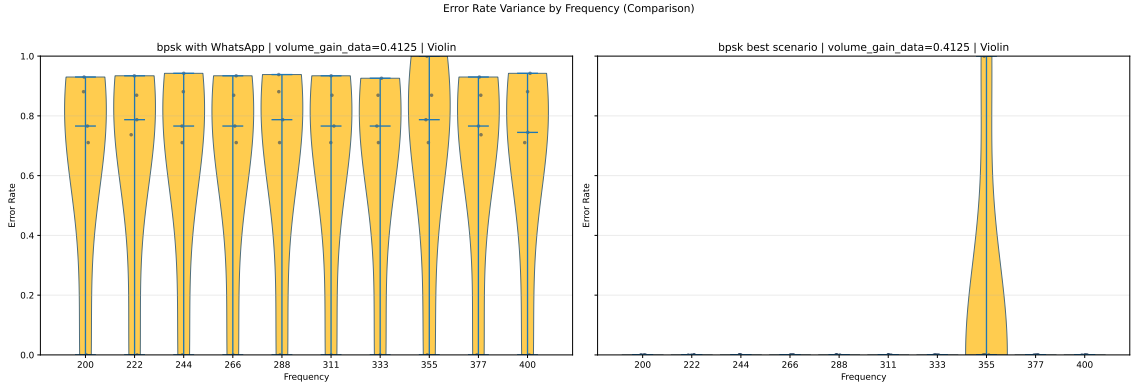


Figure 69: Violin plot for BPSK algorithm with period 5

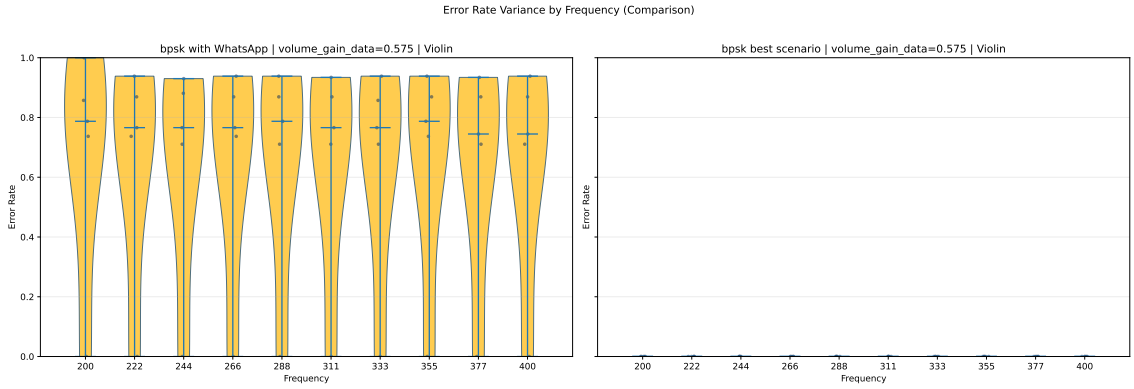


Figure 70: Violin plot for BPSK algorithm with period 5

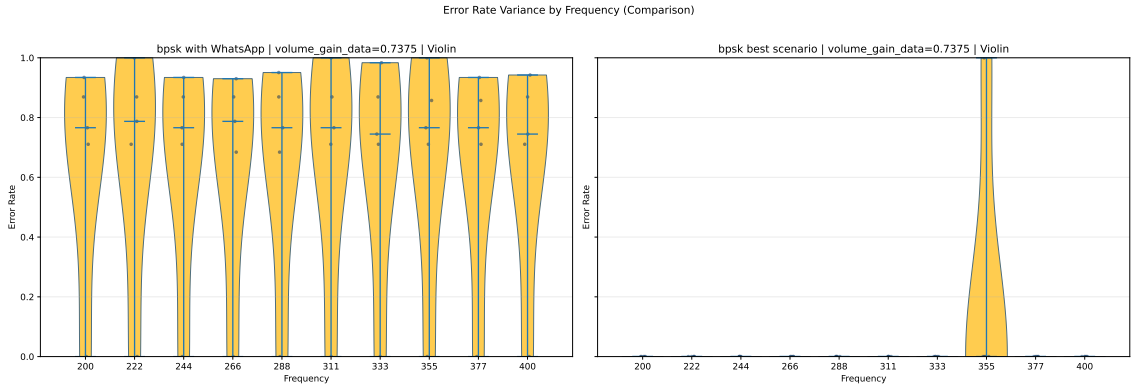


Figure 71: Violin plot for BPSK algorithm with period 5

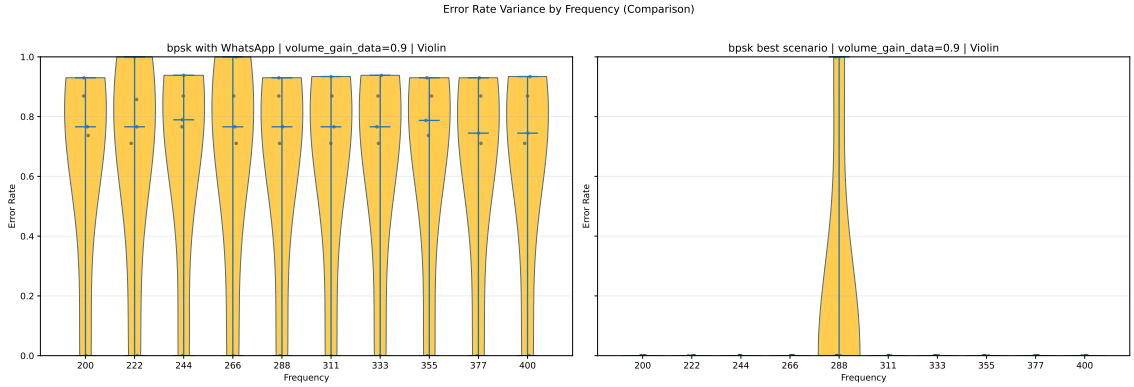


Figure 72: Violin plot for BPSK algorithm with period 5

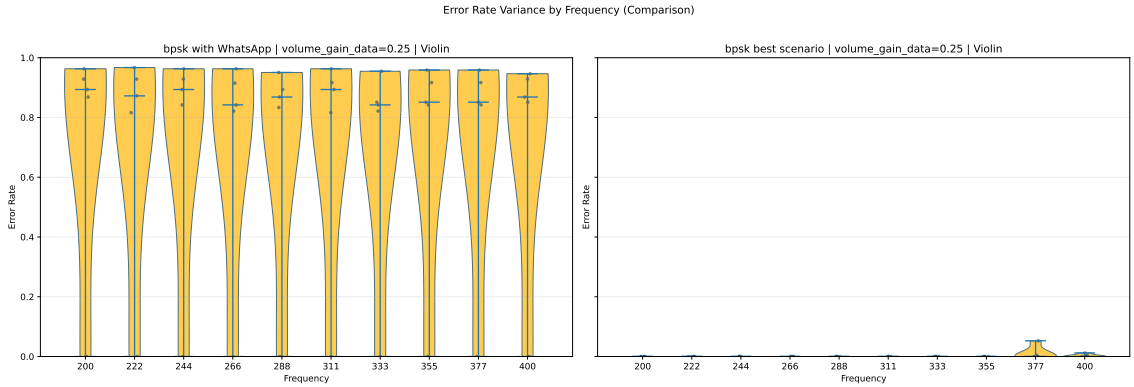


Figure 73: Violin plot for BPSK algorithm with period 7

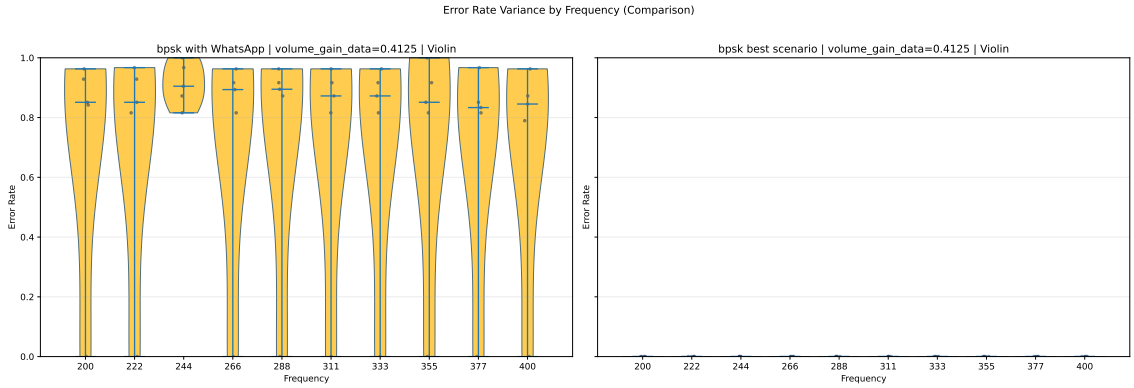


Figure 74: Violin plot for BPSK algorithm with period 7

