Performance and Decoding Complexity Analysis of Short Binary Codes

by

Bo Lian

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Performance and Decoding Complexity Analysis of Short Binary Codes

Bo Lian
Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto
2019

Motivated by emerging 5G wireless systems supporting ultra-reliable low-latency applications, this work studies performance-complexity trade-offs for short block length codes. While well-established tools exist for code optimization of long block length codes, there is no universal approach to the code design problem for short block lengths.

Three candidate approaches for short block length designs are considered:

1) tail-biting convolutional codes decoded with the wrap-around Viterbi algorithm (WAVA),

2) polar codes decoded with successive-cancellation (SC) and an SC-list algorithm aided with error detection,

3) tail-biting convolutional codes and a class of random linear codes with a particular index profile decoded with a sequential decoding algorithm.

Simulation results show that polar codes have a beneficial performance-complexity trade-off for moderate block lengths at or above 512 bits, but at shorter lengths sequentially decoded codes can have a better trade-off. WAVA decoding is competitive only at short lengths and for very low error rates.

Acknowledgements

My sincere gratitude goes to my supervisor, Dr. Frank R. Kschischang, for his kind encouragement and patient supervision. I have had the good fortune to benefit from his insightful questions and rich depth of knowledge. It has been a privilege to do research under his guidance.

I would also like to show my gratitude to my colleagues, who have graciously provided me with their time and expertise.

Finally I would like to thank my family, for their everlasting spiritual support.

Contents

Chapter 1

Introduction

1.1 Motivation

According to IMT-2020 objectives [1], the upcoming fifth generation (5G) wireless systems are classified to support three main services: enhanced mobile broadband (eMBB), massive machine-type communications (mMTC) and ultra-reliable low-latency communications (uRLLC) [1]. As shown in Figure 1.1, these three use-cases are characterized based on their attributes of quality-of-service requirements.

- **eMBB (enhanced Mobile Broadband)**: The use-cases of eMBB include data-rate-intensive applications such as streaming and virtual reality that requires high data rate and coverage. The requirement of bandwidth is 100 Mb/s per user, with peak data rate up to 20Gb/s.

- **mMTC (massive Machine-Type Communications)**: The use-cases of mMTC generally require to support a great number of end devices (up to 1 million devices/km^2 in urban environments), with a long battery life of about 10–15 years.

- **uRLLC (ultra-Reliable Low-Latency Communications)**: The use-cases of uRLLC include the services and applications that are sensitive to latency and reliability, such as tactile interaction and process automation. The reliability needed for such applications is expected to range from a frame error rate of $10^{-5} \sim 10^{-9}$, depending on the type of application [2]. Latency for this type of application is expected to be less than 10 ms for the control plane and 2 ms for the user plane [3].

The requirements of the services and applications for 5G uRLLC are more stringent than those of 4G. As mentioned above, while the end-to-end latency of typical 4G systems is

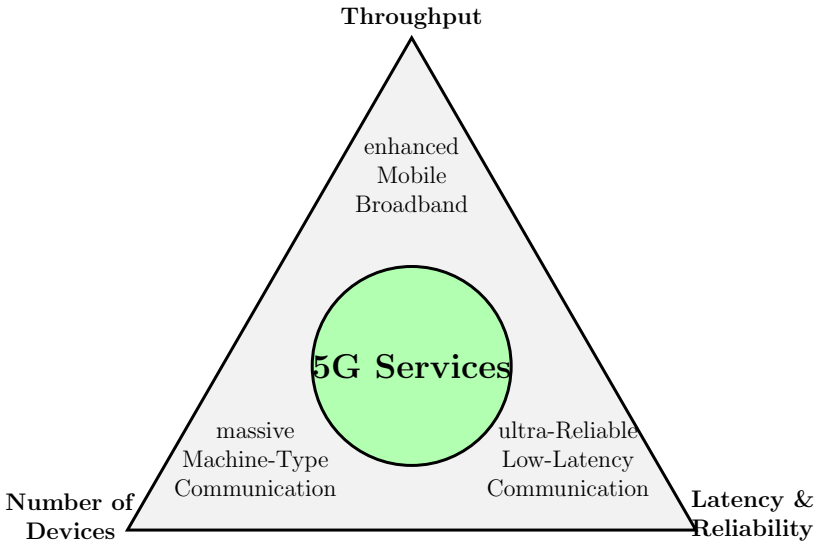

Figure 1.1: 5G service classes

about 20–80 ms with reliability greater than 99% [3], 5G uRLLC has a more stringent end-to-end latency and target reliability.

Table 1.1 lists several services and applications that require the support of relatively low latency and high reliability that are expected to be achieved by 5G technologies, where the challenging constraints are not achievable with the current 4G technologies. Most of these requirements follow the 3GPP standard, while other standards may prescribe different requirements. For instance, the ETSI standard specifies the error probability of factory automation to be 10^{-9} [4], while the error probability of discrete automation in the 3GPP standard is $10^{-4} \sim 10^{-6}$ [2].

Figure 1.1 provides an illustration of the requirements of the latency and reliability critical applications (after [5]). It shows that not all the scenarios are covered by the 5G uRLLC target. For instance, tactile interaction requires strict probability of error around 10^{-7}, with an end-to-end latency less than 1 ms for safety-related services such as remote surgery, which is beyond the range that uRLLC can provide.

One major challenge in the design of 5G uRLLC wireless systems is to satisfy the inherent conflict between ultra-high reliability and low latency, where the low latency scenarios require short block length. The typical FER of 4G wireless systems can be achieved by re-transmission mechanisms and capacity-achieving channel codes (e.g., turbo codes), but 5G uRLLC scenarios have higher reliability requirements, while the number of available channel uses and block length are limited by the latency constraints that are usually more stringent than 4G. On the other hand, the 5G wireless system is expected to support novel traffic types that inherently require short packets, such as Machine-to-Machine

Scenario	Max allowed end-to-end (E2E) latency	Reliability (Frame Error Rate)
Discrete automation [2]	1 ms ∼ 10 ms	$10^{-4} \sim 10^{-6}$
Process automation [2]	50 ms	$10^{-3} \sim 10^{-6}$
Electricity distribution – medium voltage [2]	25 ms	10^{-3}
Electricity distribution – high voltage [2]	5 ms	10^{-6}
Intelligent Transportation Systems [2] [6]	10 ms ∼ 100 ms	$10^{-3} \sim 10^{-6}$
Tactile interaction [7] [8] [9]	0.5 ms ∼ 1 ms	$10^{-5} \sim 10^{-7}$
Virtual reality [10] [3]	1 ms ∼ 10 ms	10^{-5}
Augmented reality [8] [11]	7 ms	$5 \times 10^{-2} \sim 10^{-5}$
Remote Control [2]	5 ms	10^{-5}
Smart Grid [7]	3 ms ∼ 20 ms	10^{-6}

Table 1.1: Latency and reliability requirements for 5G scenarios

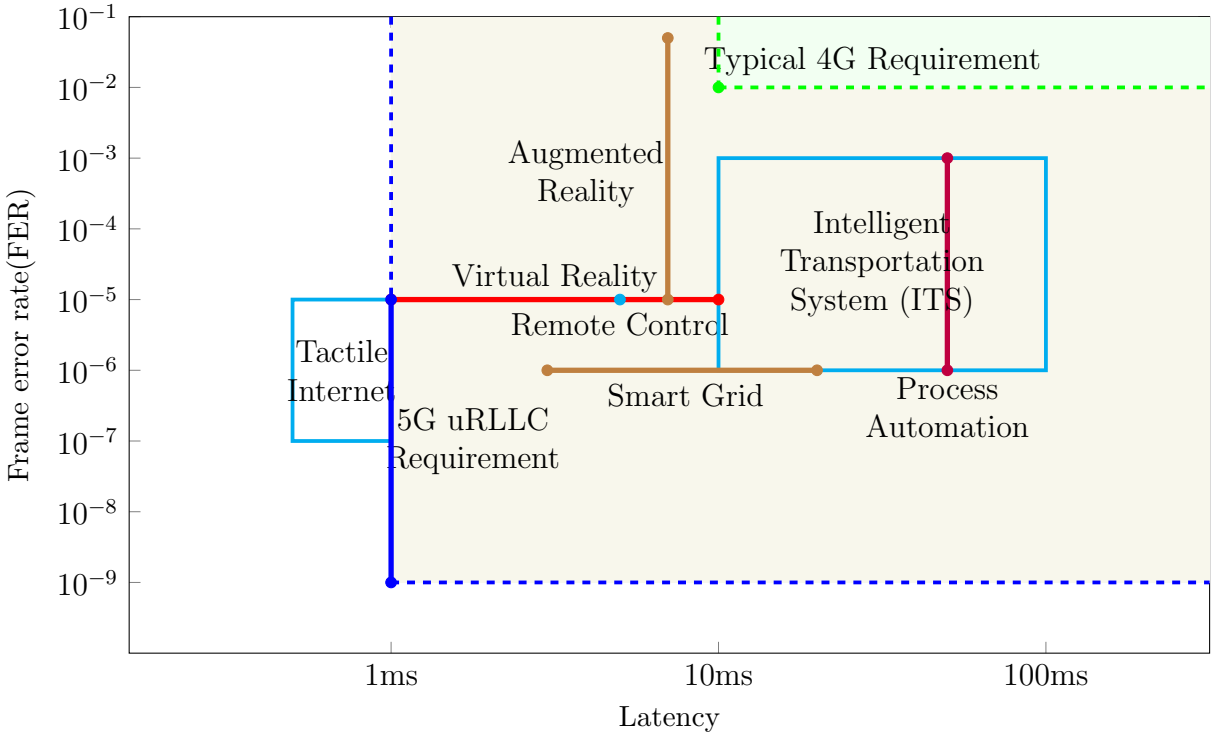

Figure 1.2: Latency and FER critical 5G applications

communications and wireless sensor networks. Therefore, short block length code design is an important aspect of 5G.

To analyze the coding schemes, decoding complexity is always a critical factor. Since the operations can be processed in parallel, multiple operations can be completed in one clock cycle, hence the computational complexity cannot be considered as proportional to the latency, but they are certainly positively correlated.

As the interest in short block length codes has risen again recently, there are numerous works about design and analysis of short codes based on existing coding schemes, e.g., [12–14]. Unlike long block length codes, the transmission of short codes usually doesn't provide efficiently many channel uses to average out the effects of noise. For long block length codes there are well-established tools to design capacity-approaching coding algorithms, while for short block length codes, with block length N less than 1000 bits, there is no universal approach to the code design problem. Many existing high-performance coding schemes are designed for sufficiently long block length, where the law of large numbers averages out the stochastic variations in the noise, but this does not apply in the case of short blocks [15]. For instance, low-density parity check (LDPC) codes can achieve performance close to the Shannon limit for long code length, as in the long length regime, the girth of the Tanner graph can be arbitrarily large. On the other hand, for short block lengths, the girth is small and the code performance is limited by it, hence there is no promise for LDPC codes to have best-in-class performance for short block lengths. It has been shown in [16] that for short block lengths the conventional decoding algorithms can be performance-wise competitive with respect to iterative decoding algorithms, with relatively low complexity. Consequently, the code design for uRLLC is still a challenging topic.

1.2 Outline

This thesis reviews and simulates different error control codes that are expected to be able to achieve satisfactory trade-off between performance and computational complexity for 5G uRLLC scenarios. The objective of this work is to extract insights from the evaluations and simulation results to provide general guidelines for the design of practical channel coding schemes for uRLLC.

The remainder of the thesis is organized as follows. Chapter 2 reviews theoretic preliminaries on error control coding. The definition of the metrics and other important related terminology are briefly reviewed. Each of Chapters 3, 4 and 5 reviews, discusses and simulates a type of competitive short block length coding scheme. Chapter 3 treats

tail-biting convolutional codes, Chapter 4 treats polar codes, and Chapter 5 treats linear block codes with sequential decoding. Chapter 6 compares the simulation results of different coding algorithms in terms of code imperfectness and computational complexity, provides conclusions based on the Pareto frontier, and makes suggestions for future work.

Chapter 2

System Model and Background

2.1 System Model

When digital data are transmitted over a communication channel, they are inevitably affected by noise, which may cause undesired data corruption. In order to reduce the probability of such scenarios, error control coding is employed to enhance the reliability of data. The key idea of error-control is to introduce redundancy in the transmitted data that can be exploited by the receiver to make better decisions.

Fig. 2.1 shows the channel model of a digital communication system. Source encoding aims to represent the data source efficiently as a sequence of binary symbols, and channel encoding formats the data to increase its immunity to noise and make digital data transmission and storage systems efficient and reliable; hence this process is known as error control. Error control coding can increase the reliability of data by adding redundancy to it during the encoding process before it is exposed to channel noise. Once the encoded data symbols that are possibly corrupted by noise are received, the decoder will attempt to recover the original data.

The technique that requests a retransmission of the data symbols if errors are detected is known as automatic repeat request or automatic repeat query (ARQ). However, ARQ requires the channel to support feedback, and can be inefficient in the sense of latency and energy.

On the other hand, forward error correction (FEC) techniques can not only detect, but also correct errors in the received signal based on the corrupted data and redundant symbols. Although the implementation of error correction is more complex than error detection, there is still a significant advantage to automatically correct certain errors without the requirement of retransmission.

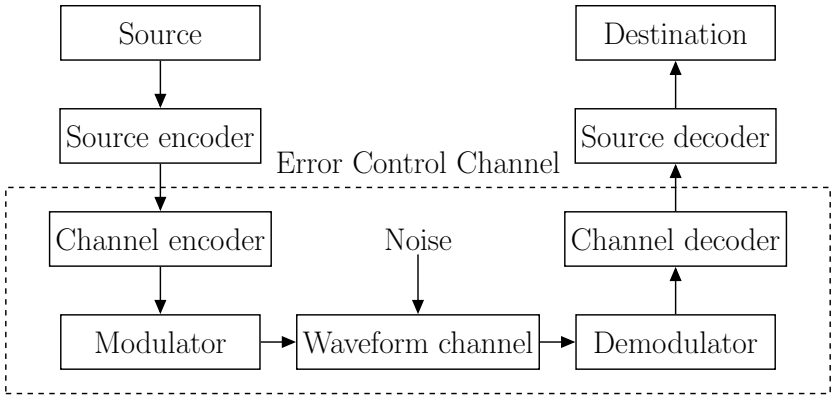

Figure 2.1: Model of a digital communication system

In this thesis, we consider the transmission by binary phase shift keying (BPSK) modulation of binary linear block codes over linear and time-invariant channels with additive white Gaussian noise (AWGN). The performance of different coding schemes is compared with Monte Carlo simulation. In simulation, uniform pseudorandom numbers are generated using the well equidistributed long-period linear (WELL) Random number generator [17], and the Box-Muller method is used to transform uniformly distributed random numbers to Gaussian distributed random numbers [18].

For the design process of a code, we first need to know the channel. In wireless systems, fading channels are commonly considered, but as long as the channel state is known and the fading process is slow and independent of the channel input, for short block length codes we can analyze the properties based on the results of the AWGN channel.

Once the channel is known, the next step of code design is to decide on the modulation scheme. In 5G scenarios, modulations with higher order are frequently used. While the modulation of BPSK has Gaussian distributed log-likelihood ratio (LLR), the modulation with higher order has different LLR distributions. However, while the channel state is independent of the input sequence, the LLRs produced by the channel can still be considered as approximately Gaussian, which is well known in bit-interleaved coded modulation (BICM) [19].

After both channel model and modulation scheme are determined, the code can be designed with signal power requirements, latency requirements, computational complexity requirements, reliability requirements, etc., which are discussed in the following sections.

2.2 Decoding

In error control coding systems, the central aim of decoding is to recover the information data from the noise-corrupted received signal. The decoding process can be either complete or incomplete. A complete decoder always decodes the received information data to some codeword, hence the errors are required to be not only detected but also corrected. On the other hand, an incomplete decoder is allowed to declare a decoding failure, where the errors are detected but not necessarily corrected. In this thesis, we focus only on complete decoders.

For an n-tuple codeword $\mathbf{v}$ and the corresponding received sequence $\mathbf{r}$, $P(\mathbf{r}|\mathbf{v})$ is the probability that $\mathbf{r}$ is received after the transmission of $\mathbf{v}$. Maximum likelihood (ML) decoding is a basic complete decoding approach with the following principle: $\mathbf{r}$ is always decoded to a codeword $\mathbf{v}$ that maximizes $P(\mathbf{r}|\mathbf{v})$. Maximum likelihood detection gives minimum probability of error (assuming codewords are equiprobable), but the computational complexity of ML decoding can be as bad as $\mathcal{O}(N2^{NR})$ for binary linear block codes, where R denotes the code rate and N denotes the block length. Since the decoding complexity of ML decoding can exceed the acceptable range, a vast variety of decoding algorithms have been developed to achieve reasonable trade-off between code performance and computational complexity. Precise definitions of performance and complexity are provided in the following chapter.

2.3 Metrics

2.3.1 Reliability

Reliability is generally defined as the probability that a piece of data can be successfully transferred to the destination. There are multiple ways in which reliability can be assessed, since a piece of data may denote a bit, frame, a packet, or a block. For instance, bit error rate (BER) is expressed as the fraction of error bits in the decoded sequences with respect to the total number of information bits. Similarly, frame error rate (FER) is the expected value of the ratio of the number of frames that contain error bits to the total number of transferred frames. Packet loss rate (PLR) and block error rate (BLER) are the measurements with the same definition but different size of data. The size of frame, packet and block are determined by the communication protocol, but in this thesis we assume they are the same length for convenience. In practice, when a piece is not successfully transferred, the whole piece is required to be re-transmitted, hence FER (or PLR, BLER) is often regarded as more stringent and commonly used than BER for the

measurement of performance. The value of these measurements is often expressed as a percentage or in the form 10^{-x}.

The reliability requirement of 5G uRLLC scenario is $1 - 10^{-5}$, which refers to the success probability of transmitting a layer 2 protocol data unit (PDU) of 32 bytes within 1 ms [2].

For modulation formats in one dimension, such as BPSK and M-PAM, we define

$$\mathrm{SNR}_{norm} = \frac{\mathrm{SNR}}{2^{2R_t} - 1} = \frac{E_s}{\sigma^2(2^{2R_t} - 1)}, \tag{2.1}$$

where R_t is the transmission rate (bits/symbol), σ^2 is the noise variance per dimension ($\sigma^2 = N_0/2$ for an AWGN channel with two-sided noise power spectral density $N_0/2$), and $E_s = E[x^2]$ is the average energy of the transmitted symbol X.

For modulation formats in two dimensions, such as QAM, we define

$$\mathrm{SNR}_{norm} = \frac{\mathrm{SNR}}{2^{R_t} - 1} = \frac{E_s}{2\sigma^2(2^{R_t} - 1)}, \tag{2.2}$$

where R_t is the transmission rate (bits/symbol), σ^2 is the noise variance per dimension, and $E_s = E[\|X\|^2]$ is the average energy of the transmitted vector X.

For example, a BPSK transmission using a signal alphabet $\{+1, -1\}$ and a code of rate $1/2$ has

$$\mathrm{SNR}_{norm} = \frac{1}{\sigma^2(2^1 - 1)} = \frac{1}{\sigma^2} = \frac{2}{N_0}. \tag{2.3}$$

A QPSK transmission using a signal alphabet $\{(+1, +1), (+1, -1), (-1, +1), (-1, -1)\}$ and a code of rate $1/2$ has $R_t = 1$ and

$$\mathrm{SNR}_{norm} = \frac{2}{2\sigma^2(2^1 - 1)} = \frac{1}{\sigma^2} = \frac{2}{N_0}. \tag{2.4}$$

This is the same as the BPSK system, since the QPSK system is composed of two BPSK systems operating in quadrature.

In general, for modulation formats in N_m dimensions, we define

$$\mathrm{SNR}_{norm} = \frac{\mathrm{SNR}}{2^{2R_t/N_m} - 1} = \frac{E_s}{N_m\sigma^2(2^{2R_t/N_m} - 1)}, \tag{2.5}$$

where R_t is the transmission rate (bits/symbol), σ^2 is the noise variance per dimension, and $E_s = E[\|X\|^2]$ is the average energy of the transmitted N-dimensional vector X.

In this thesis we consider only the case of BPSK transmission with codes of rate $R = R_t$, so SNR_{norm} is given by (2.1).

2.3.2 Latency

The overall latency of 5G services is determined by the radio interface delay, the delay in data processing, and the transmission delay within and outside the 5G system. Latency may have different definitions depending on the start point and the end point of the data. In this thesis we focus on end-to-end latency.

End-to-end latency is defined as the required time from the moment of target information transmitted by the source to the moment of the information is fully received by the destination, which includes the transmission delay, queuing delay, computing delay and re-transmission delay [2] [20].

The effect of latency always needs to be considered in code design. Without a latency constraint, arbitrary reliability is achievable by re-transmission or reducing code rate R. For the scenarios that require extremely low latency, the block length is also constrained to be short.

Referring to Fig 2.1, the overall end-to-end latency is from the encoding and decoding process, buffer-fill at the encoder and decoder, modulation, demodulation, and propagation delay, hence the overall end-to-end latency can be expressed as

$$T_{overall} = T_{dec} + T_{block\_dec} + T_{block\_enc} + T_{prop} + T_{mod} + T_{demod}. \tag{2.6}$$

where $T_{overall}$ is the overall end-to-end latency, T_{dec} is the latency of decoding process, T_{block_dec} and T_{block_dec} are the block transmission time at encoder and decoder, T_{mod} and T_{demod} are the latency from modulation and demodulation, and T_{prop} is the propagation delay.

Under a given data rate, the block transmission time at encoder and decoder can be considered as identical, which is the time of buffer filling. On the other hand, the decoding time is upper bounded by the block transmission time to prevent buffer overflow. Let T_{block} represent the block transmission time, (2.6) can be rewritten as:

$$T_{overall} \geq 3T_{block} + T_{prop} + T_{mod} + T_{demod}. \tag{2.7}$$

The delay on modulation and demodulation process depends on the modulation scheme used. If the information symbols are modulated and demodulated one by one, then the time required is a constant. If an equalizer is used in block demodulation, then the demodulation time is proportional to the block size. A more extreme case is the orthogonal frequency-division multiplexing (OFDM) with required time scaled as 2^x symbols for inverse discrete Fourier transform (IDFT) and discrete Fourier transform (DFT) at modulator and demodulator, which could dominate the overall latency when

x is large and N is relatively small, but such situation is very rare.

Consider an example of discrete automation scenario with end-to-end latency of 1ms, data rate of 1 Mb/s, with service region dimension of $100 \times 100 \times 30$ m [2]. The worst case transmission distance of this example is about 144 m, and if we assume the propagation velocity to be 3×10^8 m/s, then the propagation delay is about 480 ns. On the other hand, assume code rate $R = 1/2$, since data rate is 1 Mb/s, the average rate of buffer filling is 2 Mb/s, and the block transmission time $T_{block} = N/(2 \times 10^6)$ s. If the modulation time and demodulation time can be neglected, then N must satisfy

$$\begin{aligned} 3T_{block} + T_{prop} &\leq T_{overall} \\ \frac{3N}{2 \times 10^6} + 480 \times 10^{-9} &\leq 10^{-3} \\ N &\leq 667; \end{aligned}$$

which is a relatively short block size. This result shows that short codes can be required by 5G uRLLC scenarios.

2.3.3 Complexity

Computational complexity is an important parameter of decoding. Without a constraint on computational complexity, the ML decoding technique will always be optimal in the sense of reliability for any codes. In general, the ML decoding complexity of a binary linear block code depends on the trellis complexity, which could be up to $\mathcal{O}(N2^{NR})$ in the worst case for with rate R and block length N, which is ridiculous for practical use. Therefore, the target of decoding scheme design is to achieve near ML decoding performance with reasonable complexity level.

The decoding complexity is an important factor of latency, since the decoding time is one of the major parts in the overall end-to-end latency. We are able to control the duration of the decoding process by selecting proper algorithms, yielding a sensible trade-off between complexity and performance. The computational complexity is not directly proportional to latency due to the multi-threading in software and hardware implementations, but there is a strong positive correlation between them.

The computational complexity of a given code can be measured by the average number of binary operations required to decode per information bit of the message. In our definition, binary operations include addition, subtraction, multiplication, division,

square-root, table look-up, and comparison, which requires one clock cycle on average to process.

2.4 Sphere Packing Bound

The fact that digital communication can be error free given transmission rate R less than channel capacity C entices engineers to attempt to design communication systems that can achieve this goal. From another perspective, the upper bound of reliability under fixed transmission rate can be seen as a function of signal power. In 1959, Shannon derived a lower bound on the error probability of maximum likelihood decoding for AWGN channel independently of the modulation [21] [22]. This lower bound is called the sphere packing bound since it relies on the error probability of a code with length N whose codewords are uniformly distributed on a sphere must be less than or equal to the error probability of any code with length N whose codewords lie on the same sphere.

The sphere packing bound is given in terms of the block length and rate of the code. For a code with 2^{NR} codewords in the N-dimensional Euclidean space, we define solid angle Ω_i to be the polyhedral region surrounding the ith codeword. For a channel with Gaussian noise, Shannon derived a fundamental lower bound for the error probability in the form [21]

$$P_e \geq Q(A, N, \theta_{crit}), \tag{2.9}$$

where P_e is the probability of error for the codeword, amplitude A of the corresponding signal to noise ratio follows $A = \sqrt{\text{SNR}} = \sqrt{E_s/\sigma^2} = \sqrt{RE_b/\sigma^2}$, and θ_{crit} is the cone angle such that the solid angle in N space of half-angle θ is $1/k$ of the total solid angle of N space, where k represents the code dimension, i.e., $\Omega(\theta_{crit}, N) = \frac{1}{k}\Omega(\pi, N)$.

Let θ denote the half-angle of cones, and let $\Omega(\theta)$ denote the solid angle in N space of a cone of half-angle θ. The exact and approximate expression of $Q(A, N, \theta_{crit})$ and $\Omega(\theta_{crit}, N)$ are given in [23] and [21], respectively. The exact expression for $\Omega(\theta, N)$ is:

$$\Omega(\theta, N) = \frac{(N-1)\pi^{(N-1)/2}}{\Gamma(\frac{N+1}{2})} \int_0^{\theta} (\sin \alpha^{N-2}) d\alpha \tag{2.10}$$

and

$$Q(A,N,\theta_{crit}) = \frac{N-1}{2^{N/2}\sqrt{\pi}\Gamma(\frac{N+1}{2})}\cdot \int_{\theta_{crit}}^{\pi}\int_{0}^{\infty} r^{N-1}(\sin\theta)^{N-2}\exp\Big[\frac{2rA\sqrt{N}\cos\theta - r^2 - NA^2}{2}\Big]drd\theta. \quad (2.11)$$

When N is large, we have asymptotic expressions [21] that

$$\Omega(\theta,N) \approx \frac{\pi^{(N-1)/2}\sin\theta^{N-1}}{\Gamma(\frac{N+1}{2})\cos\theta} \quad (2.12)$$

and

$$Q(A,N,\theta_{crit}) \approx \frac{1}{\sqrt{N\pi(1+G^2(A,\theta_{crit}))}\sin\theta_{crit}}\cdot \frac{G(A,\theta_{crit})\sin\theta_{crit}\exp\big(-\frac{A^2}{2}+\frac{1}{2}AG(A,\theta_{crit})\cos\theta_{crit}\big)}{\big(AG(A,\theta_{crit})\sin^2\theta_{crit}-\cos\theta_{crit}\big)}, \quad (2.13)$$

where $G(A,\theta) = \frac{1}{2}[A\cos\theta + \sqrt{A^2\cos^2(\theta)+4}]$. To compute the sphere packing bound, one must first find the critical value of θ that $\Omega(\theta_{crit}) = \frac{1}{M}\Omega(\pi) = \frac{N\pi^{N/2}}{M\Gamma(N/2+1)}$ from the exact expression (2.10) or asymptotic expression (2.12) of $\Omega(\theta,N)$, then substitute θ_{crit} into (2.11) or (2.13) to calculate the value of $Q(A,N,\theta_{crit})$.

Figure 2.2 shows the performance gap between the sphere packing bound and the ML decoding FER of codes. The (128,64,22) extended BCH code is the best code known for block length 128 and rate 1/2, in the sense of largest minimum Hamming distance ($d_{\min}$). The (128,64,10) tail-biting convolutional code has the largest $d_{\min}$ for convolutional codes with memory length = 6. It is impossible to reach the sphere packing bound unless the non-intersecting cones around codeword can fill the N space with equal size. Although such codes exist if and only if $N = 1$ or 2 when there is more than one codeword [21], it can still provide us an intuitive view of code performance in general.

2.5 Code Imperfectness

Since we aim to analyze the trade-off between code performance and decoding complexity, a proper metric for measurement needs to be developed. We adopt the measure of code imperfectness from [23]. We already know that for any practical codes, there is always a performance gap between the actual code performance and the sphere packing

bound. Since the curve of FER versus SNR for most codes and sphere packing bound has the exponentially decreasing shape, instead of showing the whole performance curve, we define code imperfectness of a given code, ΔSNR, as the difference between the lower bound of SNR implied by the Shannon's 1959 sphere packing bound under fixed error probability P_e, and the minimum SNR required by the code to attain the same P_e, under given block size N and code rate R.

The code imperfectness, ΔSNR, for ML performance of (128,64,22) extended BCH code and (128,64,10) tail-biting convolutional code at $P_e = 10^{-5}$ are shown in Figure 2.2. The ΔSNR for (128,64,22) extended BCH code is about 0.35 dB, while the ΔSNR for (128,64,10) tail-biting convolutional code is about 1.94 dB.

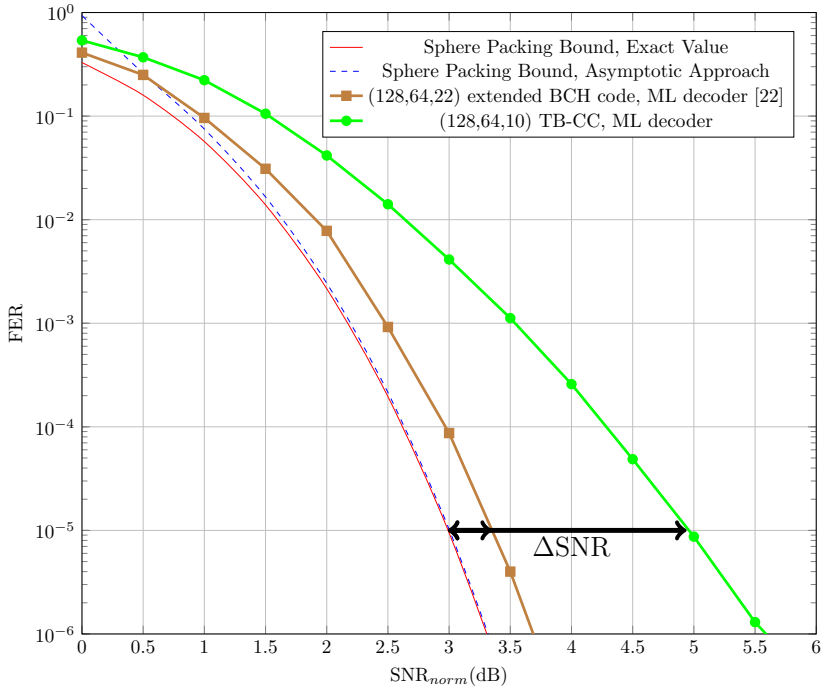

Figure 2.2: FER versus normalized SNR for sphere packing bound of (128,64) linear block code, maximum likelihood performance of (128,64,22) extended BCH code and (128,64,10) tail-biting convolutional code with memory length $M = 6$ and generator [564,634], transmitted by BPSK over AWGN channel.

Chapter 3

Tail-Biting Convolutional Codes

3.1 Introduction

A tail-biting convolutional code is a type of convolutional code that can achieve a better decoding efficiency with respect to conventional convolutional codes by replacing the fixed zero-tail with a tail-biting symbol constraint. A convolutional code is generated by a finite-state machine which has states that depend on the current and past inputs. The decoding process is more effective if both the starting state and ending state of the encoder are both known, however, this can results in a fractional rate loss. The rate loss can be neglected in the long block length regime, but is usually not acceptable with short block lengths. Tail-biting convolutional codes can overcome the rate loss caused by zero-padding, since it only ensures that the starting state of the encoder is identical to its ending state, and it is shown that the performance loss by removing the all-zeros state properties can be neglected under proper decoding algorithms [24].

3.2 The Construction of Tail-Biting Convolutional Codes

Convolutional codes are a type of error-correcting code with memory, where the codeword can be constructed by performing the convolution of an input symbol sequence with the impulse response of the encoder. The encoder of a convolutional code can be considered as a linear time-invariant (LTI) system, hence the structure of encoder can be represented as a rational function multiplication circuit. If we segment a codeword into multiple sets, then the current output set is not only determined by the corresponding

current input, but also determined by M_i previous inputs, where M_i here denotes the length of the memory registers of input u_i, and let M be the total number of memory registers.

For example, Fig. 3.1 shows a binary convolutional encoder with number of input bits $b_{in} = 1$ and number of output bits $b_{out} = 2$. This encoder consists of an $M = 2$ stage shift register with mod-$(q = 2)$ adders, and a parallel-to-serial multiplexer for serializing the output bits. Similarly, Fig. 3.2 shows a circuit example of rate 2/3 binary convolutional encoder with $M = 5$, $b_{in} = 2$ and $b_{out} = 3$.

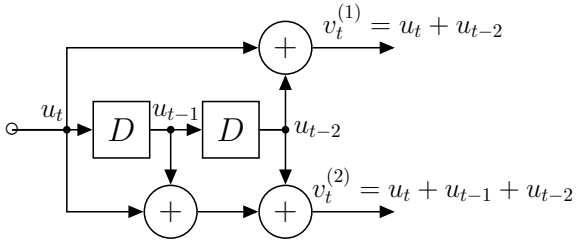

Figure 3.1: rate 1/2 Convolutional Encoder

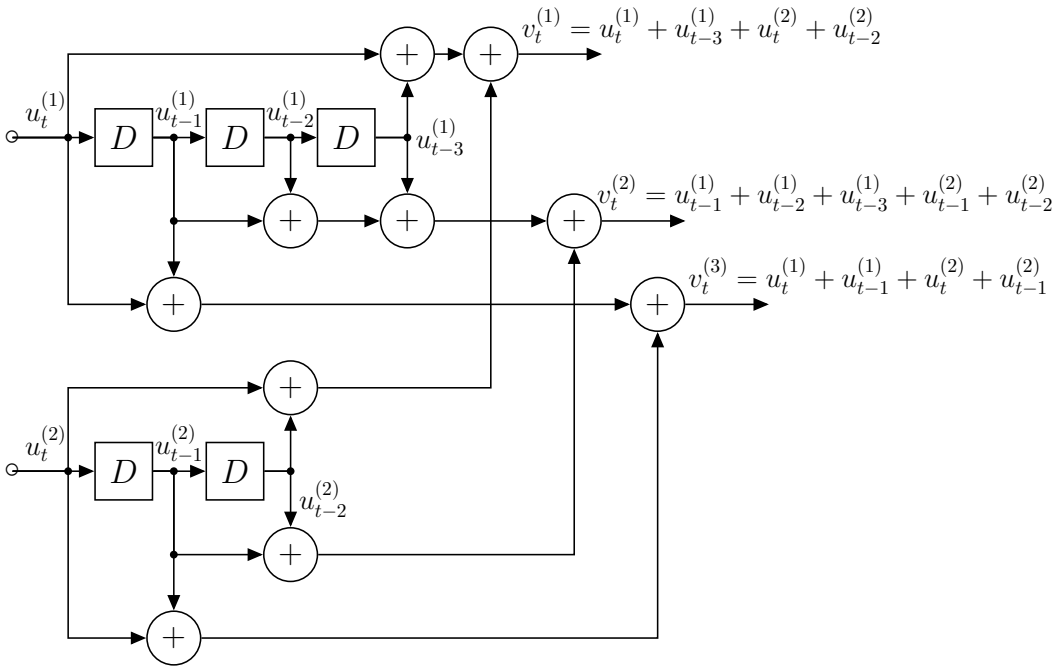

Figure 3.2: Rate $\frac{2}{3}$ Convolutional Encoder

Let $u_t^{(i)}$ be the ith input, $v_t^{(i)}$ be the ith output at time t. As shown on Fig. 3.1 and Fig. 3.2, the transfer function of the outputs of the example rate 1/2 encoder has the

form that

$$\begin{aligned} v_t^{(1)} &= u_t + u_{t-2}, \\ v_t^{(2)} &= u_t + u_{t-1} + u_{t-2}, \end{aligned} \tag{3.1}$$

and for the rate $\frac{2}{3}$ encoder example,

$$\begin{aligned} v_t^{(1)} &= u_t^{(1)} + u_{t-3}^{(1)} + u_t^{(2)} + u_{t-2}^{(2)}, \\ v_t^{(2)} &= u_{t-1}^{(1)} + u_{t-2}^{(1)} + u_{t-3}^{(1)} + u_{t-1}^{(2)} + u_{t-2}^{(2)}, \\ v_t^{(3)} &= u_t^{(1)} + u_{t-1}^{(1)} + u_t^{(2)} + u_{t-1}^{(2)}. \end{aligned} \tag{3.2}$$

Equations (3.1) and (3.2) show that each set of output bits can be represented as a function of current input and M previous inputs. Since the encoder of a convolutional code is a linear time-invariant system, the convolution operation can be replaced by polynomial multiplication, and the encoding equations can be replaced by a corresponding polynomial equations.

The field of Laurent series is defined as

$$F((D)) = \left\{ \sum_{i=r}^{\infty} x_i D^i, r \in \mathbb{Z}, x_i \in \mathbb{F}_q \right\}, \tag{3.3}$$

where x_i denotes the ith input. The Laurent series is the "D-transform" of the associated sequence and the letter D denotes the "delay operator" in such expressions [25], since if $v(D)$ is the D-transform of u, then the D-transform of Du is $Dv(D)$, which only adds same delay to both input and output in this LTI system. Therefore, the power of D denotes the number of time units that a bit is delayed with respect to the current input bit. Laurent series can be infinite, but contain only finite terms of D with negative powers. The delay of a Laurent series is the "time index" at the starting point of Laurent series. For example,

$$x(D) = D^{-4} + D^{-1} + 1 + D^3 + D^6 + D^{11} \ldots$$

is a Laurent series with delay of -4. If a Laurent series start has only non-negative powers of D while the constant term has non-zero coefficient, then this Laurent series has delay of 0 and is said to be "delay-free" (i.e., $x(D) = \sum_{i=0}^{\infty} x_i D^i$ is delay-free Laurent series if $x_0 \neq 0$).

The transfer function of convolutional encoder can be represented as delay-free Lau-

rent series by expressing the previous input in terms of "delay operator" D. Recall the rate 1/2 encoder example of Fig. 3.1, the transfer function can be rewritten as

$$\begin{aligned} v_t^{(1)} &= u_t(1+D^2), \\ v_t^{(2)} &= u_t(1+D+D^2), \end{aligned}$$

and for the rate $\frac{2}{3}$ encoder example of Fig. 3.2,

$$\begin{aligned} v_t^{(1)} &= u_t^{(1)}(1+D^3) + u_t^{(2)}(1+D^2), \\ v_t^{(2)} &= u_t^{(1)}(D+D^2+D^3) + u_t^{(2)}(D+D^2), \\ v_t^{(3)} &= u_t^{(1)}(1+D) + u_t^{(2)}(1+D). \end{aligned}$$

The encoding process of a binary linear convolutional code with rate $\frac{b_{in}}{b_{out}}$ is a linear mapping from the input vector $u(D) \in \mathbb{F}_2^{b_{in}}((D))$ to the output vector $v(D) = u(D)G(D) \in \mathbb{F}_2^{b_{out}}((D))$, which can be represented by a $b_{in} \times b_{out}$ transfer function matrix $G(D)$. To ensure the convolutional encoder is practical, the map needs to be injective. That is, every input vector $u(D)$ must map to a unique output vector $v(D)$. For this to happen, the $b_{in} \times b_{out}$ transfer function matrix $G(D)$ is required to have rank b_{in}. For the rate 1/2 example of 3.1,

$$G(D) = \begin{pmatrix} 1+D^2 & 1+D+D^2 \end{pmatrix},$$

and for the rate $\frac{2}{3}$ example of Fig. 3.2,

$$G(D) = \begin{pmatrix} 1+D^3 & D+D^2+D^3 & 1+D \\ 1+D^2 & D+D^2 & 1+D \end{pmatrix}.$$

.

The linear convolutional codes can be generated by a generator matrix multiplied by the information sequence. Assume an (n,k) code with input $1 \times k$ vector $\mathbf{u}$ denotes the information sequence and output $1 \times n$ vector $\mathbf{v}$ denotes the encoded sequence, then the generator matrix $\mathbf{G}$ of linear convolutional codes follows $\mathbf{v} = \mathbf{uG}$. The information sequence can be written as

$$\mathbf{u} = (u_1^{(1)}, u_1^{(2)}, \ldots, u_1^{(b_{in})}, u_2^{(1)}, u_2^{(2)}, \ldots, u_2^{(b_{in})}, u_3^{(1)}, u_3^{(2)}, \ldots\ldots),$$

where $u_l^{(1)}, u_l^{(2)}, \ldots, u_l^{(b_{in})}$ is the lth set of inputs to the encoder, and there are $N_{in} = \frac{k}{b_{in}}$ sets of input for the information sequence. Similarly,

$$\mathbf{v} = (v_1^{(1)}, v_1^{(2)}, \ldots, v_1^{(b_{out})}, v_2^{(1)}, v_2^{(2)}, \ldots, v_2^{(b_{out})}, v_3^{(1)}, v_3^{(2)}, \ldots\ldots),$$

where $v_l^{(1)}, v_l^{(2)}, \ldots, v_l^{(b_{out})}$ is the lth set of output. The generator matrix G maps from $\mathbf{u}$ to $\mathbf{v}$ with structure

$$\mathbf{G} = \begin{bmatrix} G_0 & G_1 & \ldots & G_{M-1} & G_M & & \\ & G_0 & G_1 & \ldots & G_{M-1} & G_M & \\ & & G_0 & G_1 & \ldots & G_{M-1} & G_M \\ & & & \ddots & \ddots & \ldots & \ddots & \ddots \end{bmatrix},$$

where

$$G_l = \begin{bmatrix} G_l^{(1,1)} & G_l^{(1,2)} & \ldots & G_l^{(1,b_{out})} \\ G_l^{(2,1)} & G_l^{(1,2)} & \ldots & G_l^{(2,b_{out})} \\ \vdots & \vdots & \ddots & \vdots \\ G_l^{(b_{in},1)} & G_l^{(1,2)} & \ldots & G_l^{(b_{in},b_{out})} \end{bmatrix}$$

is the $b_{in} \times b_{out}$ submatrix represents the impulse of delay l. For linear convolutional code, every element in G_l is equal to the coefficient of D^l term at the corresponding position of $G(D)$. For the rate 1/2 example of Fig. 3.1, we have

$$G_0 = \begin{bmatrix} 1 & 1 \end{bmatrix}, \; G_1 = \begin{bmatrix} 0 & 1 \end{bmatrix}, \; G_2 = \begin{bmatrix} 1 & 1 \end{bmatrix},$$

and for the rate $\frac{2}{3}$ example of Fig. 3.2,

$$G_0 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \; G_1 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \; G_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \; G_3 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Except for the start and tail, by viewing the generator matrix G horizontally, every set of inputs affects current set of output and M future sets of output through the map G_l; vertically, every set of output is affected by current input and M past inputs.

A convolutional code is called "tail-biting" if the initial state of memory is identical to the end state, and codes with such tail-biting constraint are called tail-biting convolutional codes (TB-CC). The generator matrix G of tail-biting convolutional codes has the form

$$\mathbf{G} = \begin{bmatrix}
G_0 & G_1 & \dots & G_{M-1} & G_M & & & & & \\
 & G_0 & G_1 & \dots & G_{M-1} & G_M & & & & \\
 & & G_0 & G_1 & \dots & G_{M-1} & G_M & & & \\
 & & & \ddots & \ddots & \dots & \ddots & \ddots & & \\
 & & & & G_0 & G_1 & \dots & G_{M-1} & G_M & \\
G_M & & & & & G_0 & G_1 & \dots & G_{M-1} & \\
G_{M-1} & G_M & & & & & G_0 & \dots & G_{M-2} & \\
\vdots & \vdots & \ddots & & & & \ddots & \vdots & \vdots & \\
G_2 & \dots & G_M & & & & & \ddots & G_1 & \\
G_1 & \dots & G_{M-1} & G_M & & & & & G_0 &
\end{bmatrix}.$$

For example, for the (12, 6) and (18, 12) binary linear tail-biting convolutional code with the encoder of Fig. 3.1 and Fig. 3.2, respectively, the generator matrices are

$$\mathbf{G} = \begin{bmatrix}
1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{bmatrix}$$

and

$$\mathbf{G} = \begin{bmatrix}
1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1
\end{bmatrix}.$$

If an information sequence $\mathbf{u} = (10, 11, 00, 10, 01, 11)$ is encoded by the encoder of the above (18, 12) tail-biting convolutional code, the corresponding codeword $\mathbf{v}$ is $\mathbf{v} = \mathbf{uG} = (101, 111, 100, 111, 000, 001)$.

For the encoding process of tail-biting convolutional code, the last M bits of the input sequence represent the starting state. Assume the input sequence $\mathbf{u} = (1, 1, 0, 1, 0, 1)$ is encoded by the encoder of the above (12,6) tail-biting convolutional code, then the corresponding output is $\mathbf{v} = \mathbf{uG} = (10, 01, 10, 00, 01, 00)$, where the starting state of the encoder is the last 2 bits of the input sequence, $0, 1$, which is identical to the ending state.

3.3 Convolutional Decoding

3.3.1 The Viterbi Decoding Algorithm

The encoding process of convolutional codes can be viewed as a random walk through 2^M states. Recall the encoder of convolutional code is a finite state machine, the encoding process can be represented by a state diagram.

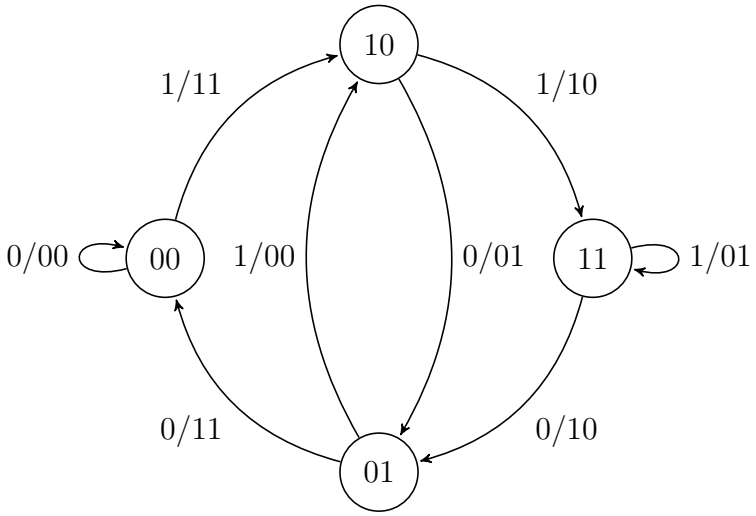

Figure 3.3: State diagram for rate 1/2 convolutional encoder

Fig. 3.3 is the state diagram of the example encoder shown in Fig. 3.1, where each circle represents a state, and each branch is labeled by input/output bits. The transition between states is controlled by an input sequence, and the serialized corresponding output is the codeword.

With known initial state, the codeword can be obtained by following the path determined by input sequence through the state diagram. For example, in Fig. 3.3, as-

suming that the encoder is initially in state 00 and the input information sequence is $\mathbf{u} = (1, 1, 1, 0, 1, 0, 0)$, the corresponding codeword $\mathbf{v} = (11, 10, 01, 10, 00, 01, 11)$. For finite convolutional codes, only the initial state or end state need to be known to ensure the path to be unique, but practically both initial and end state are expected to be known for the robustness of decoding.

A typical decoding algorithm for a convolutional code is the Viterbi algorithm, which ensures the end state to be the all-zero state by zero-padding the information sequence. This zero-padding process can result in rate loss of M bits, which can be unacceptable under short block length regime. This is because the length of zero-padding bits is fixed with fixed encoding structure, thus the fractional rate loss for long block length is smaller than short block length. Tail-biting convolutional codes avoid the rate loss by only forcing the end state to be identical to the initial state, but this can result in extra decoding complexity with respect to the normal Viterbi algorithm.

The state transition process can be represented by trellis structure described in [26] by expanding the state diagram of the encoder with respect to time units, which is convenient to understand Viterbi based decoding algorithm. For the trellis representation of a convolutional code, the number of states of each section are the same and the path between every two adjacent sections are identical. The random walk through the trellis structure from left to right represents the forward transition between states, where the full path of walking through one trellis period denotes the information sequence and corresponding codeword.

Assuming that the transmission rate is $R = b_{in}/b_{out}$, the number of information bits is k, then the length of code is $N = \frac{kb_{out}}{b_{in}}$. Given an information sequence with k bits, then for a binary tail-biting convolutional code there are $\frac{k}{b_{in}}$ stages, each with 2^M states, and there are 2_{in}^b branches leaving each state. The number of total paths is 2^k, where every codeword is associated with a unique path starting and ending in the same state.

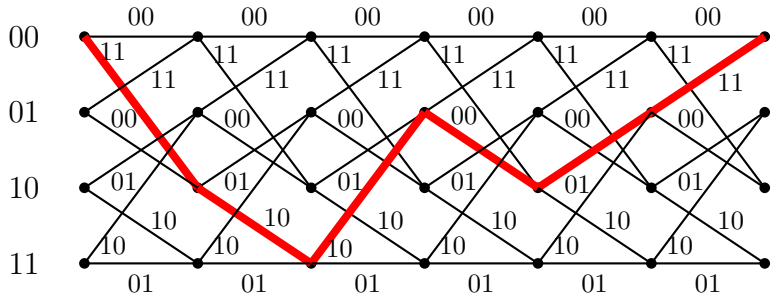

Figure 3.4: Trellis diagram of rate 1/2 convolutional code with encoded sequence $(11, 10, 10, 00, 01, 11)$

Consider the example trellis diagram shown in Fig. 3.4 for (12,6) convolutional code, where the bold line represents the encoding path. Here the initial state is 01, the input information sequence is $\mathbf{u} = (1, 1, 0, 1, 0, 0)$, and the codeword $\mathbf{v} = (11, 10, 10, 00, 01, 11)$.

Assume a sequence $\mathbf{r}$ is the output of a noise-free channel. To determine the information sequence, we can simply match up the received bits with the output labels set by set. At each node, we can simply follow the branch with label that matches next b_{out} received bits, and the corresponding input bits represents the information sequence.

When the channel is not noise-free, we need to define a metric that can compare the various paths and properly guess the correct path given a particular received sequence. Recall $N_{in} = \frac{k}{b_{in}}$ is the number of sets of input for the information sequence. Assume an information sequence

$$\mathbf{u} = (u_1^{(1)}, u_1^{(2)}, \ldots, u_1^{(b_{in})}, u_2^{(1)}, u_2^{(2)}, \ldots, u_2^{(b_{in})}, \ldots, u_{N_{in}}^{(1)}, u_{N_{in}}^{(2)}, \ldots, u_{N_{in}}^{(b_{in})})$$

with length k is encoded into codeword

$$\mathbf{v} = (v_1^{(1)}, v_1^{(2)}, \ldots, v_1^{(b_{out})}, v_2^{(1)}, v_2^{(2)}, \ldots, v_2^{(b_{out})}, \ldots, v_{N_{in}}^{(1)}, v_{N_{in}}^{(2)}, \ldots, v_{N_{in}}^{(b_{out})})$$

by convolutional encoder, and the sequence received through discrete memoryless channel is

$$\mathbf{r} = (r_1^{(1)}, r_1^{(2)}, \ldots, r_1^{(b_{out})}, r_2^{(1)}, r_2^{(2)}, \ldots, r_2^{(b_{out})}, \ldots, r_{N_{in}}^{(1)}, r_{N_{in}}^{(2)}, \ldots, r_{N_{in}}^{(b_{out})}).$$

A maximum likelihood (ML) decoder estimates $\mathbf{v}$ that maximizes the probability $P(\mathbf{r}|\mathbf{v})$, which follows that

$$\begin{aligned} P(\mathbf{r}|\mathbf{v}) &= \prod_{i=1}^{N_{in}} (P(r_i^{(1)}|v_i^{(1)})(P(r_i^{(2)}|v_i^{(2)}) \ldots (P(r_i^{(b_{out})}|v_i^{(b_{out})}) \\ &= \prod_{i=1}^{N_{in}} \big(\prod_{j=1}^{b_{out}} (P(r_i^{(j)}|v_i^{(j)})\big), \end{aligned} \tag{3.4}$$

where $P(r_i^{(j)}|v_i^{(j)})$ is the channel transition probability, and equation (3.4) is called the likelihood function. In hardware and software implementations, summations are usually faster than multiplications, we can obtain the following equation by taking the logarithm

of both sides:

$$\ln P(\mathbf{r}|\mathbf{v}) = \sum_{i=1}^{N_{in}} \Big(\sum_{j=1}^{b_{out}} \ln \left(P(r_i^{(j)}|v_i^{(j)}) \right) \Big), \tag{3.5}$$

which is the log likelihood function. We can define a term "cost" as a linear transformation of the log likelihood function, to make the value to be more convenient for implementation. This is also referred to "likelihood cost" or "likelihood distance", where

$$C(r_i^{(j)}|v_i^{(j)}) = a_1 \Big(- \ln \left(P(r_i^{(j)}|v_i^{(j)}) + a_2 \right) \tag{3.6}$$

is called bit cost, which is the cost of a jth bit of the branch in depth i. This is a linear transfer of log likelihood function, where a_1 and a_2 are constant numbers that are chosen to make the bit cost value positive and small. Similarly, branch cost $C(\mathbf{r}_i|\mathbf{v}_i)$ is the cost of an edge at depth i of the trellis, which can be expressed as

$$C(\mathbf{r}_i|\mathbf{v}_i) = \sum_{j=1}^{b_{out}} C(P(r_i^{(j)}|v_i^{(j)})). \tag{3.7}$$

The path cost is defined as the total cost of the whole codeword, which follows that

$$C(\mathbf{r}|\mathbf{v}) = \sum_{i=1}^{N_{in}} C(\mathbf{r}_i|\mathbf{v}_i) = \sum_{i=1}^{N_{in}} \sum_{j=1}^{b_{out}} C(P(r_i^{(j)}|v_i^{(j)})). \tag{3.8}$$

The accumulative cost of a path from depth 0 to depth d is called partial path cost, which can be written as

$$C_d(\mathbf{r}|\mathbf{v}) = \sum_{i=1}^{d} C(\mathbf{r}_i|\mathbf{v}_i) = \sum_{i=1}^{d} \sum_{j=1}^{b_{out}} C(P(r_i^{(j)}|v_i^{(j)})). \tag{3.9}$$

According to equation (A.8) in Appendix A.1.1 and A.1.2, the maximum likelihood estimation of the transmitted sequence satisfies

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v}\in\mathcal{C}} (\ln L(\mathbf{v}|\mathbf{r})) = \arg\min_{\mathbf{v}\in\mathcal{C}} (C(\mathbf{r}|\mathbf{v})), \tag{3.10}$$

where to maximize the channel transition probability of a sequence of received bits is equivalent to minimize its likelihood cost. As shown in (A.11), the likelihood cost for an i.i.d. sequence received through an AWGN channel to a codeword is the sum of the squared Euclidean distance between the received bit and the corresponding bit in the

codeword. Therefore, the bit cost, branch cost, path cost can be expressed, respectively, as follows:

$$C(r_i^{(j)}|v_i^{(j)}) = (r_i^{(j)} - v_i^{(j)})^2 \tag{3.11}$$

$$C(\mathbf{r}_i|\mathbf{v}_i) = \sum_{j=1}^{b_{out}} (r_i^{(j)} - v_i^{(j)})^2 \tag{3.12}$$

$$C(\mathbf{r}|\mathbf{v}) = \sum_{i=1}^{N_{in}} \sum_{j=1}^{b_{out}} (r_i^{(j)} - v_i^{(j)})^2; \tag{3.13}$$

and the partial path cost of a path from depth 0 to depth d has the form

$$C_d(\mathbf{r}|\mathbf{v}) = \sum_{i=1}^{d} \sum_{j=1}^{b_{out}} (r_i^{(j)} - v_i^{(j)})^2. \tag{3.14}$$

The cost of the path can be calculated by finding the squared Euclidean distance between the received sequence and a path on the trellis diagram. For the trellis diagram of Fig. 3.4, assuming the received sequence is $(0, 0, 1.5, 0, 0, 0.9, 0, 0, 0, 0, 1)$, then the squared Euclidean distance to codeword $(00, 00, 00, 00, 00, 00)$ is 4.06, the squared Euclidean distance to codeword $(00, 10, 01, 10, 00, 01)$ is 0.26. The procedure to calculate the cost of all valid paths separately and choose the path with minimum cost as the decoded sequence is equivalent to ML decoding with complexity $\mathcal{O}(N2^{NR})$, which can be too high for implementation. On the other hand, the decoding complexity for Viterbi algorithm is $\mathcal{O}(N2^M)$.

Let S denote the set of all states, $S(i, j)$ denote the jth state in depth i, and N_s denote the size of state space, where $S(i, j), j = 0, 1, \ldots, N_s - 1$ are all the states in depth i. Let $C(i, j)$ denote the assigned value of partial path cost for state $S(i, j)$. The steps of the Viterbi algorithm are shown in Algorithm 1.

Viterbi decoding obtains maximum likelihood performance. That is, the path decoded by Viterbi decoder is guaranteed to be a maximum likelihood path. Since the trellis structure has only one initial state and only one end state, when the ML path is not selected, it means there exist a partial path from depth i_a to depth i_b such that the decoded path is different from ML path, with the state in depth i_a and i_b identical to the ML paths. The cost of the decoded partial path is required to be lower than the cost of the ML partial path for the decoded path to survive, which contradicts the fact that ML path has the lowest cost among all possible paths.

Algorithm 1 Viterbi Algorithm

Initialize:
  Set $C_{i,j} \leftarrow +\infty$ for all $i = 0, 1, \ldots, N_{in}$ and $j = 0, 1, \ldots, N_s - 1$, then set $C_{0,0} \leftarrow 0$
for depth counter $i = 0$ to $N_{in} - 1$
  for state index $j = 0$ to $N_s - 1$
    for each branch leaving $S(i, j)$
      Find the state $S(i+1, j')$ that the branch enters
      Set $C_{new} \leftarrow C(i, j) + C_{branch}$
      if $C_{new} < C(i+1, j')$
        Set $C(i+1, j') \leftarrow C_{new}$
      end if
    end for
  end for
end for
Trace back from state $S(N_{in}, 0)$ to find the survivor path $\hat{\mathbf{v}}$
Return $\hat{\mathbf{v}}$

3.3.2 The Wrap-Around Viterbi Algorithm

Although the Viterbi Algorithm achieves maximum likelihood performance, it requires the starting state and the ending state of the encoder to be normally all-zeros state (or other known state), which can result in a fractional rate loss. The zero-padding of the encoding process of Viterbi algorithm adds M redundant bits to the codeword, which may result in unacceptable rate loss, especially in the short block length regime.

There is a decoding algorithm called Wrap-around Viterbi algorithm (WAVA) for tail-biting convolutional codes that can overcome the rate loss caused by zero-padding, since it only ensures the starting state of the encoder is identical to its ending state. WAVA is shown to be performance-wise sub-optimal with respect to the maximum likelihood decoder, while it can achieve near maximum likelihood performance [24].

Instead of tracing back from the known ending state, WAVA traces back from every state and find its path following the same update rule as the Viterbi Algorithm. As a result, a trace-back path is called tail-biting (TB) path if its starting state and ending state are identical. Otherwise, it is called a non tail-biting (NTB) path. Fig. 3.5 shows an example. Assume the bold paths are the trace-back paths of states, then the paths of C_{00}, C_{01}, C_{11} are tail-biting and the path of C_{10} is non tail-biting.

In a tail-biting convolutional code, a codeword is constrained to be a TB path. If the survivor is NTB, then we know there are errors. One option is to find the surviving path only among the set of TB paths, and output it as the decoded sequence, but this may result in a significant performance loss. Another option is to update the starting state

cost to be the corresponding end state cost, then repeat the Viterbi decoding process by another round over the trellis, which can be considered as one iteration.

Since TB paths have the same starting and ending state, the paths have the potential to converge to be tail-biting as the number of iterations increases. This is because after sufficient number of iterations, the paths are likely to be periodic, where only TB paths can have a period of 1 trellis length. On the other hand, an NTB path needs to average out the accumulated cost of longer incorrect path as the number of iteration increases, hence it is less possible to be the path with lowest cost. It is shown that WAVA is suboptimal with respect to ML, but setting a maximum number of iterations of $I_{\max} = 4$ is usually sufficient for the suboptimality to be negligible [24].

Recall the decoding complexity of VA is $\mathcal{O}(N2^M)$. WAVA has the same decoding complexity level as VA, except there is a constant factor $I_{\max}$, the number of iterations. To reduce the effect of this constant factor, a sufficient condition is derived for identifying the maximum likelihood tail-biting path (MLTBP), which can terminate the decoding process before reaching the maximum number of iterations.

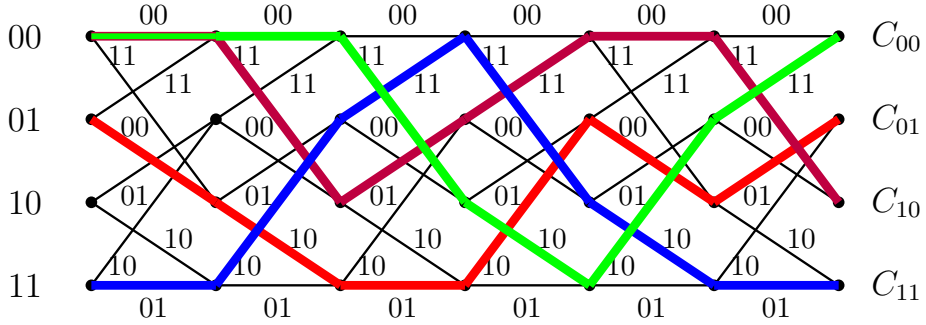

Figure 3.5: Trellis Diagram of $R = 1/2$ Convolutional Code with encoded sequence $(00, 10, 01, 10, 00, 01)$

Let S_{TB} and S_{NTB} denote the set of tail-biting paths and non tail-biting paths, respectively, and let $I_{init}(i, j)$ denote the index of the initial state at depth 0 of the survivor path to state $S(i, j)$. The steps of the Wrap-around Viterbi algorithm are described in Algorithm 2.

3.4 Performance and Complexity Trade-off with Short Block Length

The decoding complexity of WAVA is $\mathcal{O}(N2^M)$, which increases linearly with code length N, and increases exponentially with the memory order M, since the number of

Algorithm 2 Wrap-around Viterbi Algorithm

Initialize:
  Set $S_{TB} = \varnothing$, $S_{NTB} = \varnothing$
  Set $C(0, j) \leftarrow 0$ for all $j = 0, 1, \ldots, N_s - 1$
  Set $C(i, j) \leftarrow +\infty$ for all $i = 0, 1, \ldots, N_{in}$ and $j = 0, 1, \ldots, N_s - 1$
  Set $C_{worst}(j) \leftarrow 0$ for all $j = 0, 1, \ldots, N_s - 1$
  Set $n_{iter} \leftarrow 0$
while iteration counter $n_{iter} < I_{\max}$
  for depth counter $i = 0$ to $N_{in} - 1$
    for state index $j = 0$ to $N_s - 1$
      for Each branch leaving $S(i, j)$
        Find the state $S(i + 1, j')$ that the branch enters
        Set $C_{new} \leftarrow C(i, j) + C_{branch}$
        if $C_{new} < C(i + 1, j')$
          Set $C(i + 1, j') \leftarrow C_{new}$
          Set $I_{init}(i + 1, j') \leftarrow I_{init}(i, j)$
        end if
      end for
    end for
  end for
  for state index $j = 0$ to $N_s - 1$
    Compute $C_\Delta(j) = C(N_{in}, j) - C(0, j)$
    if $I_{init}(N_{in}, j) = j$ ▷ final state has same index as initial state
      Add $S(N_{in}, j)$ to S_{TB}
      Set $C_{worst}(j) \leftarrow \max(C_{worst}(j), C_\Delta(j))$
    else
      Add $S(N_{in}, j)$ to S_{NTB}
      Set $C_{worst}(j) \leftarrow C_\Delta(j)$
    end if
  end for
  Set $j^* = \arg\min(C_{worst}(j))$, where $S(N_{in}, j) \in S$
  if $S(N_{in}, j^*) \in S_{TB}$
    Trace back from state $S(N_{in}, j^*)$ to find the survivor path $\hat{\mathbf{v}}$
    Return $\hat{\mathbf{v}}$
  else
    Set $C(0, j) \leftarrow C(N_{in}, j)$ for all $j = 0, 1, \ldots, N_s - 1$
    Set $C(i, j) \leftarrow +\infty$ for all $i = 0, 1, \ldots, N_{in}$ and $j = 0, 1, \ldots, N_s - 1$
    $n_{iter} \leftarrow n_{iter} + 1$
  end if
end while
if $S_{TB} \neq \varnothing$
  Set $j^* = \arg\min(C_{worst}(j))$, where $S(N_{in}, j) \in S_{TB}$
  Trace back from state $S(N_{in}, j^*)$ to find the survivor path $\hat{\mathbf{v}}$
  Return $\hat{\mathbf{v}}$
else
  Decoding failed
end if

states increases exponentially with M. Therefore, as long as M is fixed to be small, the complexity of WAVA decoding for tail-biting convolutional only linearly increases with the code length N.

Although WAVA decoding achieves the near ML performance, the ML performance of a block code depends on its minimum Hamming distance, which is determined by M for convolutional codes. As shown in Fig. 3.6 and 3.7, a good tail-biting convolutional code with larger M also has better performance. Based on the construction of convolutional codes, we know $d_{\min}$ is upper bounded by $(M+1)/R$. When the code is long, the $d_{\min}$ of the code with best performance is expected to be large. For instance, for rate 1/2 linear block code with $N = 64$ and $K = 32$, the code with largest $d_{\min}$ is the (64, 32, 12) extended BCH code. On the other hand, for (256, 128) linear block code, the known optimal code is the (256, 128, 38) extended BCH code [27]. To achieve the same $d_{\min}$ with a convolutional code, M is required to be greater than 20, which makes the WAVA decoding hopelessly complex. Therefore, the performance loss of convolutional codes is more acceptable under short block length regime compared to long length.

Since the computational complexity of a tail-biting convolutional code with WAVA decoding algorithm is relatively stable, to use it for applications with strict latency constraints under varying SNR could be good, but for applications that have a high SNR environment, the stable complexity can be a disadvantage as lower complexity approaches may give the same performance.

3.5 Simulation Results

The WAVA decoding algorithm is applied to the rate 1/2 convolutional code with memory order from 2 to 8, where code length N is 128 and the maximum number of iterations is 4. Fig. 3.6 shows at FER $= 10^{-4}$, the code imperfectness between FER versus normalized SNR curve for $M = 2$ is about 4 dB, while for $M = 8$ the gap is about 1 dB. The dashed line on the plot represents the Polyanskiy-Poor-Verdú (PPV) approximation for AWGN channels [28]. From Fig. 3.7, we can see that the computational complexity increases exponentially with M, while for the same M the computational complexity does not vary much with changing SNR. From Fig. 3.6, as the value of M increases, the frame error rate also decreased, which shows the larger M results in higher reliability. Due to the trade-off between the complexity and reliability, M needs to be carefully selected in practical applications.

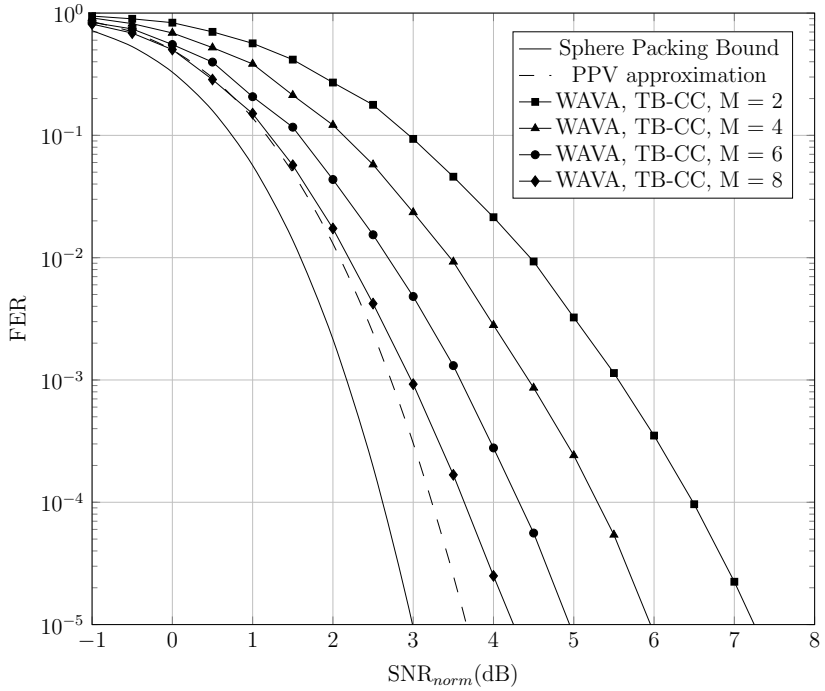

Figure 3.6: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/2$, $N = 128$, maximum number of iterations = 4.

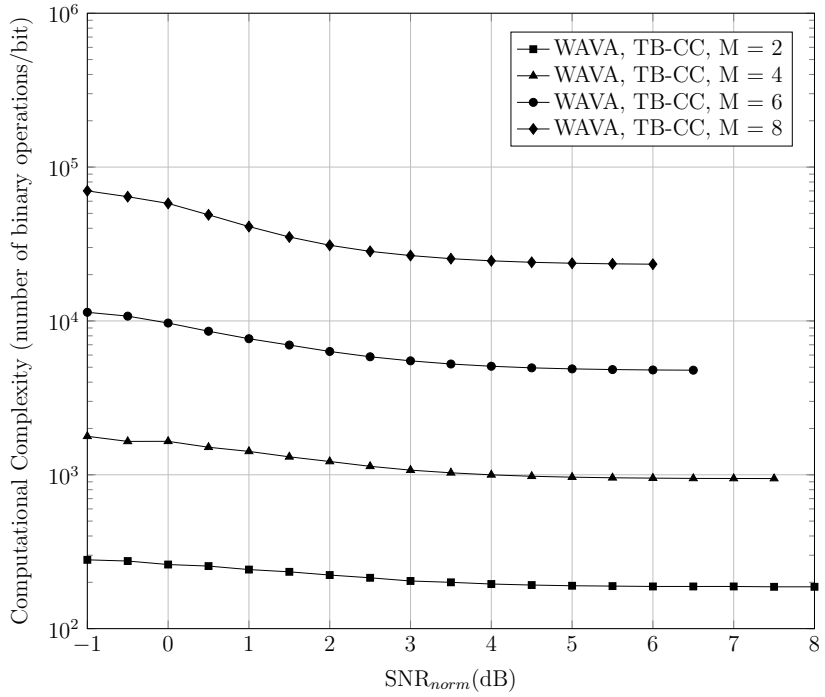

Figure 3.7: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/2$, $N = 128$, maximum number of iterations = 4.

Chapter 4

Polar Codes

4.1 Introduction

The concept of channel polarization was first introduced and further elaborated by Erdal Arıkan in 2008, then he developed a new channel coding technique called polar codes based on channel polarization in 2009 [29]. Channel polarization is a method proposed to construct capacity-achieving codes for any binary input discrete memoryless channels (B-DMC) with symmetric properties. It refers to the information lossless process that converts N independent copies of a given B-DMC channel to a mixture of binary-input extreme channels. For code with length N, polar codes have both encoding complexity and decoding complexity of $\mathcal{O}(N \log N)$, which makes it competitive in many applications with respect to other channel coding techniques.

4.2 Channel Polarization

The purpose of channel polarization process is to enhance the probability of correctly estimating a proportion of the transmitted bits over a discrete memoryless channel (DMC). Assume W is a binary input discrete memoryless channel (B-DMC) with input alphabet $\mathcal{U}$ and output alphabet $\mathcal{R}$, and $P(r|u)$ is the channel transition probability where $r \in \mathcal{R}$ and $u \in \mathcal{U}$. For symmetric B-DMC, the input alphabet $\mathcal{U}$ is always $\{0, 1\}$, while $\mathcal{R}$ can be arbitrary.

Polar transform focuses on the channel capacity, $I(W)$, of symmetric B-DMC given by

$$I(W) = \sum_{r \in \mathcal{R}} \sum_{u \in \mathcal{U}} \frac{1}{2} P(r|u) \log \frac{P(r|u)}{\frac{1}{2}P(r|0) + \frac{1}{2}P(r|1)}. \tag{4.1}$$

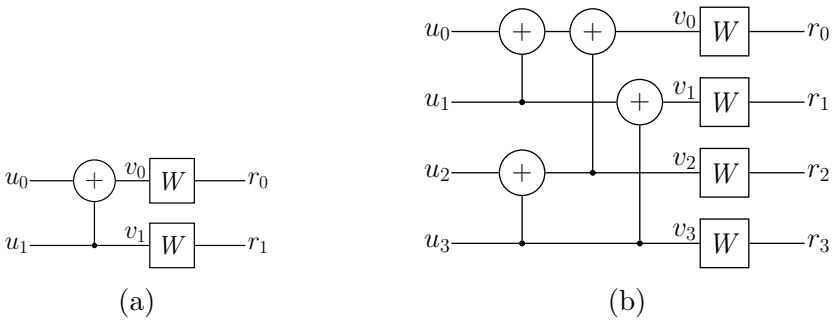

Figure 4.1: Channel polarization for $N = 2$ and $N = 4$

The so called Bhattacharyya parameter [29] has the form

$$Z(W) = \sum_{r \in \mathcal{R}} \sqrt{P(r|0)P(r|1)}, \tag{4.2}$$

where $P(r|u) = Pr\{R = r|U = u\}$. Both $I(W)$ and $Z(W)$ have values $\in [0, 1]$. When the transmission is over a B-DMC, they are related by the following 2 inequalities:

$$\begin{cases} I(W) \geq \log \dfrac{2}{1 + Z(W)} \\ I(W) \leq \sqrt{1 - Z(W)^2}. \end{cases} \tag{4.3}$$

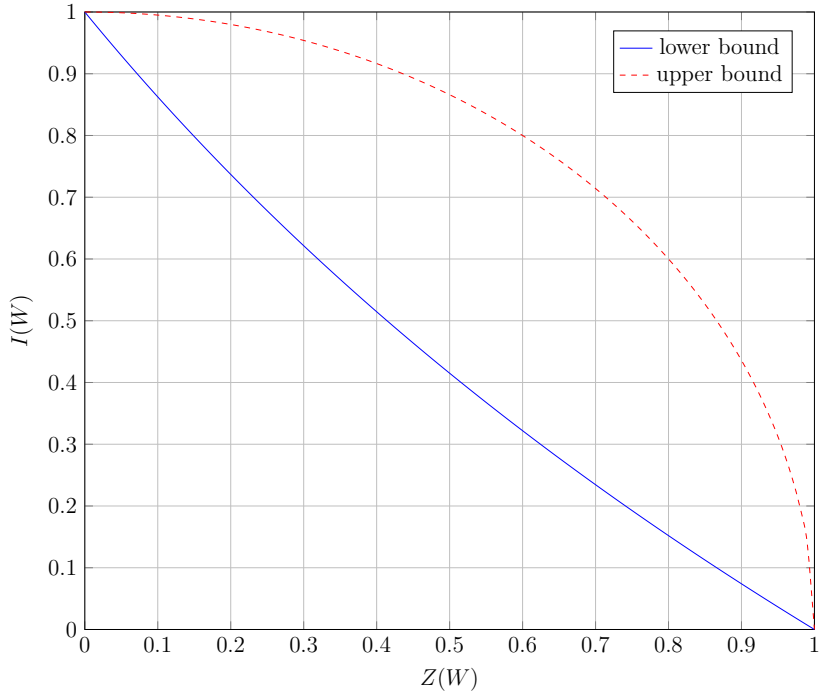

Figure 4.2: The upper and lower bounds of $I(W)$ as a function of $Z(W)$.

Fig. 4.2 illustrates the two bounds as a function of $Z(W)$. From (4.3) and Fig. 4.2 it can be observed that $I(W) \approx 1$ if and only if $Z(W) \approx 0$, which indicates a perfect channel, and $I(W) \approx 0$ if and only if $Z(W) \approx 1$, which indicates a completely noisy channel. For perfect channels, the output determines the input, which means error free transmission can be performed. For a completely noisy channel, the output is independent of the input, hence information transmission is not possible.

By channel polarization, most of the channels can be transformed to these two type of extreme channels with no information loss. First consider 2 input bits u_0 and u_1 are transmitted through two independent and identically distributed copies of the channel W, where r_0 and r_1 are the corresponding received symbols. Now consider the following transform for the inputs,

$$\begin{aligned} v_0 &= u_0 + u_1, \\ v_1 &= u_1, \end{aligned} \tag{4.4}$$

which refers to Fig. 4.1a. Given this transform, we can define two synthetic channels:

$$\begin{aligned} W^- &: U_0 \to (R_0, R_1), \\ W^+ &: U_1 \to (R_0, R_1, U_0). \end{aligned} \tag{4.5}$$

Since the input symbols U_0 and U_1 are assumed to be independent, we have

$$\begin{aligned} I(W^-) &= I(U_0; R_0, R_1), \\ I(W^+) &= I(U_1; R_0, R_1, U_0) = I(U_1; R_0, R_1 | U_0). \end{aligned} \tag{4.6}$$

Consider the independent use of the two copies of channel W, there is

$$I(W^-) + I(W^+) = I(U_0, U_1; R_0, R_1) = 2I(W), \tag{4.7}$$

which shows the preservation of symmetric capacity is preserved via channel polarization. The symmetry capacity of the synthetic channels follows inequality:

$$I(W^-) \leq I(W) \leq I(W^+). \tag{4.8}$$

The polar transform of 2 channels can be seen as providing an decrease in the reliability of the estimation of u_0, while that of u_1 increases. This can be viewed as a channel better than W and a channel worse than W. If we consider the two channel system as a whole and repeat the polarization process again, we will achieve 4 polarized channels, which is shown in Fig. 4.1b.

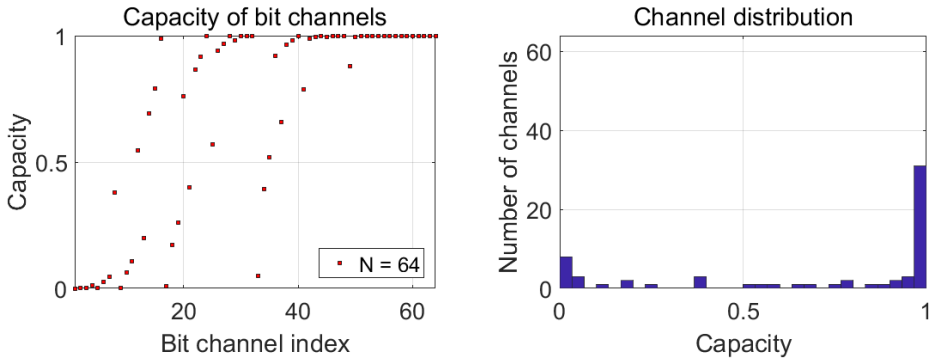

Figure 4.3: Polarization for BEC(0.3) with code length $N = 64$.

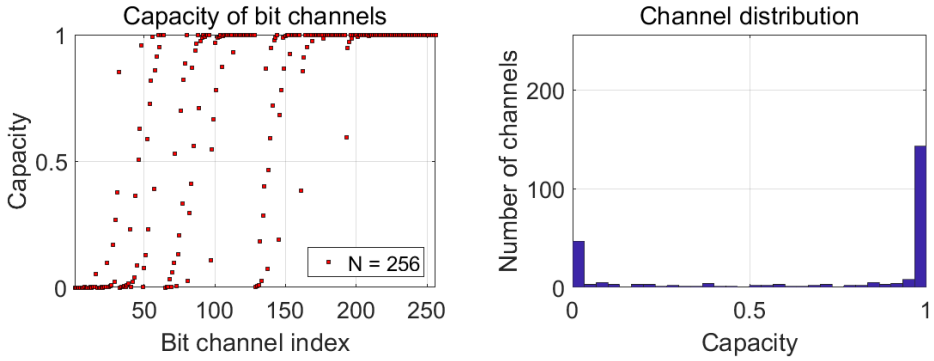

Figure 4.4: Polarization for BEC(0.3) with code length $N = 256$.

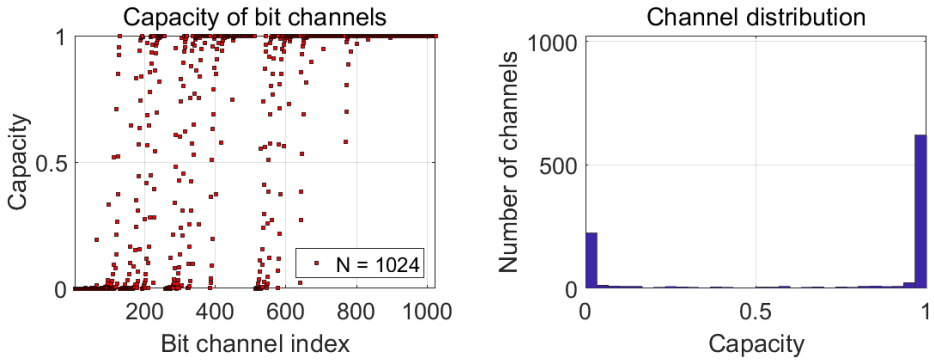

Figure 4.5: Polarization for BEC(0.3) with code length $N = 1024$.

We can repeat this process by n times to construct $N = 2^n$ channels. As N goes to infinity, the number of near perfect channels approaches CN, where C represents the channel capacity, while the number of near complete noisy channels approaches $(1-C)N$ and the number of mediocre channels goes to 0. The proof of this is presented in [29] and [30]. An example of polarization effect is illustrated in Fig. 4.3 $\sim$ 4.5 for W to be BEC with erasure probability $\epsilon = 0.3$.

The construction of polarized channels and polar codes are all based on such B-DMCs. Recall for binary discrete memoryless channel $W : U \to R$, the transition probability is $P(r|u), r \in \mathcal{R}, u \in \mathcal{U}$. As the defined in [29], through the polarization process of N identical copies, the new polarized channel $W : U \to R^N \times U^{i-1}$ is formed, with transition probability $P_N^{(i)}(r_1^N, u_1^{i-1}|u_i)$, where i is the index of channel.