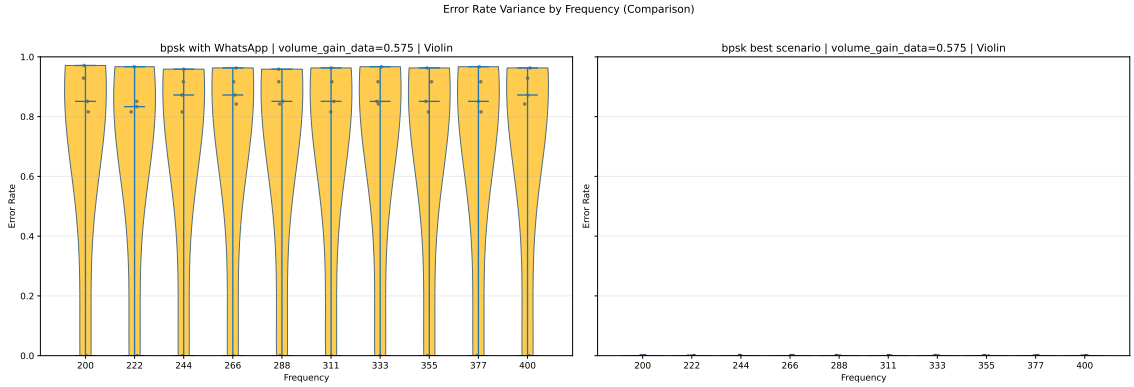


Figure 75: Violin plot for BPSK algorithm with period 7

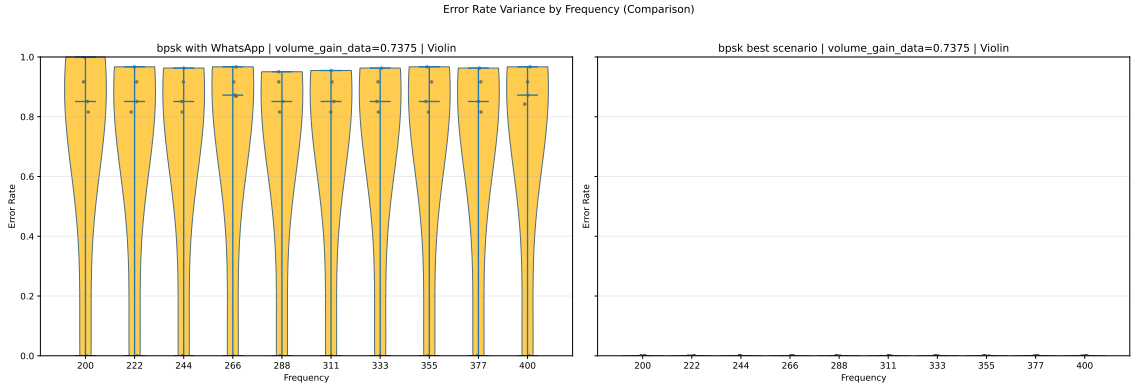


Figure 76: Violin plot for BPSK algorithm with period 7

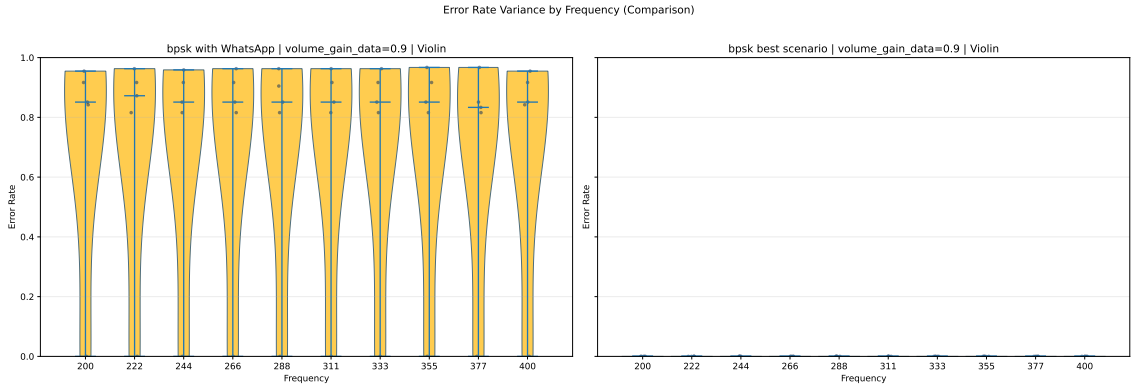


Figure 77: Violin plot for BPSK algorithm with period 7

B Supplementary DBPSK Plots

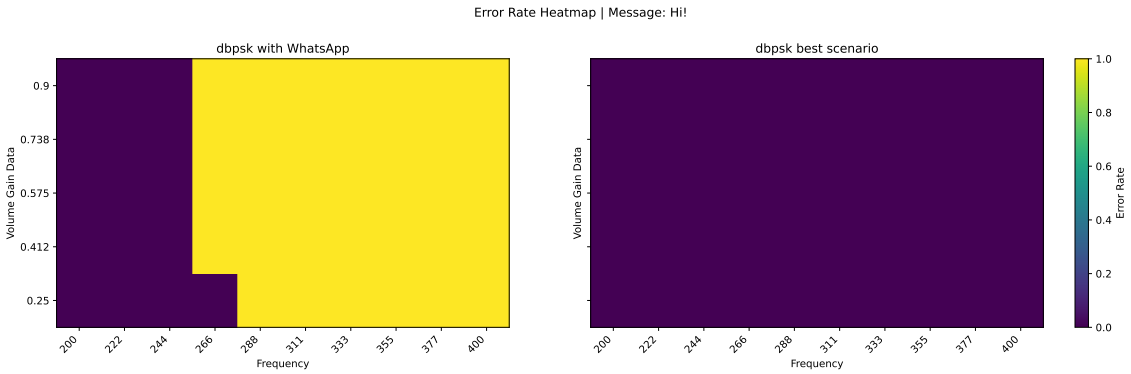


Figure 78: Heatmap plot for DBPSK algorithm with period 5

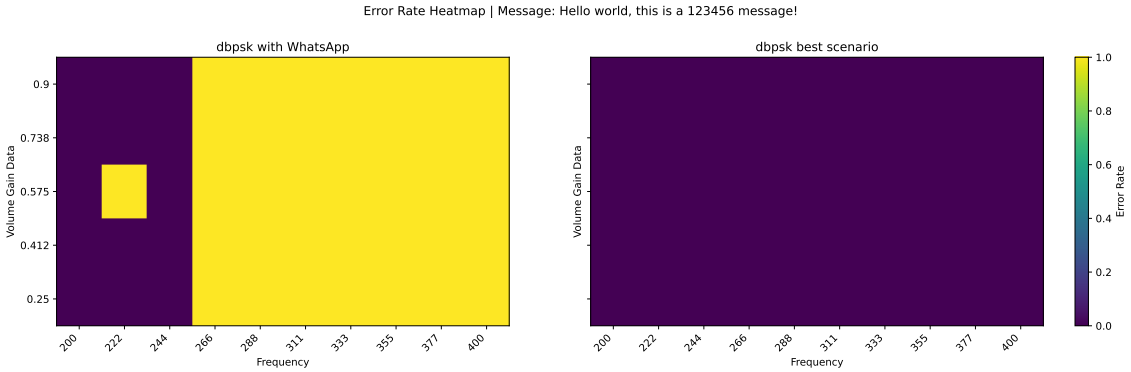


Figure 79: Heatmap plot for DBPSK algorithm with period 5

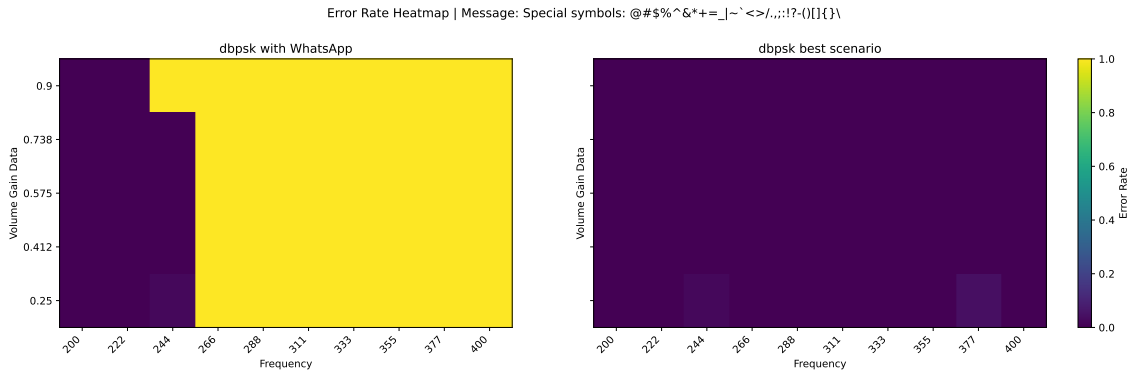


Figure 80: Heatmap plot for DBPSK algorithm with period 5