For any $n \geq 0, N = 2^n, 1 \leq i \leq N/2$, for transition probability $P_N^{(i)}(r_1^N, u_1^{i-1}|u_i)$, there are recursive equations

$$
\begin{aligned}
P_N^{(2i-1)}(r_1^N, u_1^{2i-2}|u_{2i-1}) &= \sum_{u_{2i}} \frac{1}{2} P_{N/2}^{(i)}\left(r_1^{N/2}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|u_{2i-1} \oplus u_{2i}\right) P_{N/2}^{(i)}\left(r_{N/2+1}^N, u_{1,e}^{2i-2}|u_{2i}\right), \\
P_N^{(2i)}(r_1^N, u_1^{2i-1}|u_{2i}) &= \frac{1}{2} P_{N/2}^{(i)}\left(r_1^{N/2}, u_{1,o}^{2i-2} \oplus u_{1,e}^{2i-2}|u_{2i-1} \oplus u_{2i}\right) P_{N/2}^{(i)}\left(r_{N/2+1}^N, u_{1,e}^{2i-2}|u_{2i}\right).
\end{aligned}
\tag{4.9}
$$

4.3 The Construction of Polar Codes

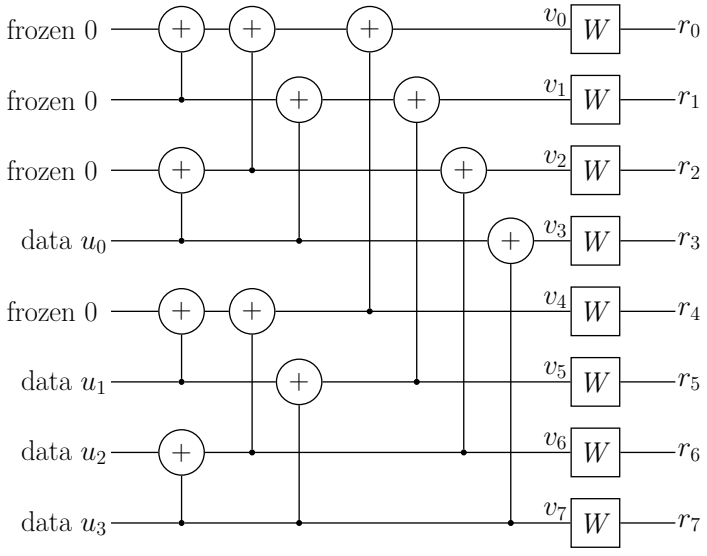

Figure 4.6: Channel polarization for $N = 8$ with frozen bits

To construct a polar code with length N and dimension k, where the code length N of polar code is a power of 2, the channel polarizing transformation is applied $\log_2 N$ times to construct N channels, then freeze the most unreliable $N-k$ channels and choose the remaining k channels to transmit information. For instance, in Fig. 4.5, we want to preserve the synthetic channels with capacity close to 1 to transmit information, and freeze the channels with capacity close to 0. Freezing a channel means the channel is not used to transmit informaiton, where the indices and default values of frozen bits are known by the receiver. Fig. 4.6 is an example of size 8 polarized channels with 4 frozen bits, which refers to an $(8, 4)$ polar code.

Recall that via the polar transform, most of the B-DMC channels can be transmitted to extreme channels with no information loss. The construction of polar codes starts with channel polarization. The two channel polarization in Fig. 4.1a can be represented in matrix form

$$F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}. \tag{4.10}$$

The transformation matrix of the code with length N, is constructed via the Kronecker product

$$F^{\otimes log_2 N+1} = F \otimes F^{\otimes log_2 N} \tag{4.11}$$

where $F^{\otimes 1} = F$, and $F^{\otimes log_2 N}$ is the generator matrix of length N polar code with frozen bits. Following the Kronecker structure, the complexity of this transformation process is $\mathcal{O}(N \log N)$, which is also the encoding complexity. The actual generator matrix can be achieved by deleting the rows corresponding to frozen bits. For example, the generator matrix with frozen bits for the polar code shown in Fig. 4.6 has the form

$$F^{\otimes 3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \tag{4.12}$$

while the actual generator matrix G with channel $0, 1, 2, 4$ frozen is

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \tag{4.13}$$

which is the $(8, 4, 4)$ extended Hamming code.

In matrix form, the encoding process can be represented as $\mathbf{v} = \mathbf{u}G$, where $\mathbf{u}$ of size k is the information sequence and $\mathbf{v}$ of size N is the codeword of $\mathbf{u}$. When polar codes was firstly proposed in [29], bit-reversal indexing was applied for the convenience of implementation, which can be achieved by multiplying the generator matrix by the bit-reversal permutation matrix. While bit-reversal indexing changes the bit order for encoding and decoding process, the error probability remains the same.

The selection of frozen bits can be achieved by numerous estimation methods, including Monte-Carlo simulation, estimation of bit-channel Bhattacharyya bounds [29], bit-channel transition probability estimation [31], Trifonov's Gaussian approximation [32], and genetic optimization [33]. Each of these algorithms is good enough for AWGN channels; in this thesis we select the Bhattacharyya bounds estimation method. Let z_i denote the Bhattacharyya parameter of channel i, the procedure of the Bhattacharyya bounds estimation is summarized in Algorithm 3.

Algorithm 3 Bhattacharyya Bounds Estimation

Initialize:
  Set $z_0 = \mathrm{SNR}_{norm}$
  $\log_2 N$
for $i = 0$ to $i = \log_2 N - 1$
  $u \leftarrow 2^j$
  for $t = 0$ to $t = u - 1$
    $T \leftarrow z_t$
    Find the upper channel $z_t \leftarrow 2T - T^2$
    Find the lower channel $z_{u+t} \leftarrow T^2$
  end for
end for
Return the best $N - k$ channels with greatest z value

The selection of frozen bits is based on the SNR assumed in design, which is called design SNR. This means the design of polar codes needs to be tailored to the channel. There exist a few attempts of universal polar codes design, but with higher decoding complexity [34] [35].

In this thesis, we assume the noise level is known, and set the design SNR equals to

the SNR of channel. Then the frozen bits are selected based on the design SNR, and the remaining unfrozen bits are used to transmit the information.

4.4 Decoding Algorithms for Polar Codes

4.4.1 Successive-Cancellation Decoding

Successive-cancellation (SC) decoding algorithm is a decoding scheme for polar codes that has been proved to achieve the symmetric channel capacity [29]. During the decoding process, from the transition probability indicated in equation (4.9), the transition probability of ith polarized channel $W_N^{(i)}$ includes received signal r_1^N and the inputs of the $i-1$ previous polarized channels, where

$$P_N^{(i)}(r_1^N, u_1^{i-1}|u_i) = \sum_{u_{i+1}^N \in \mathcal{U}^{N-i}} \frac{1}{2^{N-1}} P_N(r_1^N, u_1^N). \tag{4.14}$$

For $i \in 1, 2, \ldots, N$, SC decoding algorithm calculates the estimation of u_i by the transition probability $P_N^{(i)}$ when $\hat{u}_i = 0$ and $\hat{u}_i = 1$, with knowledge of the received bits r_1^N and the sequence of estimation, u_1^{i-1}. The estimation value can be obtained by

$$\hat{u}_i = \begin{cases} h_i(r_1^N, \hat{u}_1^{i-1}), & \text{if } i \in \mathcal{A} \\ u_1, & \text{if } i \in \mathcal{A}^{\mathcal{C}}, \end{cases} \tag{4.15}$$

where $i \in \mathcal{A}^{\mathcal{C}}$ indicates bit i is frozen bit that $u_i = \hat{u}_i$, and $i \in A^{\mathcal{C}}$ indicates bit i is information bit. The decision function of information bit i has the form

$$h_i(r_1^N, \hat{u}_1^{i-1}) = \begin{cases} 0, & \text{if } L_N^{(i)}(r_1^N, \hat{u}_1^{i-1}) \geq 0 \\ 1, & \text{if } L_N^{(i)}(r_1^N, \hat{u}_1^{i-1}) < 0, \end{cases} \tag{4.16}$$

where $L_N^{(i)}(r_1^N, \hat{u}_1^{i-1})$ is the log-likelihood ratio (LLR) defined as

$$L_N^{(i)}(r_1^N, \hat{u}_1^{i-1}) \triangleq \ln\left(\frac{P_N^{(i)}(r_1^N, \hat{u}_1^{i-1}|0)}{P_N^{(i)}(r_1^N, \hat{u}_1^{i-1}|1)}\right). \tag{4.17}$$

For any polar code with block length N, and rate R less than $I(W)$, the FER under SC decoder is bounded by $P_e(N, R) = o(2^{-\sqrt{N}+o(\sqrt{N})})$, where $I(W)$ is the symmetric capacity of any binary-input discrete memoryless channel W [36].

As the name implies, the SC decoding algorithm takes LLRs as the input, performs hard decisions on each bit, and sequentially estimate the bits from u_0 to u_{N-1}. The

channels approach complete polarization when the code length approaches infinity. For completely polarized channels, the channel capacity becomes either 0 or 1, and every information bit transmitted through a channel with capacity 1 is corrected decoded, which can theoretically make the polar code achieve the symmetric capacity $I(W)$ of the channel. On the other hand, the complexity of the SC decoder is only $O(N \log N)$. However, for a code with finite length, since the channel polarization is not complete, there may still exist some information bits that cannot be correctly decoded. Once an error occurs in the decoding of the previous $i-1$ information bits, since the SC decoder needs to use the estimated value of the previous information bits while decoding the following bits, this can lead to serious error propagation. Therefore, for a short block length code, the SC decoder is often unable to achieve ideal performance.

4.4.2 Successive-Cancellation List Decoding

This SC decoding process can be viewed as a greedy algorithm though a code tree. Fig. 4.7 gives an example of code tree representation of length 4 polar code. The number next to each node indicates the corresponding transition probability, where the path with the highest transition probability is selected at each node. In this example, the decoding sequence is $(1, 1, 0, 0)$.

A length N polar code is the concatenation of two length $N/2$ polar codes. Therefore, the structure of polar codes can be represented as a tree of depth $\log_2 N$. Since the move on code tree is unidirectional, once the optimal path is found for current depth, the decoder directly moves to next depth and there is no chance to correct any existing error. Although the number of mediocre channels approaches to 0 as code length N increases to infinity, for small N the number of mediocre channels is not negligible.

The successive cancellation list (SCL) decoding algorithm was developed as an improvement of the SC decoding algorithm, which increase the number of candidate paths after the path searching in each depth [37] [38]. Same as the SC algorithm, SCL decoding starts from the root node of the code tree and performs path search to the leaf nodes from depth 0 to depth N, but instead of reserving the path with best metric, the SCL decoder allows up to L best candidate paths to be preserved, and the path with best metric is selected from the list at the end, where $L \geq 1$ represents the list size. When $L = 1$, the SCL decoding is equivalent to SC decoding; when $L \geq 2^k$, it is equivalent to ML decoding.

Fig. 4.7 shows an example of SCL decoding of a length 4 polar code with list size $L = 2$. The two candidate sequences is $(1, 1, 0, 0)$ and $(0, 1, 1, 0)$, where $(0, 1, 1, 0)$ is

selected as the decoding sequence. This is an example where the SC decoder makes an error while the SCL decoder finds the correct path.

As described in [37], the code performance of SCL decoding converges to ML performance as SNR increases, where larger L can accelerate the convergence process with respect to lower L, and larger code length N refers to greater convergence speed.

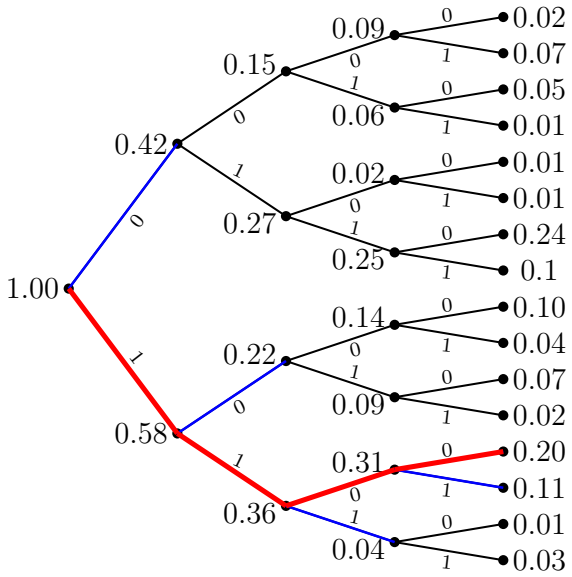

Figure 4.7: code tree for SC decoding of polar code with code length $N = 4$

In addition, cyclic redundancy check (CRC) can be proposed to improve the performance of SCL decoding of polar codes [39]. Cyclic redundancy check (CRC) is a channel error detection algorithm that is widely used in digital communication systems. For CRC-aided SCL (CA-SCL) decoding algorithm, the cyclic redundancy check bits are added to the information bit sequence before encoding. At the end the normal SCL decoding process, a list of candidate paths is obtained. The decoder can select the optimal sequence with the prior information "the correct information sequence can pass the CRC check" to output the decoding path, thereby enhancing the error correction capability of the SCL decoding algorithm with very low complexity, but with rate loss due to the redundant bits. Assume the code length of polar code is N, and the length of CRC is k_{CRC}. If the length of polarization information channels is k, then the length of the information bits is k_{info}, where $k = k_{\text{info}} + k_{\text{CRC}}$. In this case the code rate of polar code still remains $R = k/N$.

Recall that $\mathcal{A}$ and $\mathcal{A}^{\mathcal{C}}$ indicate the set of active channels and the set of frozen channels, respectively. Let $\mathcal{L}$ denote the set of candidate paths, and let $||$ represent the concatenation operator. As described in [38], the steps of CA-SCL decoding algorithm of polar

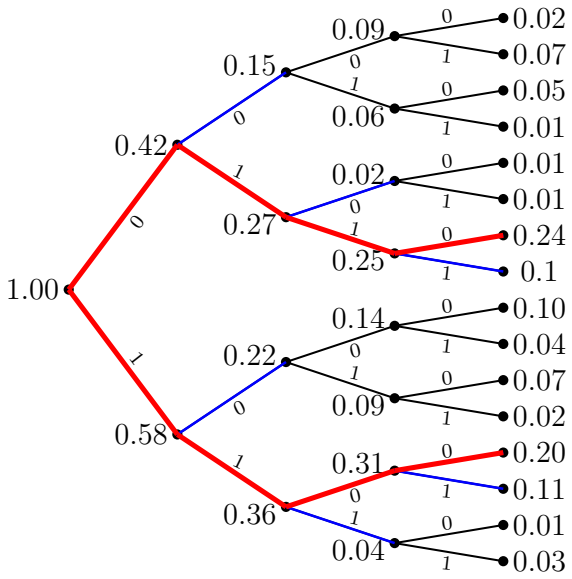

Figure 4.8: Code tree for SCL decoding of polar code with code length $N = 4$ and list size $L = 2$.

codes are shown in Algorithm 4.

4.5 Simulation Results

The simulation result of SC and SCL decoding of a $(128, 64)$ polar code is shown in Fig. 4.9, where for SCL decoder the list size is chosen to be from 2 to 16, with CRC of size 7. We can see at FER $= 10^{-4}$, the performance gap between sphere packing bound and SC decoding polar code is about 2.5 dB. SCL decoder with higher list size has better code performance, but from Fig. 4.10, the decoding complexity with greater L is also higher, which involves a trade-off between complexity and performance. For instance, for CA-SCL decoder with list size $L = 8$, the code imperfectness at FER $= 10^{-4}$ is about 1.8 dB, with around 1420 number of binary operations per bit.

In Fig. 4.9, the performance of polar code with SC decoder performs better than SCL decoders under low SNR. This is because there are 7 redundant bits for the polar code with SCL decoders, which results in performance deficiency. On the other hand, the performance of polar code with different SCL decoders converges under high SNR, this is because as SNR increases, the probability for the correct path to fall into the best L branches in each depth converges to 1, where the higher L refers to higher rate of convergence.

In contrast to the SC decoding algorithm, the SCL decoding algorithm enables us to

Algorithm 4 CRC-aided Successive Cancellation List decoding Algorithm

Initialize:
  Set $\mathcal{L} = \varnothing$
  Find the set of information bits A
for depth counter $i = 0$ to $N_{in} - 1$
  if $i \in \mathcal{A}^{\mathcal{C}}$
    $\hat{u}_i(\ell) \leftarrow \hat{u}_i$ for $\forall \ell \in \mathcal{L}$
  else
    if current list size $|\mathcal{L}| < L$
      for each $\ell \in \mathcal{L}$
        $\ell_{new} \leftarrow \ell || 0$
        $\ell \leftarrow \ell || 1$
        Add ℓ_{new} to Set $\mathcal{L}$
      end for
    else
      Compute $P_{\ell,u} = P_N^{(i)}(r_1^N, u_1^{i-1}(\ell)|u_i$ for $\forall \ell \in \mathcal{L}$ and $\forall u_i \in \mathbb{F}_2$
      $P_{\text{med}} \leftarrow$ the median of $2L$ numbers $P_{\ell,u}$
      for each $\ell \in \mathcal{L}$
        for each $u_i \in \mathbb{F}_2$
          if $P_{\ell,u} < P_{\text{med}}$
            Delete path ℓ from $\mathcal{L}$
          else
            $\ell \leftarrow \ell || u$
          end if
        end for
      end for
    end if
  end if
end for
while $|\mathcal{L}| > 0$
  $\ell^* \leftarrow \arg\max_{\ell \in \mathcal{L}} P_N^{(N)}(r_1^N, u_1^{N-1}(\ell)|u_N)$
  if ℓ^* passes the CRC test
    Return ℓ^*
  else
    Delete path ℓ^* from $\mathcal{L}$
  end if
end while
Decoding failed

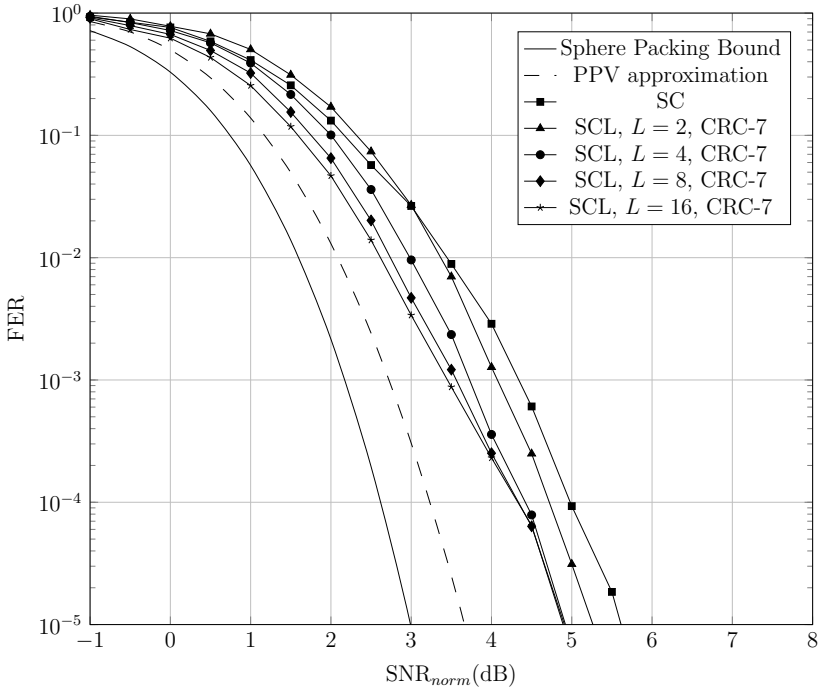

Figure 4.9: Frame error rate versus normalized SNR for Polar codes with SC and SCL decoder, $R = 1/4$, $N = 128$

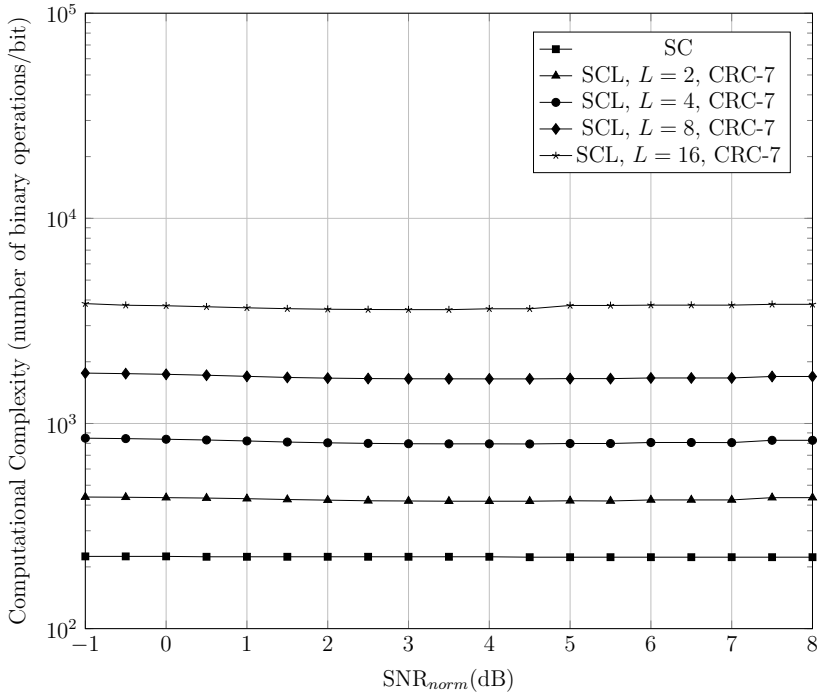

Figure 4.10: Computational complexity versus normalized SNR for Polar codes with SC and SCL decoder, $R = 1/4$, $N = 128$

implement better performance for short length polar codes with extra cost on computational complexity. The complexity of exploring L branches is $\mathcal{O}(LN \log N)$, while the complexity of finding L best paths in all the depths is $\mathcal{O}(LN \log 2L)$. Since L cannot be greater than $N/2$, the complexity of exploring the L branches always dominates, and thus the decoding complexity of SCL decoder is $\mathcal{O}(LN \log N)$. However, for short block length code, L is relatively big with respect to small code length N, hence the cost of finding L best paths in each depth cannot be neglected.

The linear block code sequential decoding and tail-biting convolutional code with WAVA decoding can decode the received block sequentially, hence the decoding process can be started immediately once the first bit of the sequence is received. On the other hand, for polar code, the decoding process cannot be started until the whole block is received. For short length codes the delay caused by waiting until the end of the block can usually be neglected, but for scenarios that require extremely low latency, this needs to be noticed.

Chapter 5

Sequential Decoding

5.1 Introduction

In contrast to the limitations of the decoding techniques based on the Viterbi algorithm, the concept of sequential decoding was firstly introduced as a suboptimal decoding technique for convolutional codes by Wozencraft in 1957 [40]. In 1963, Fano [41] developed the Fano algorithm, which saves a tremendous amount of computer memory, at the expense of an increase in the error probability and computational complexity. Thereafter, Zigangirov [42] and Jelinek [43] independently proposed the stack algorithm, which achieves lower error probability and complexity but with heavy use of memory, hence it is also called ZJ stack algorithm. A more recent innovation is the Creeper algorithm [44], which provides a trade-off between complexity, performance and memory with respect to the Fano algorithm and stack algorithms.

The popularity of sequential decoding algorithms declined after the development of the Viterbi algorithm for convolutional codes [45], but sequential decoding algorithms are still a competitive choice for many applications. There are several limitations of Viterbi based algorithms. Recall the complexity of Viterbi decoding algorithm for convolutional code and Wrap-around Viterbi algorithm for tail-biting convolutional code grows exponentially with the memory order of the encoder. The error probability is constrained by the minimum Hamming distance of the code, and minimum Hamming distance is constrained by the memory order. In the case that the error probability is required to be extremely low, memory order is forced to be long, and the computational complexity would be too high for the latency to be acceptable. On the other hand, Viterbi based algorithms have fixed computational complexity of $\mathcal{O}(N2^M)$, which is not always needed, especially under high SNR. Moreover, while the Wrap-around Viterbi algorithm can only be used on tail-biting convolutional codes, sequential decoding algorithms can be applied

to any linear block code.

5.2 Sequential Decoding Procedure

With respect to Viterbi-related algorithms, the encoding process and decoding process of sequential decoding can be viewed as walk through a tree structure instead of a trellis. The fundamental idea behind the sequential decoding is that we only explore the most probable branches, and temporarily discard all the improbable branches until they become relatively probable. The precise definition of "probable" is determined by the metric and algorithm of the sequential decoding approach.

Recall the advantages and disadvantages of different algorithms. Fano algorithm has very low memory requirement since the decoder always visit a single node in the tree at a time, which makes Fano algorithm more suitable for hardware implementations with strict memory constraint. However, as a price, since the decoding procedure of the Fano algorithm is based on the move from a certain node to its predecessor or to one of its immediate successors, it generally requires visiting more nodes than the stack algorithm, which results in higher computational time and complexity. On the other hand, to move from one certain node can results in higher error probability compared to the stack algorithm, which takes multiple nodes into consideration.

In this thesis, we consider the code length to be short, where the amount of computer memory size is assumed to be relatively sufficient. Therefore, we only focus on the stack algorithm in the following sections. For the metrics used in the stack algorithm, we will introduce the Fano metric and the variable bias-term metric.

5.2.1 Stack Algorithm

The stack algorithm of sequential decoding is based on priority-first search. In computer science the A* search algorithm [46] for path finding follows the same idea as sequential decoding in error-control coding [46]. The A* search algorithm selects the path that minimizes the cost $C = C_{\text{exp}} + C_{\text{unexp}}$, where C_{exp} is the cost of the explored path from the start node to current node, and C_{unexp} is the estimation of the cost of the unexplored path from current node to the end.

The stack algorithm saves nodes in a "stack" in memory, hence it requires sufficient memory size for implementations. Without constraints on computer memory and complexity, the stack algorithm of sequential decoding can achieve near ML performance with simple implementations, where performance depends on the metric for the calculation of

"priority". There are two commonly used metrics for stack algorithm: Fano metric and variable bias-term metric, which are discussed later.

Sequential decoding with stack algorithm decode linear block codes by guessing the paths through the code tree. Fig. 5.1 shows an example of code tree representation for $(2, 1, 2)$ convolutional code. Each branch of the tree diagram is labeled by input/output bits.

Similar to a trellis structure, the final path of the code tree determines the information sequence decoded, where the path can be searched by matching up the received bits with the output labels.

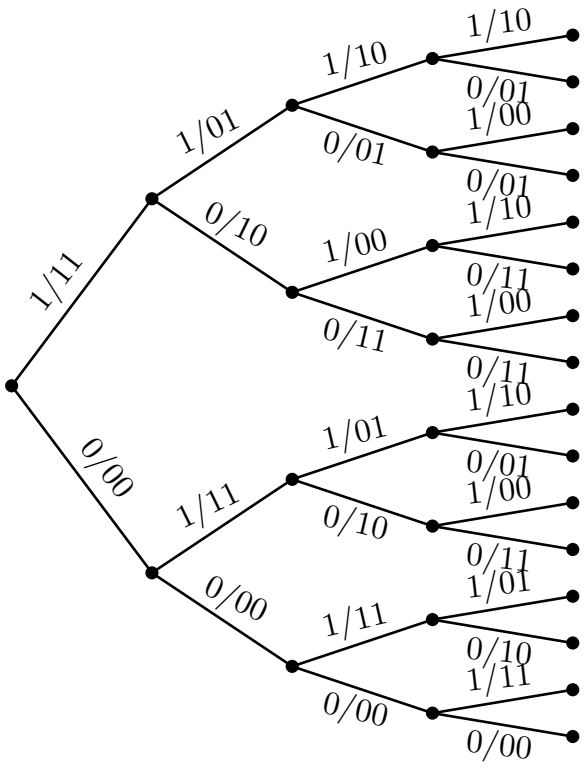

Figure 5.1: Tree diagram of (2,1,2) convolutional code

Recall that sequential decoding with stack algorithm is based on priority first search, where the explored nodes are saved in a priority queue. A priority queue always has the element with highest priority served first, hence the priority queue can be implemented with heap structure, where the term "priority" is determined by the cost based on particular metric.

Due to the data structure of a priority queue, sequential decoding with the stack algorithm is memory dependent. As stack overflow occurs, some of the nodes saved in the stack must be erased, which could possibly include the correct path. Stack overflow

is not a such dramatic issue if the number of paths thrown out is small. Once the stack capacity is exceeded, one of the bottom nodes of the stack can be thrown out with a slight performance reduction since the bottom nodes usually have highest path cost. However, throwing out most of the paths can still result in a significant performance reduction.

Let S denote the set of all states and S_{exp} denote the set of all states and the stack of explored states, where S_{exp} is a priority queue. Let $S(i, P)$ denote the state with partial path P_i in depth i, where the path P of state $S(i, P)$ is the truncated codeword sequence from depth 0 to depth i. Let $C(i, \ell)$ denote the assigned value of partial path cost for state $S(i, \ell)$. Let $||$ represent the concatenation operator. The procedure of the stack algorithm is summarized in Algorithm 5.

Algorithm 5 The Stack algorithm of Sequential Decoding

Initialize:
    Set $S_{\text{exp}} = \varnothing$
    Add $S(0, \ell)$ to S_{exp}, where ℓ is empty, $C(i, \ell) = 0$
while iteration counter $n_{iter} < I_{\max}$
    Find state $S(i, \ell)$ in stack S_{exp} with minimum $C(i, \ell)$
    if $i >= N_{in}$
        Return $\hat{\mathbf{v}}$
    else
        if $|S(j, \ell)| > L_{\text{stack,max}}$
            Delete $S(j, \ell)$ with one of the highest $C(j, \ell)$ from stack S_{exp}, $S(j, \ell) \in S_{\text{exp}}$
        end if
        Delete $S(i, \ell)$ from S_{exp}
        for Each branch leaving $S(i, \ell)$
            Compute C_{branch} depends on the metric
            Set $\ell_{new} \leftarrow \ell || u_i$
            Set $C(i + 1, \ell_{new}) \leftarrow C(i, \ell) + C_{branch}$
            Add $S(i + 1, \ell_{new})$ to stack S_{exp}
        end for
    end if
end while
Find state $S(i, \ell)$ in stack S_{exp} with minimum $C(i, \ell)$
if $i >= N_{in}$
    Return $\hat{\mathbf{v}}$
else
    Decoding failed
end if

5.2.2 Fano Metric

A probabilistic branch metric called the Fano metric has become a typical path metric for sequential decoding algorithms since it was proposed in 1963 [41]. For binary input discrete memoryless channel (B-DMC), the bit Fano metric has the form

$$M(r_i|v_i) = \log_2\left(\frac{P(r_i|v_i)}{P(r_i)}\right) - R, \tag{5.1}$$

where r_i is the received symbol, v_i is the transmitted symbol, $P(r_i|v_i)$ is the channel transition probability, $P(r_i)$ is the probability of the received bit, and B is a constant that represents bias term. The value of $M(r_i|v_i)$ is maximized by ML sequential decoder, hence for convenience we can define a term "cost" as the metric, where $C(r_i|v_i) = -M(r_i|v_i)$.

Consider BPSK modulation with information symbol $v \in \{+\mu, -\mu\}$ with additive white Gaussian noise (AWGN) $\mathcal{N}(0, \sigma^2)$, the Fano metric can be derived as

$$M(r_i|v_i) = 1 - R - \log_2\left(1 + \exp\left(-\frac{2r_iv_i}{\sigma^2}\right)\right), \tag{5.2}$$

where the calculation procedure is described in Appendix B.1.

The statistic to be maximized by the optimum decoder depends on the unexplored part of the code tree, where this dependency is removed by averaging over all possible random tails [47]. Fano metric ensures good overall performance over a large code ensemble by the assumption of code randomness, but it does not ensure superior performance of any individual cases.

5.2.3 Variable Bias-term Metric

The variable-bias term (VBT) branch metric is introduced in [48], which is based on the concept of Viterbi metric. Recall for Viterbi metric, the cost C of a path is the partial path cost of a branch from depth 0 to depth d, which can be expressed as

$$C_d(\mathbf{r}|\mathbf{v}) = \sum_{i=1}^{d}(r_i - v_i)^2. \tag{5.3}$$

This is the cost of the explored part of paths. Unlike Viterbi algorithm, stack algorithm for sequential decoding requires an approach that allows comparisons between paths with different depth. Therefore, the estimation of the unexplored part of branches is required. In [48], the variable-bias term branch cost is defined as

$$C(d) = C_{\text{exp}}(d) + C_{\text{unexp}}(d), \tag{5.4}$$

where $C_{\text{exp}}(d)$ can be calculated from (5.3), and $C_{\text{unexp}}(d)$ is the accumulated squared Euclidean distance of the hard decision that

$$C_{\text{unexp}}(d) = \sum_{i=d+1}^{N} (\min\{\| r_i - (-1) \|^2, \| r_i - 1 \|^2\}), \tag{5.5}$$

where the accumulated bias term cost can be considered as a lower bound for the corresponding accumulated cost of the ML codeword. Since both the cost of explored part and the cost of unexplored part are lower bounded by ML cost, the sequential decoder with the stack algorithm and VBT metric is promised to be ML, as long as the size of stack is not constrained.

Let C_{bias} denote the accumulated bias term cost of the whole codeword. Then $C_{\text{bias}} = C_{\text{unexp}}(0) = \sum_{i=1}^{N}(\min\{\| r_i - (-1) \|^2, \| r_i - 1 \|^2\})$, which is a constant for given received sequence $\mathbf{r}$. By subtracting C_{bias} from $C(d)$, the cost of variable-bias term branch metric is equivalent to

$$C(d) = C_{\text{exp}}(d) - \sum_{i=1}^{d}(\min\{\| r_i - (-1) \|^2, \| r_i - 1 \|^2\}), \tag{5.6}$$

which refers to the difference between the accumulated distance of the path and the accumulated distance of the hard decision branches. The calculation of VBT metric is particularly simple, but for some short codes such as $(24, 12)$ Golay code and $(48, 24)$ QR code, VBT may require to explore fewer nodes than Fano metric [48].

Sequential decoding with ordinary stack algorithm can be considered a first-passage process, where every node with accumulated cost less than the total cost of the decoded path is expanded during decoding. Fig. 5.2 shows an example of all the explored paths of a sequential decoder with variable bias term metric for a $(128, 64)$ tail-biting convolutional code. The total cost of the decoded path is about 15.1, and every node with cost below this value is explored. The magnitude of the expected cost $\overline{c}$ of a single bit has the form

$$\overline{c} = \frac{2\sqrt{2}\mu\sigma}{\sqrt{\pi}} e^{-\frac{\mu^2}{2\sigma^2}} - 2\mu^2 \text{erfc}\left(\frac{\mu}{\sqrt{2}\sigma}\right), \tag{5.7}$$

and for the expected accumulated cost $\overline{c}_N$ there is

$$\overline{c}_N = N\overline{c}, \tag{5.8}$$

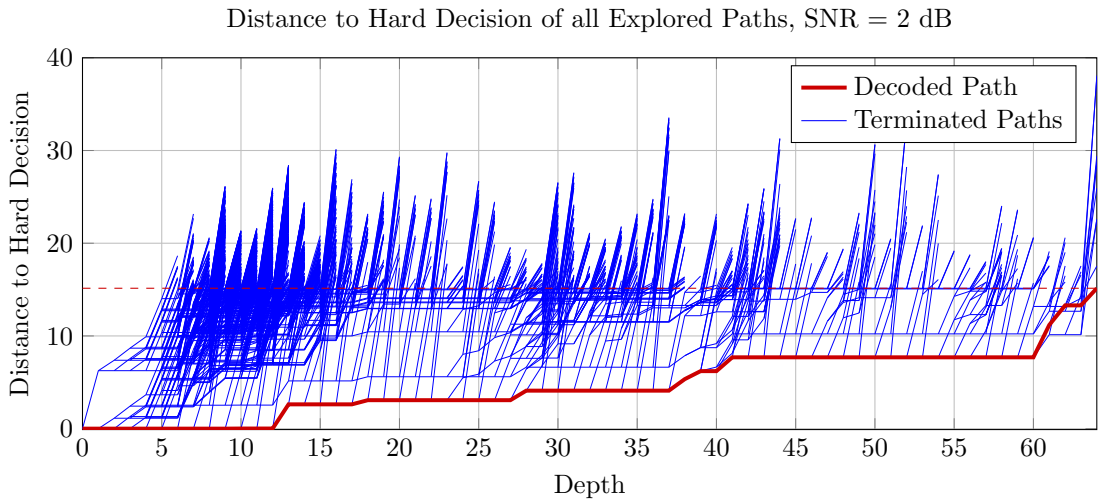

Figure 5.2: The explored paths of $(128, 64)$ TB-CC ($M = 3$ $[54, 74]$) with sequential decoding. The decoder uses stack algorithm with variable bias term metric, SNR = 2 dB.

where the calculating process is shown in Appendix B.2. For particular noise level, $\overline{c}_N$ is proportional to code length N, hence the number of explored branches rises as an power function of N. This fact indicates that the ordinary stack algorithm is more suitable for short block length, rather than long block length.

5.2.4 Improved Variable Bias-term Metric

According to Fig. 5.2, a tremendous number of nodes are explored, where some of them are not necessary to be visited. For instance, assume the decoded path of a $(128, 64)$ linear block code has a total cost of 10, and there exists a path with accumulated cost 9.5 in depth 1, then this path will still be expanded during the decoding process, although it is very unlikely to be the correct path.

In order to handle this problem, we propose an improved scheme of VBT metric, where an extra bias term is added to the cost expression. Recall the cost of VBT metric is the difference between the accumulated distance of the path and the accumulated distance of the hard decision branches, thus the cost is monotonically increasing. Therefore, when there are two paths with same cost and different depth, the path with higher depth is more likely to be the correct path than the path with lower depth.

To eliminate the influence of depth, we can add a extra bias term to the cost. The purpose of the extra bias term is to make the paths with greater depth to have higher chance to be considered first. One appropriate choice of the extra bias term is the expected cost, $\overline{c}$. The VBT cost with extra bias term $-\overline{c}$ per bit has expression

$$C(d) = C_{\text{exp}}(d) - \sum_{i=1}^{d} (\min\{\| r_i - (-1) \|^2, \| r_i - 1 \|^2\}) - d\bar{c}. \tag{5.9}$$

By the law of large numbers, the accumulated cost of the correct path converges to $N\bar{c}$ of as $N \to \infty$, hence this metric can eliminate the impact of depth for sufficiently large N.

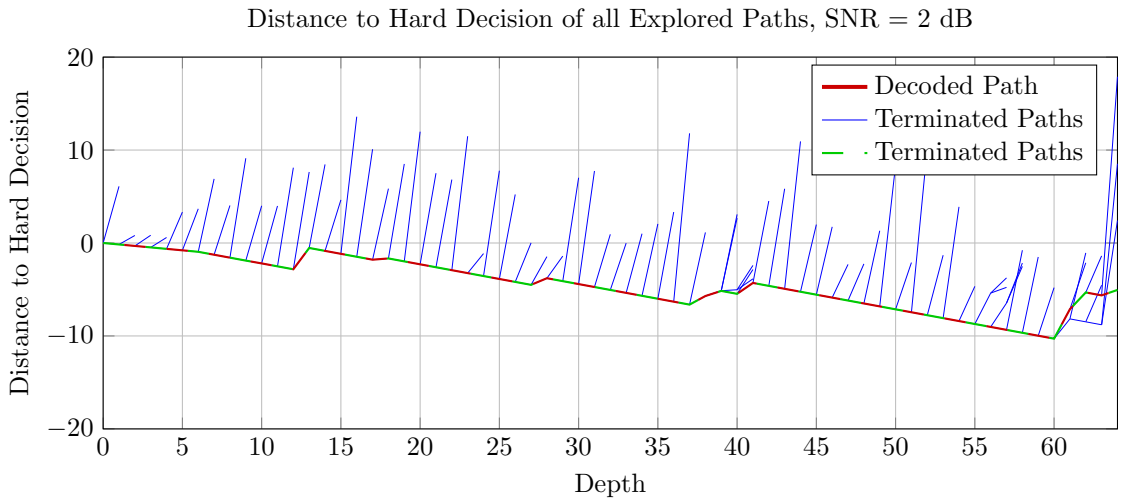

Figure 5.3: The explored paths of $(128, 64)$ TB-CC ($M = 3$ [54, 74]) with sequential decoding. The decoder uses stack algorithm with improved variable bias term metric, SNR = 2 dB.

Fig. 5.3 shows the paths explored with improved VBT metric, with codeword and noise identical to the example of Fig. 5.2, which indicates that the improved VBT metric can significantly reduce the redundant explored nodes. Instead of monotonically increasing cost with ordinary VBT metric, the cost of improved VBT metric decreases as depth increases, hence the paths with greater depth are more likely to be expanded. As the decoding process ends at depth $d = 128$, the final cost is already less than the initial cost with depth $d < 30$. With high probability, only a small range of depth is active as depth increases, thus the average number of nodes visited per bit is much less than the ordinary VBT metric, where an example is shown in Fig. 5.4.

Throwing out such low depth paths will cause a loss of the guarantee of ML performance, but the performance reduction is relatively slight. Fig. 5.5 and 5.6 compare the performance and complexity of TB-CC code decoded by sequential decoder with stack algorithm and different metrics. It shows that for stack algorithm sequential decoding, Fano metric and VBT metric can result in similar code performance and decoding complexity for linear block codes. On the other hand, the improved VBT metric can have

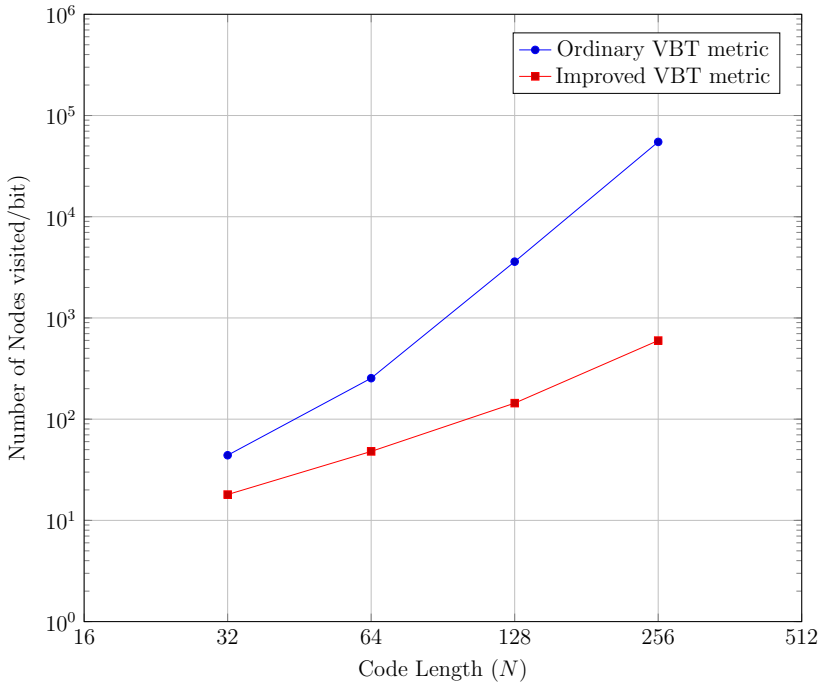

Figure 5.4: Number of nodes visited per bit for stack algorithm sequential decoding on $(128, 64)$ TB-CC ($M = 6$ $[564, 634]$) code, with VBT metric and improved VBT metric.

a significant complexity reduction compared to VBT metric, especially under low SNR. Sequential decoding with such improved VBT metric can be applied on low noise level cases with reasonable complexity, which allows sequential decoder to be used for the scenario that the channel noise that cannot be accurately estimated.

5.3 Applicability of Sequential Decoding

While Viterbi-based algorithms exhaust the whole trellis to locate the best codeword, sequential decoders only concentrate on a certain number of the most promising codewords. Since the computational complexity is directly determined by the number of branches explored in the code tree, for sequential decoding to be an appropriate choice of decoding algorithm, we want the number of nodes explored to be small.

Under particular noise level, for code rate R below a certain threshold R_{comp}, which is known as the computational cutoff rate, the number of explored nodes per bit is upper bounded. The computational cutoff rate R_{comp} for a binary-input, unquantized output channel is [25]

$$R_{comp} = 1 - \log\left(1 + \int_{-\infty}^{\infty} \sqrt{p(r|0)p(r|1)}dx\right). \tag{5.10}$$

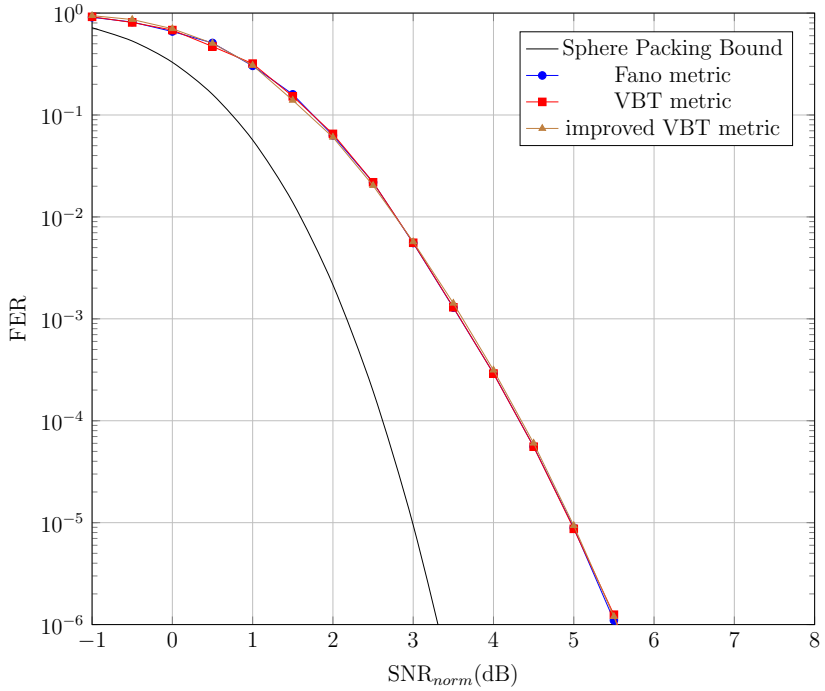

Figure 5.5: Code performance for stack algorithm sequential decoding with different metric on TB-CC codes, $N = 128$, $M = 6$ $[564, 634]$

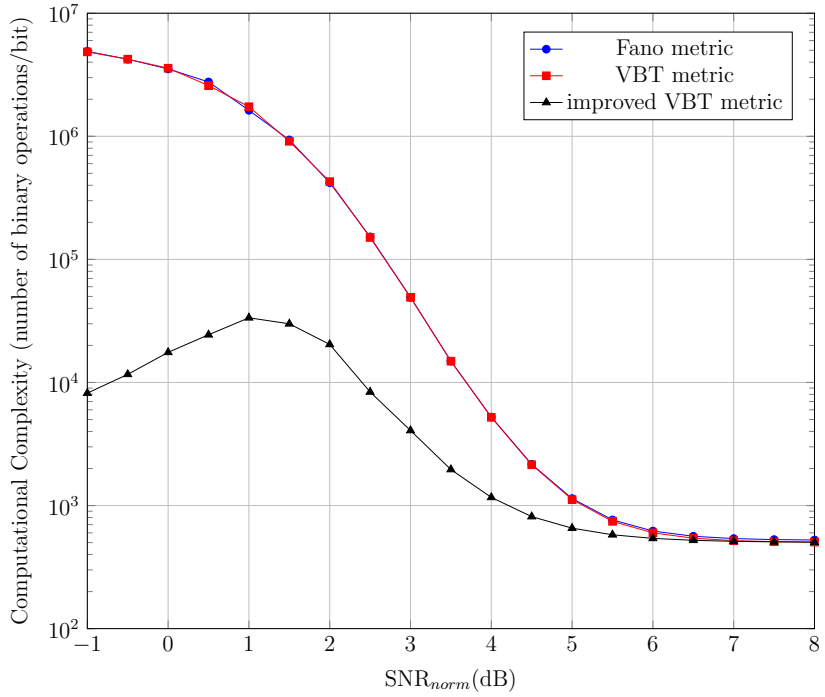

Figure 5.6: Computational complexity for stack algorithm sequential decoding with different metric on TB-CC codes, $N = 128$, $M = 6$ $[564, 634]$

Therefore, for the binary symmetric channel with cross over probability ϵ, the cutoff rate is given by [49]

$$R_{comp} = 1 - \log(1 + 2\sqrt{\epsilon(1-\epsilon)}), \tag{5.11}$$

and for the AWGN channel with BPSK modulation, the computational cutoff rate is [25]

$$R_{comp} = 1 - \log(1 + e^{-E_s/2\sigma^2}). \tag{5.12}$$

Conversely, under particular code rate R, the applicability of sequential decoding is affected by the channel noise. For example, for $R = 1/2$ code, the minimum SNR level for the number of explored nodes per bit is upper bounded is $\text{SNR}_{norm} = 0.8655$. Lower noise level allows the decoder to locate the best codeword more easily without expanding an excess of incorrect branches. The best-case complexity refers to the case that only the correct path is expanded, which has complexity of $\mathcal{O}(N)$. On the other hand, the worst-case complexity occurs when the whole tree structure is expanded, with complexity level $\mathcal{O}(N2^{NR})$, which is equivalent to exhaustive search ML decoding. When the SNR is extremely high, the received sequence can match up the target codeword easily, hence almost only the correct path is explored. When the SNR is extremely low, the information is considered to be nearly completely corrupted, and a significant number of branches are required to be visited before a word decoded.

Fig. 5.7 represents an example of all the explored paths of a sequential decoder with variable bias term metric for the $(128, 64)$ binary linear code with SNR = 2, 4 and 6, respectively. For uRLLC scenario, the requirement of ultra-high reliability forces the SNR to be relatively high, which plays a positive role in reducing complexity of sequential decoding compared to decoding algorithms with fixed complexity.

As the code approaches the PPV approximation, the code imperfectness at a certain FER is smaller, which refers to the code with low SNR. As mentioned above, a sequential decoder can achieve much lower decoding complexity under high SNR rather than low SNR, hence there is a trade-off between complexity and performance.

For sequential decoding to be reliable, the difference on the path cost between ML path and other paths needs to be large enough. Here we consider an example with $(40, 20)$ TB-CC. A code has 2^k paths in total, hence for $k = 20$, there are 1048576 paths. The accumulated cost of a path follows noncentral chi-squared distribution, and the distribution of the ML path can be found through the order statistic method. For $R = 1/2$ and $N = 40$, the PPV bound is about 5.3 dB, and the sphere packing bound is about 4 dB, at FER level 10^{-4}. Assume we can achieve the sphere packing bound with actual code with sequential decoder, and assume the code is $M = 8$ TB-CC, then the

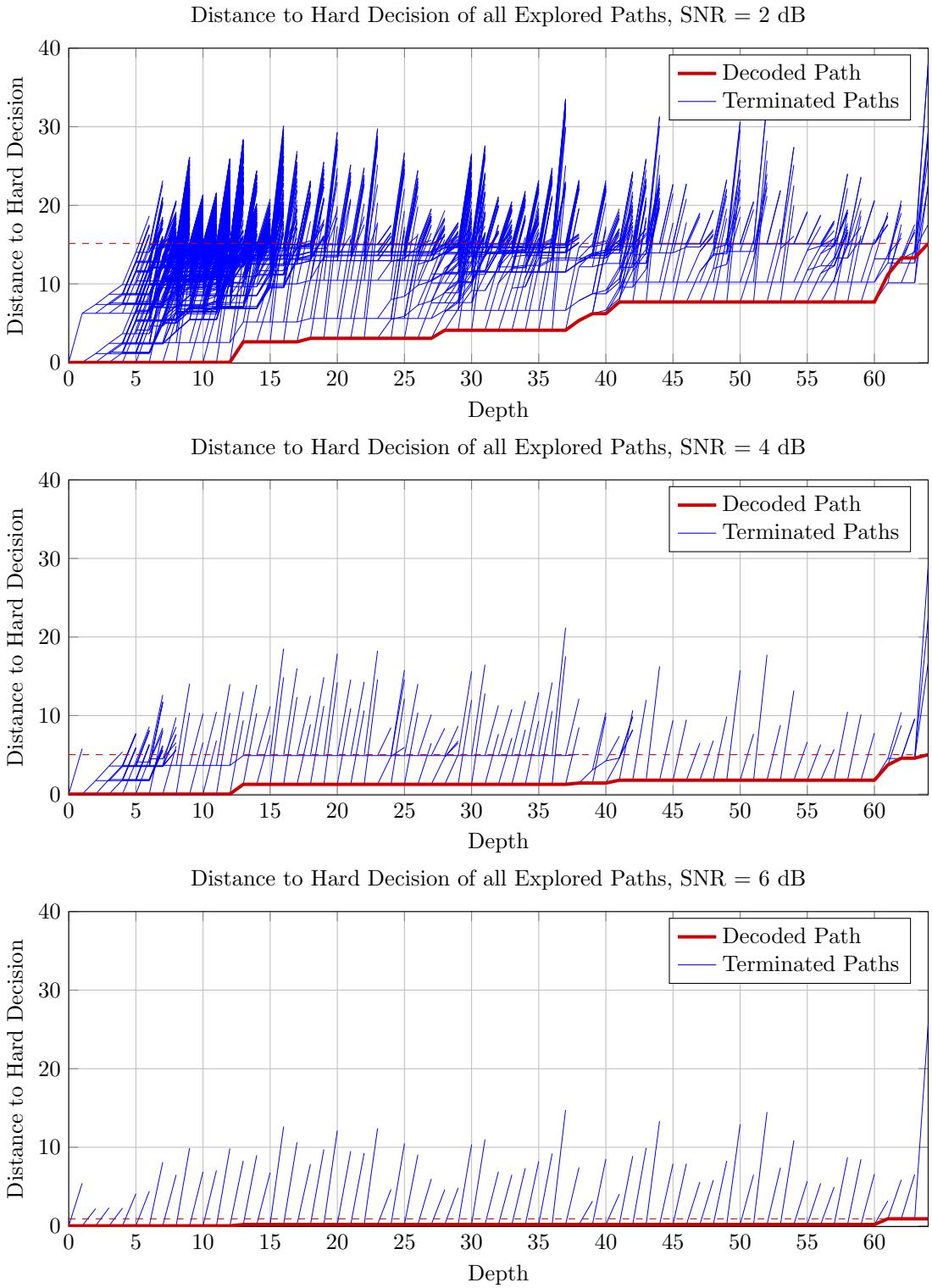

Figure 5.7: The explored paths of $(128, 64)$ TB-CC ($M = 3$, $[54, 74]$) with sequential decoding. The decoder uses stack algorithm with variable bias term metric, SNR = 2, 4 and 6 dB.

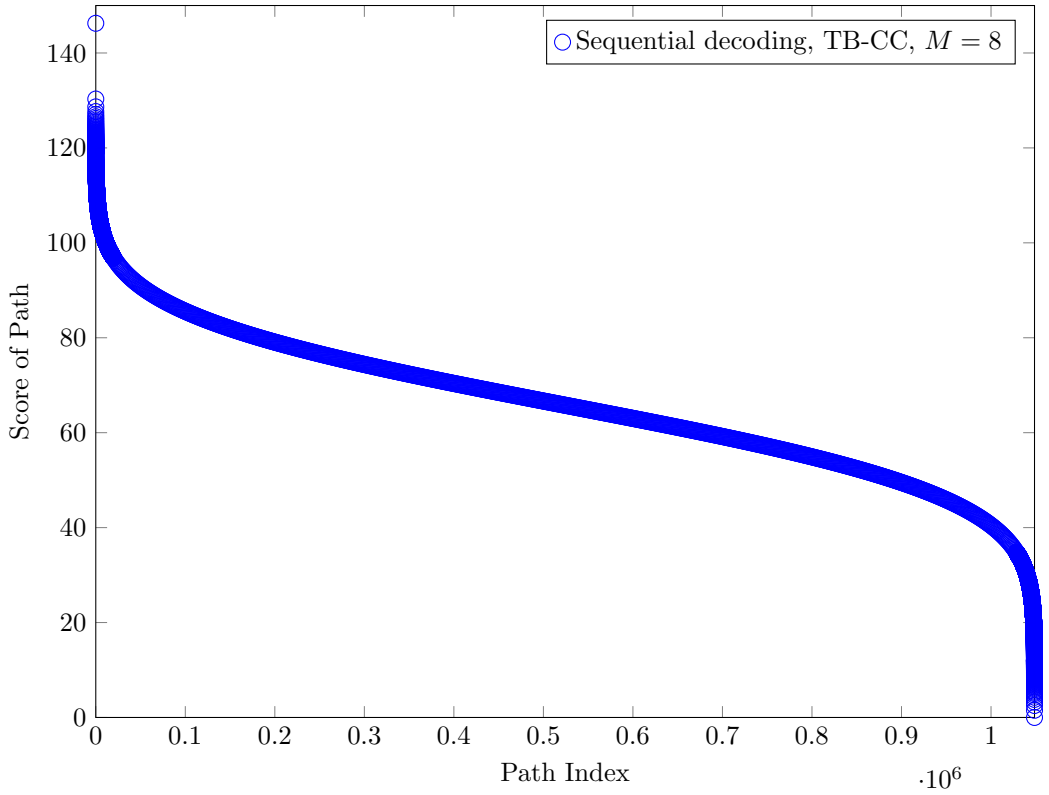

Figure 5.8: The score of paths in descending order of $(40, 20)$ TB-CC ($M = 8$ $[751, 557]$) with sequential decoding, SNR $= 4$ dB.

distribution of score of all paths is shown in Fig. 5.8 in descending order. Here the score is defined as the maximum path cost minus the accumulated cost of each path. It shows that the score of ML path and the score of the second best path has a relatively huge gap, which is about 11% of the ML score.

In another respect, the code structure also affects the decoding complexity. We start with convolutional codes, since many conventional research of sequential decoding are based on them. For convolutional code with particular code rate, superior sequential decoding performance requires large free distance for minimum error probability, and under given free distance, an optimal sequential decoder requires rapid initial column distance growth for low decoding complexity [50]. The definitions of column distance and free distance are described below.

The ith order column distance d_i^c of generator matrix $G(D)$ refers to the minimum Hamming distance between two encoded sequence $(v_1, v_2, \ldots, v_{i+1})$ up to index $i + 1$ resulting from two causal information sequences $\mathbf{u} = (u_1, u_2, \ldots)$ with different u_1 [51].

Free distance d_{free} refers to the minimum Hamming distance between any two differing codewords of a particular code, i.e., $d_{\text{free}} = \min(d_H(\mathbf{v}, \hat{\mathbf{v}})), \mathbf{v} \neq \hat{\mathbf{v}}$. From the code linearity

it follows that d_{free} is upper bounded by the minimum Hamming weight over the non-zero codewords.

According to the structure of convolutional codes, we know d_{free} is determined by the memory order M. Larger M refers to greater minimum Hamming weight, which results in larger d_{free}. On the other hand, larger M can result in higher decoding complexity. The decoding complexity of sequential decoder is considered to be independent of the memory order of convolutional code, since the fraction of the weight of tail branch approaches 0 as the code length N goes to infinity, but this does not hold for block codes. Tail branch of the code tree with greater weight indicates higher column distance growth of the tail branch, which increases the proportion of tail branch in the code decision. Thus the probability for more nodes in lower depth to be explored is also higher, which will lead to higher computational complexity. Therefore, there is a trade-off between performance and complexity in the tail length selection.

Although sequential decoding is a technique developed for the decoding process of convolutional codes, it can be applied to any code as long as the code can be represented as a code tree structure, which generally contains any linear block codes. This fact allows the extensive use of sequential decoder, but only the codes with special properties are suitable for it. As mentioned above, the tail length of the code needs to be relatively short. For example, BCH code has relatively high lower bound of d_{min} under moderate block length regime, which indicates it has good ML performance, but the structure of BCH code is not suitable for sequential decoding since the tail length is about $N-k+1$, which can results in very large computational complexity.

On the other hand, rapid increase on initial column distance growth is expected for lower decoding complexity. First consider the structure of a binary linear block code with generator matrix G that has transformed into an upper triangular matrix through Gaussian elimination. Each row of G starts with a certain number of 0's then followed by a 1, except first row starts with 1. Define the positions of first 1's as index profile. For example, for a convolutional code with generator matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix},$$

the corresponding index profile is $\{1, 3, 5, 7\}$. For a linear code, the branch size of the tree structure are determined by the difference of adjacent terms of index profile. For terminated convolutional codes, index profile is an arithmetic progression and branch

size is a constant value, while the index profile of a general block code can any possible set. Setting the initial branch length to be large makes a rapid growth of initial column distance, but since the corresponding section of excess of rows remains all 0, the minimum Hamming distance of the code is relatively low, which leads to high error error probability. This is another performance-complexity trade-off.

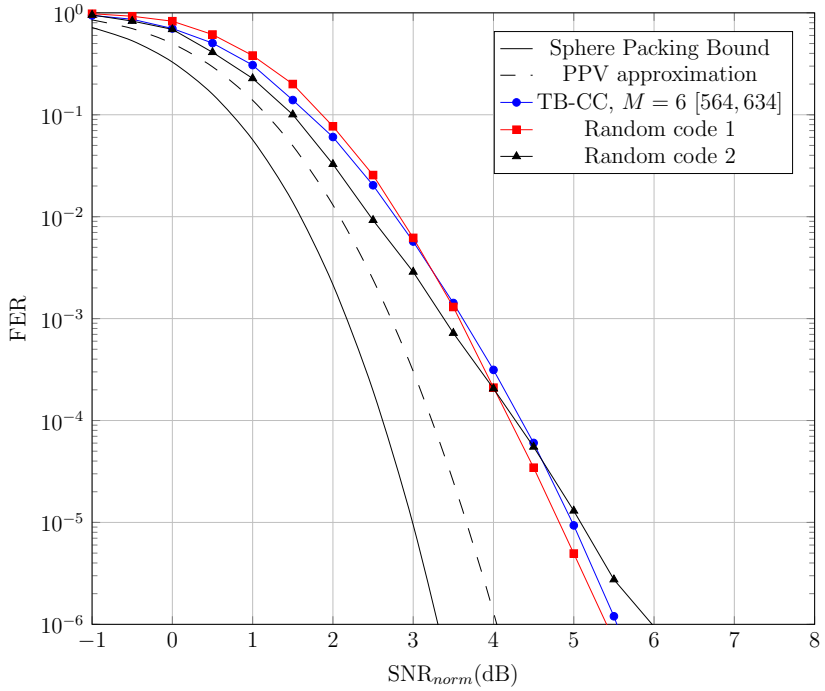

Figure 5.9: Code performance for stack algorithm sequential decoding with improved VBT metric on different $(128, 64)$ codes

Since most linear codes have parameters close to the Gilbert-Varshamov bound [52], there is substantial probability for a randomly generated code to have sufficient minimum Hamming distance. Therefore, the generator matrices can be randomly generated to construct linear block codes for sequential decoding. One possible way to generate random codes is to fix the tail length of G for target complexity and performance level. After that for convenience, we can generate the index profile manually or randomly, then fill the upper triangle of G with randomly generated 0 or 1. Some of the codes are unsatisfactory, but there exist randomly generated codes that are competitive compared to convolutional codes. Fig. 5.9 and 5.10 represent an example the performance and complexity of $M = 6$ tail-biting convolutional codes and two randomly generated codes, where the generator are searched among 100 randomly generated matrices with tail length 14. In Appendix C, for rate 3/4 and rate 2/3 codes with sequential decoder, we randomly generated 100 codes and applied sequential decoding algorithm to them, where some of the codes are

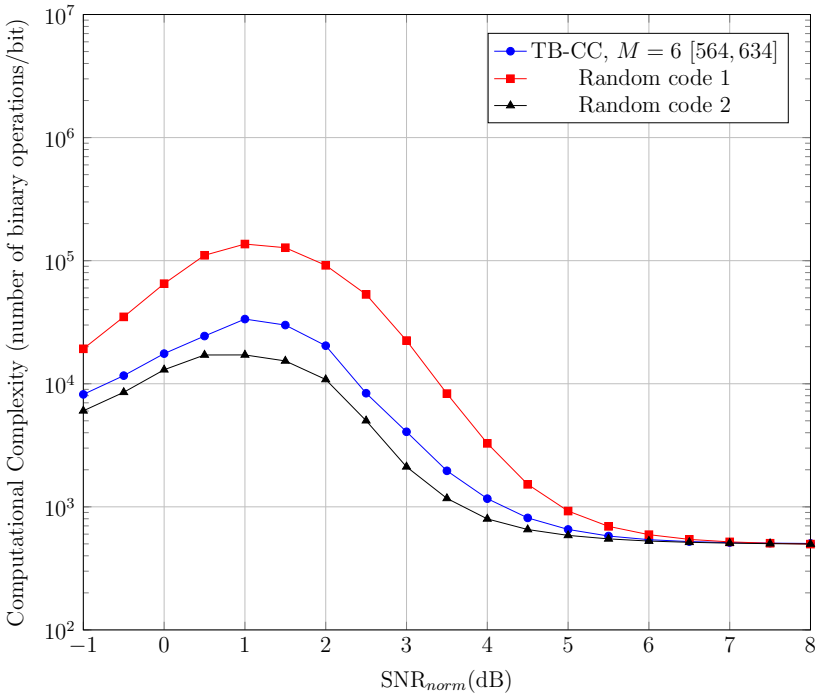

Figure 5.10: Computational complexity for stack algorithm sequential decoding with improved VBT metric on different $(128, 64)$ codes

relatively competitive in the sense of the trade-off between performance and complexity. This fact indicates the extensive applicability of sequential decoding algorithms.

5.4 Simulation Results

Fig. 5.11 and 5.12 shows the code performance and decoding complexity of sequential decoding applied to different $(128, 64)$ linear block codes, and Fig. 5.13 represents the number of nodes visited for each bit decoded. Here we consider the stack algorithm with improved VBT metric described above. Codes with superior free distance and index profile are selected, including the tail-biting convolutional codes with memory order from $M = 2$ to $M = 8$ described in [50] and several randomly generated block codes. The size of priority queue is not constrained in the simulation for convenience. With constrained size of priority queue, performance is slightly sacrificed for lower memory usage and lower computational complexity.

Fig. 5.11 and 5.12 shows that the codes with lower error probability always have higher complexity. The complexity of sequential decoding converges to a constant value, which refers to the case that only the correct path is explored. For uRLLC scenarios a relatively low error rate is required, hence the decoding complexity is close to the

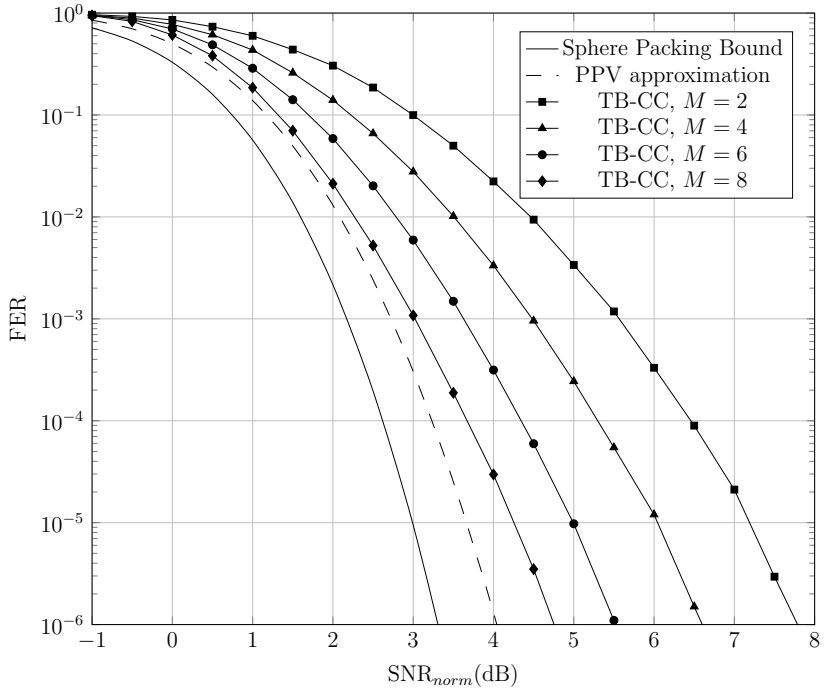

Figure 5.11: Frame error rate versus normalized SNR for tail-biting convolutional code with sequential decoder, $R = 1/2$, $N = 128$.

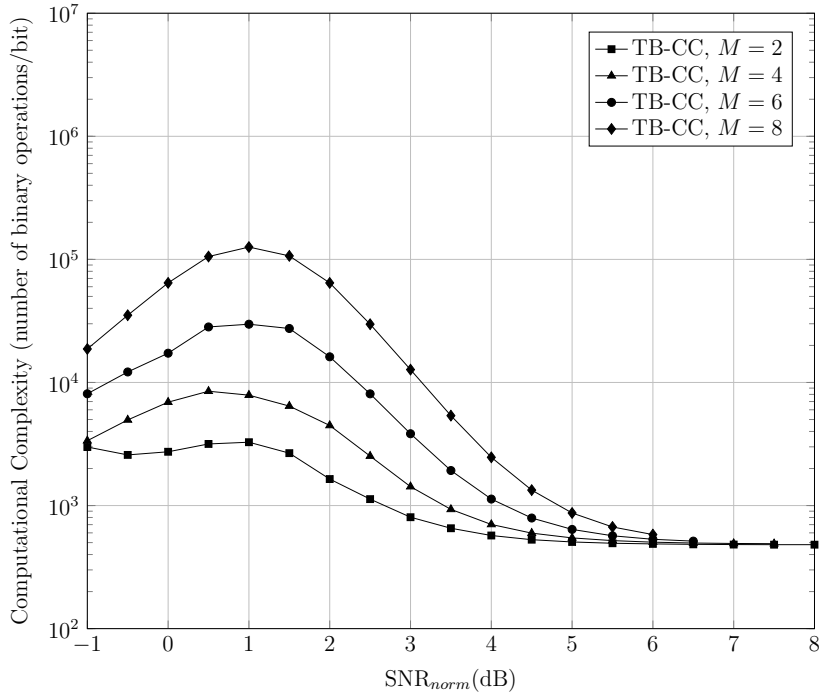

Figure 5.12: Computational complexity versus normalized SNR for tail-biting convolutional code with sequential decoder, rate $= 1/2$, $N = 128$.

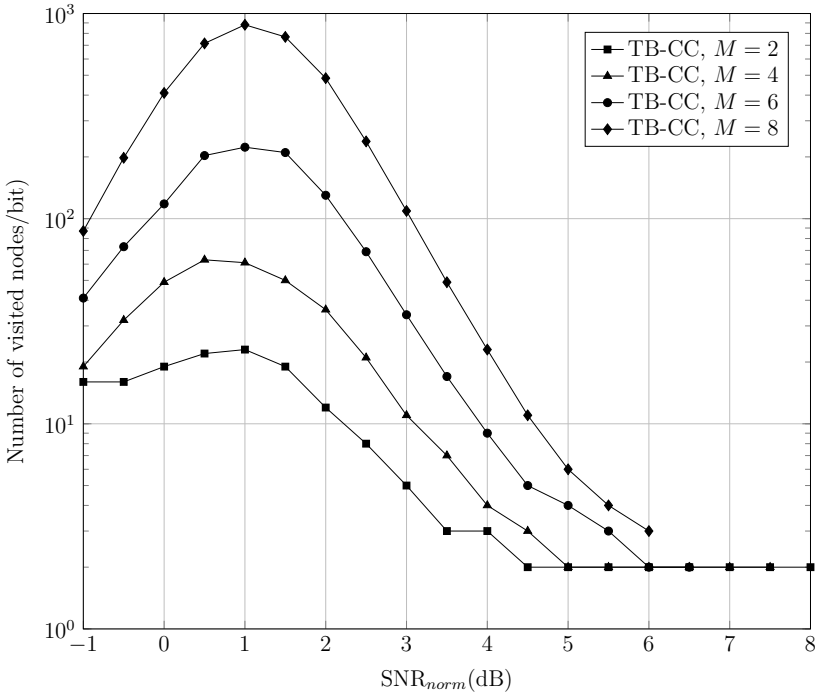

Figure 5.13: Number of visited nodes versus normalized SNR for tail-biting convolutional code with sequential decoder, rate = 1/2, $N = 128$.

minimum level. Although the ML performance of codes that are suitable for sequential decoder is not superior, the sequential decoder is still competitive for the decoding of short block length codes.

Chapter 6

Conclusions and Future Directions

6.1 Trade-off Between Complexity and Performance

In this chapter we will extract insights from the simulations to eventually come to a general conclusion. Recall that we consider the transmission by BPSK modulation of binary linear block codes over linear and time-invariant channels with additive white Gaussian noise (AWGN). In this section, we will choose the case study of rate $R = 1/2$ codes with block length $N = 64, 128, 256, 512$. We performed 5×10^5 simulations for each random code decoded with sequential decoder, and 10^7 simulations for other codes. The result of LDPC code is referenced from [5] and [53]. The generator matrices for tail-biting convolutional codes for WAVA decoding algorithm are referenced from [54], and the tail-biting convolutional codes for sequential decoding algorithm are referenced from [55].

In Fig. 6.1 we compare the trade-off between performance and complexity for different coding schemes with block length $N = 128$. In the figure, the x axis represents the computational complexity per bit, and the y axis represents the code imperfectness, which is the FER gap between the sphere packing bound and the actual code performance in dB at FER $= 10^{-4}$. The codes that are closer to the left side have lower complexity, while the codes that are closer to the bottom side have lower error probability, hence codes achieving points near the bottom left corner are considered as achieving a good trade-off between complexity and reliability.

A code is called Pareto efficient if there does not exist a code has better complexity or code imperfectness without having a worse value on the other criterion. In Fig. 6.1, the following codes are considered for Pareto efficiency: polar code with successive cancellation decoder, polar codes with successive cancellation list decoder with list size 2 and 4, $M = 6$ and $M = 8$ tail-biting convolutional codes with sequential decoder,

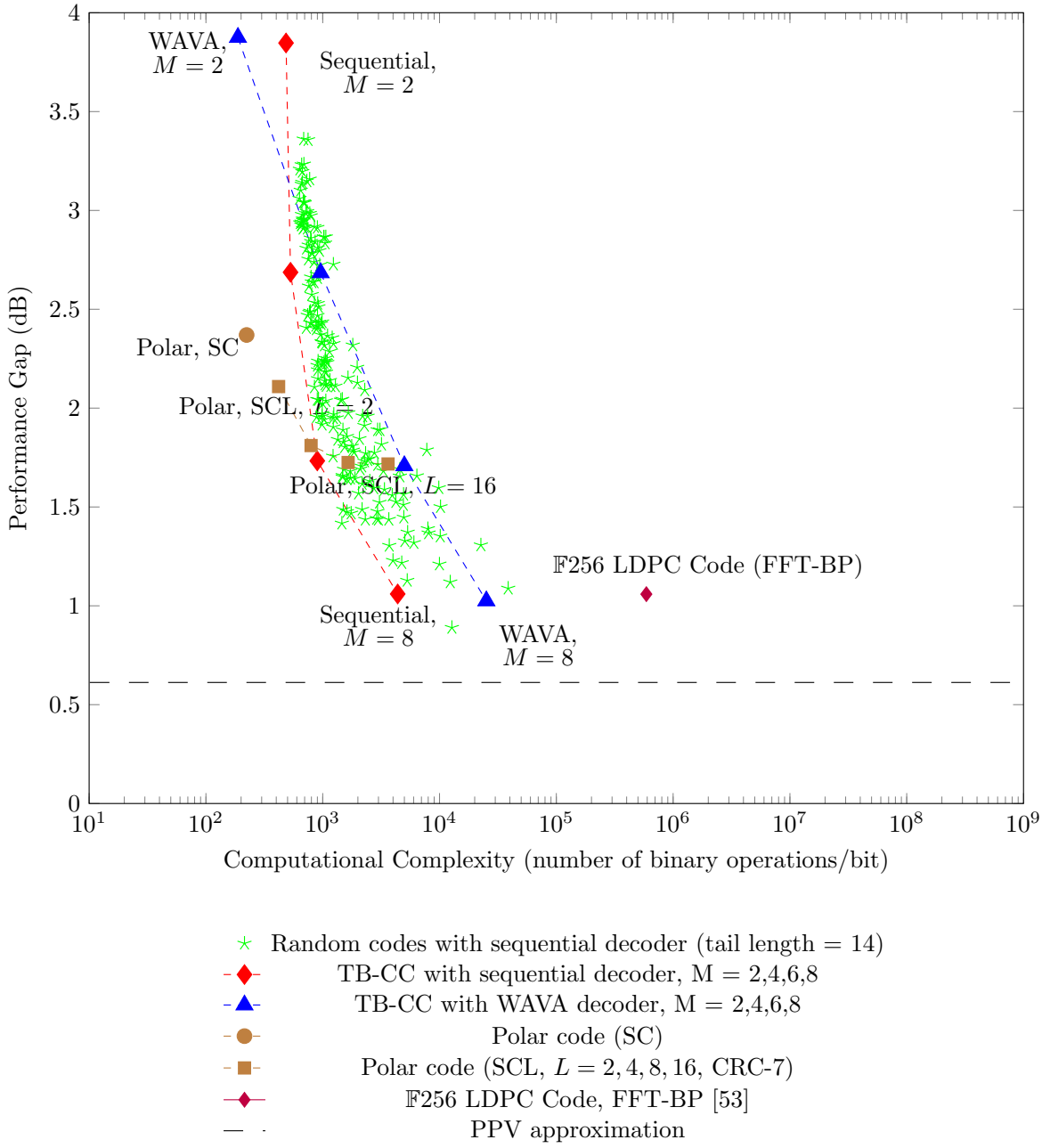

Figure 6.1: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 1/2$, $N = 128$

tail-biting convolutional code of $M = 8$ with Wrap-around Viterbi algorithm decoder, and several random codes with tail length = 14 with sequential decoder. All the codes located on the Pareto frontier are considered as optimal solutions for scenarios with special performance and complexity requirements. For example, if an application requires performance gap to be less than 1.1 dB with complexity less than 10^4 operations/bit, then $M = 8$ TB-CC with sequential decoding algorithms will be the only solution that satisfies the constraints among the codes in Fig. 6.1.

There are 200 random codes with sequential decoder in Fig. 6.1. These codes have randomly generated index profiles with tail length = 14 (which is same as the tail length of $M = 6$ TB-CC), and the upper triangle of generator matrices are randomly filled with 0 or 1. Based on the plot, many of these codes are very close to the frontier formed by TB-CC codes, while some of them are even Pareto efficient. This fact shows that randomly generated codes under specific constraints can have good performance-complexity trade-off with sequential decoding.

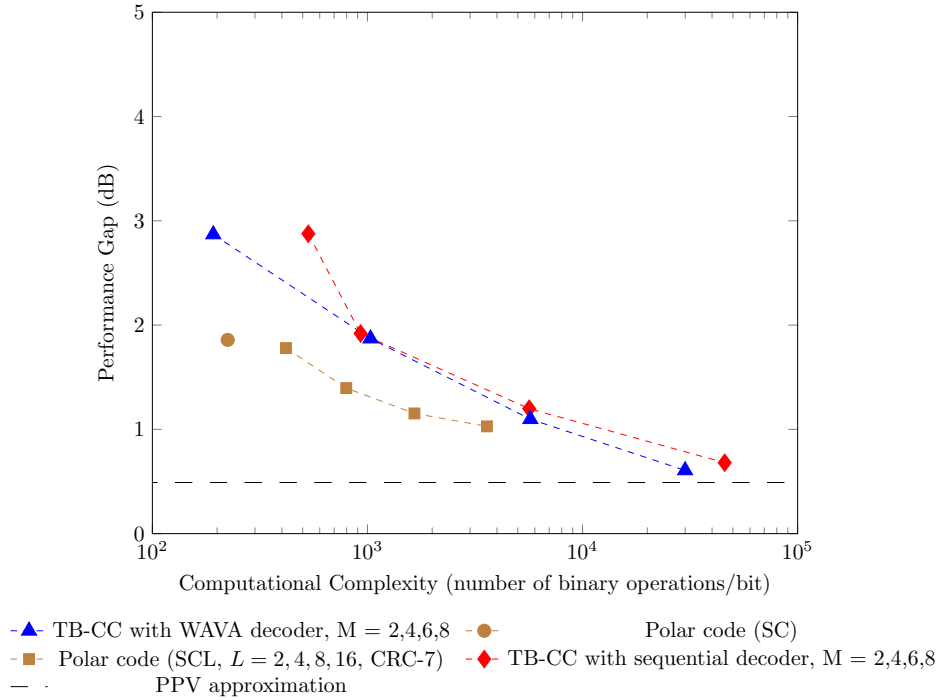

Figure 6.2: Code imperfectness versus computational complexity for different codes at FER $= 10^{-2}$, $R = 1/2$, $N = 128$

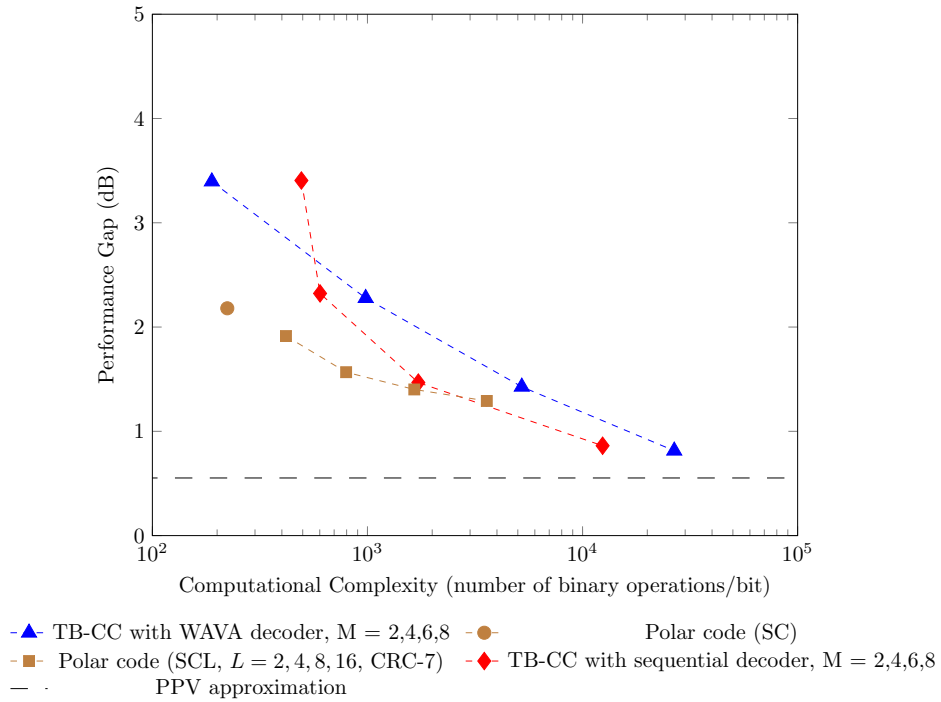

Figure 6.3: Code imperfectness versus computational complexity for different codes at FER $= 10^{-3}$, $R = 1/2$, $N = 128$

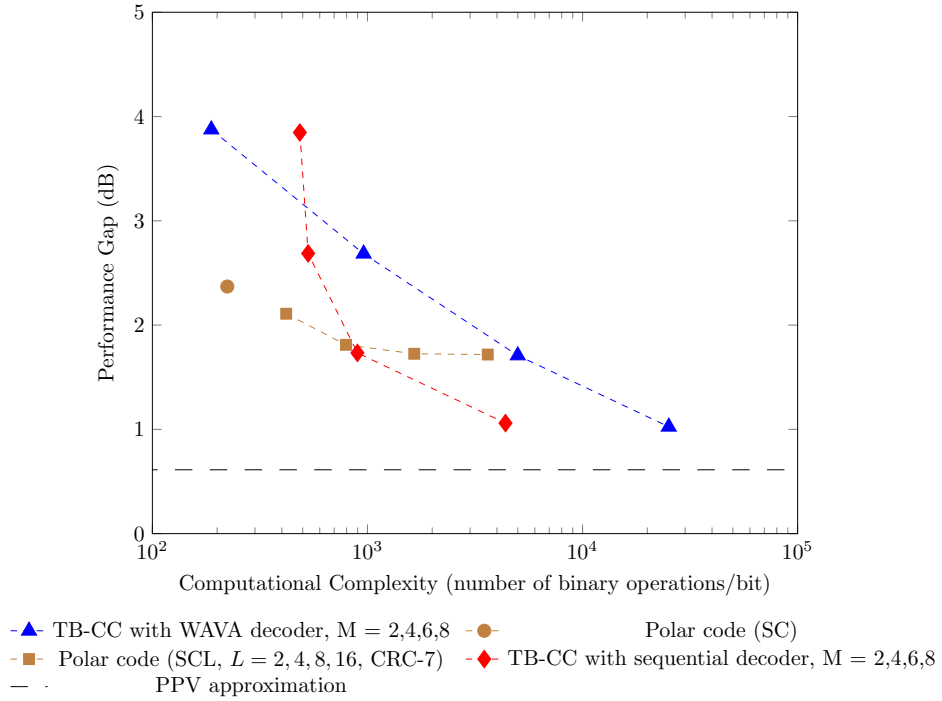

Figure 6.4: Code imperfectness versus computational complexity for different codes at FER $= 10^{-4}$, $R = 1/2$, $N = 128$

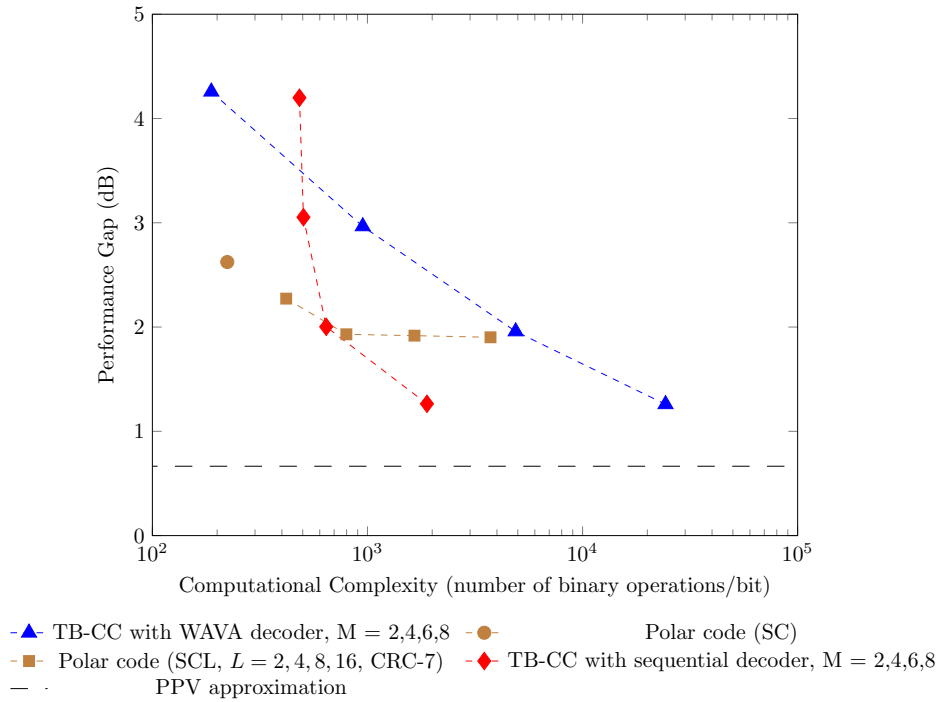

Figure 6.5: Code imperfectness versus computational complexity for different codes at FER $= 10^{-5}$, $R = 1/2$, $N = 128$

Figs. 6.2, 6.3, 6.4 and 6.5 show the trade-off between code imperfectness and computational complexity under different FER levels, from 10^{-2} to 10^{-5}. While TB-CC with WAVA decoder and polar codes have almost the same computational complexity under different FER levels, the complexity of codes decoded by sequential algorithm is low under low FER level. For $M = 8$ TB-CC with sequential decoder, under FER $= 10^{-2}$, the computational complexity is about 4×10^4 operations per bit; under FER $= 10^{-5}$, the complexity is about 2×10^3 operations per bit, which is about 1/20 of the complexity under FER $= 10^{-2}$. In Fig. 6.2, none of the codes with sequential decoder is Pareto efficient under FER $= 10^{-2}$, but in Fig. 6.5, $M = 6$ and $M = 8$ TB-CC with sequential decoder are Pareto efficient under FER $= 10^{-5}$. Therefore, codes with sequential decoder are more competitive when the scenario requires higher reliability.