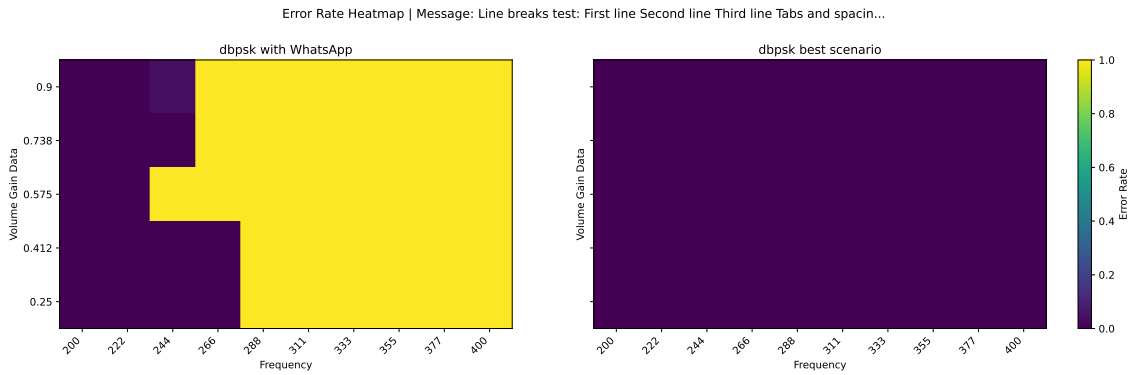


Figure 81: Heatmap plot for DBPSK algorithm with period 5

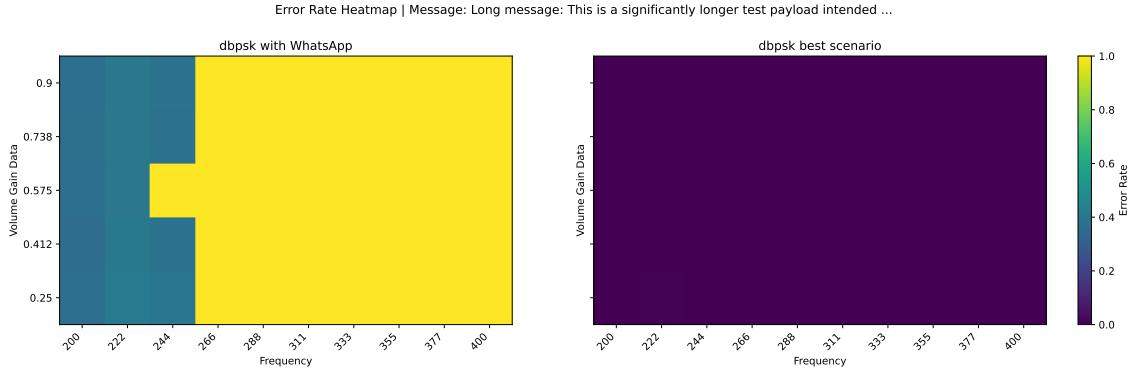


Figure 82: Heatmap plot for DBPSK algorithm with period 5

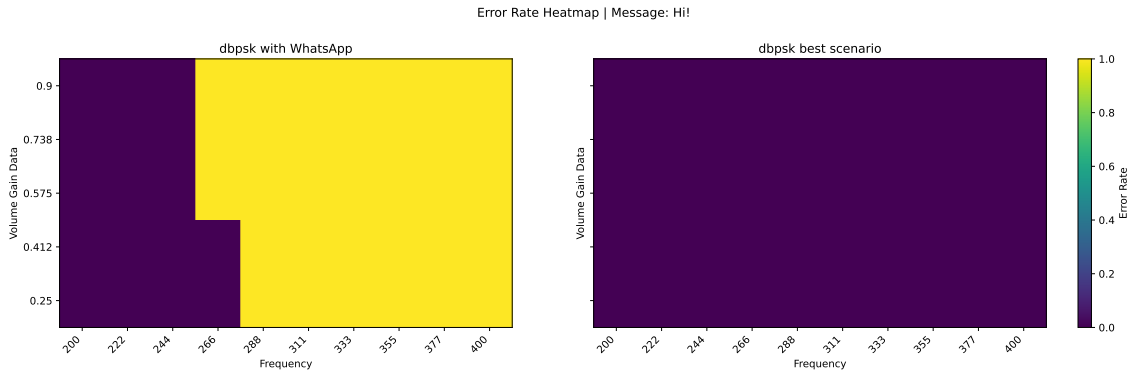


Figure 83: Heatmap plot for DBPSK algorithm with period 7

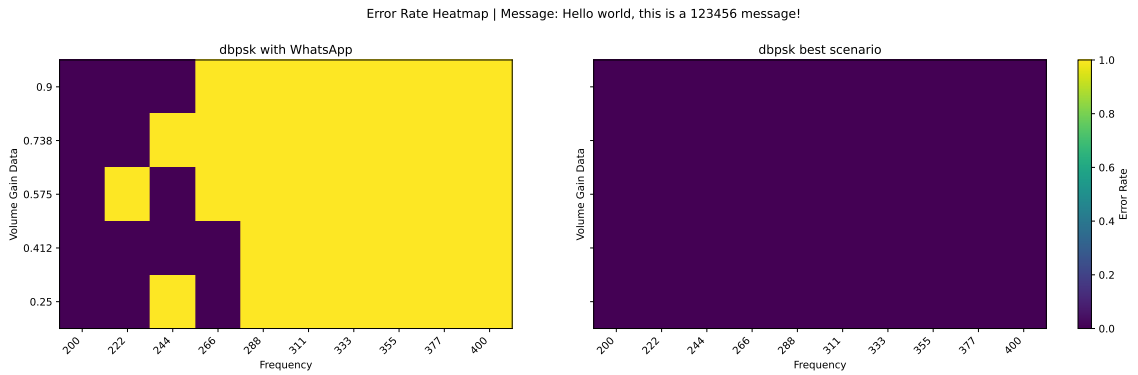


Figure 84: Heatmap plot for DBPSK algorithm with period 7

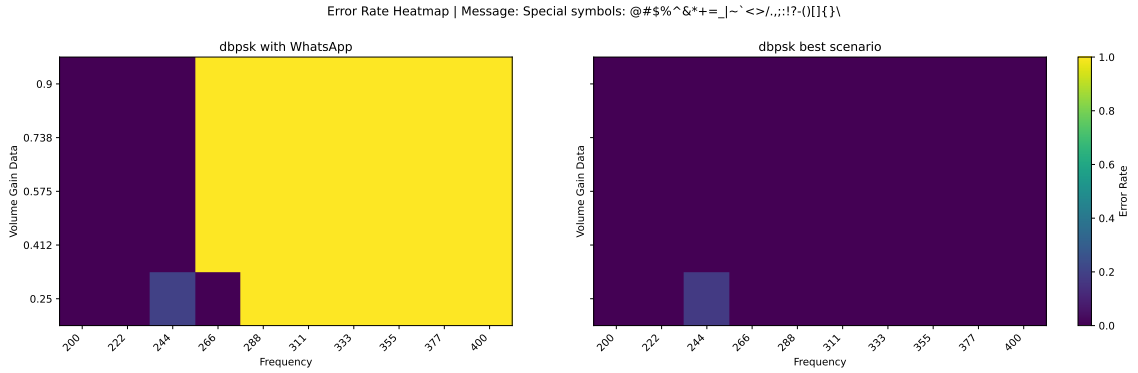


Figure 85: Heatmap plot for DBPSK algorithm with period 7

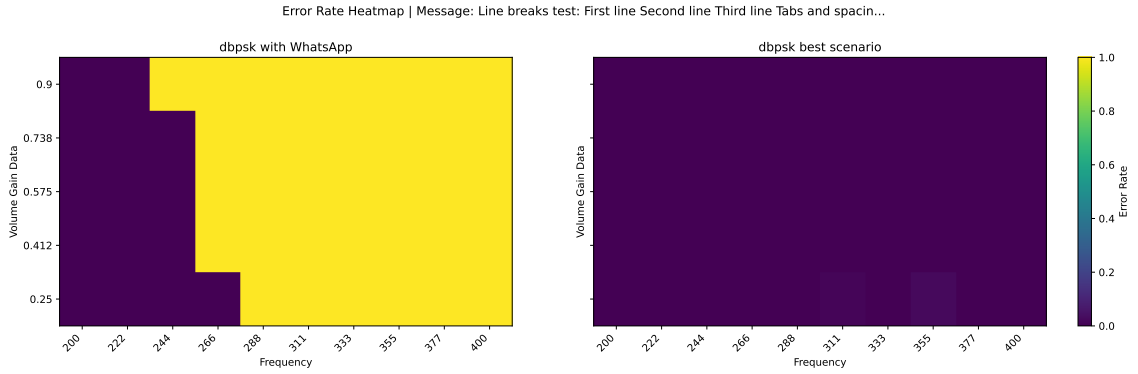


Figure 86: Heatmap plot for DBPSK algorithm with period 7