On the other hand, Figs. 6.6, 6.7, 6.8 and 6.9 compare the performance-complexity trade-off with different code length, from $N = 512$ to $N = 64$. These results show that with relatively large code length, polar codes with SC and SCL decoders have lower complexity and code imperfectness than the other two type of codes, but when the code is short enough, codes with sequential decoder can be very competitive in the sense of performance-complexity trade-off. This is because with short length codes, the channel polarization is far from complete, which results in a loss of performance, while the performance of sequential decoding and WAVA decoding are not designed based on sufficient large code length. In Fig. 6.9, with code length $N = 64$ and FER level $= 10^{-5}$, the performance gap of $L = 16$ polar code with SCL decoder is about 1.3 dB, while the performance gap of $M = 8$ TB-CC with sequential decoder and $M = 8$ TB-CC with WAVA decoder is about only 0.7 dB. On the other hand, since the decoding complexity of polar code with SC decoder is $\mathcal{O}(N \log N)$, the computational complexity of SC decoded polar codes in Fig. 6.9 is about only 2×10^2 operations per bit, which is still very competitive compared to other codes.

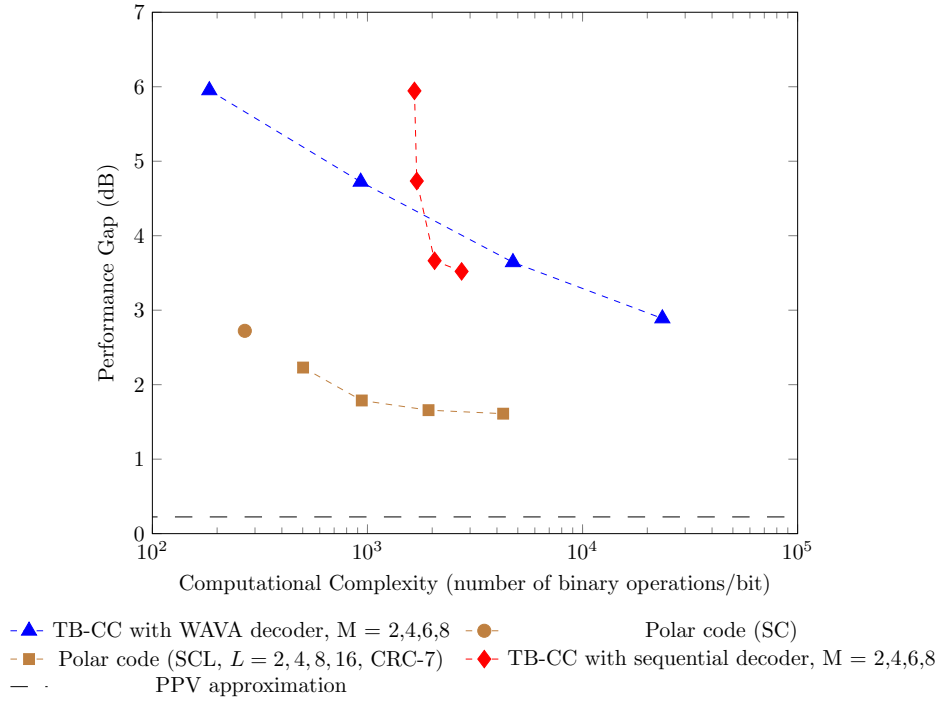

Figure 6.6: Code imperfectness versus computational complexity for different codes at FER $= 10^{-5}$, $R = 1/2$, $N = 512$

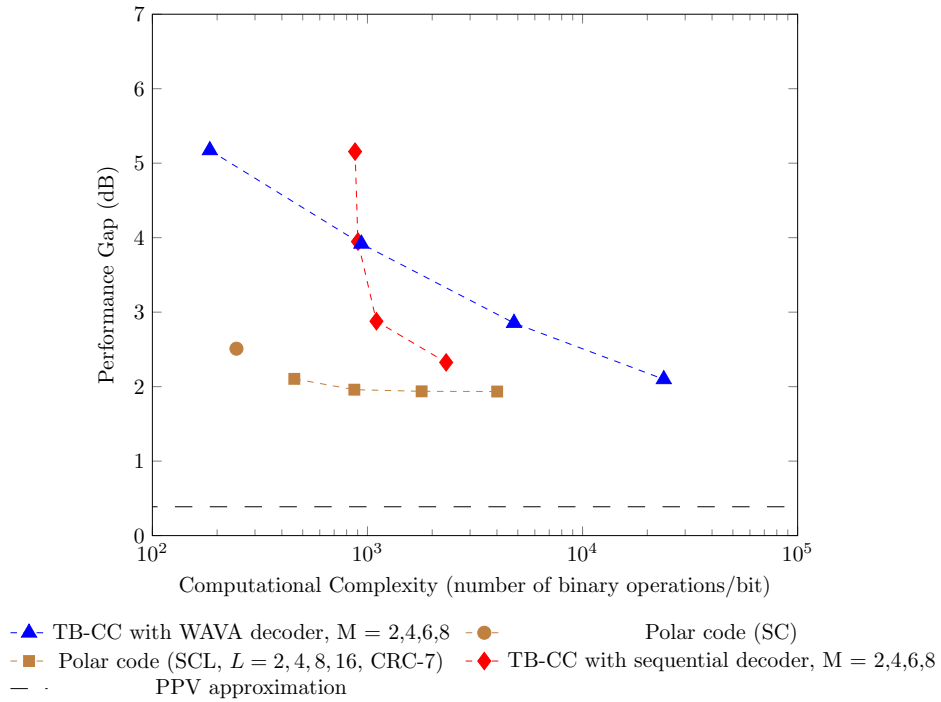

Figure 6.7: Code imperfectness versus computational complexity for different codes at FER $= 10^{-5}$, $R = 1/2$, $N = 256$

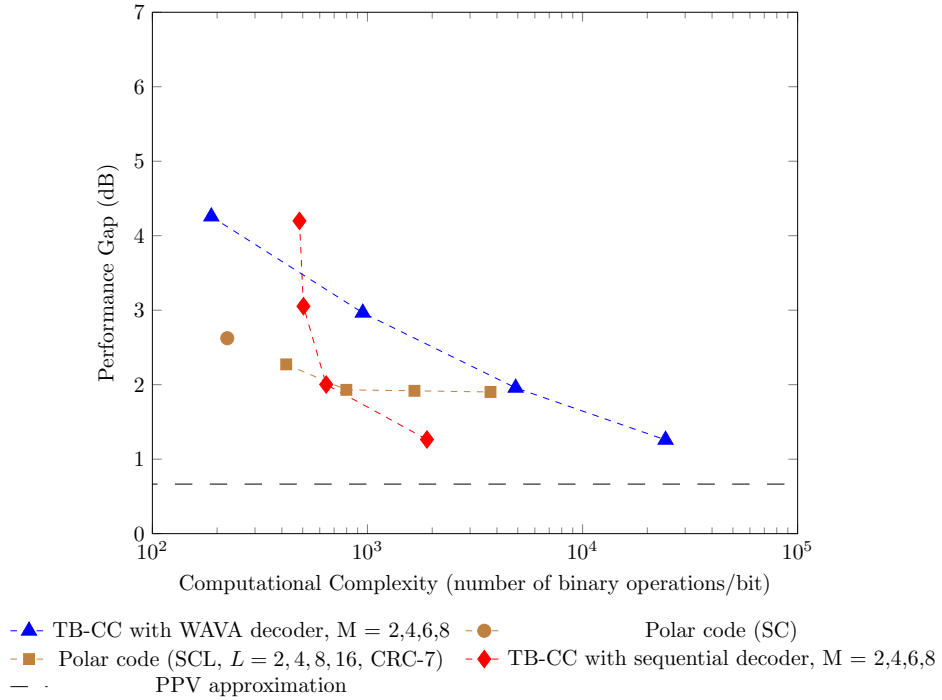

Figure 6.8: Code imperfectness versus computational complexity for different codes at FER $= 10^{-5}$, $R = 1/2$, $N = 128$

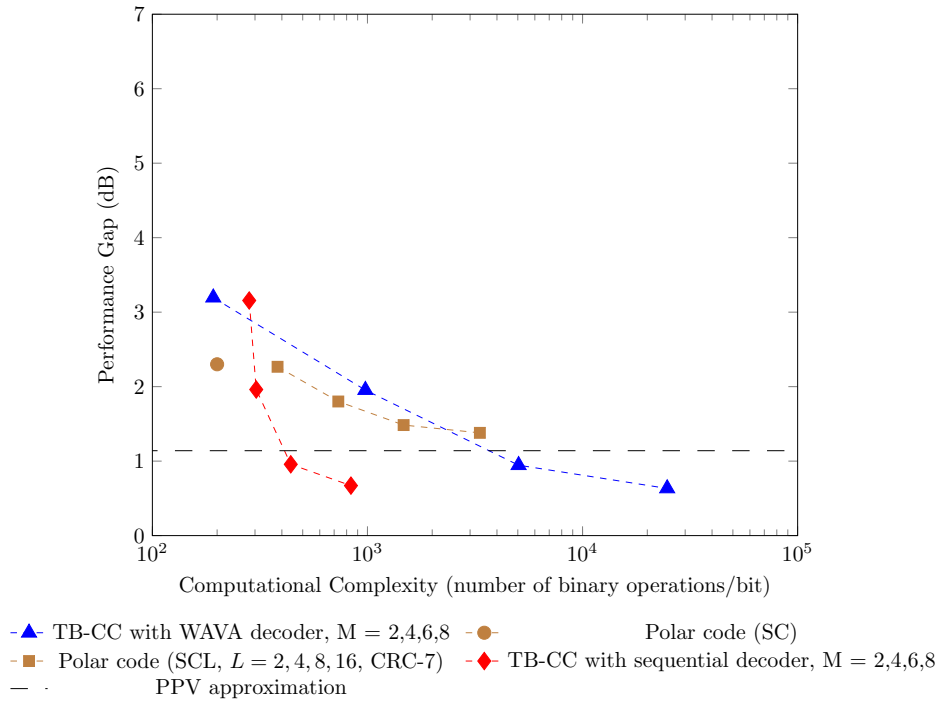

Figure 6.9: Code imperfectness versus computational complexity for different codes at FER $= 10^{-5}$, $R = 1/2$, $N = 64$

6.2 Conclusions

The idea of sequential decoding has been well-known for decades. However, the potential of sequential decoding under ultra high reliability and short block length requires further exploration. This might be because most of the existing research focuses on low reliability requirements, which is not suitable to sequential decoding. However, in 5G and future standards, the high reliability scenario will be taken into consideration. For instance, in this thesis we discussed the FER level down to 10^{-5}, while 5G uRLLC scenarios have extremely high reliability requirements from 10^{-5} to 10^{-9} with code length commonly less than 1000.

Polar codes with SCL decoder have been considered as the undisputed best type of codes under short block length regime. In this thesis, we discussed how the codes with sequential decoder can be competitive compared to polar codes with SCL decoder under short block length regime, especially with high reliability requirements.

According to the simulation results above and in Appendix C, with code length $N \leq 2^7$ and FER requirement $\leq 10^{-4}$, there usually exist some codes with sequential decoder that are located on the Pareto frontier in the trade-off between performance and average complexity, while the index profile of such competitive codes can even be generated randomly.

As mentioned before, the sequential decoding process with the improved variable bias-term metric can be viewed as a first passage problem; hence one future research direction of this work is to develop a more effective algorithm of sequential decoding under short block length regime. Another direction is to develop an improved method to construct of codes with superior free distance and index profile, which can achieve better performance and complexity trade-off for sequential decoding.

Bibliography

[1] ITU-R Rec., "IMT vision–framework and overall objectives of the future development of IMT for 2020 and beyond," *Recommendation ITU*, pp. 2083–0, Sep. 2015.

[2] 3GPP TS 22.261, "3rd generation partnership project; technical specification group services and system aspects; service requirements for the 5G system; stage 1 (release 16)," *Tech. Spec.*, Jun. 2018.

[3] A. Machwe, C. Dent-Young, J. A. Moreno, P. Ciria, M. A. Alonso, G. Lyberopoulos, H. Theodoropoulou, I. Mesogiti, K. Filis, A. Polydorou, C. Tsironas, S. Spiliadis, A. D. Giglio, A. Pagano, A. Percelsi, L. Serra, J. Francis, J. Bartelt, J.-K. Chaudhary, A. Tzanakaki, M. Anastasopoulos, and D. Simeonidou, "5G and vertical services, use cases and requirements," *5G PPP*, Jan. 2018.

[4] ETSI TR 102 889-2, "Electromagnetic compatibility and radio spectrum matters (ERM); system reference document; short range devices (SRD); part 2: Technical characteristics for SRD equipment for wireless industrial applications using technologies different from ultra-wide band (UWB)," *Tech. Report*, Aug. 2011.

[5] M. Shirvanimoghaddam, M. S. Mohammadi, R. Abbas, A. Minja, C. Yue, B. Matuz, G. Han, Z. Lin, W. Liu, Y. Li *et al.*, "Short block-length codes for ultra-reliable low latency communications," *IEEE Commun. Mag.*, vol. 57, no. 2, pp. 130–137, Dec. 2018.

[6] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai, "A survey on low latency towards 5G: RAN, core network and caching solutions," *arXiv preprint arXiv:1708.02562*, 2017.

[7] P. Schulz, M. Matthe, H. Klessig, M. Simsek, G. Fettweis, J. Ansari, S. A. Ashraf, B. Almeroth, J. Voigt, and I. Riedel, "Latency critical IoT applications in 5G: Perspective on the design of radio interface and network architecture," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 70–78, Feb. 2017.

[8] M. Fallgren, B. Timus *et al.*, "Scenarios, requirements and KPIs for 5G mobile and wireless system," *METIS deliverable D*, vol. 1, p. 1, May 2013.

[9] G. P. Fettweis, "The tactile Internet: Applications and challenges," *IEEE Veh. Technol. Mag.*, vol. 9, no. 1, pp. 64–70, Mar. 2014.

[10] D. Warren and C. Dewar, "Understanding 5G: Perspectives on future technological advancements in mobile," *Tech. Rep.*, Dec. 2014.

[11] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, "5G-enabled tactile Internet," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 3, pp. 460–473, Feb. 2016.

[12] Y. Mao and A. H. Banihashemi, "A heuristic search for good low-density parity-check codes at short block lengths," *Proc. 2001 IEEE Int. Conf. Commun.*, vol. 1, pp. 41–44, Aug. 2001.

[13] D. J. C. MacKay and M. C. Davey, "Evaluation of Gallager codes for short block length and high rate applications," in *Proc. IMA Workshop Codes, Systems, and Graphical Models*, 2001, pp. 113–130.

[14] T. Jerkovits and B. Matuz, "Turbo code design for short blocks," *Adv. Satellite Multimedia Syst. Conf. 14th Signal Process. Space Commun. Workshop (ASMS/SPSC), 2016 8th*, pp. 1–6, Sep. 2016.

[15] P. Popovski, J. J. Nielsen, C. Stefanovic, E. de Carvalho, E. Strom, K. F. Trillingsgaard, A.-S. Bana, D. M. Kim, R. Kotaba, J. Park, and R. B. Sorensen, "Wireless access for ultra-reliable low-latency communication: Principles and building blocks," *IEEE Network*, vol. 32, no. 2, pp. 16–23, Apr. 2018.

[16] G. Liva, L. Gaudio, T. Ninacs, and T. Jerkovits, "Code design for short blocks: A survey," *arXiv preprint arXiv:1610.00873*, Oct. 2016.

[17] F. Panneton, P. L'ecuyer, and M. Matsumoto, "Improved long-period generators based on linear recurrences modulo 2," *ACM Transactions on Mathematical Software (TOMS)*, vol. 32, no. 1, pp. 1–16, Mar. 2006.

[18] R. Paley, E. A. Christopher, and N. Wiener, *Fourier transforms in the complex domain*. American Mathematical Soc., 1934, vol. 19.

[19] G. Caire, G. Taricco, and E. Biglieri, "Bit-interleaved coded modulation," *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 927–946, May 1998.

[20] M. Bennis, M. Debbah, and H. V. Poor, "Ultra-reliable and low-latency wireless communication: Tail, risk and scale," *arXiv preprint arXiv:1801.01270*, Jan. 2018.

[21] C. E. Shannon, "Probability of error for optimal codes in a Gaussian channel," *Bell Labs Tech. J.*, vol. 38, no. 3, pp. 611–656, May 1959.

[22] A. Valembois and M. P. Fossorier, "Sphere-packing bounds revisited for moderate block lengths," *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 2998–3014, Nov. 2004.

[23] S. Dolinar, D. Divsalar, and F. Pollara, "Code performance as a function of block size," *TMO Progress Report*, vol. 42, no. 133, May 1998.

[24] R. Y. Shao, S. Lin, and M. P. Fossorier, "Two decoding algorithms for tailbiting codes," *IEEE Trans. Commun.*, vol. 51, no. 10, pp. 1658–1665, Oct. 2003.

[25] R. Johannesson and K. S. Zigangirov, *Fundamentals of convolutional coding*. John Wiley & Sons, 2015, vol. 15.

[26] F. R. Kschischang and V. Sorokine, "On the trellis structure of block codes," *IEEE Trans. Inf. Theory*, vol. 41, no. 6, pp. 1924–1937, Aug. 1995.

[27] S. Johnson, "A new upper bound for error-correcting codes," *IRE Trans. Inf. Theory*, vol. 8, no. 3, pp. 203–207, Apr. 1962.

[28] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Channel coding rate in the finite block-length regime," *IEEE Trans. Inf. Theory*, vol. 56, no. 5, pp. 2307–2359, 2010.

[29] E. Arıkan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[30] M. Alsan and E. Telatar, "A simple proof of polarization and polarization for non-stationary memoryless channels," *IEEE Trans. Inf. Theory*, vol. 62, no. 9, pp. 4873–4878, 2016.

[31] I. Tal and A. Vardy, "How to construct polar codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 10, pp. 6562–6582, Oct. 2013.

[32] P. Trifonov, "Efficient design and decoding of polar codes," *IEEE Trans. Commun.*, vol. 60, no. 11, pp. 3221–3227, Aug. 2012.

[33] A. Elkelesh, M. Ebada, S. Cammerer, and S. t. Brink, "Decoder-in-the-loop: Genetic optimization-based ldpc code design," *arXiv preprint arXiv:1903.03128*, Mar. 2019.

[34] E. Şaşoğlu and L. Wang, "Universal polarization," *IEEE Trans. Inf. Theory*, vol. 62, no. 6, pp. 2937–2946, Apr. 2016.

[35] S. H. Hassani and R. Urbanke, "Universal polar codes," *ISIT 2014,IEEE Int. Symp. on Inf. Theory*, Aug. 2014.

[36] E. Arıkan and E. Telatar, "On the rate of channel polarization," *ISIT 2009, IEEE Int. Symp. on Inf. Theory*, pp. 1493–1495, Aug. 2009.

[37] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Trans. Inf. Theory*, pp. 1–5, Mar. 2015.

[38] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "Llr-based successive cancellation list decoding of polar codes," *IEEE Trans. Signal Process.*, vol. 63, no. 19, pp. 5165–5179, Jun. 2015.

[39] K. Niu and K. Chen, "Crc-aided decoding of polar codes," *IEEE Commun. Lett.*, vol. 16, no. 10, pp. 1668–1671, Sep. 2012.

[40] J. M. Wozencraft, *Sequential decoding for reliable communication*. Research Laboratory of Electronics, Massachusetts Institute of Technology, 1957.

[41] R. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. Inf. Theory*, vol. 9, no. 2, pp. 64–74, Apr. 1963.

[42] K. Zigangirov, "Some sequential decoding procedures," *Probl. Peredachi Inf.*, vol. 2, no. 4, pp. 13–25, 1966.

[43] F. Jelinek, "Fast sequential decoding algorithm using a stack," *IBM Journal of Research and Development*, vol. 13, no. 6, pp. 675–685, Nov. 1969.

[44] V. Imtawil, "Creeper: An algorithm for decoding convolutional codes," *Proc. 2001 IEEE Int. Conf. Commun. Syst.*, pp. 332–336, Feb. 2002.

[45] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. 13, no. 2, pp. 260–269, Apr. 1967.

[46] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[47] J. Massey, "Variable-length codes and the fano metric," *IEEE Trans. Inf. Theory*, vol. 18, no. 1, pp. 196–198, Jan. 1972.

[48] V. Sorokine and F. R. Kschischang, "A sequential decoder for linear block codes with a variable bias-term metric," *IEEE Trans. Inf. Theory*, vol. 44, no. 1, pp. 410–416, Jan. 1998.

[49] E. Arıkan, "An upper bound on the cutoff rate of sequential decoding," *IEEE Trans. Inf. Theory*, vol. 34, no. 1, pp. 55–63, Jan. 1988.

[50] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[51] D. J. Costello, "A construction technique for random-error-correcting convolutional codes," *IEEE Trans. Inf. Theory*, vol. 15, no. 5, pp. 631–636, Sep. 1969.

[52] R. Roth, *Introduction to coding theory*. Cambridge University Press, 2006.

[53] M. Helmling, S. Scholl, F. Gensheimer, T. Dietz, K. Kraft, S. Ruzika, and N. Wehn, "Database of Channel Codes and ML Simulation Results," www.uni-kl.de/channel-codes, 2017.

[54] P. Stahl, J. B. Anderson, and R. Johannesson, "Optimal and near-optimal encoders for short and moderate-length tail-biting trellises," *IEEE Trans. Inf. Theory*, vol. 45, no. 7, pp. 2562–2571, Nov. 1999.

[55] R. Johannesson and P. Stahl, "New rate 1/2, 1/3, and 1/4 binary convolutional encoders with an optimum distance profile," *IEEE Trans. Inf. Theory*, vol. 45, no. 5, pp. 1653–1658, Jul. 1999.

[56] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi, "5G wireless network slicing for eMBB, URLLC, and mMTC: A communication-theoretic view," *arXiv preprint arXiv:1804.05057*, Aug. 2018.

[57] S. B. Wicker, *Error control systems for digital communication and storage*. Englewood Cliffs, NJ: Prentice-Hall, 1995, vol. 1.

[58] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*. Courier Corporation, 1965, vol. 55.

Appendices

Appendix A

Calculations for Tail-Biting Convolutional Codes

A.1 Maximum Likelihood Estimations for Gaussian Channel

A.1.1 Maximum Likelihood Decoding

Following the definition, the likelihood function is the density function

$$L(\theta|\mathbf{x}) = f(\mathbf{x}|\theta), \tag{A.1}$$

where θ denotes the parameter vector, and $\mathbf{x}$ denotes the set of random samples. The maximum likelihood estimation is defined as

$$\hat{\theta}(\mathbf{x}) = \arg\max_{\theta\in\Theta} L(\theta|\mathbf{x}), \tag{A.2}$$

where Θ is the set of all possible θ values.

We formulate the maximum-likelihood decoding problem as follows. Suppose $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ is the codeword selected for transmission, and $\mathbf{r} = (r_1, r_2, \ldots, r_n)$ is the corresponding received sequence through a Gaussian channel.

With known channel state information, we can assume codeword $\mathbf{v}$ is the parameter vector that we want to estimate based on the known Gaussian channel and the received codeword $\mathbf{r}$. Considering equation (A.2), here we want to find a code $\hat{v}$ that satisfies

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v}\in\mathcal{C}} L(\mathbf{v}|\mathbf{r}), \tag{A.3}$$

where C represents the codebook. Since the logarithm function is monotonically increasing, (A.3) is equivalent to

$$\begin{aligned}\hat{\mathbf{v}} &= \arg\max_{\mathbf{v}\in\mathcal{C}} \Big(\ln L(\mathbf{v}|\mathbf{r}) \Big) \\ &= \arg\min_{\mathbf{v}\in\mathcal{C}} \Big(-\ln L(\mathbf{v}|\mathbf{r}) \Big) \\ &= \arg\max_{\mathbf{v}\in\mathcal{C}} \Big(-\ln \prod_{i=1}^{n} P(r_i|v_i) \Big) \\ &= \arg\max_{\mathbf{v}\in\mathcal{C}} \Big(-\sum_{i=1}^{n} \ln P(r_i|v_i) \Big). \end{aligned} \tag{A.4}$$

For the additive white Gaussian noise channel with noise variance σ^2, r_i has the distribution $\mathcal{N}(v_i, \sigma^2)$. The probability density function of r_i has the form

$$f(r_i|v_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left(\frac{r_i - v_i}{2\sigma^2}\right)^2}, \tag{A.5}$$

where $v_1, v_2, \ldots, v_i$ are the transmitted bits that need to be estimated. Similar to (A.13) and (A.14), the log likelihood function of ith bit, v_i can be expressed as

$$\ell(v_i|r_i) = \ln L(v_i|r_i) = \ln f(r_i|v_i) = -\frac{1}{2}\ln(2\pi\sigma^2) - \frac{(r_i - v_i)^2}{2\sigma^2}. \tag{A.6}$$

We can define a term "metric" as a linear transformation of the log likelihood function, which is also referred to "likelihood cost" or "likelihood distance", where

$$C(r_i|v_i) = a_1\big(-\ell(v_i|r_i) + a_2 \big), \tag{A.7}$$

where a_1 and a_2 are constants and a_1 is positive. For the codeword it satisfies

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v}\in\mathcal{C}}(\ln L(\mathbf{v}|\mathbf{r})) = \arg\min_{\mathbf{v}\in\mathcal{C}}(C(\mathbf{r}|\mathbf{v})), \tag{A.8}$$

and for a single bit it satisfies

$$\hat{v_i} = \arg\max_{v_i\in q}(\ln L(v_i|r_i)) = \arg\min_{v_i\in q}(C(r_i|v_i)). \tag{A.9}$$

Hence minimizing the likelihood cost is equivalent to maximizing the transition probability. Choosing $a_1 = 2\sigma^2$, $a_2 = \frac{1}{2}\ln(2\pi\sigma^2)$, we have

$$C(r_i|v_i) = (r_i - v_i)^2. \tag{A.10}$$

Since all the bits are considered as i.i.d., the likelihood cost between the received vector $\mathbf{r}$ and a codeword $\mathbf{v}$ is therefore shown as

$$C(\mathbf{r}|\mathbf{v}) = \sum_{i=1}^{n}(r_i - v_i)^2, \tag{A.11}$$

where the magnitude of the likelihood cost is geometrically equal to the squared Euclidean distance. A decoder with this measure is usually called a "soft-decision decoder", while a "hard-decision decoder" makes decision about every individual bit prior to decoding process of the whole codeword.

A.1.2 Gaussian Channel Estimation

Maximum likelihood estimation is also important in channel estimate procedures. Assume all the bits in v has the same magnitude, then we can assume $\mathbf{r}$ is a length n sequence of i.i.d. Gaussian random variables with mean μ and variance σ^2. The probability density function of ith term in this sequence can be expressed as

$$f_R(r_i|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left(\frac{r_i-\mu}{2\sigma^2}\right)^2}, \tag{A.12}$$

where μ and σ are the parameters that required to be estimated. The likelihood function can be written as

$$\begin{aligned} L(\mu,\sigma^2|\mathbf{r}) &= f_R(r_1, r_2, \ldots, r_n|\mu,\sigma^2) \\ &= \prod_{i=1}^{n} f_R(r_i|\mu,\sigma^2) \\ &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\left(\frac{r_i-\mu}{2\sigma^2}\right)^2} \\ &= \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\Big(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(r_i-\mu)^2\Big). \end{aligned} \tag{A.13}$$

Taking the natural logarithm of both side, we can obtain

$$\ell(\mu,\sigma^2|\mathbf{r}) = \ln L(\mu,\sigma^2|\mathbf{r}) = -\frac{n}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(r_i-\mu)^2. \tag{A.14}$$

To find the value of μ and σ that can maximize $\ell(\mu,\sigma^2|\mathbf{r})$, we consider the first order

partial derivative for μ,

$$\begin{aligned}\frac{\partial}{\partial\mu}\ell(\mu,\sigma^2|\mathbf{r}) &= \frac{\partial}{\partial\mu}\Big(-\frac{n}{2}\ln(2\pi\sigma^2)-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(r_i-\mu)^2\Big)\\ &= \frac{1}{\sigma^2}\sum_{i=1}^{n}(r_i-\mu)\\ &= \frac{1}{\sigma^2}\Big(\sum_{i=1}^{n}r_i-n\mu\Big),\end{aligned} \tag{A.15}$$

which equal to zero if and only if

$$\mu = \frac{1}{n}\sum_{i=1}^{n}r_i. \tag{A.16}$$

Since the second partial derivative with respect to μ is negative, the maximum likelihood estimation of the mean is

$$\hat{\mu}(\mathbf{r}) = \bar{r}. \tag{A.17}$$

Similarly, for σ,

$$\begin{aligned}\frac{\partial}{\partial\sigma^2}\ell(\mu,\sigma^2|\mathbf{r}) &= \frac{\partial}{\partial\sigma^2}\Big(-\frac{n}{2}\ln(2\pi\sigma^2)-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(r_i-\mu)^2\Big)\\ &= \frac{1}{2(\sigma^2)^2}\sum_{i=1}^{n}(r_i-\mu)^2-\frac{n}{2\sigma^2}\\ &= \frac{n}{2(\sigma^2)^2}\Big(\frac{1}{n}\sum_{i=1}^{n}(r_i-\mu)^2-\sigma^2\Big),\end{aligned} \tag{A.18}$$

which equal to zero if and only if

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(r_i-\mu)^2, \tag{A.19}$$

thus the maximum likelihood estimation of the variance is

$$\hat{\sigma}^2(\mathbf{r}) = \frac{1}{n}\sum_{i=1}^{n}(r_i-\hat{\mu})^2. \tag{A.20}$$

Note that the maximum likelihood estimator $\hat{\mu}$ is the sample mean that is unbiased since $\mathbf{E}[\hat{\mu}] = \mu$, while the maximum likelihood estimator $\hat{\sigma}^2$ is the unadjusted sample variance that is biased since $\mathbf{E}[\hat{\sigma}^2] = \frac{n-1}{n}\sigma^2$.

Appendix B

Calculations for Sequential Decoding

B.1 The Fano Metric for Gaussian channel

Assume v_i is the information bit, r_i is the corresponding received symbol, z_i is the additive white Gaussian noise $\mathcal{N}(0, \sigma^2)$, where $r_i = v_i + z_i$. For BPSK signalling with information symbol $v_i \in \{+\mu, -\mu\}$, the channel transition probability can be expressed as

$$f(r_i|v_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(r_i - v_i)^2}{2\sigma^2}}, \tag{B.1}$$

and the probability of the received bit is

$$f(r_i) = P(v_i = \mu) f(r_i|v_i = \mu) + P(v_i = -\mu) f(r_i|v_i = -\mu) \tag{B.2}$$

$$= \frac{1}{2}\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(r_i-\mu)^2}{2\sigma^2}} + \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(r_i+\mu)^2}{2\sigma^2}}\right). \tag{B.3}$$

Recall equation (5.1), Fano metric has the form

$$M(r_i|v_i) = \log_2\left(\frac{f(r_i|v_i)}{f(r_i)}\right) - R. \tag{B.4}$$

For the convenience of calculation, assume $a = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(r_i-\mu)^2}{2\sigma^2}}$, $b = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(r_i+\mu)^2}{2\sigma^2}}$, then (B.4) can be rewritten as

$$M(r_i|v_i) = \log_2\left(\frac{a}{1/2(a+b)}\right) - R. \tag{B.5}$$

B.2 The Expected Cost of the Correct Path

Let $d_{tran,i}$ denote the distance from the branch in depth i to the received symbols in depth i. Let $d_{hard,i}$ denote the distance from the received symbols in depth i to the hard decision of it. For variable bias metric, the cost of a branch in depth i is measure by

$$c_i = d_{tran,i} - d_{hard,i}, \tag{B.6}$$

where the distance refers to the squared Euclidean distance for variable bias metric. Since $d_{tran,i} \geq d_{hard,i}$, there is $c_i \geq 0$, and for a codeword identical to the hard decision result, the value of $c_i \geq 0$ is 0.

Assume x is the information symbol in transmitter, y is the corresponding received symbol, z is the additive white Gaussian noise, where $y = x + z$. For BPSK modulation with information symbol $x \in \{+\mu, -\mu\}$, without loss of generality, consider the information symbol $x = +\mu$ and received symbol $y = x + z$, where $z \sim \mathcal{N}(0, \sigma^2)$, then the probability density function of the received symbol y is

$$f_Y(y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu)^2}{2\sigma^2}}. \tag{B.7}$$

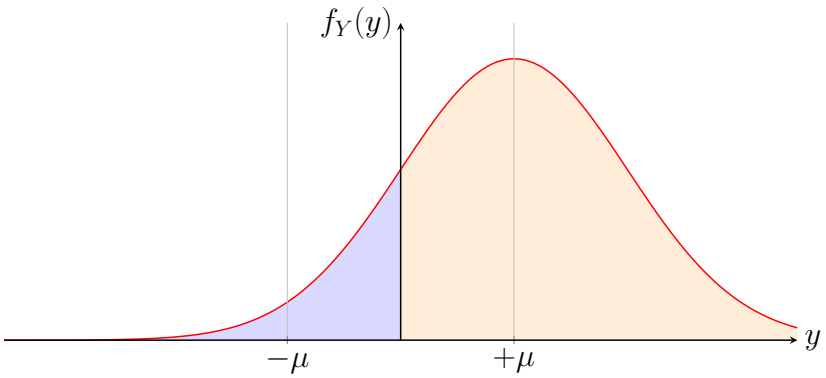

Figure B.1: Graph of the distribution of input $+\mu$ with noise $\mathcal{N}(0, 1)$

When the received symbol y is positive, the hard decision of y is $+\mu$, thus $c = d_{tran} - d_{hard} = \|y - \mu\|^2 - \|y - \mu\|^2 = 0$; when y is negative, the hard decision of y is $-\mu$, thus $c = d_{tran} - d_{hard} = \|y - \mu\|^2 - \|y + \mu\|^2$. Therefore, The distribution of c is

$$c(y) = \begin{cases} 0 & \text{if } y > 0 \\ -4\mu y & \text{if } y \leq 0. \end{cases} \tag{B.8}$$

Thus the expected value of c can be derived as

$$\overline{c} = E[c(Y)] = \int_{-\infty}^{+\infty} c(y) f_Y(y) \mathrm{d}y \tag{B.9a}$$

$$= \int_{-\infty}^{0} -4x\mu \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} \tag{B.9b}$$

$$= -\frac{4\sqrt{2}\mu\sigma}{\sqrt{\pi}} \int_{-\infty}^{-\frac{\mu}{\sqrt{2}\sigma}} w e^{-w^2} \mathrm{d}w - \frac{4\mu^2}{\sqrt{\pi}} \int_{-\infty}^{-\frac{\mu}{\sqrt{2}\sigma}} e^{-w^2} \mathrm{d}w \tag{B.9c}$$

$$= \frac{2\sqrt{2}\mu\sigma}{\sqrt{\pi}} \left(e^{-w^2} \right) \Big|_{-\infty}^{-\frac{\mu}{\sqrt{2}\sigma}} - 2\mu^2 \int_{\frac{\mu}{\sqrt{2}\sigma}}^{\infty} \frac{2}{\sqrt{\pi}} e^{-w^2} \mathrm{d}w \tag{B.9d}$$

$$= \frac{2\sqrt{2}\mu\sigma}{\sqrt{\pi}} e^{-\frac{\mu^2}{2\sigma^2}} - 2\mu^2 \mathrm{erfc}\left(\frac{\mu}{\sqrt{2}\sigma} \right), \tag{B.9e}$$

which is the expected cost per bit of the correct path with BPSK modulation. By replacing the complementary error function in B.9 with its upper and lower bound [58]

$$\frac{2e^{-x^2}}{\sqrt{\pi}\left(x + \sqrt{x^2+2}\right)} < \mathrm{erfc}(x) \leq \frac{2e^{-x^2}}{\sqrt{\pi}\left(x + \sqrt{x^2 + \frac{4}{\pi}}\right)}, \tag{B.10}$$

the corresponding upper and lower bound of $\overline{c}$ can be derived as

$$\frac{2\mu^2}{\sqrt{\pi}} e^{-\xi^2} \left(\frac{1}{\xi} - \frac{2}{\xi + \sqrt{\xi^2 + \frac{4}{\pi}}} \right) \leq \overline{c} < \frac{2\mu^2}{\sqrt{\pi}} e^{-\xi^2} \left(\frac{1}{\xi} - \frac{2}{\xi + \sqrt{\xi^2 + 2}} \right), \tag{B.11}$$

where $\xi = \frac{\mu}{\sigma\sqrt{2}}$. Assume $\mathcal{Y} = (y_1, y_2, y_3, ..., y_N)$ is the received sequence with length N, since the expected value of sums is equal to the sum of expected values for any set of random variables due to the independence, the expected cost of codeword $\mathcal{Y}$ is $N\overline{c}$. Therefore, the expression of $\overline{c}_N$ is

$$\overline{c}_N = N\overline{c} = N \left(\frac{2\sqrt{2}\mu\sigma}{\sqrt{\pi}} e^{-\frac{\mu^2}{2\sigma^2}} - 2\mu^2 \mathrm{erfc}\left(\frac{\mu}{\sqrt{2}\sigma} \right) \right), \tag{B.12}$$

which is Then the expected cost of the correct path with BPSK modulation of $(+\mu, -\mu)$ over AWGN channel with noise $\mathcal{N}(0, \sigma^2)$. The upper and lower bound of $\overline{c}_N$ are

$$\frac{2N\mu^2}{\sqrt{\pi}} e^{-\xi^2} \left(\frac{1}{\xi} - \frac{2}{\xi + \sqrt{\xi^2 + \frac{4}{\pi}}} \right) \leq \overline{c}_N < \frac{2N\mu^2}{\sqrt{\pi}} e^{-\xi^2} \left(\frac{1}{\xi} - \frac{2}{\xi + \sqrt{\xi^2 + 2}} \right). \tag{B.13}$$

B.3 The Expected Cumulative Cost of a Random Path

Let $\mathcal{X} = (x_1, x_2, x_3, ..., x_N)$ denote the information bit sequence in transmitter, $\mathcal{Y} = (y_1, y_2, y_3, ..., y_N)$ denote the sequence received by receiver, $\mathcal{Z} = (z_1, z_2, z_3, ..., z_N)$ denote the additive white Gaussian noise that is i.i.d., where $y_j = x_j + z_j$, $1 \leq j \leq N$.

Let $\mathcal{X}(i) = (x_1, x_2, x_3, ..., x_i)$ denote the information bit sequence truncated at depth i, then without loss of generality, let $x_1 = x_2 = x_3 = ... = x_i = +\mu$ (i.e.,$\mathcal{X}(i) = (\underbrace{+\mu, +\mu, +\mu, ..., +\mu}_{\text{i bits}})$). With noise $z_j = \mathcal{N}(0, \sigma^2)$, $1 \leq j \leq i$, the distribution of the received sequence is

$$Y_1, Y_2, Y_3, ..., Y_i \overset{iid}{\sim} \mathcal{N}(\mu, \sigma^2). \tag{B.14}$$

Since $Y_1, Y_2, Y_3, ..., Y_N$ are independent of each other, the distance from the branch in depth i to the received symbols in depth i, $d_{tran,i}$, can be can be described by chi-squared distribution. Without loss of generality, we consider codeword with depth N, then $d_{tran,N} = \sum_{j=1}^{N} (Y_j - \mu)^2$ is distributed according to the chi-squared distribution with N degrees of freedom. The PDF of $d_{tran,N}$ is

$$f(y; N) = \begin{cases} \frac{1}{2^{N/2}\sigma^N \Gamma(\frac{N}{2})} e^{-\frac{y}{2\sigma^2}} y^{\frac{N}{2}-1} & \text{if } y > 0 \\ 0 & \text{if } y \leq 0, \end{cases} \tag{B.15}$$

where $E[Y] = k\sigma^2$ and $VAR[Y] = 2k\sigma^4$.

According to (B.9e) in Appendix B.2, the expression for the expected squared Euclidean distance from the received symbols of length N to the hard decision of it can be derived as

$$d_{hard,N} = d_{tran,N} - \overline{c}_N = N\sigma^2 - N\frac{2\sqrt{2}\mu\sigma}{\sqrt{\pi}} e^{-\frac{\mu^2}{2\sigma^2}} + 2N\mu^2 \text{erfc}\left(\frac{\mu}{\sqrt{2}\sigma}\right), \tag{B.16}$$

and the expected cost for a random codeword of length N has the expression

$$\overline{c}_{rand,N} = 2N\sigma^2 + N\frac{2\sqrt{2}\mu\sigma}{\sqrt{\pi}} e^{-\frac{\mu^2}{2\sigma^2}} - 2N\mu^2 \text{erfc}\left(\frac{\mu}{\sqrt{2}\sigma}\right). \tag{B.17}$$

Appendix C

Plots of Performance, Complexity and the Trade-offs

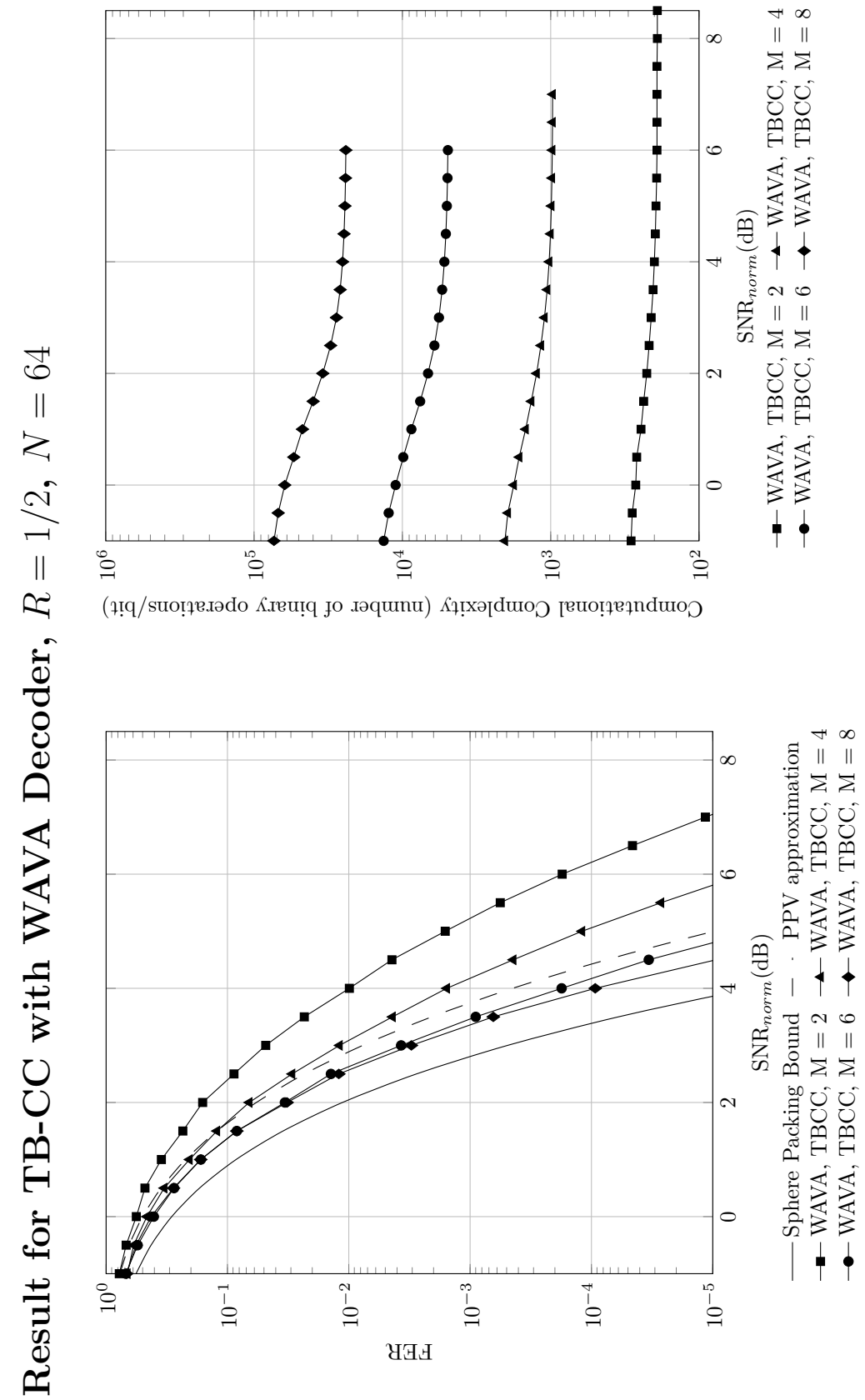

Figure C.1: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/2$, $N = 64$, maximum number of iterations = 4.

Figure C.2: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/2$, $N = 64$, maximum number of iterations = 4

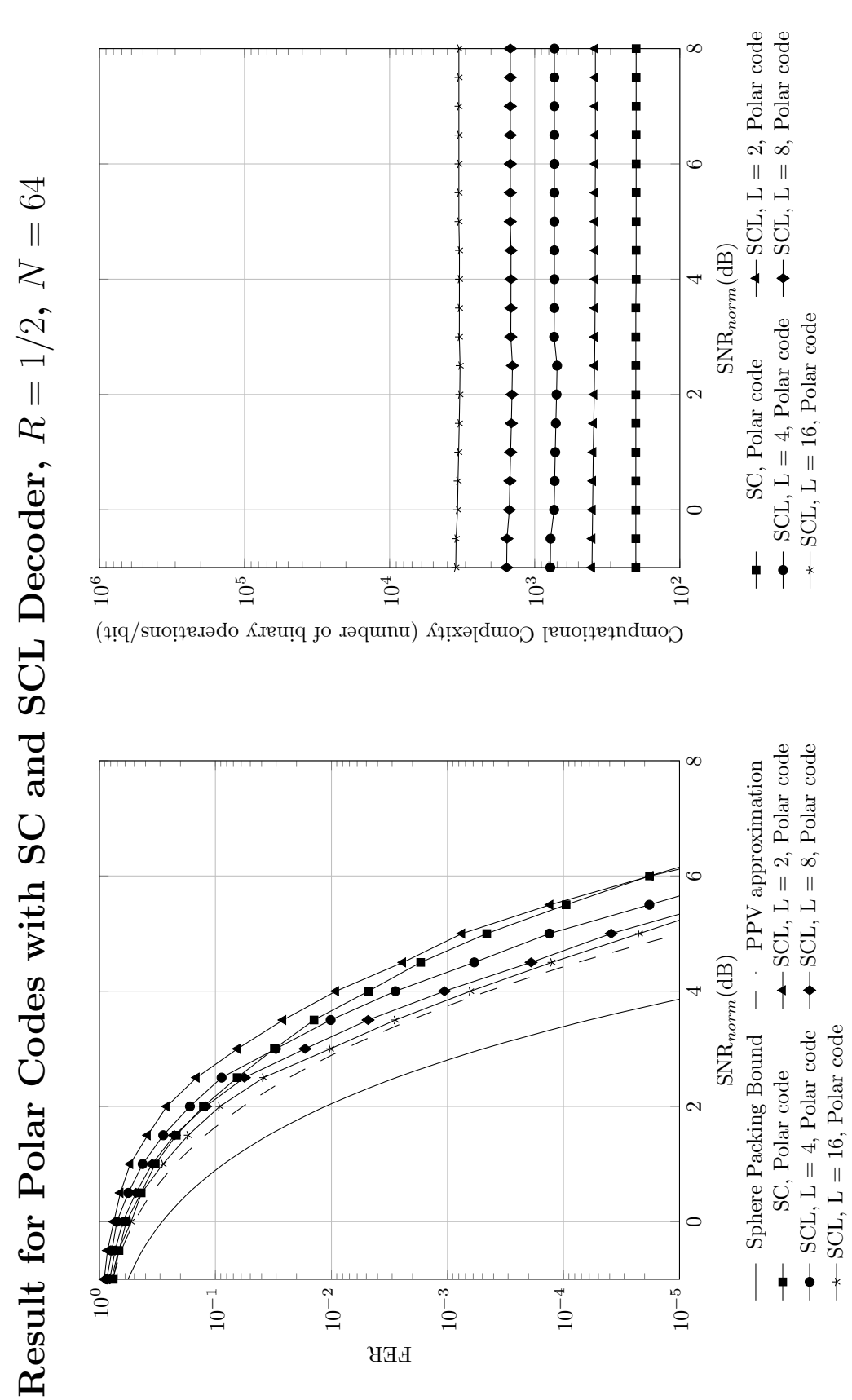

Figure C.3: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/2$, $N = 64$

Figure C.4: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/2$, $N = 64$

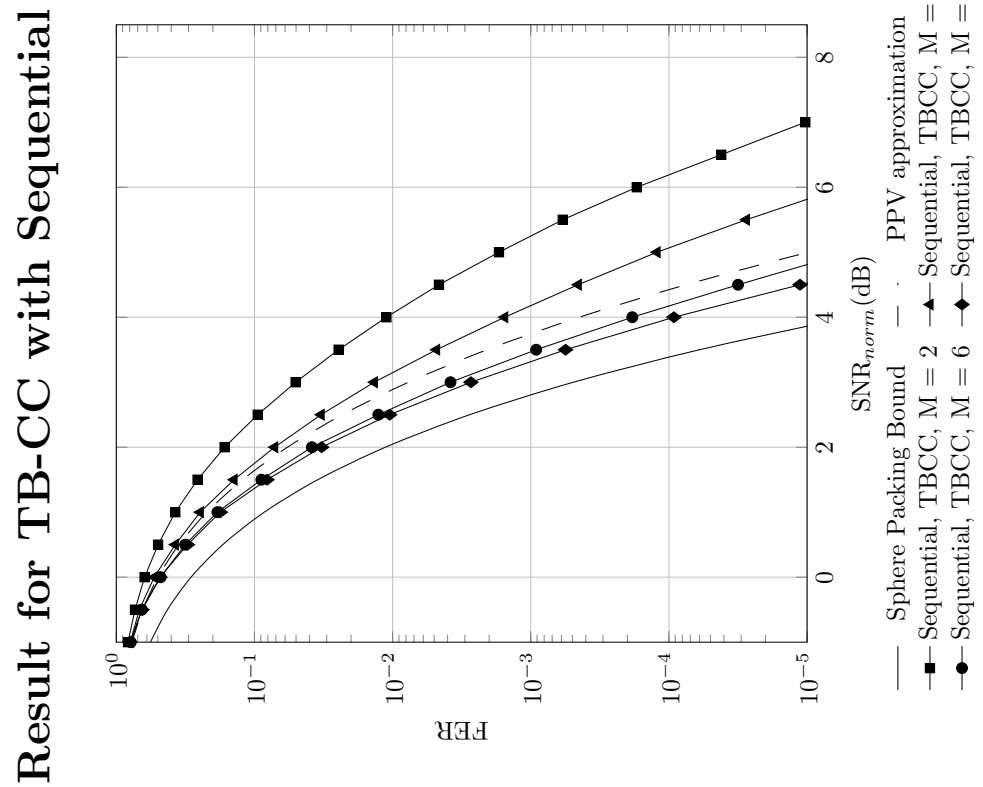

Figure C.5: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/2$, $N = 64$

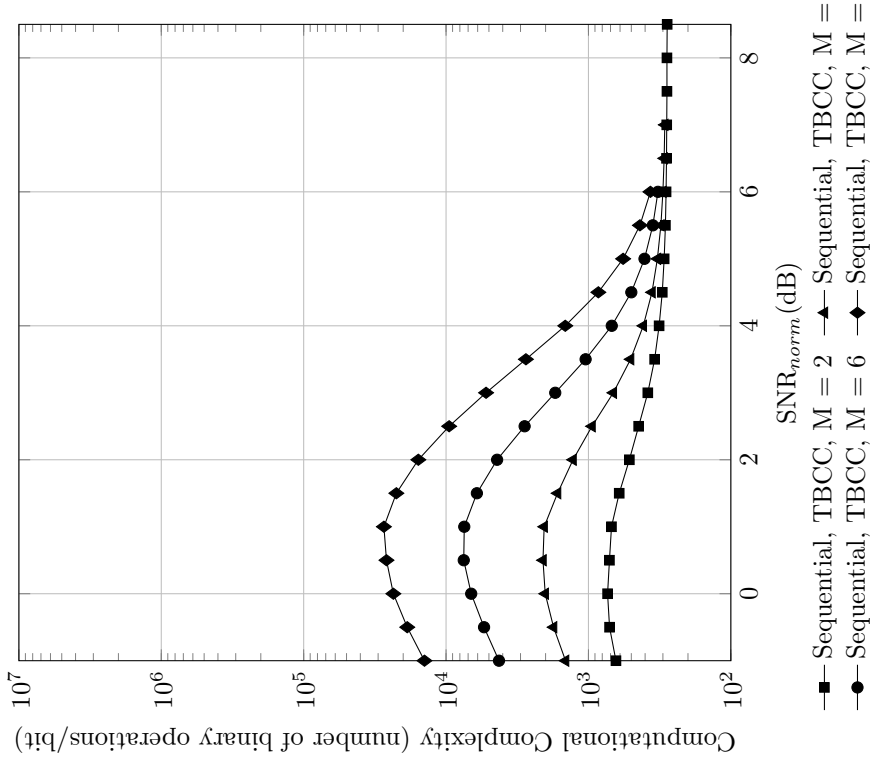

Figure C.6: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/2$, $N = 64$

Code imperfectness versus computational complexity for different codes at FER $= 10^{-2}$, $R = 1/2$, $N = 64$

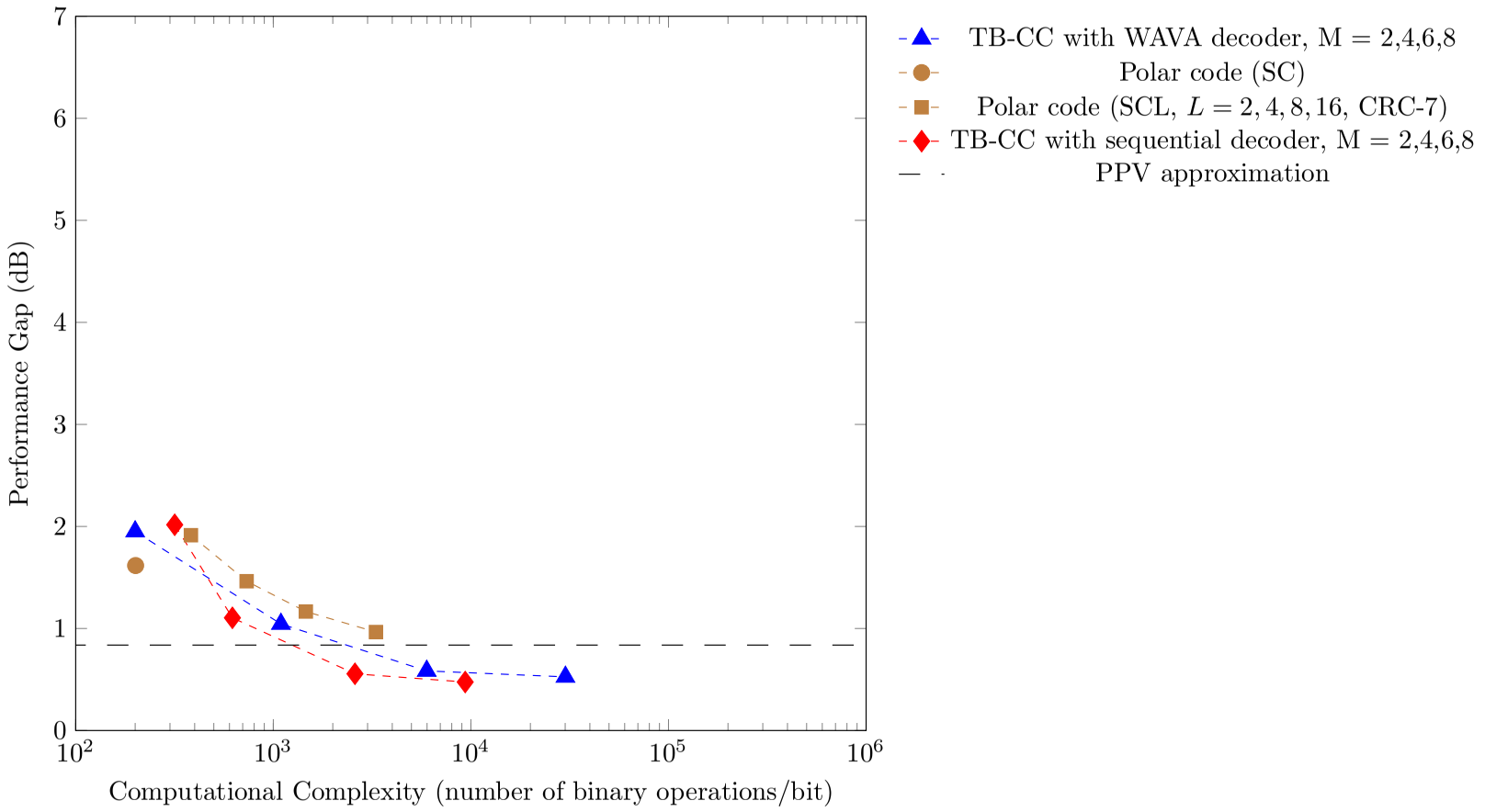

Figure C.7: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 1/2$, $N = 64$

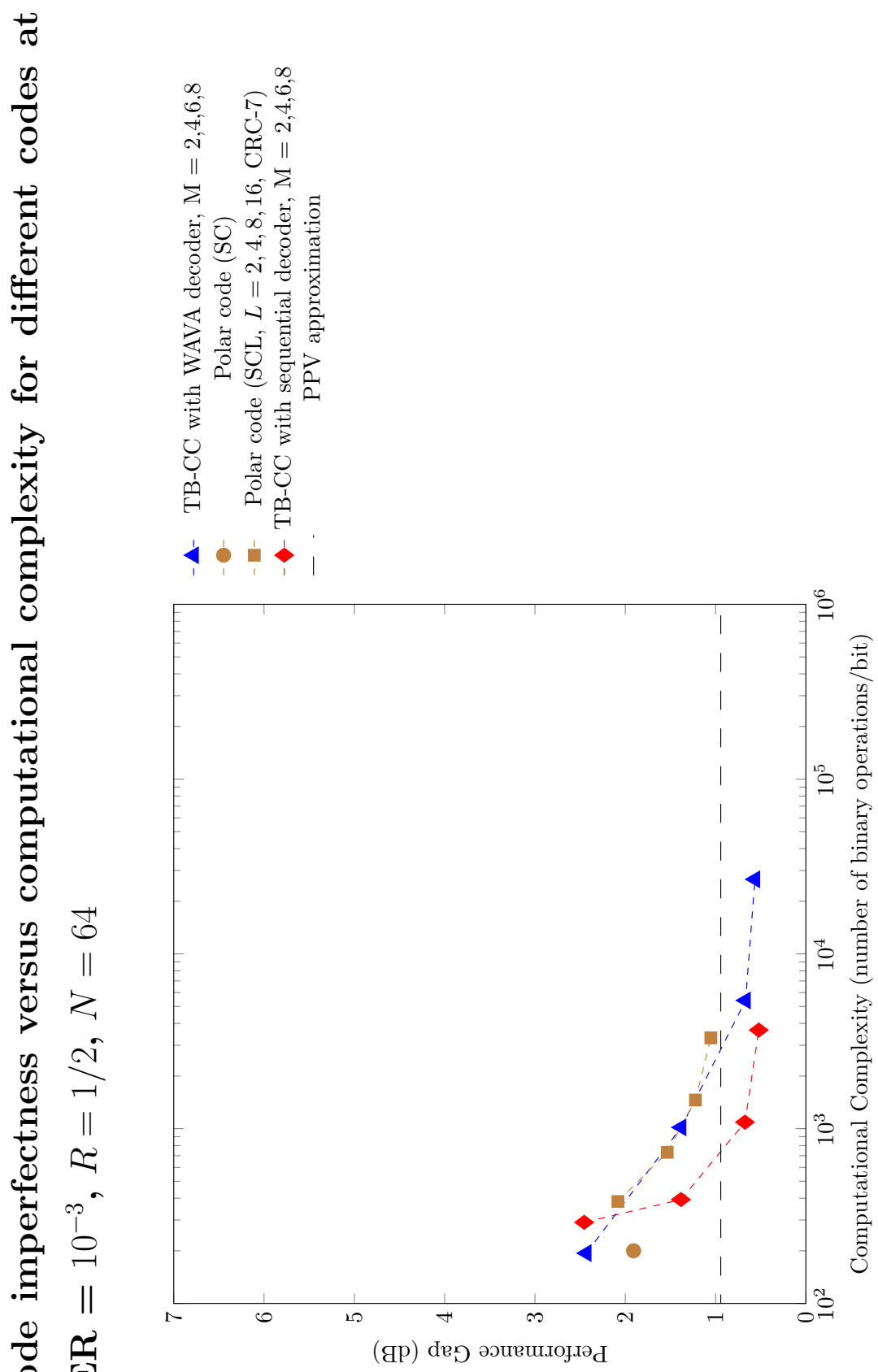

Figure C.8: Code imperfectness versus computational complexity at FER = 10^{-3} for different codes with $R = 1/2$, $N = 64$

Code imperfectness versus computational complexity for different codes at FER $= 10^{-4}$, $R = 1/2$, $N = 64$

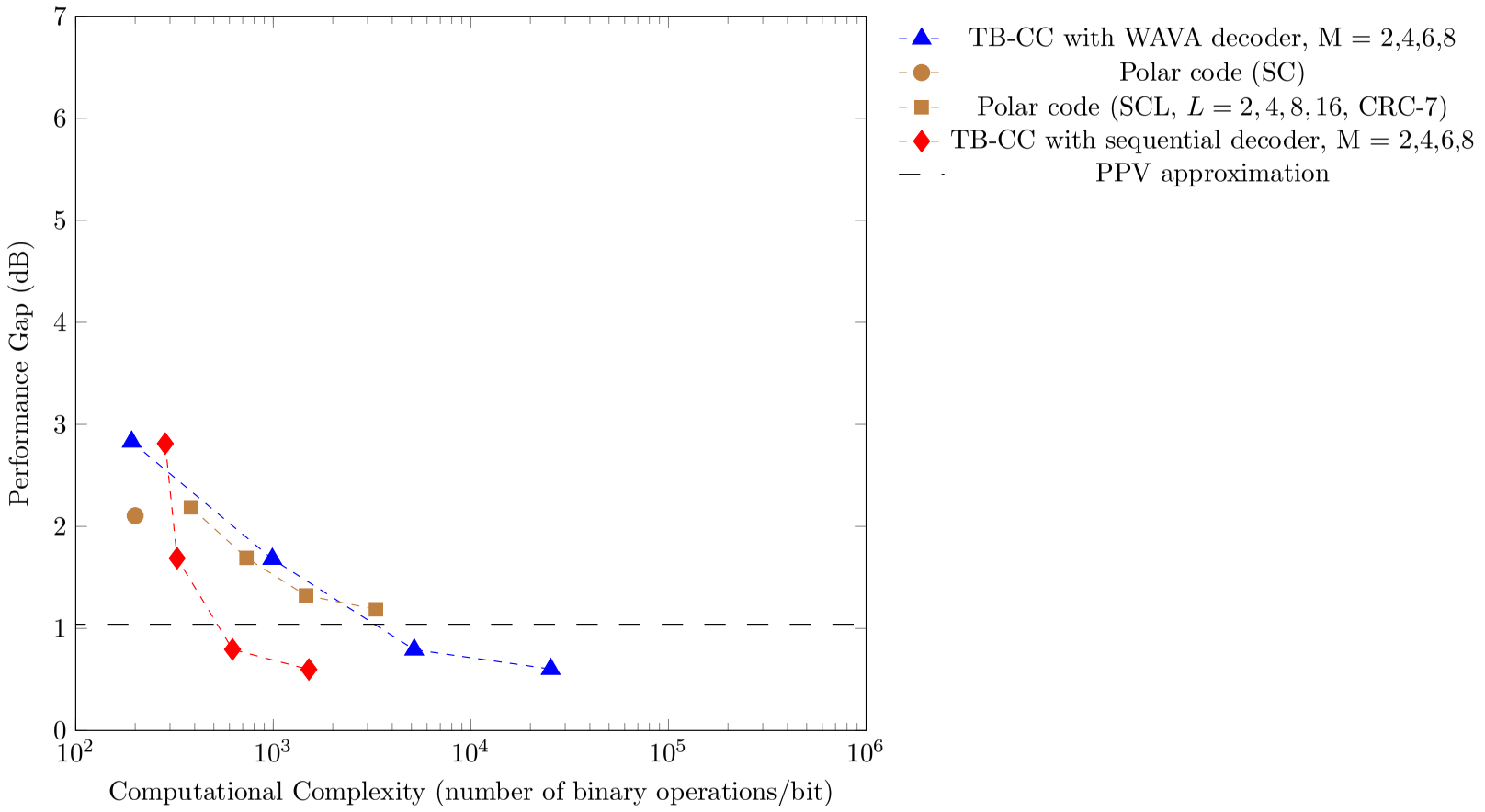

Figure C.9: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 1/2$, $N = 64$

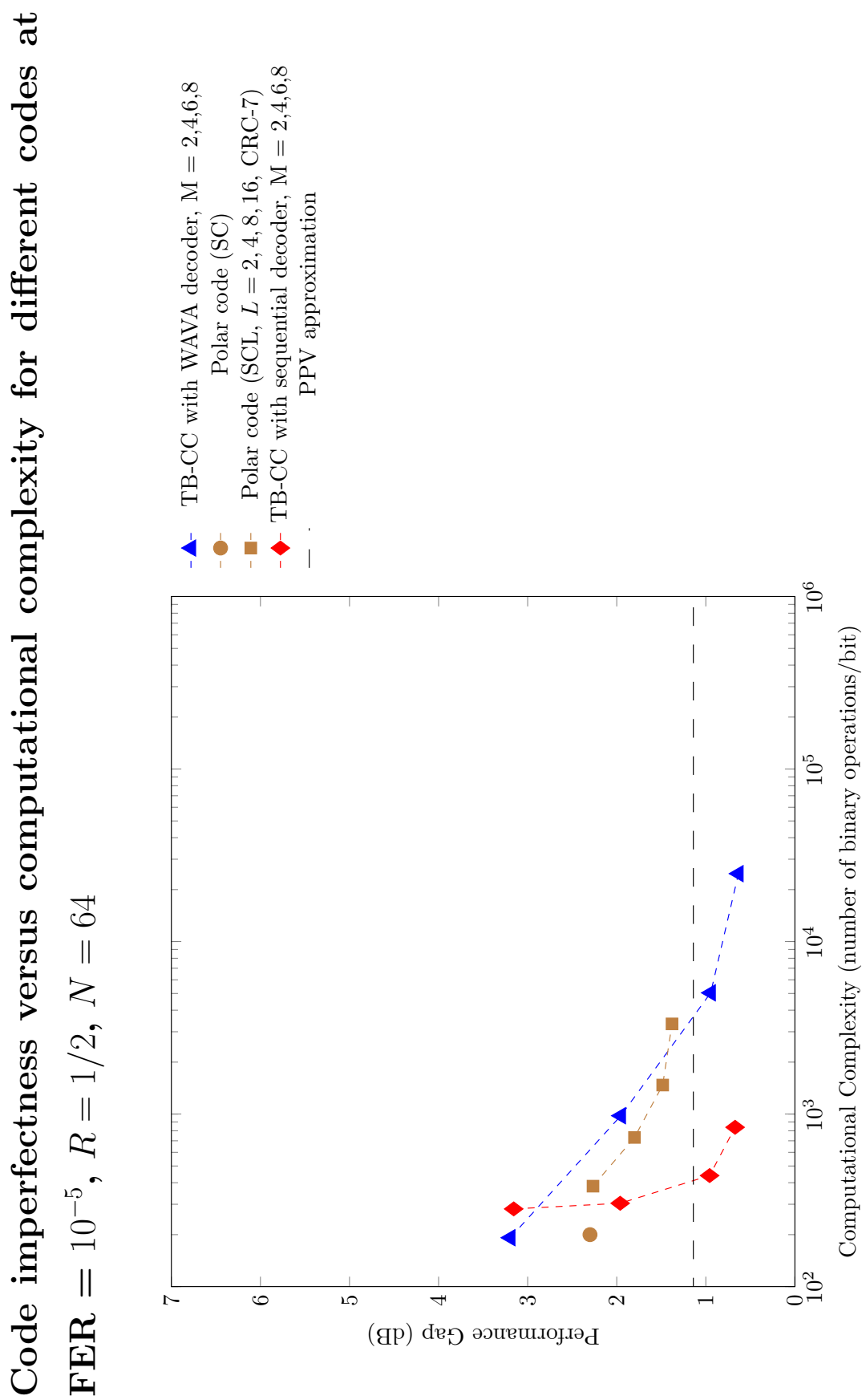

Figure C.10: Code imperfectness versus computational complexity at FER = 10^{-5} for different codes with $R = 1/2$, $N = 64$

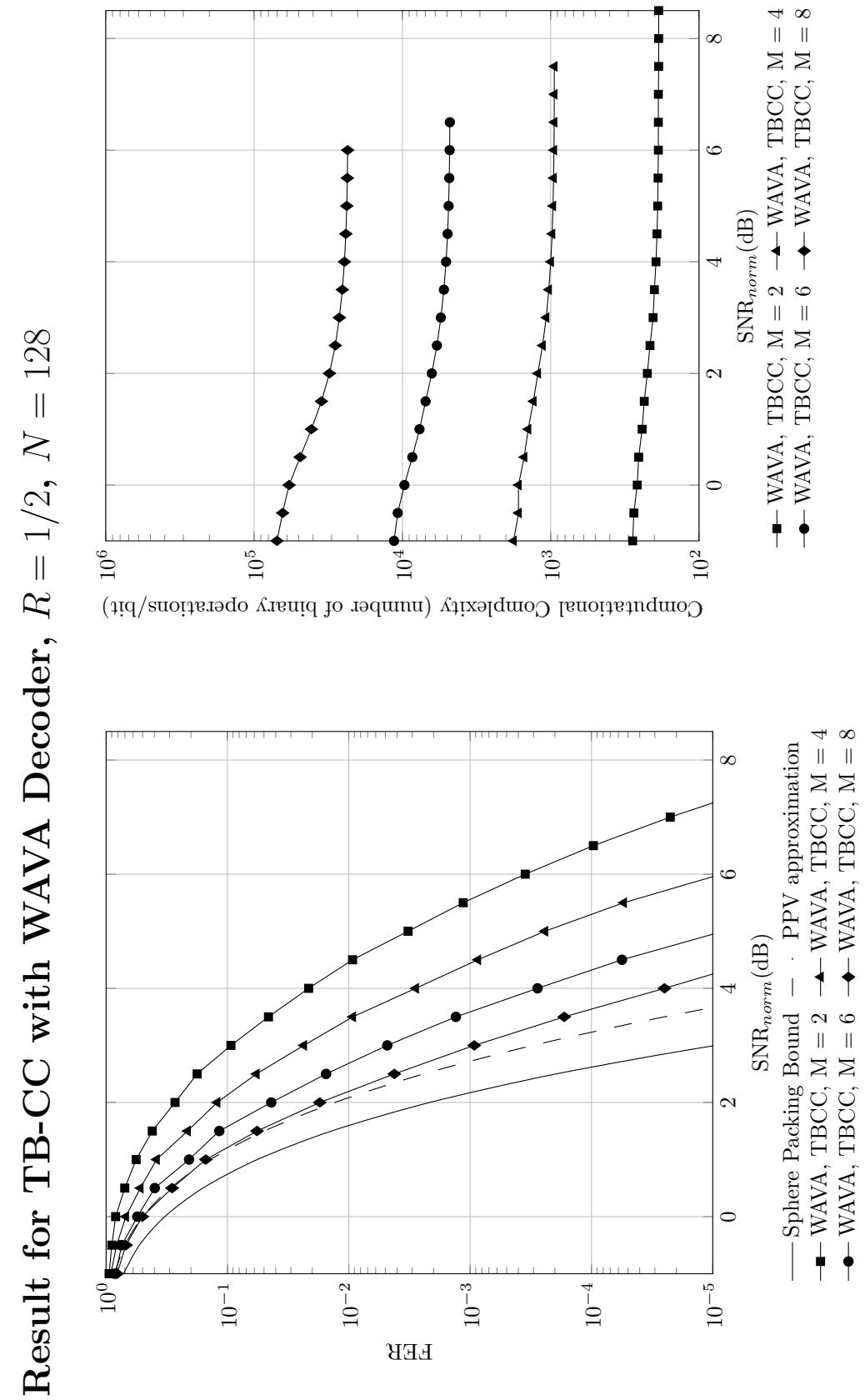

Figure C.11: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/2$, $N = 128$, maximum number of iterations = 4.