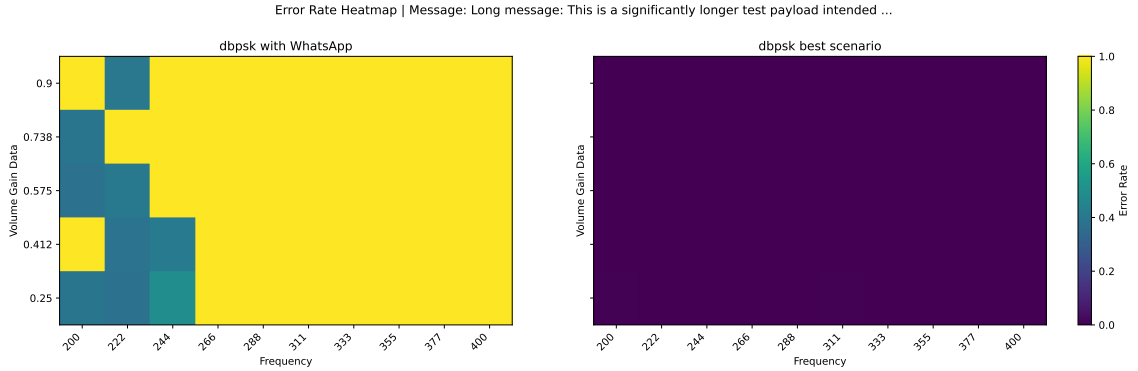


Figure 87: Heatmap plot for DBPSK algorithm with period 7

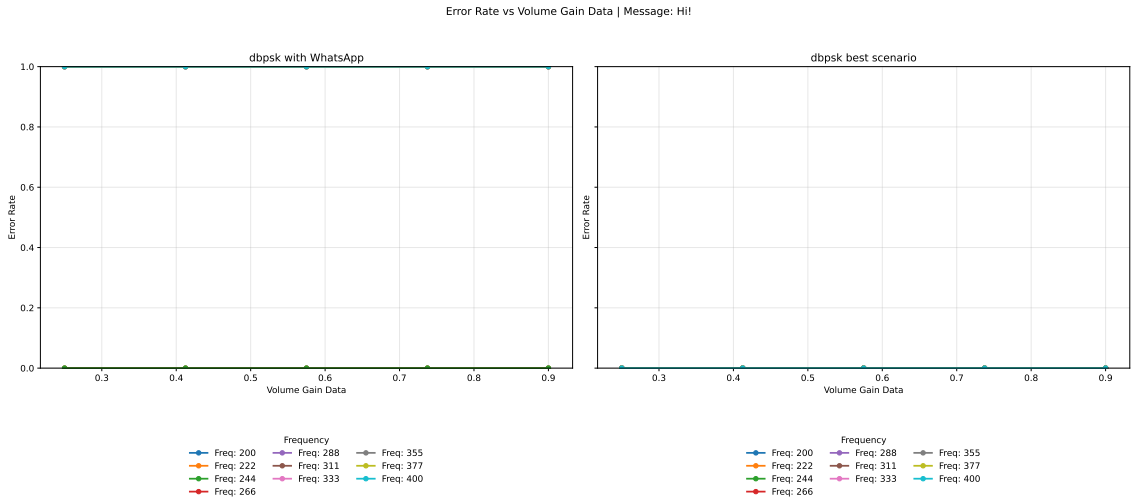


Figure 88: Line plot for DBPSK algorithm with period 2

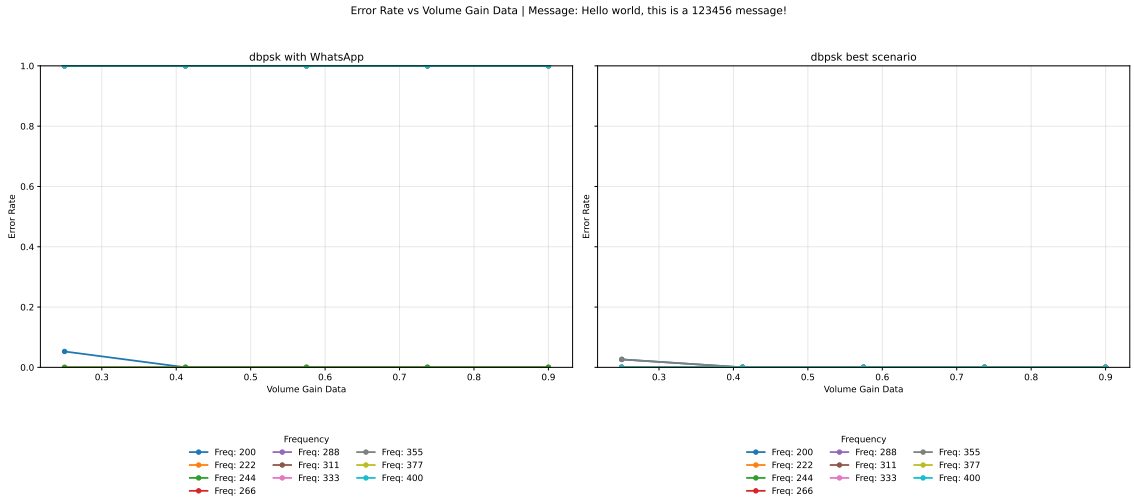


Figure 89: Line plot for DBPSK algorithm with period 2

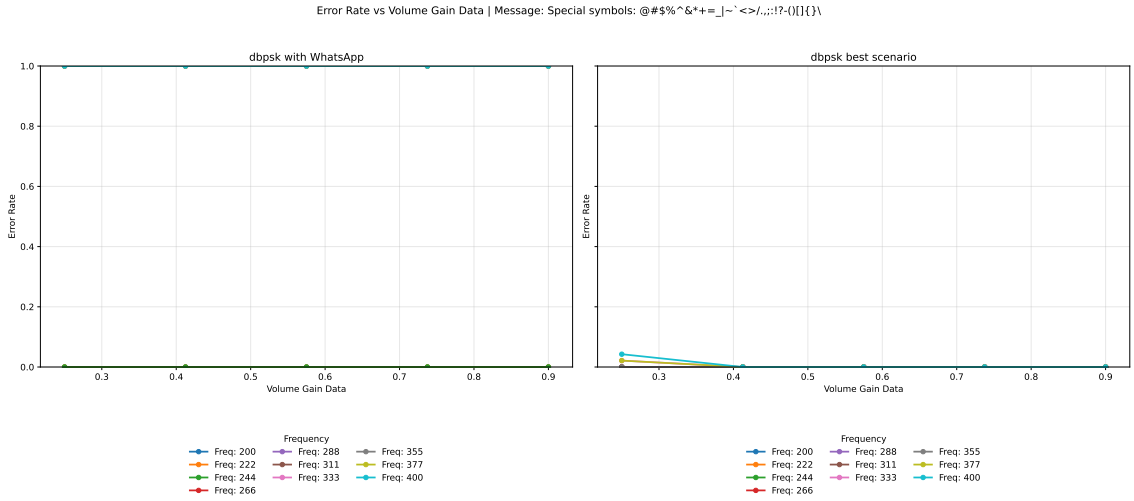


Figure 90: Line plot for DBPSK algorithm with period 2

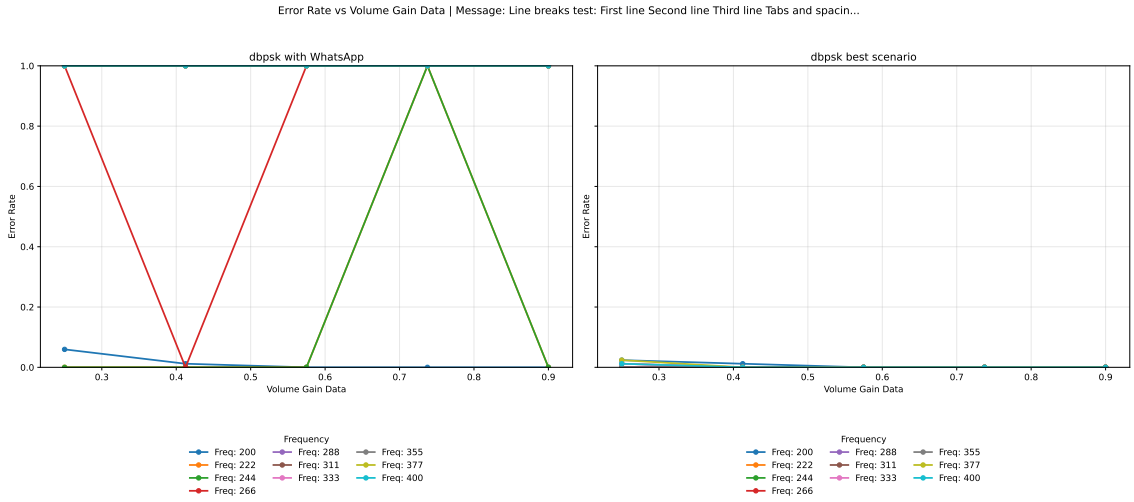


Figure 91: Line plot for DBPSK algorithm with period 2

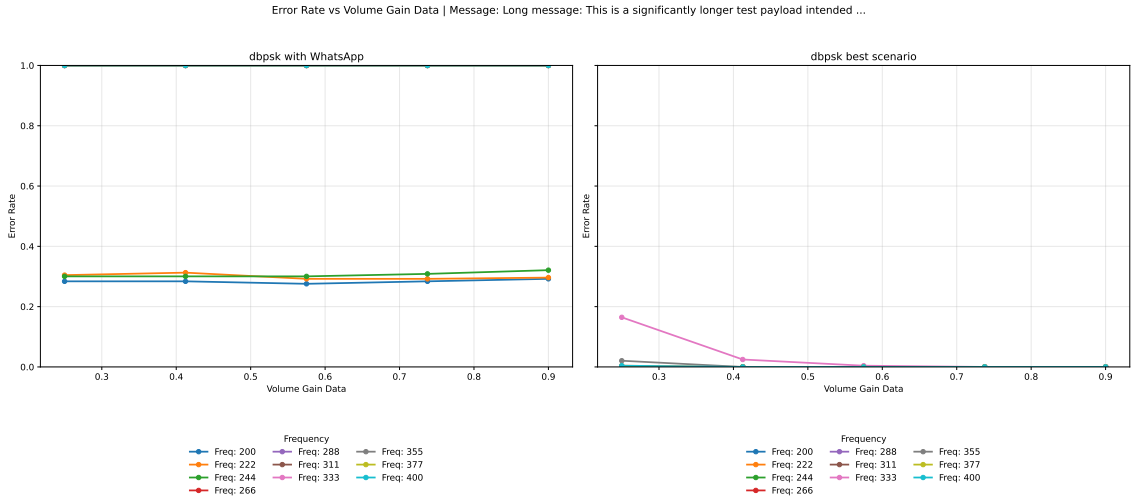


Figure 92: Line plot for DBPSK algorithm with period 2

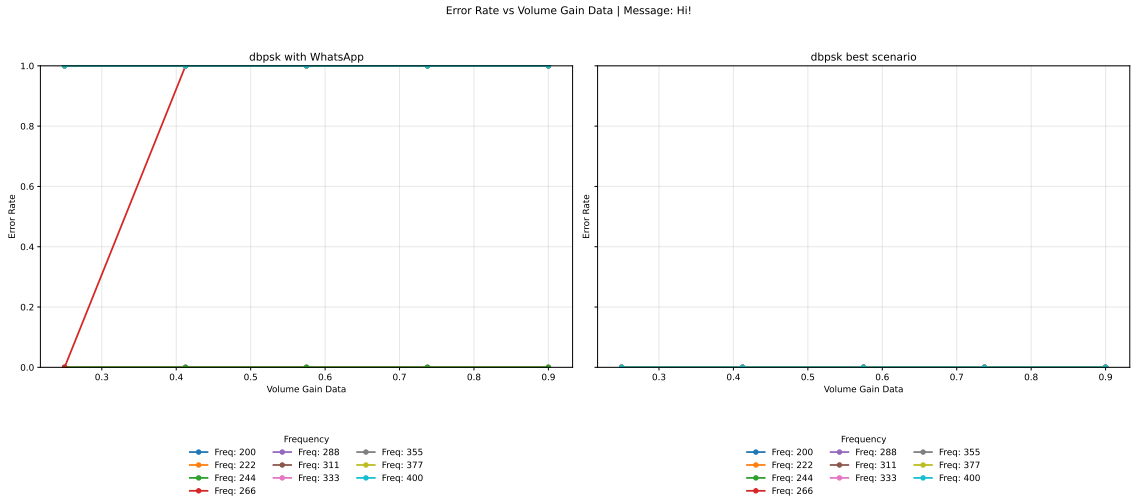