Figure C.12: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/2$, $N = 128$, maximum number of iterations = 4

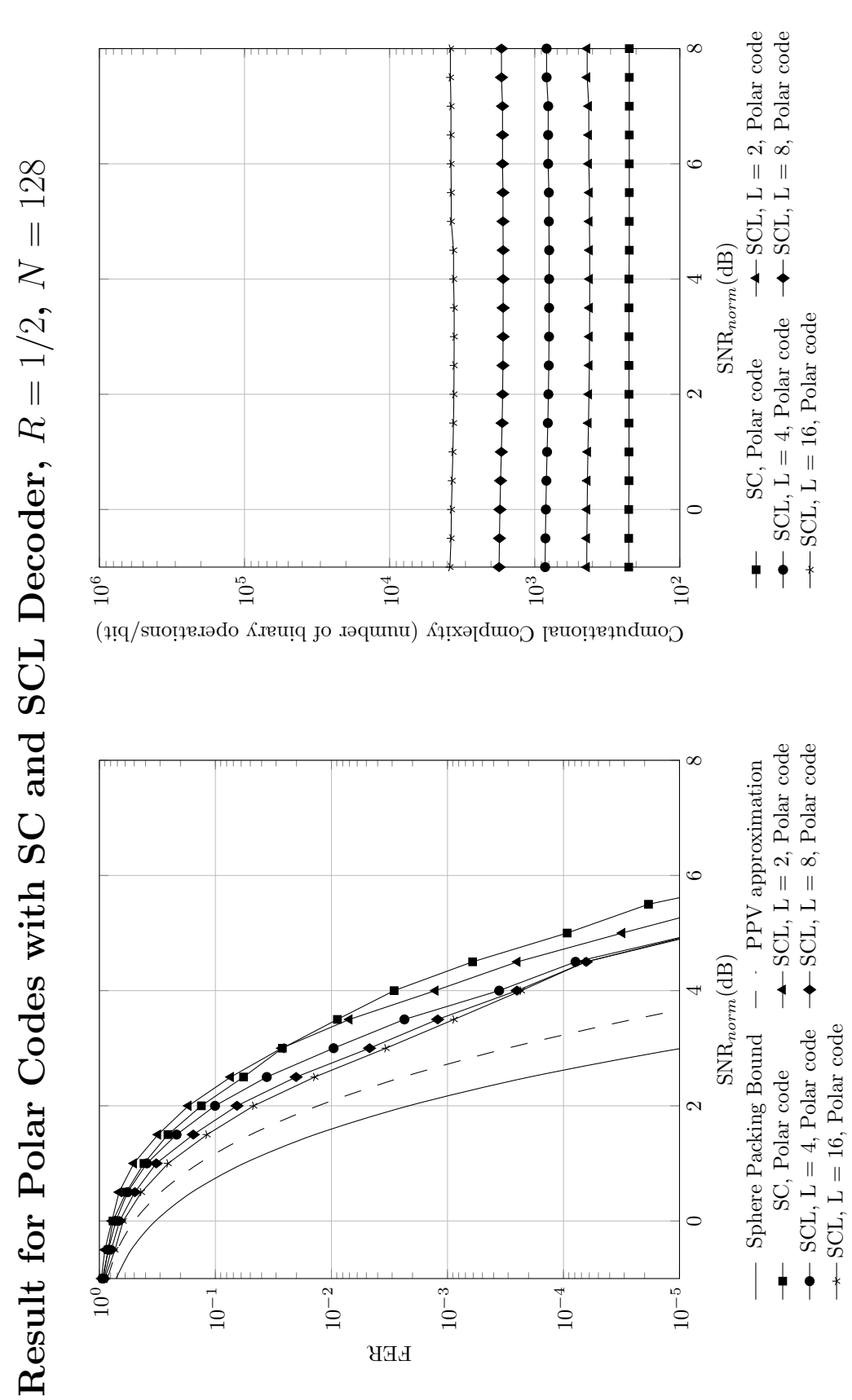

Figure C.13: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/2$, $N = 128$

Figure C.14: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/2$, $N = 128$

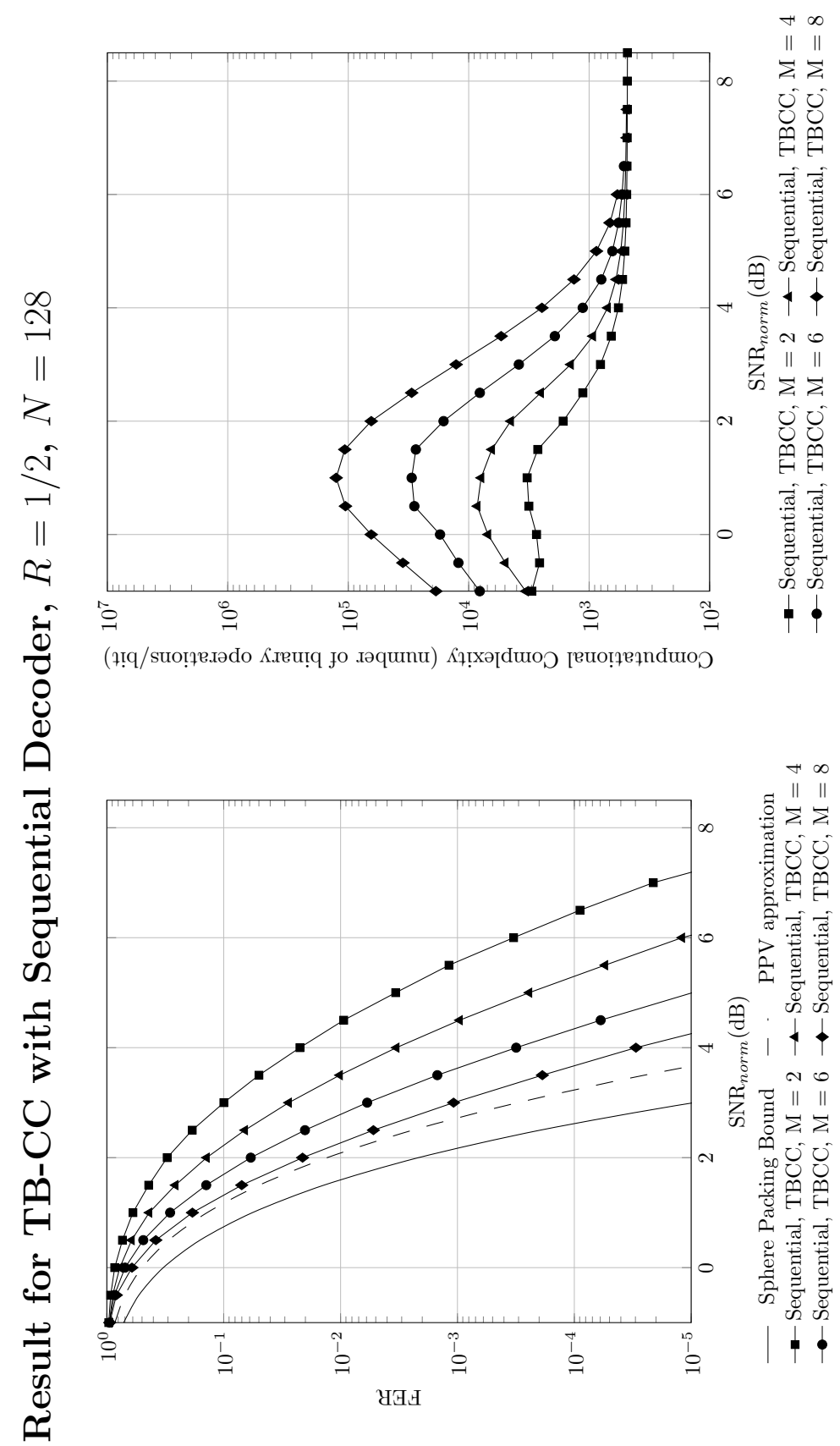

Figure C.15: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/2$, $N = 128$

Figure C.16: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/2$, $N = 128$

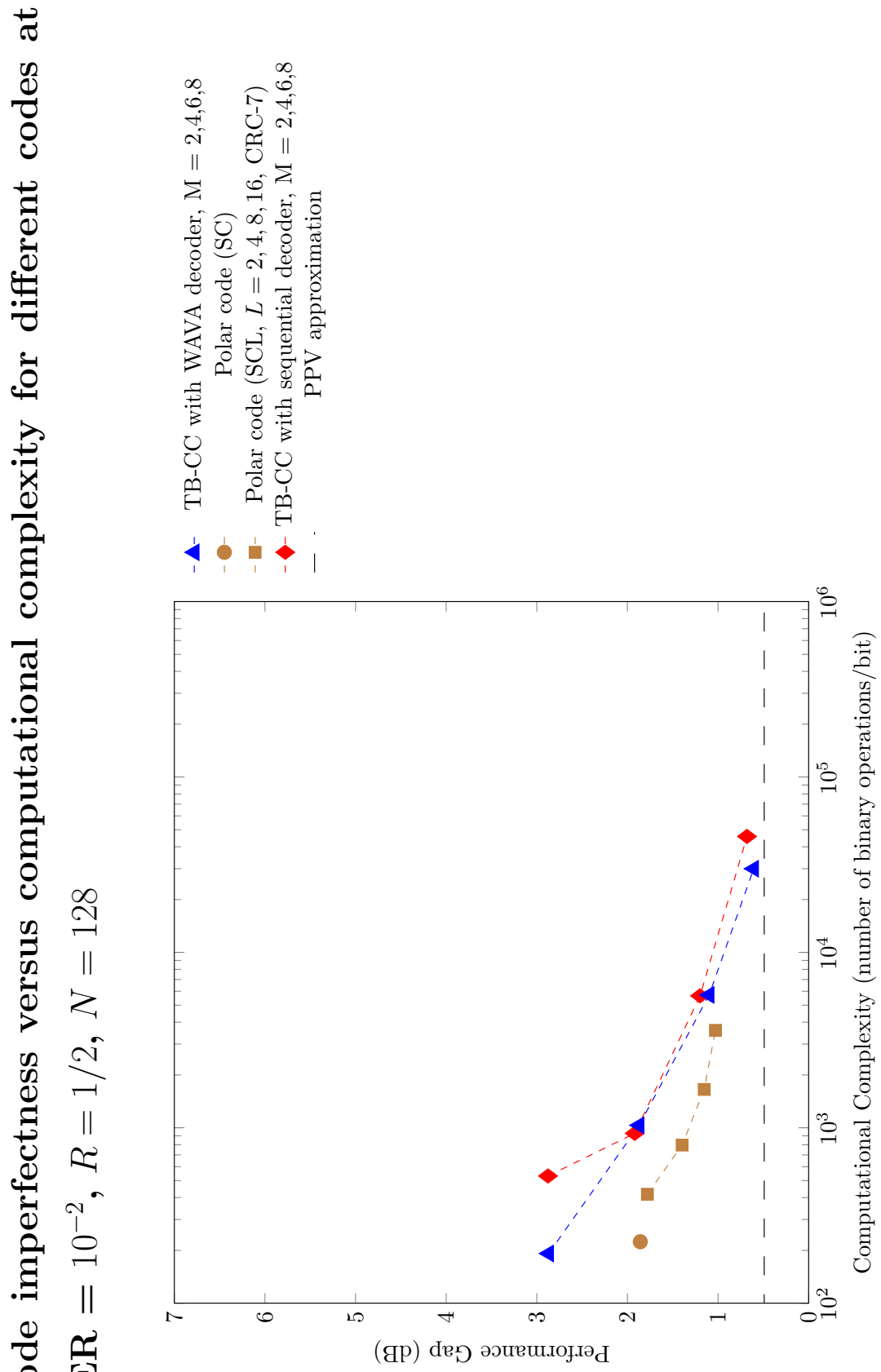

Figure C.17: Code imperfectness versus computational complexity at FER = 10^{-2} for different codes with $R = 1/2$, $N = 128$

Code imperfectness versus computational complexity for different codes at FER $= 10^{-3}$, $R = 1/2$, $N = 128$

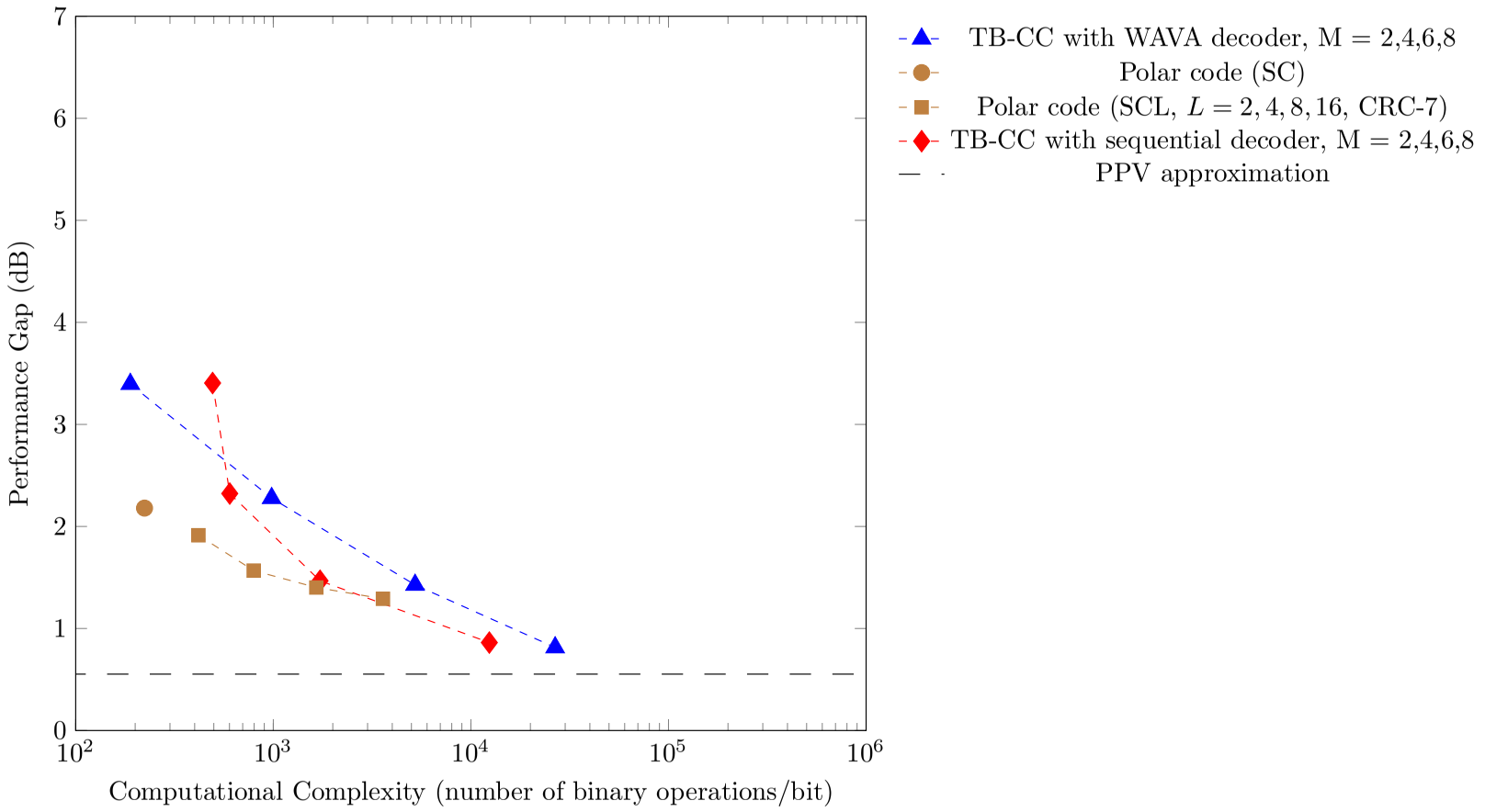

Figure C.18: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 1/2$, $N = 128$

Code imperfectness versus computational complexity for different codes at FER $= 10^{-4}$, $R = 1/2$, $N = 128$

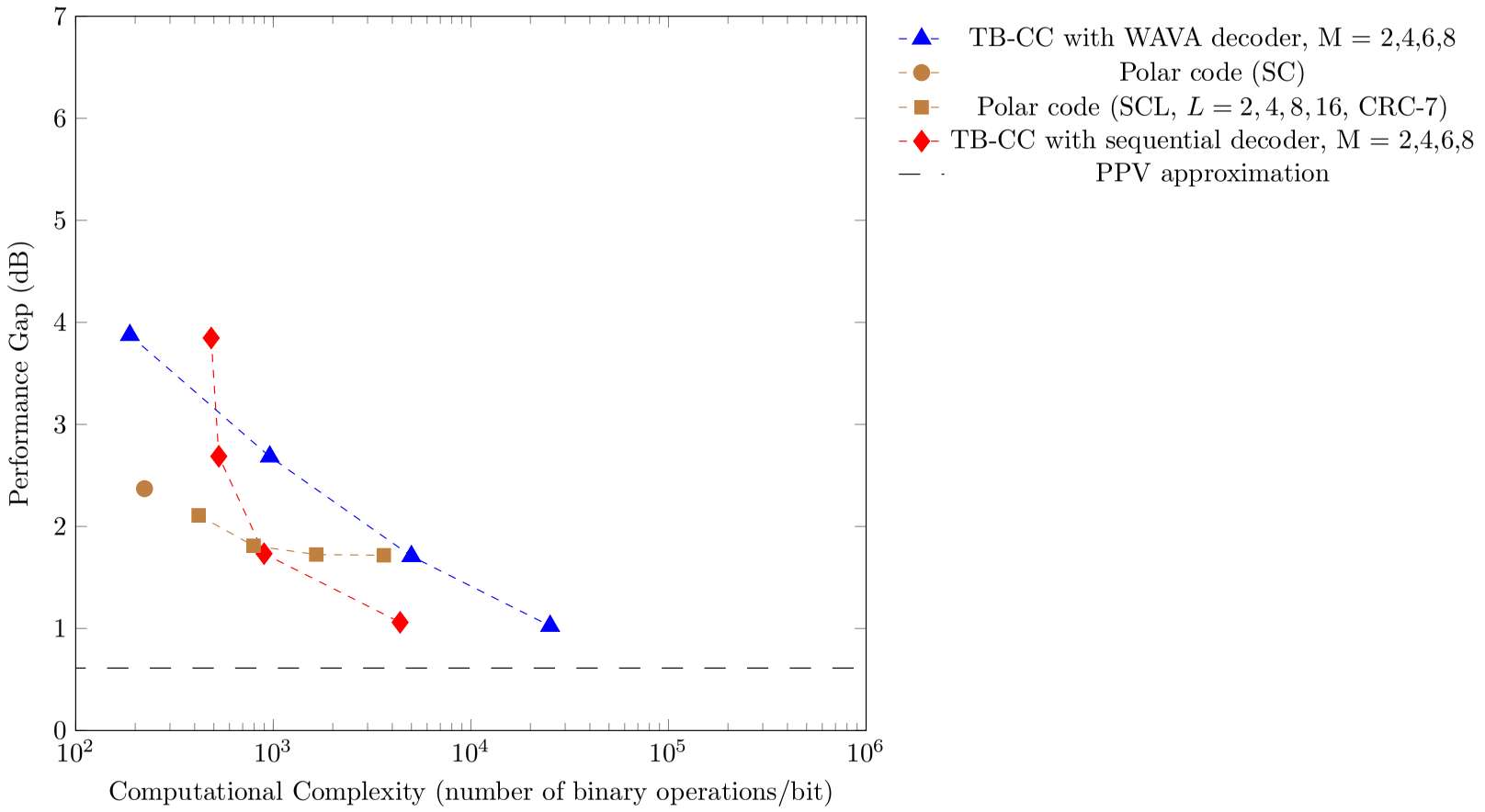

Figure C.19: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 1/2$, $N = 128$

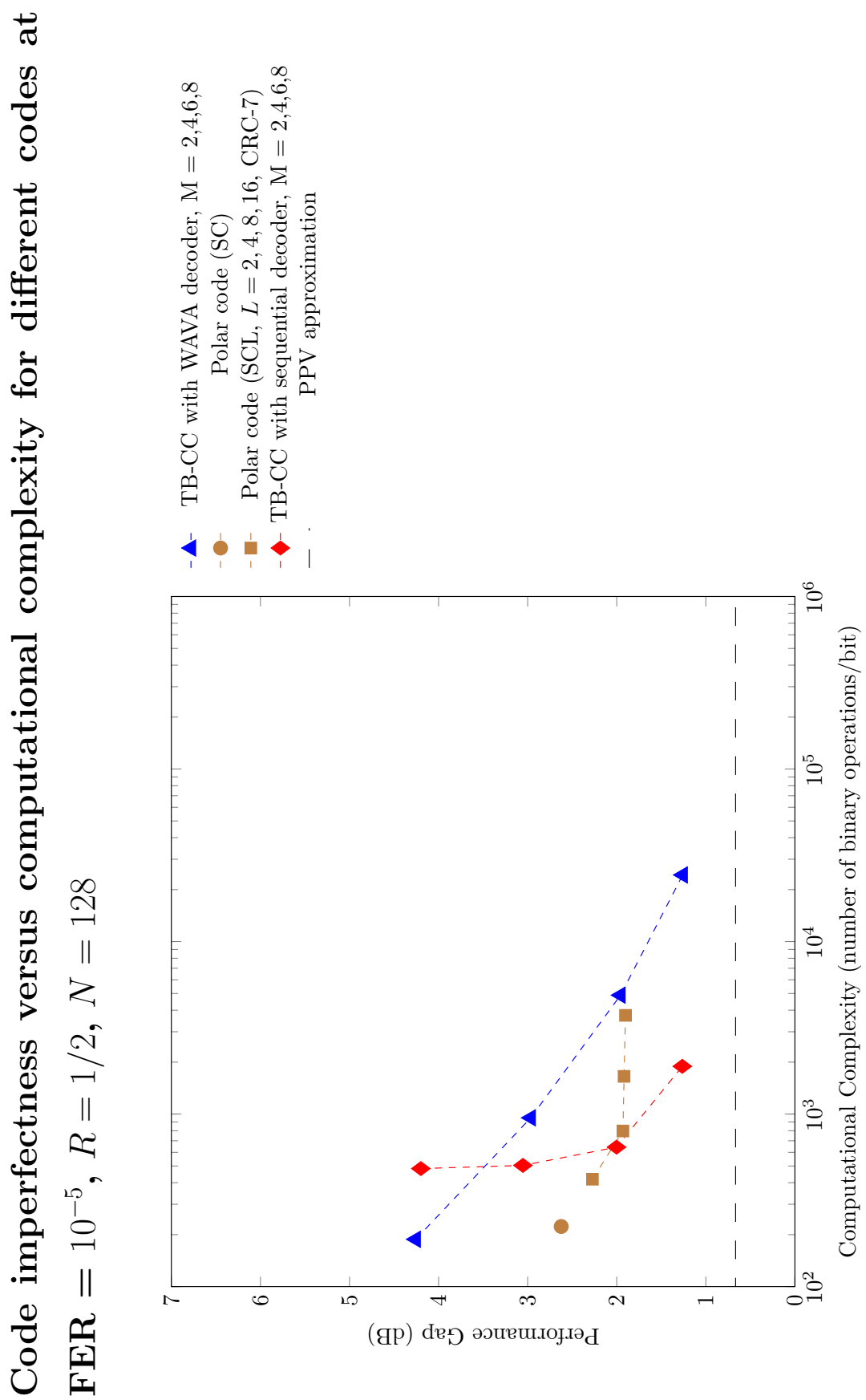

Figure C.20: Code imperfectness versus computational complexity at FER = 10^{-5} for different codes with $R = 1/2$, $N = 128$

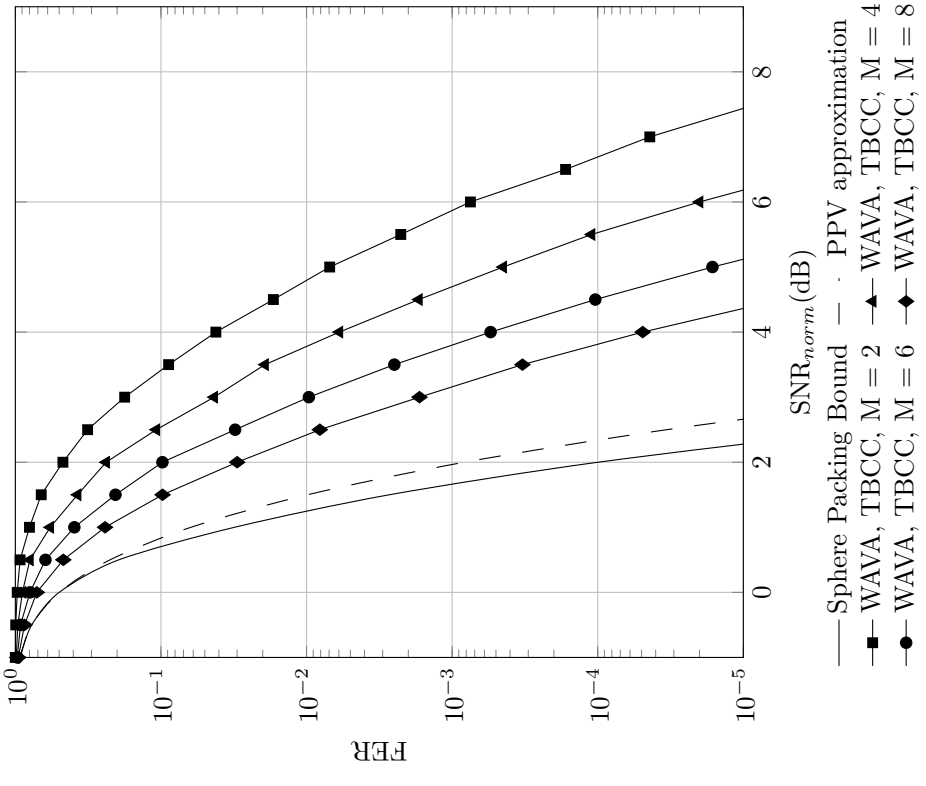

Figure C.21: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/2$, $N = 256$, maximum number of iterations = 4.

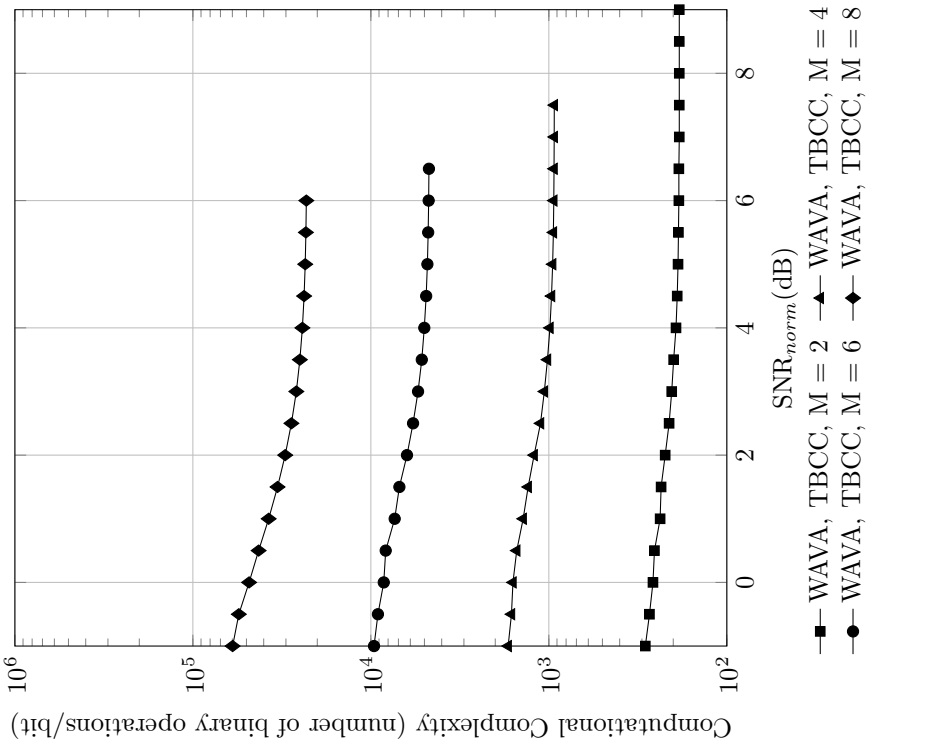

Figure C.22: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/2$, $N = 256$, maximum number of iterations = 4

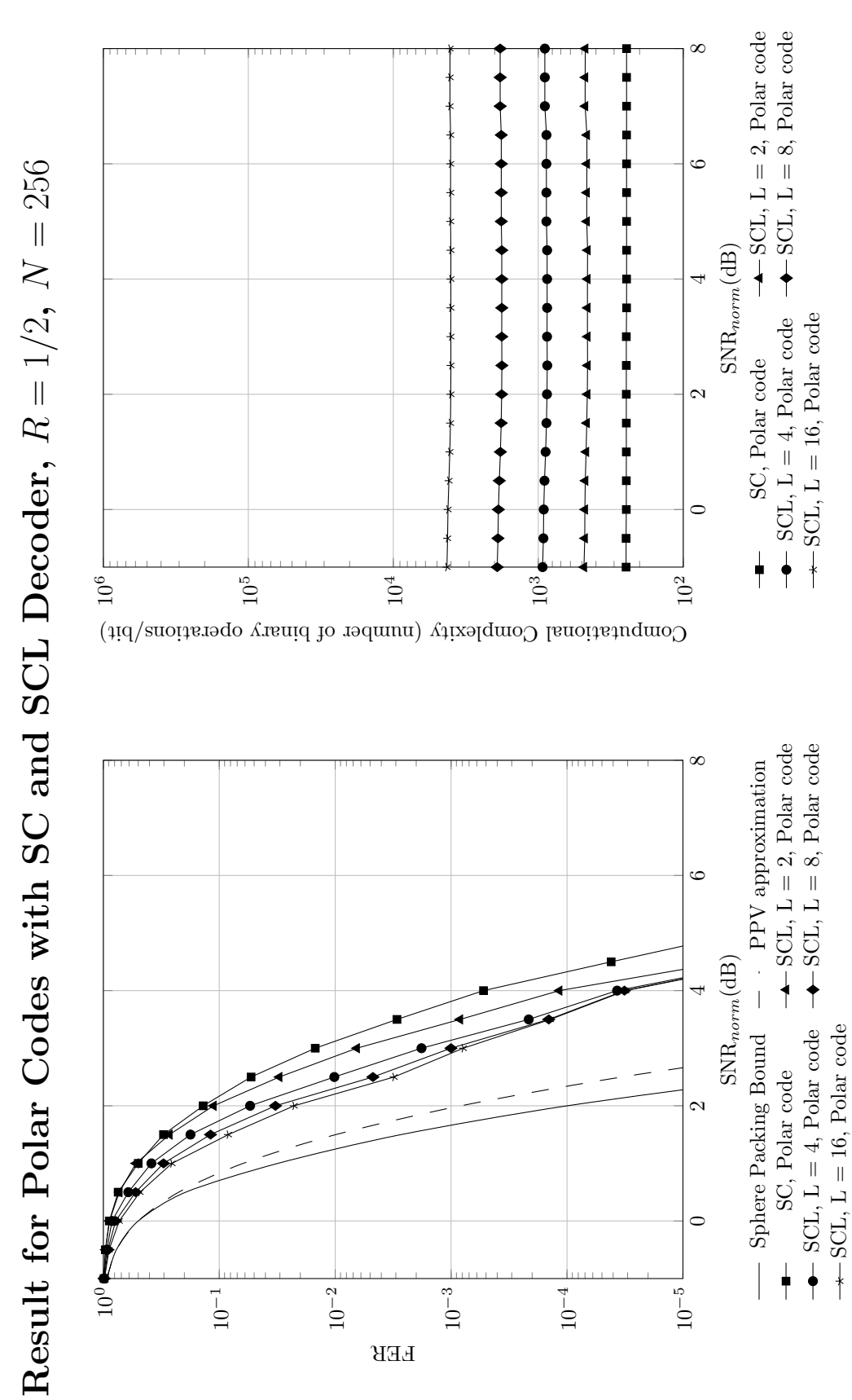

Figure C.23: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/2$, $N = 256$

Figure C.24: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/2$, $N = 256$

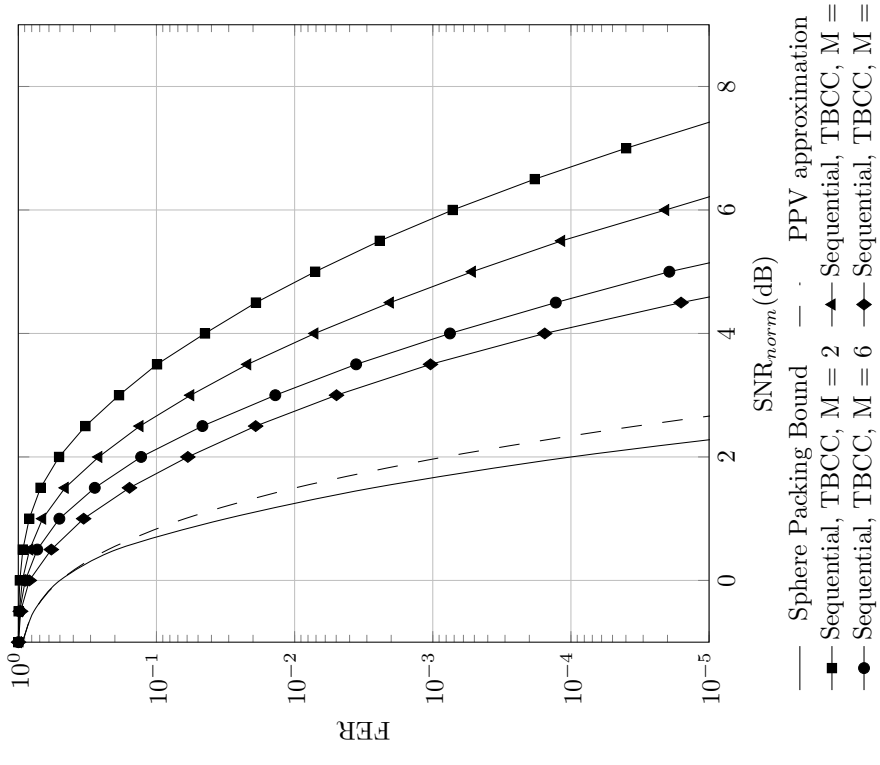

Figure C.25: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/2$, $N = 256$

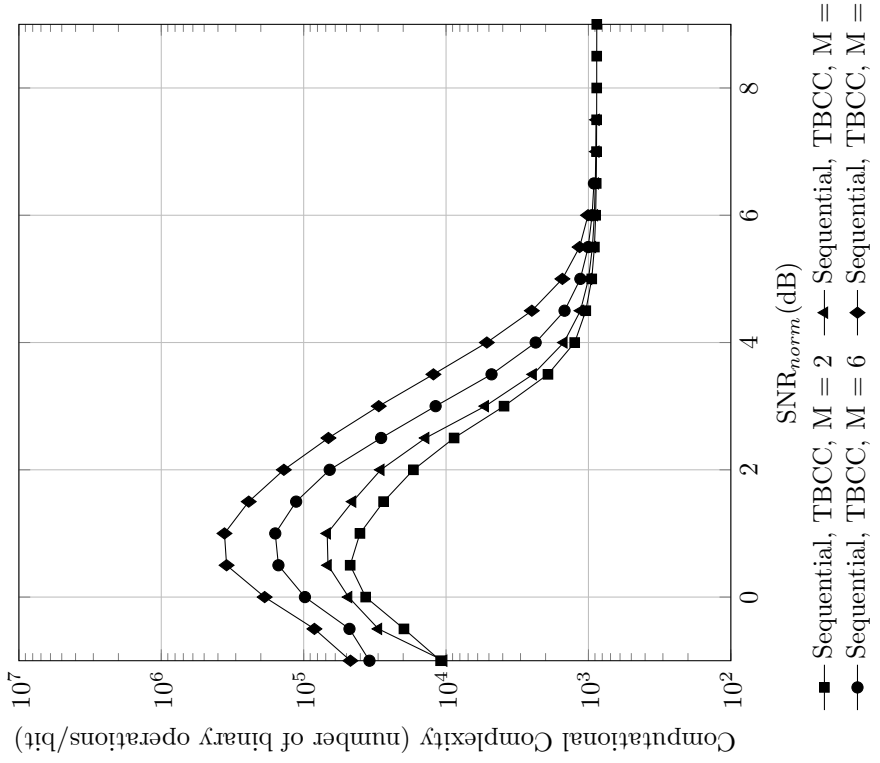

Figure C.26: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/2$, $N = 256$

Code imperfectness versus computational complexity for different codes at FER $= 10^{-2}$, $R = 1/2$, $N = 256$

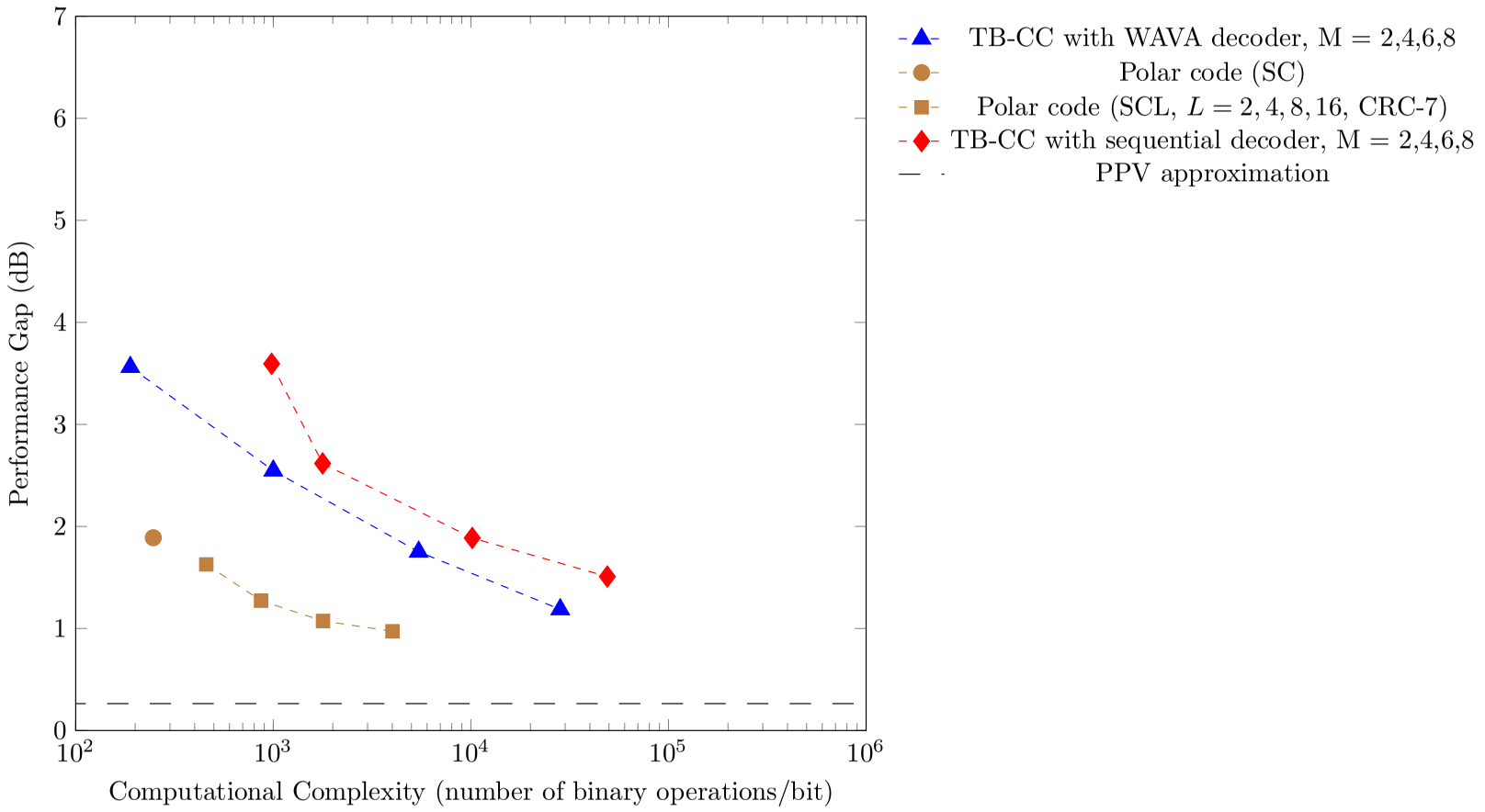

Figure C.27: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 1/2$, $N = 256$

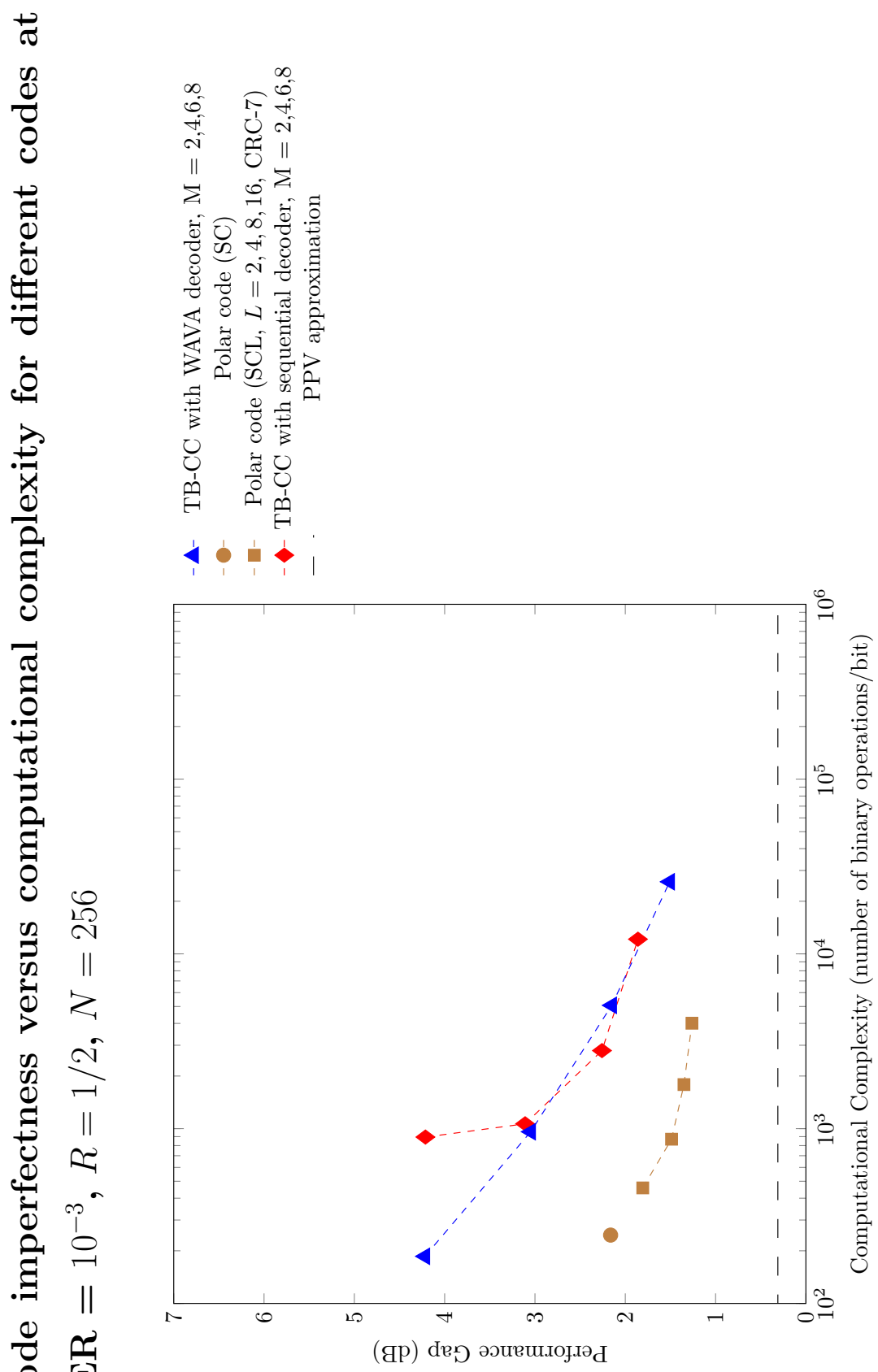

Figure C.28: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 1/2$, $N = 256$

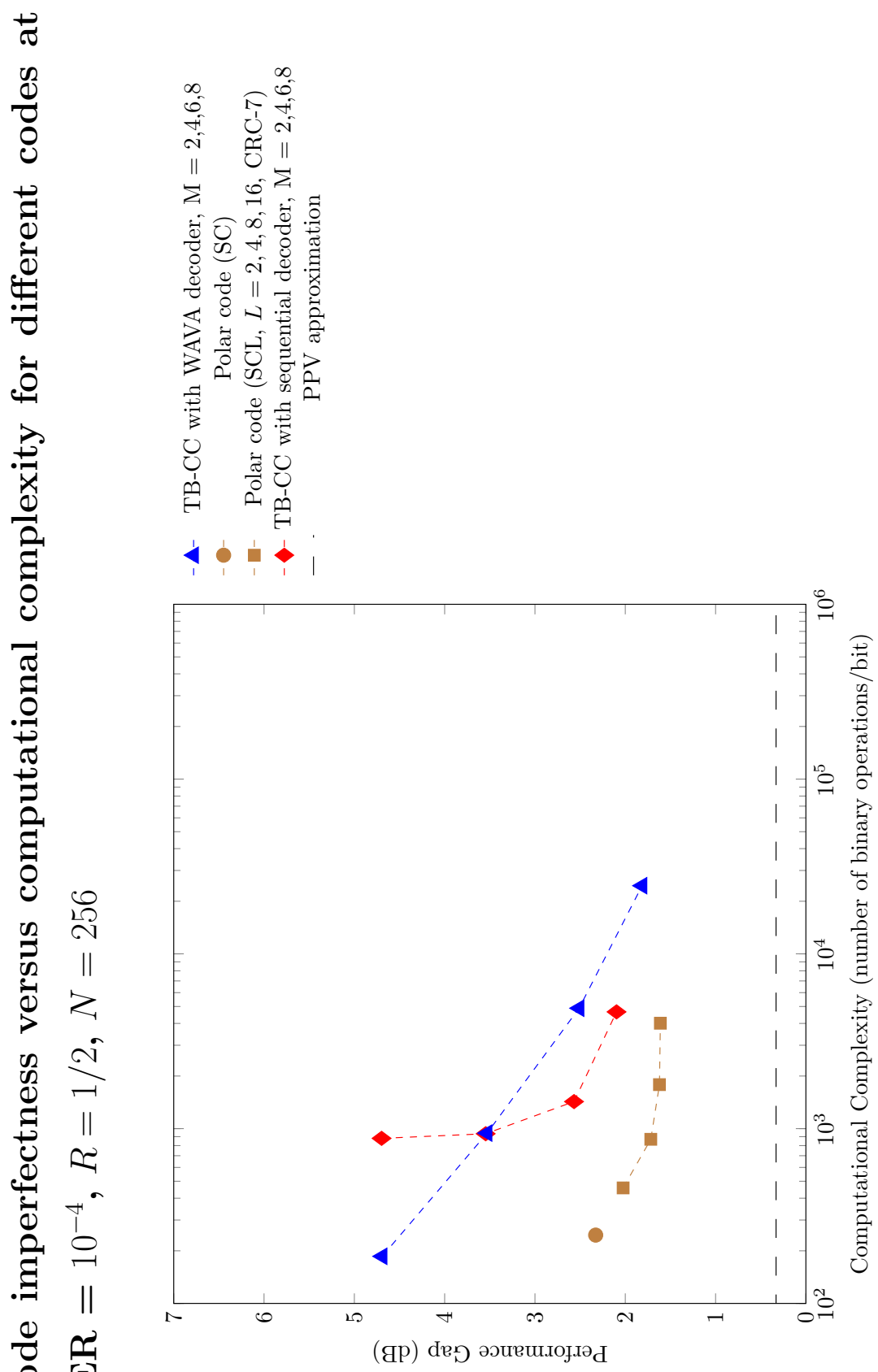

Figure C.29: Code imperfectness versus computational complexity at FER = 10^{-4} for different codes with $R = 1/2$, $N = 256$