Figure 93: Line plot for DBPSK algorithm with period 5

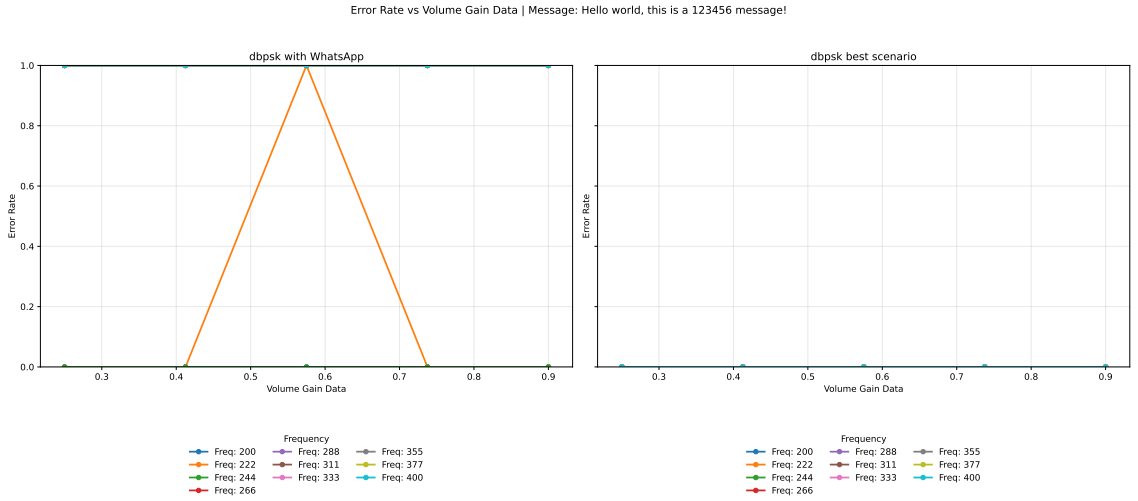


Figure 94: Line plot for DBPSK algorithm with period 5

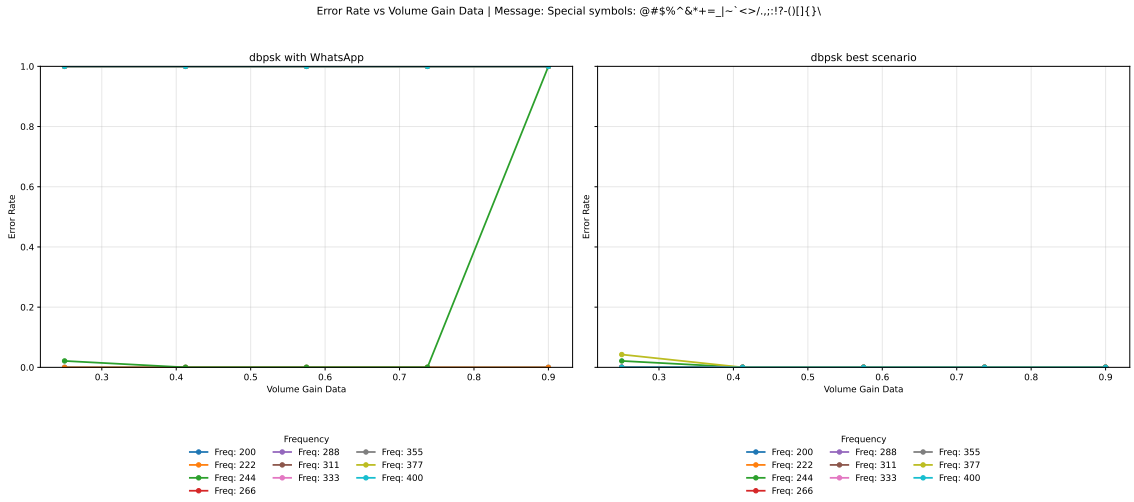


Figure 95: Line plot for DBPSK algorithm with period 5

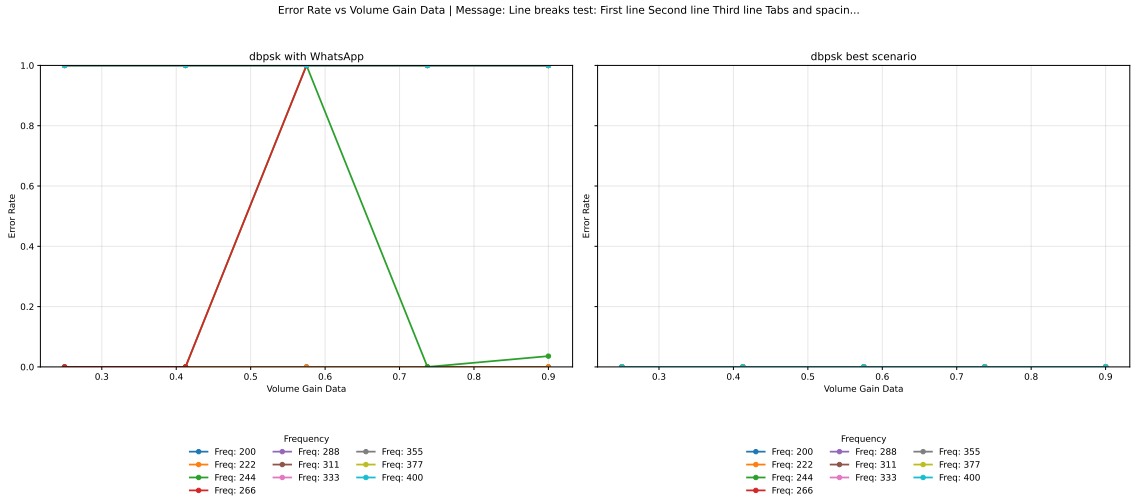


Figure 96: Line plot for DBPSK algorithm with period 5

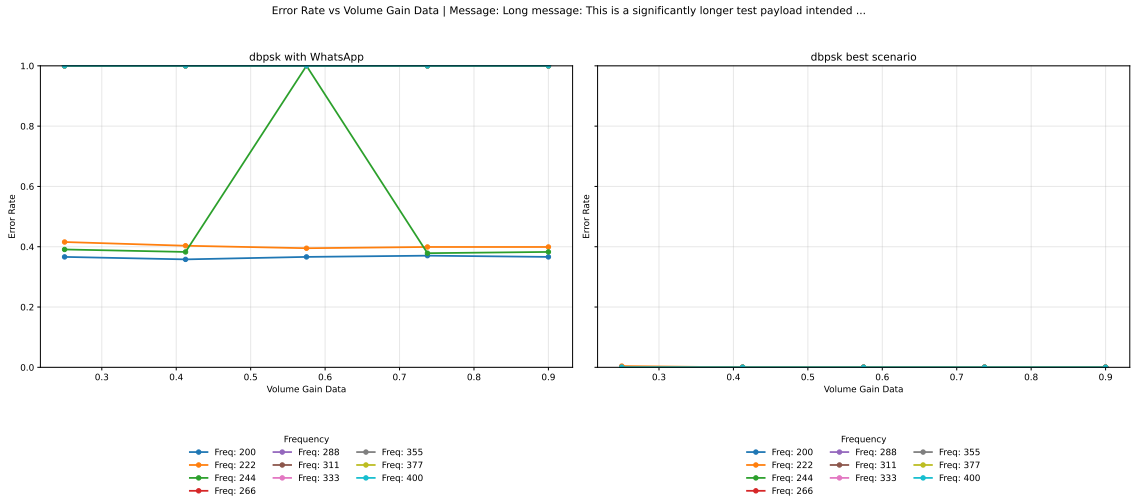


Figure 97: Line plot for DBPSK algorithm with period 5

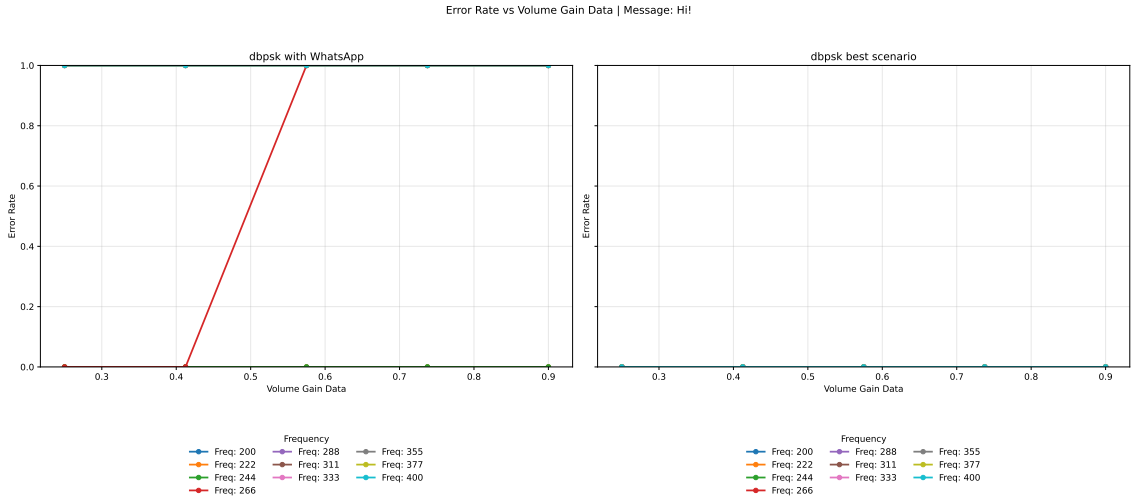


Figure 98: Line plot for DBPSK algorithm with period 7

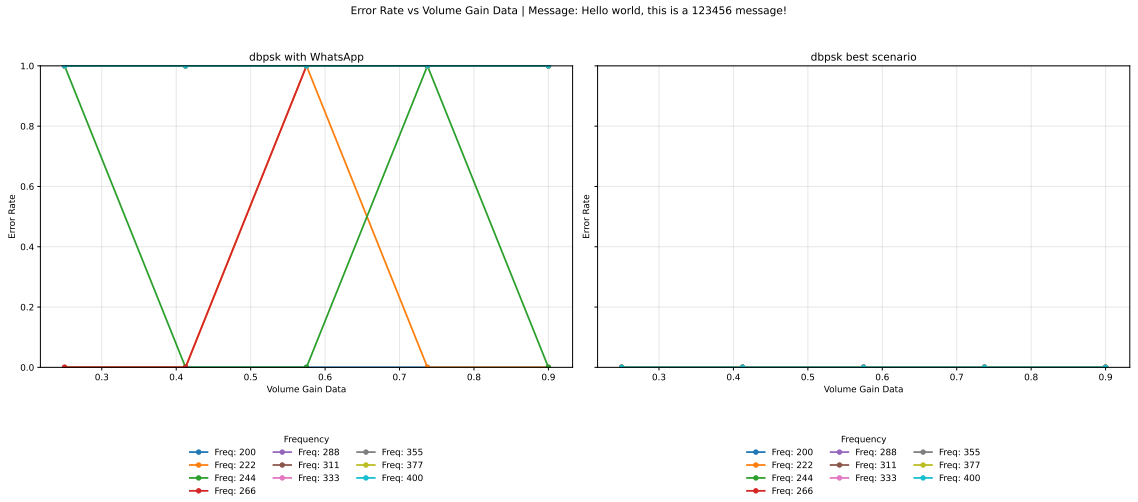


Figure 99: Line plot for DBPSK algorithm with period 7

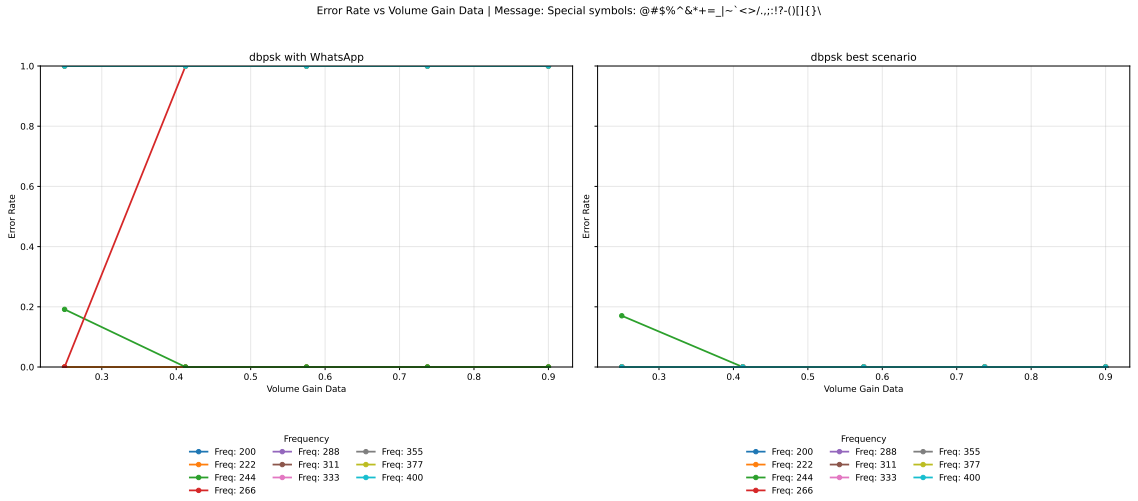


Figure 100: Line plot for DBPSK algorithm with period 7

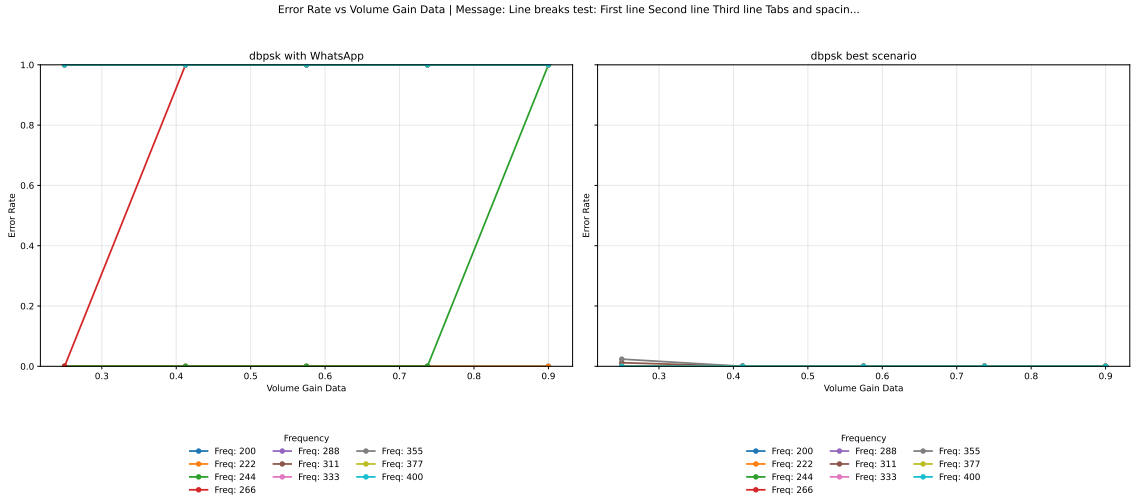


Figure 101: Line plot for DBPSK algorithm with period 7

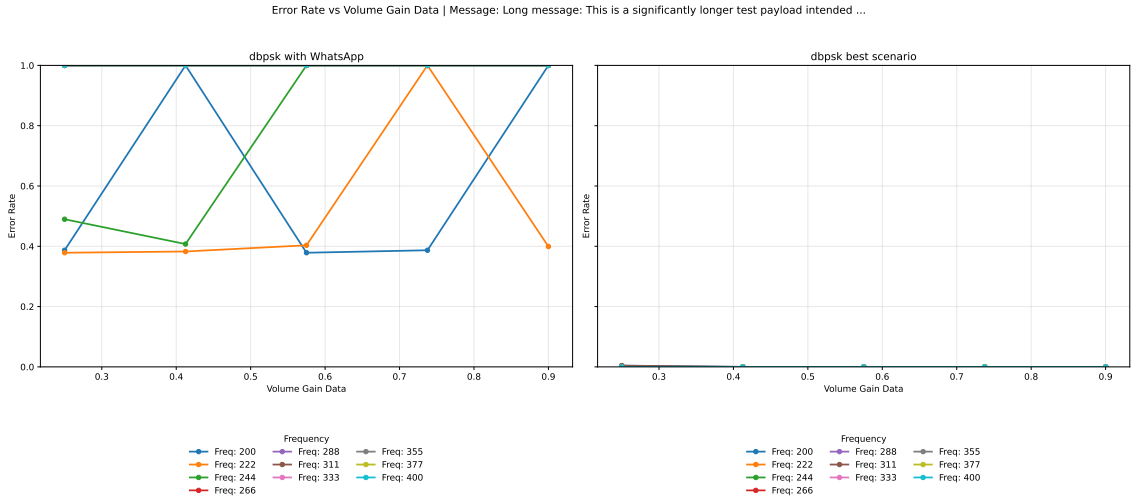


Figure 102: Line plot for DBPSK algorithm with period 7

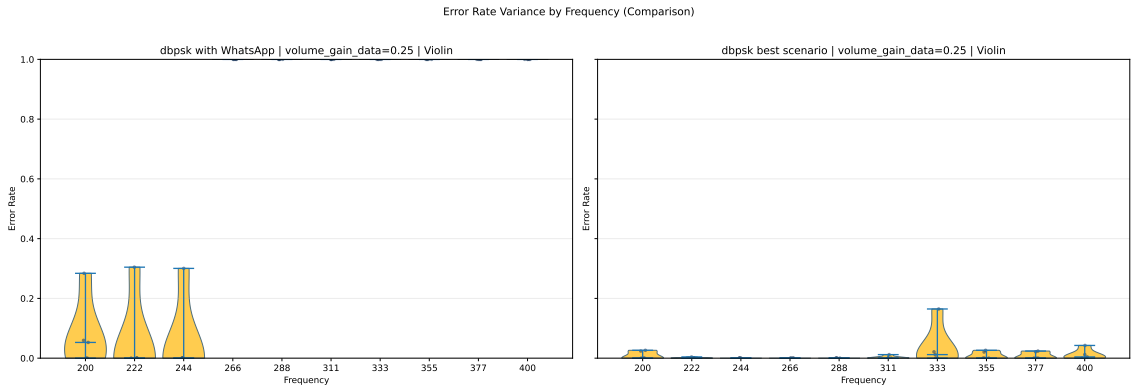


Figure 103: Violin plot for DBPSK algorithm with period 2

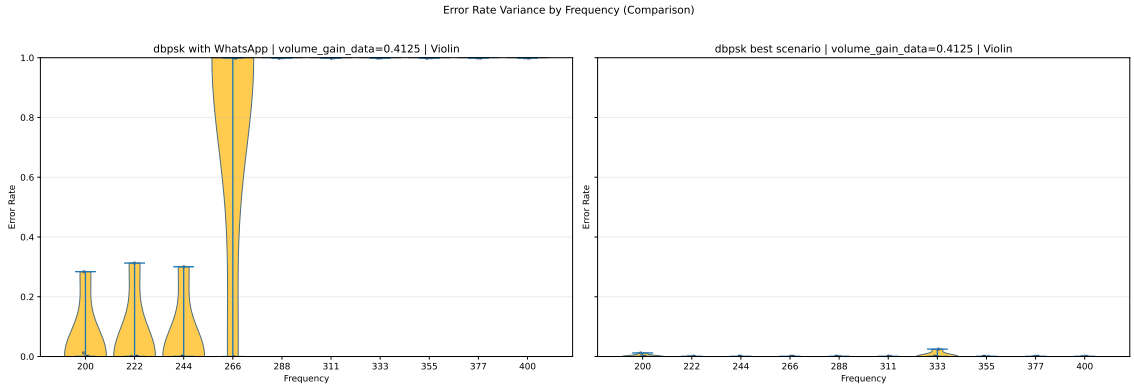


Figure 104: Violin plot for DBPSK algorithm with period 2

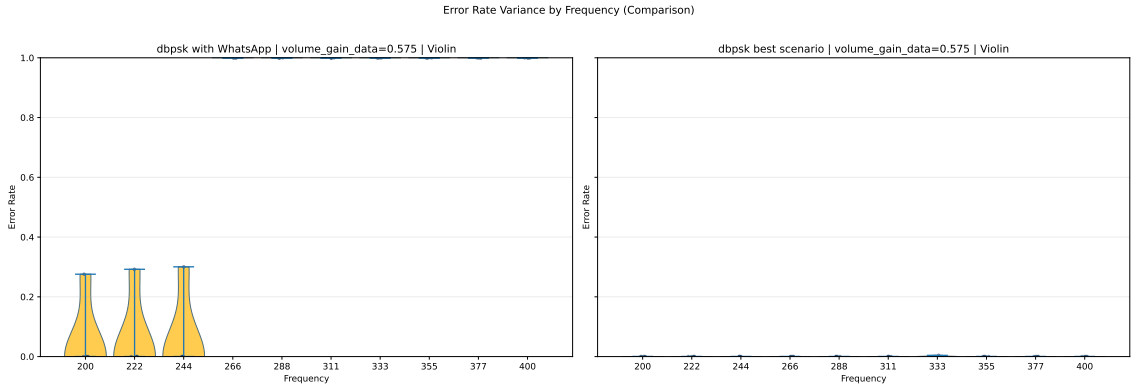


Figure 105: Violin plot for DBPSK algorithm with period 2

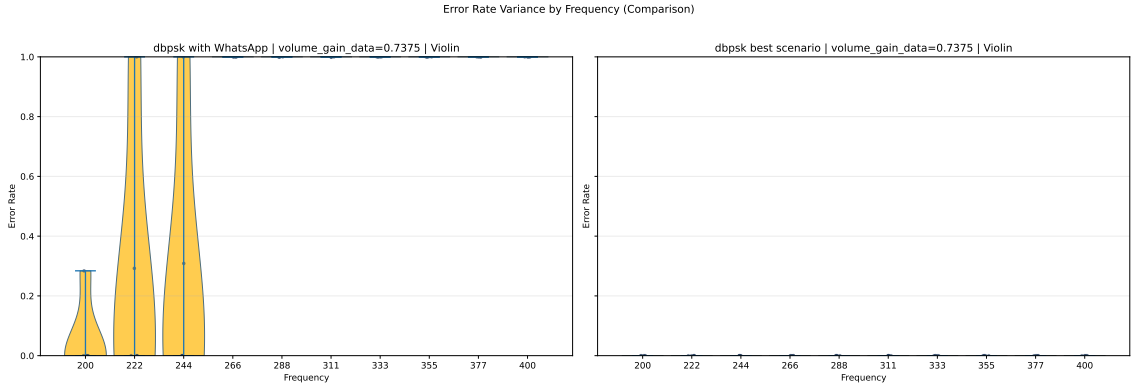


Figure 106: Violin plot for DBPSK algorithm with period 2

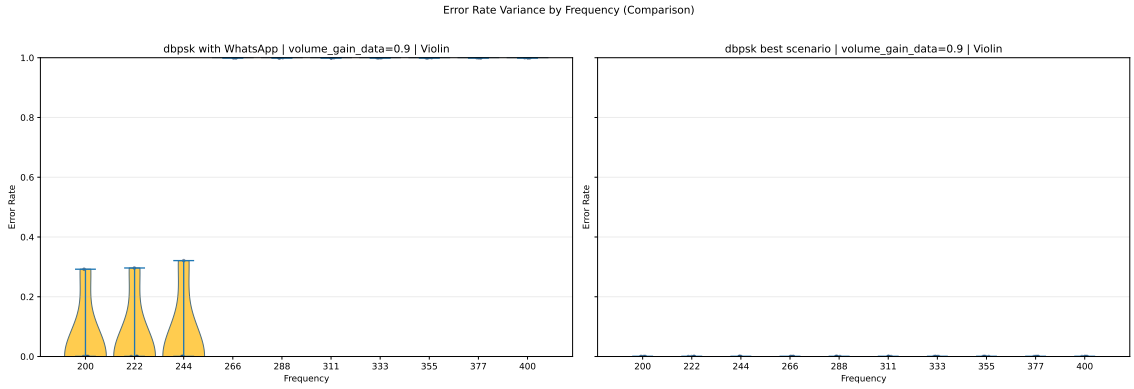


Figure 107: Violin plot for DBPSK algorithm with period 2

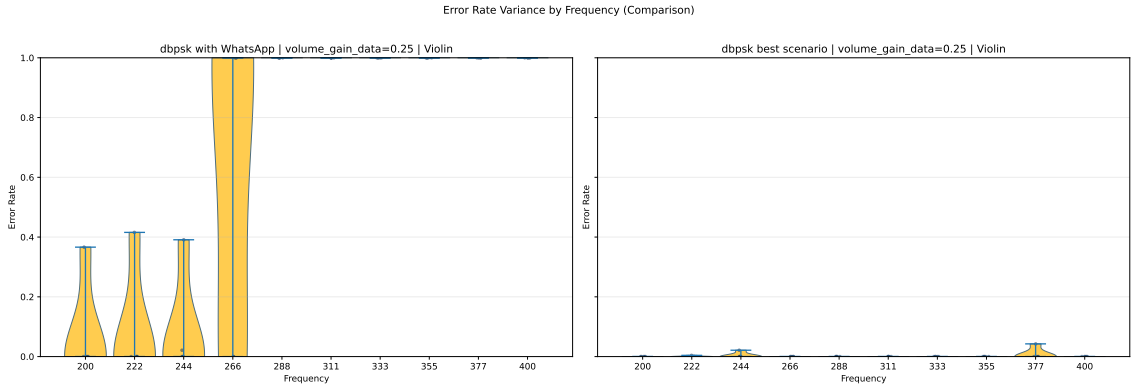


Figure 108: Violin plot for DBPSK algorithm with period 5

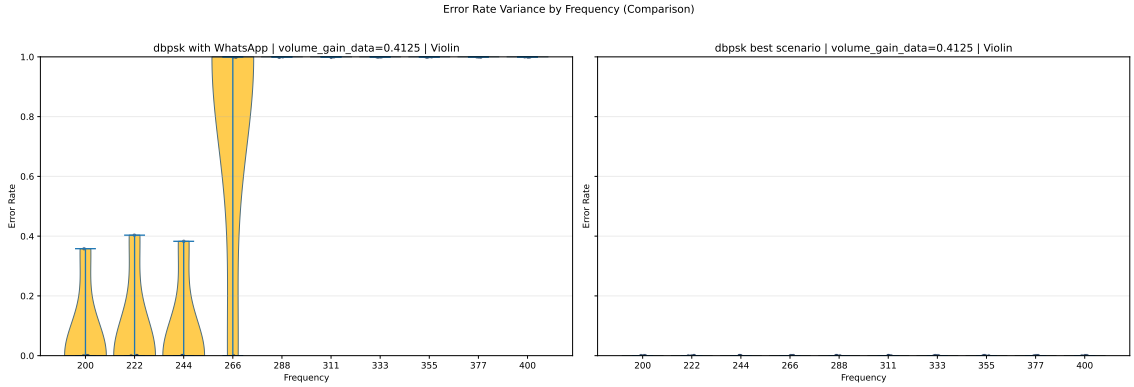


Figure 109: Violin plot for DBPSK algorithm with period 5

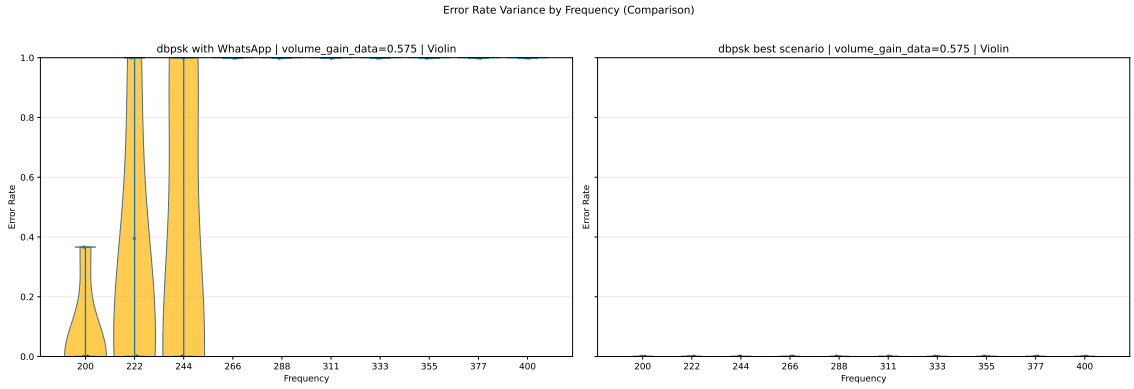


Figure 110: Violin plot for DBPSK algorithm with period 5

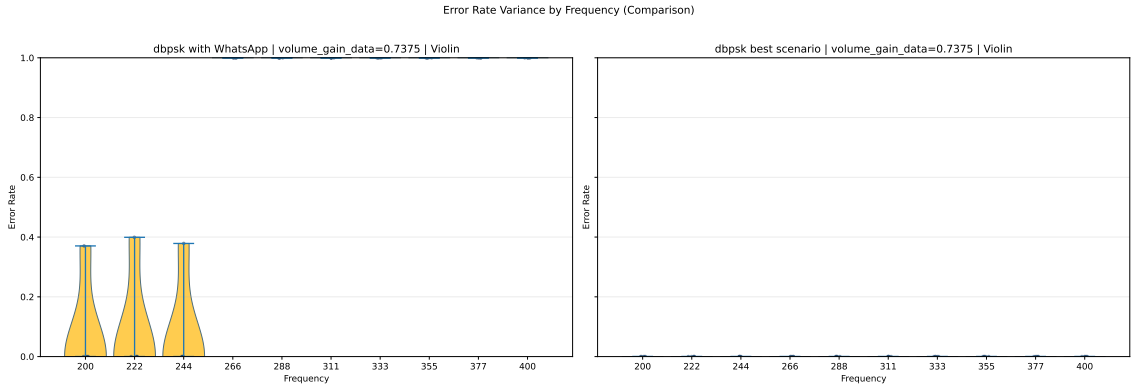


Figure 111: Violin plot for DBPSK algorithm with period 5

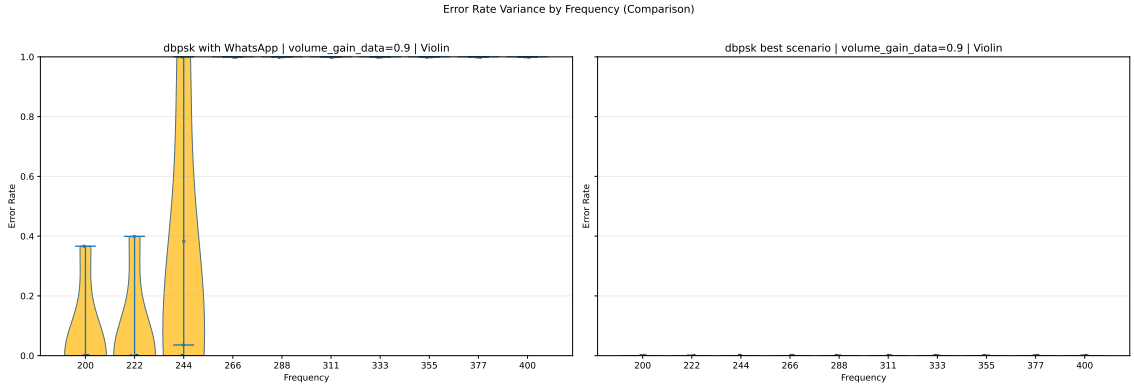


Figure 112: Violin plot for DBPSK algorithm with period 5

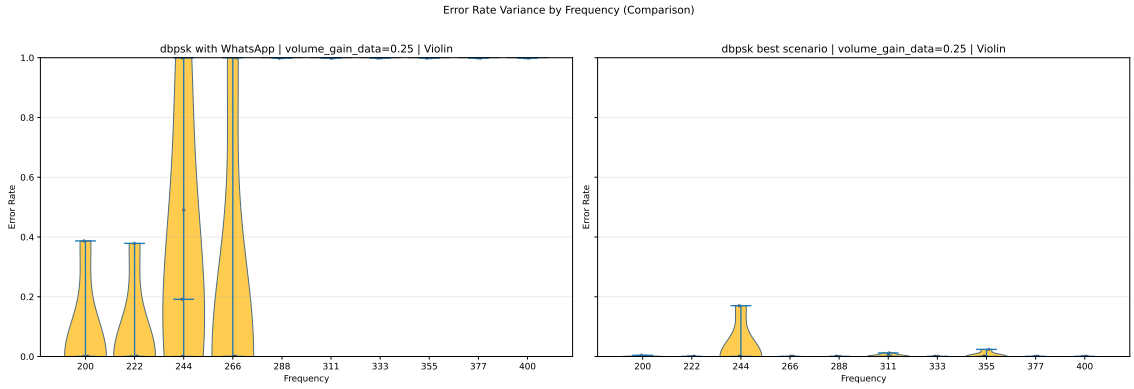


Figure 113: Violin plot for DBPSK algorithm with period 7

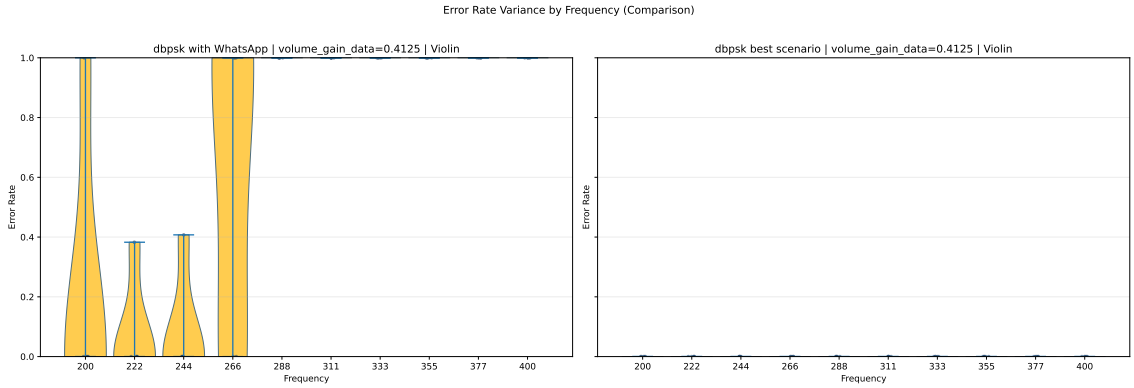


Figure 114: Violin plot for DBPSK algorithm with period 7

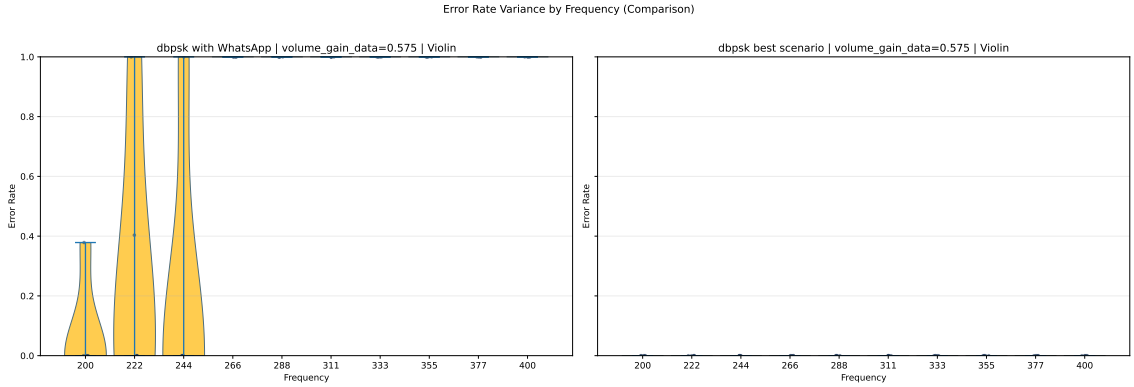


Figure 115: Violin plot for DBPSK algorithm with period 7

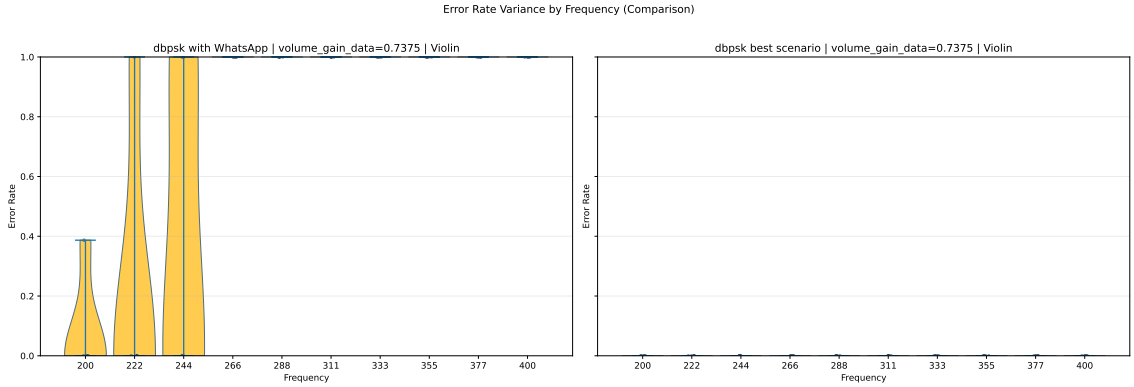


Figure 116: Violin plot for DBPSK algorithm with period 7

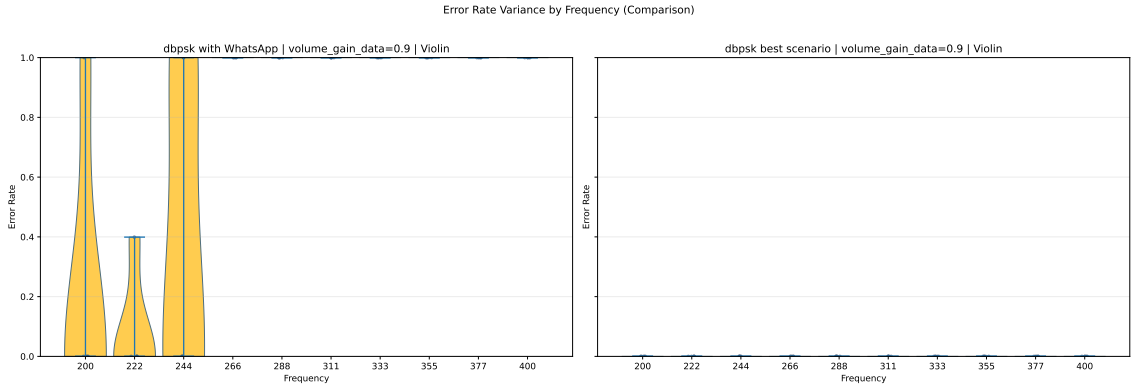


Figure 117: Violin plot for DBPSK algorithm with period 7