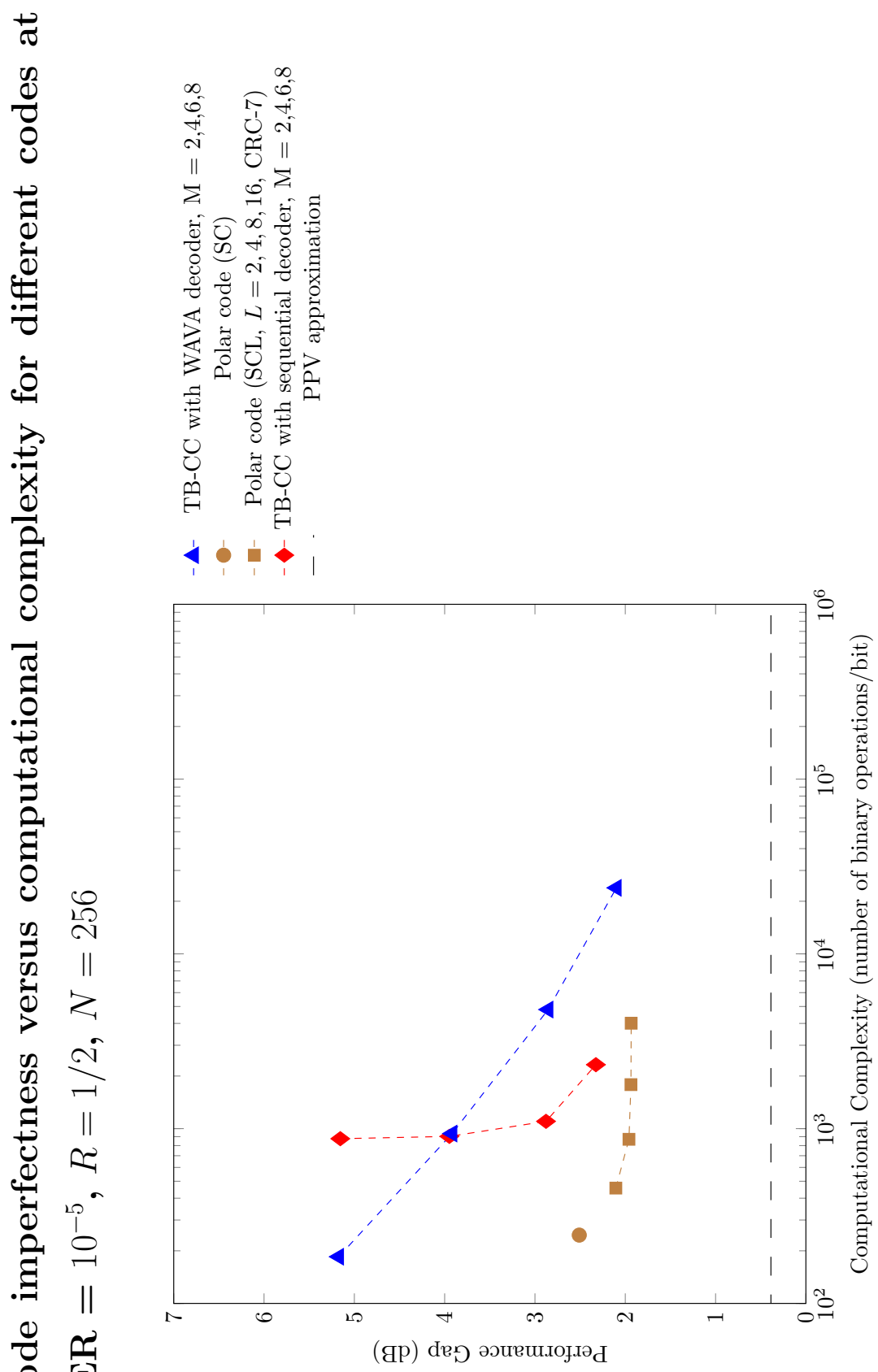

Figure C.30: Code imperfectness versus computational complexity at FER = 10^{-5} for different codes with $R = 1/2$, $N = 256$

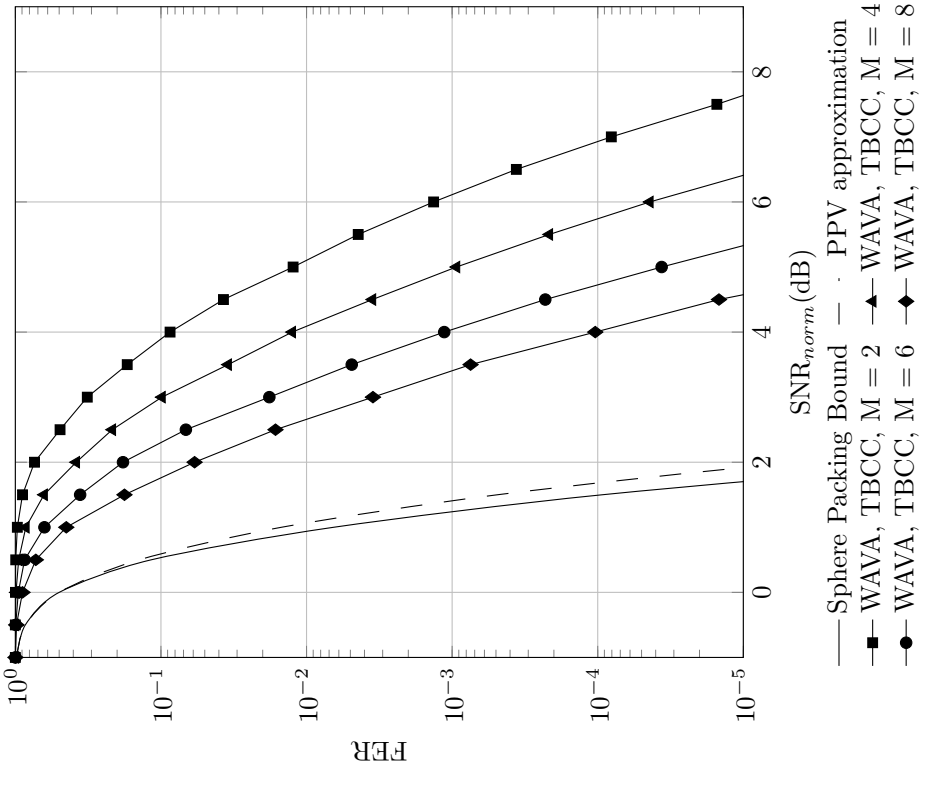

Figure C.31: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/2$, $N = 512$, maximum number of iterations = 4.

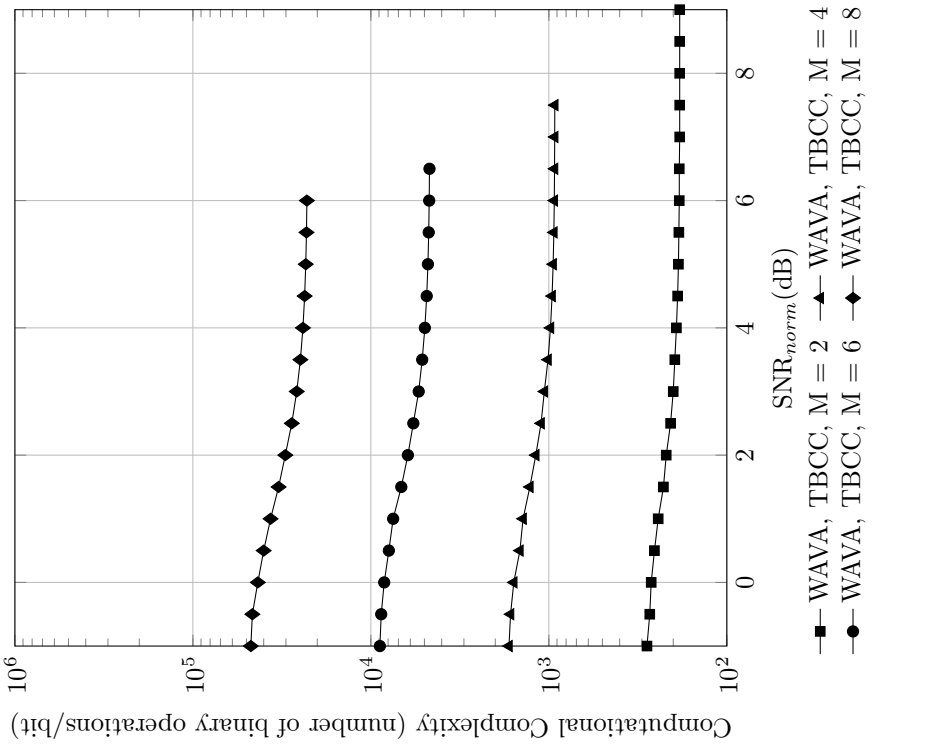

Figure C.32: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/2$, $N = 512$, maximum number of iterations = 4

Result for Polar Codes with SC and SCL Decoder, $R = 1/2$, $N = 512$

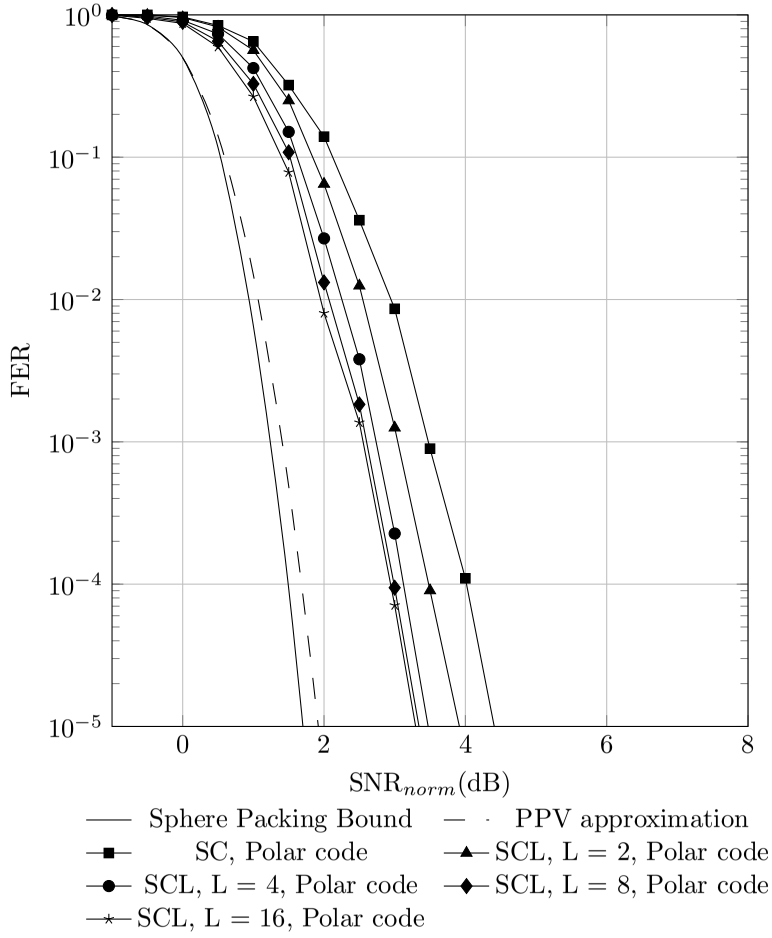

Figure C.33: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/2$, $N = 512$

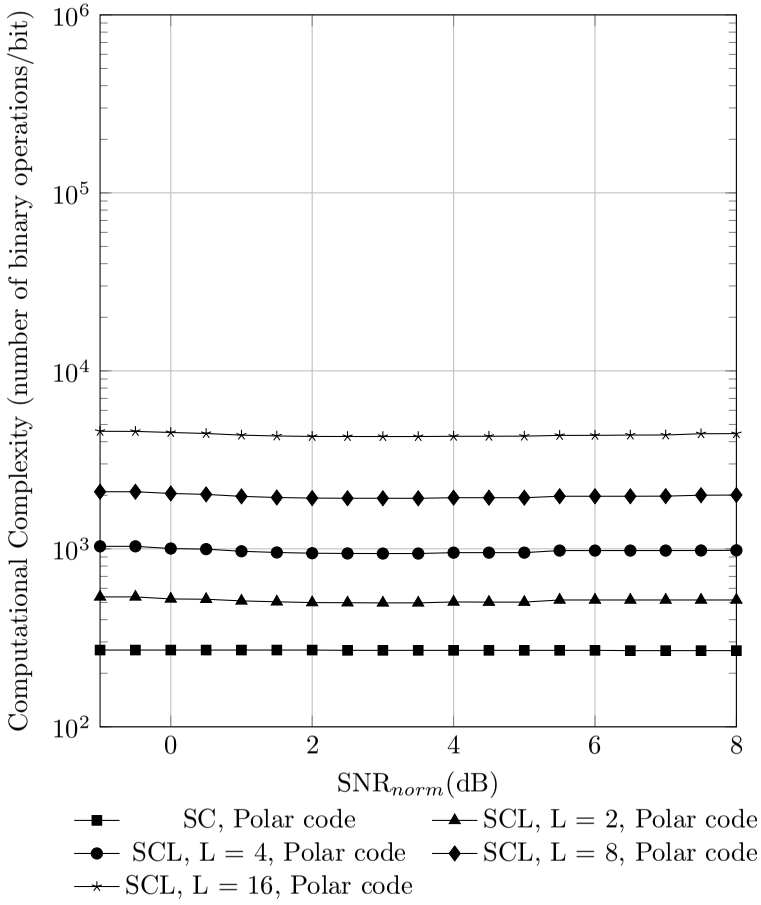

Figure C.34: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/2$, $N = 512$

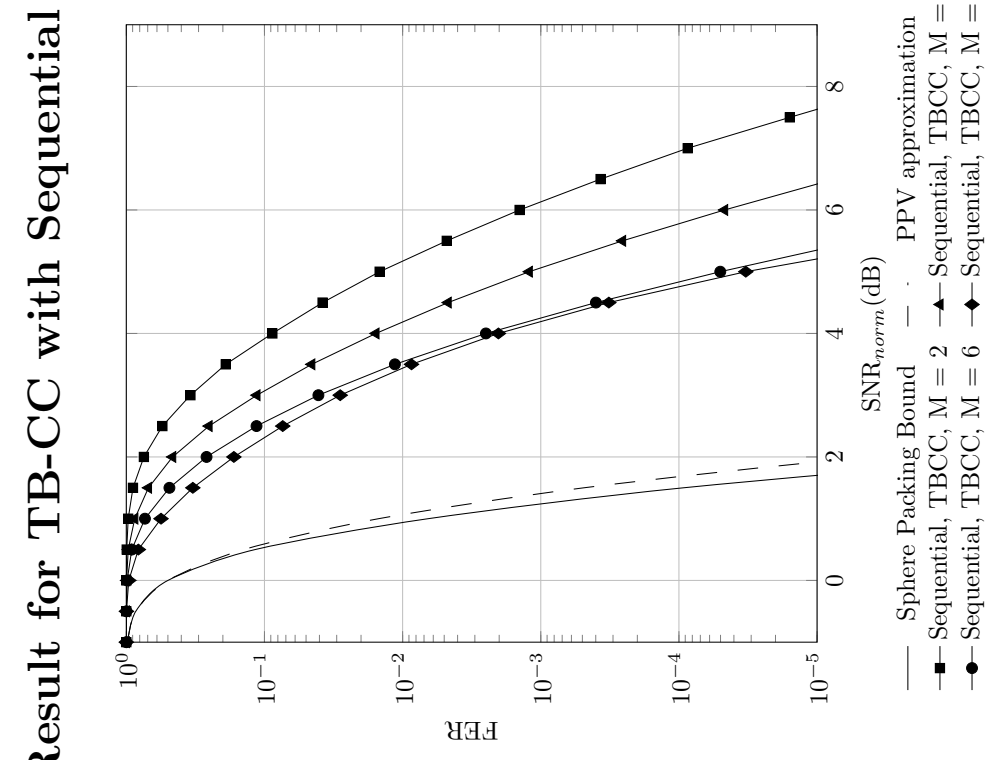

Figure C.35: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/2$, $N = 512$

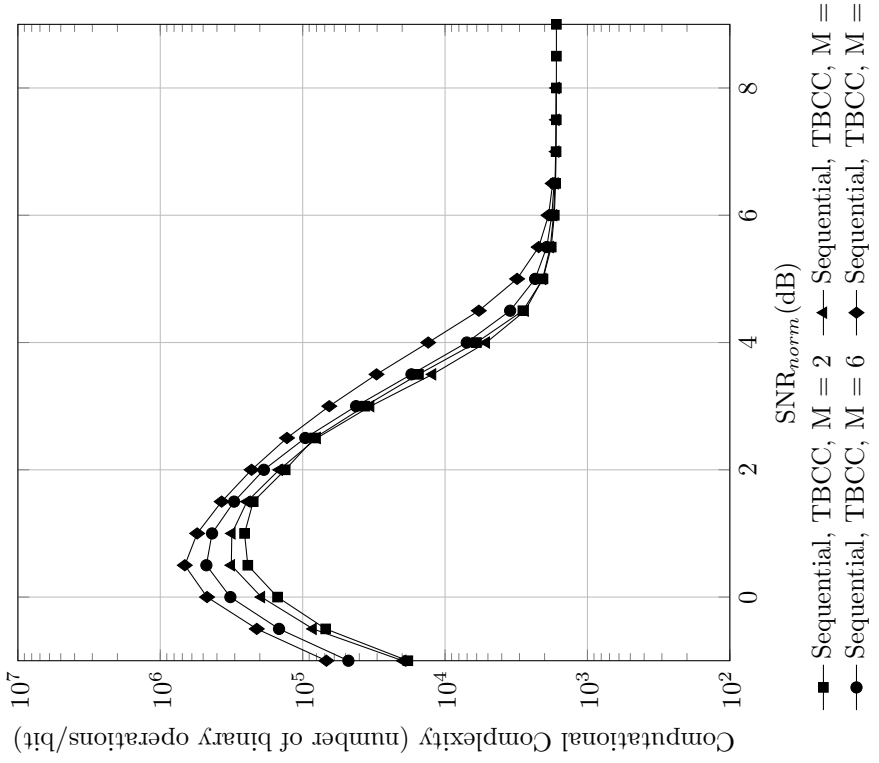

Figure C.36: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/2$, $N = 512$

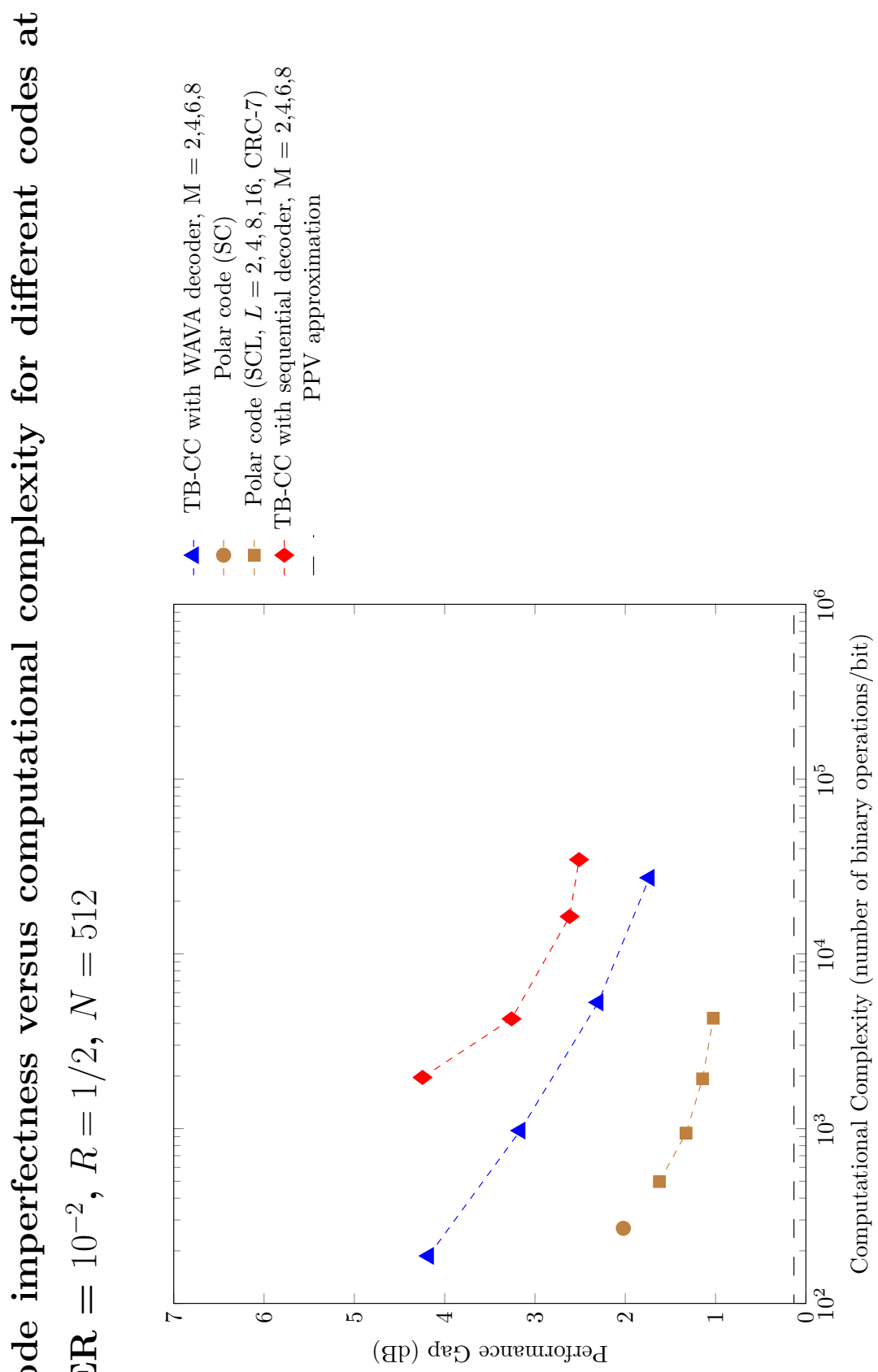

Figure C.37: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 1/2$, $N = 512$

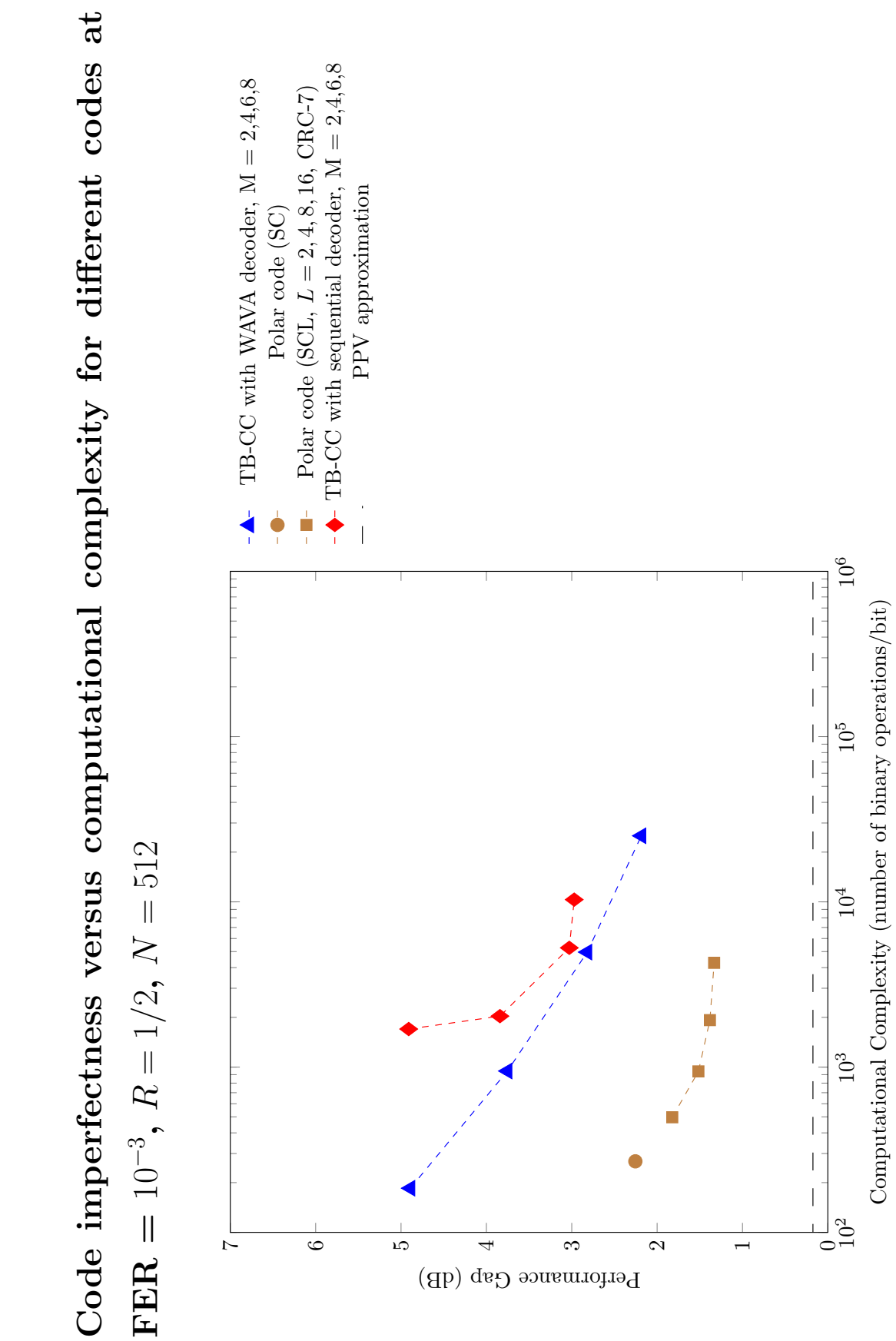

Figure C.38: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 1/2$, $N = 512$

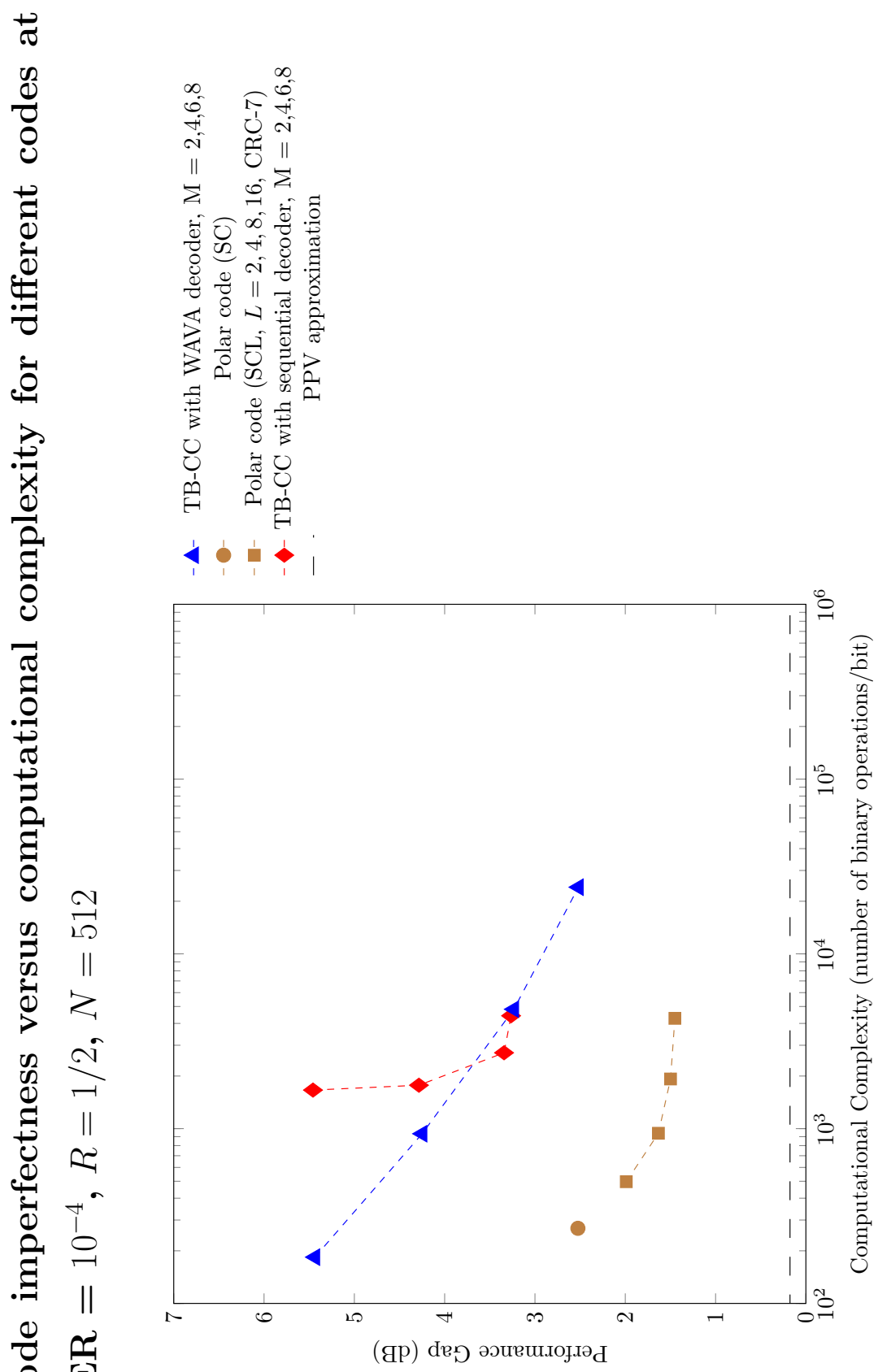

Figure C.39: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 1/2$, $N = 512$

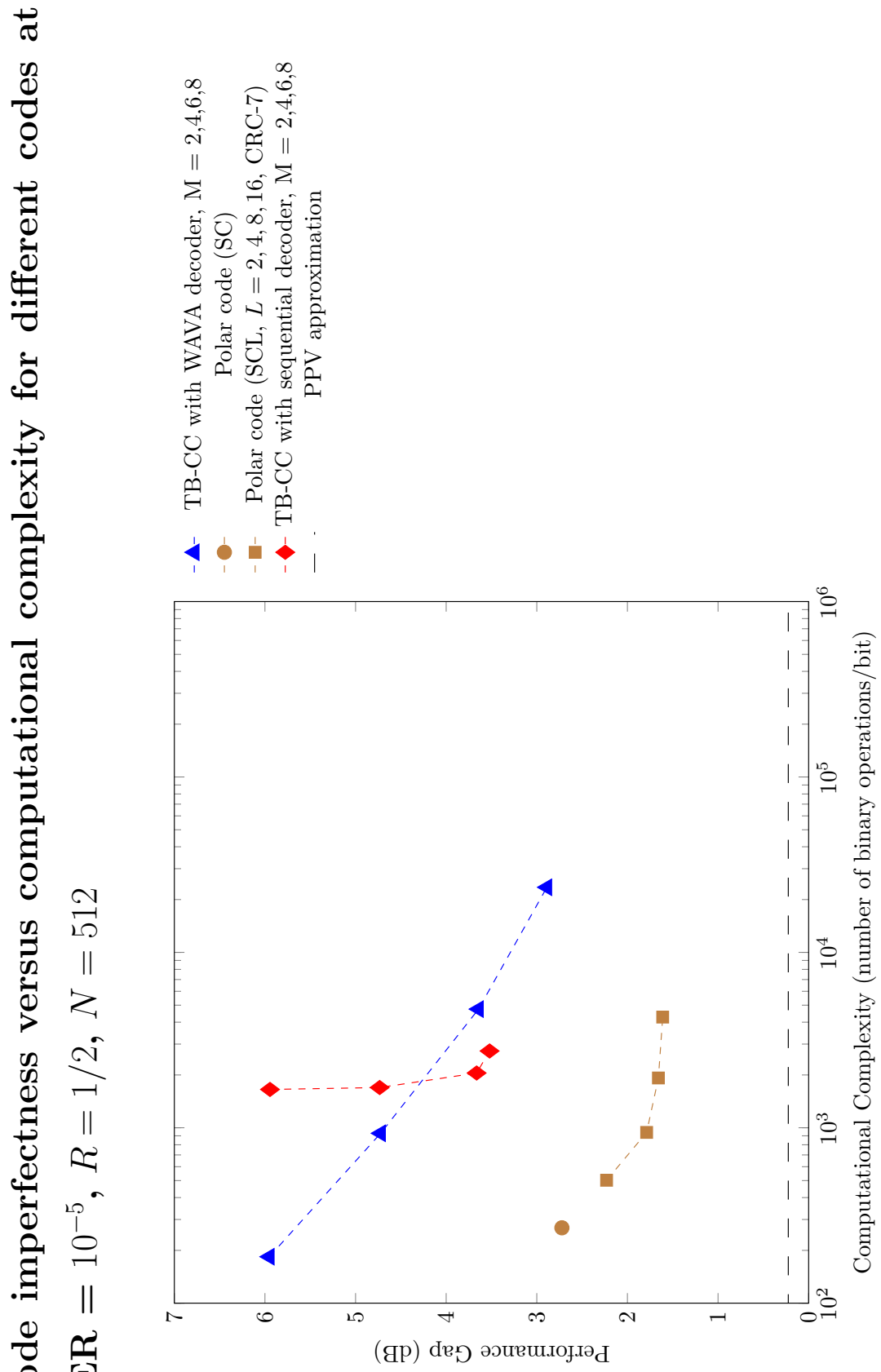

Figure C.40: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 1/2$, $N = 512$

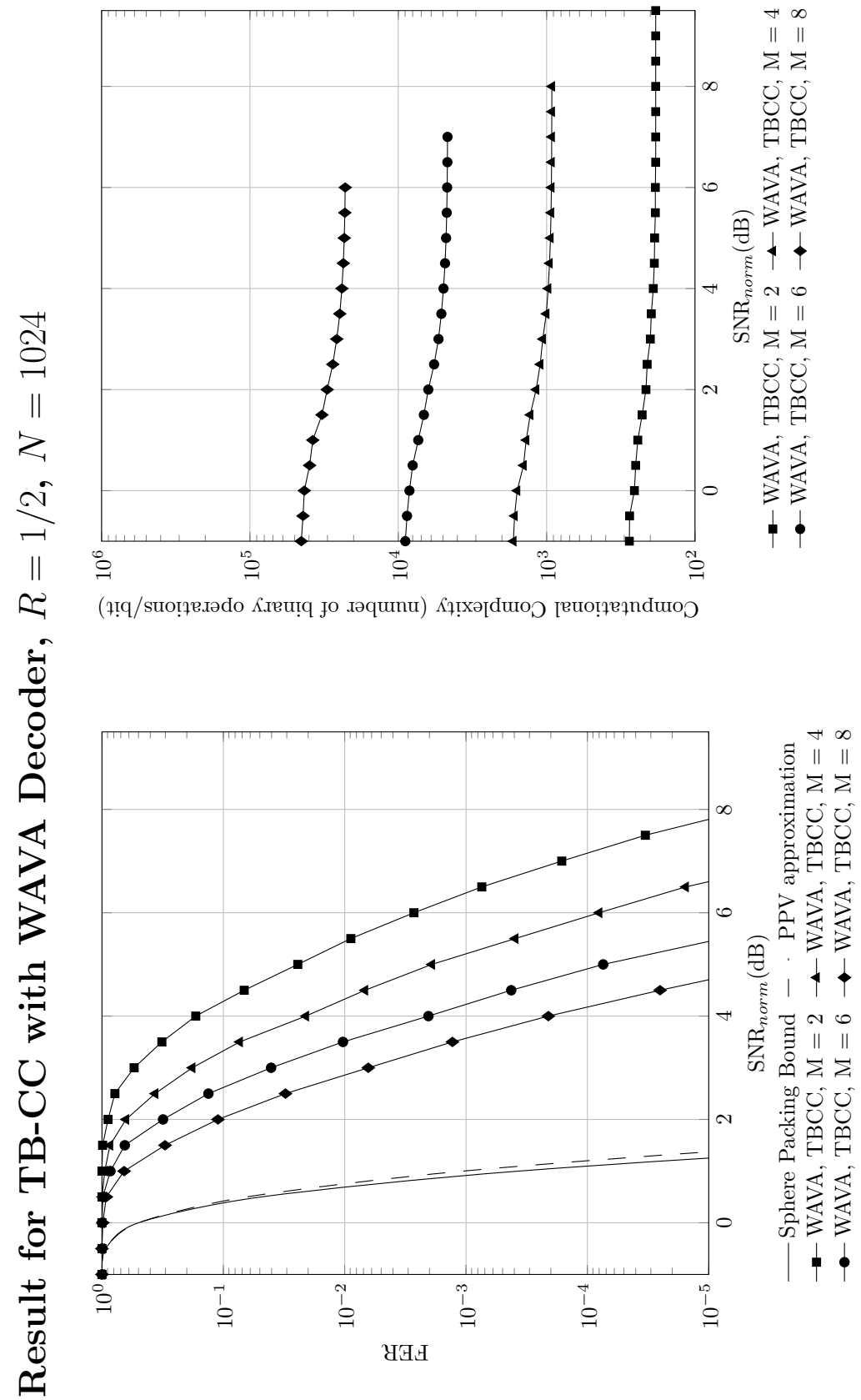

Figure C.41: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/2$, $N = 1024$, maximum number of iterations = 4.

Figure C.42: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/2$, $N = 1024$, maximum number of iterations = 4

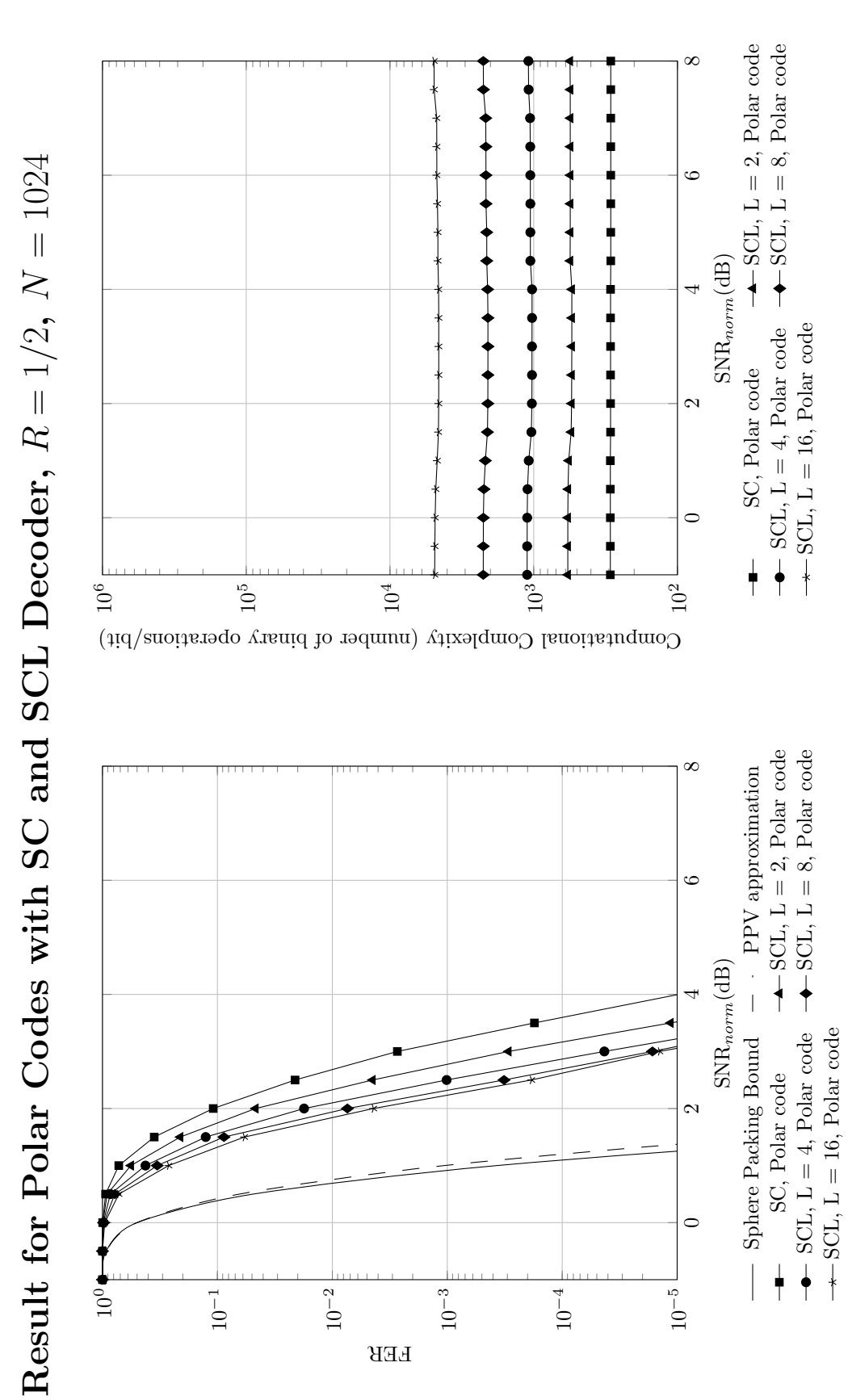

Figure C.43: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/2$, $N = 1024$

Figure C.44: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/2$, $N = 1024$

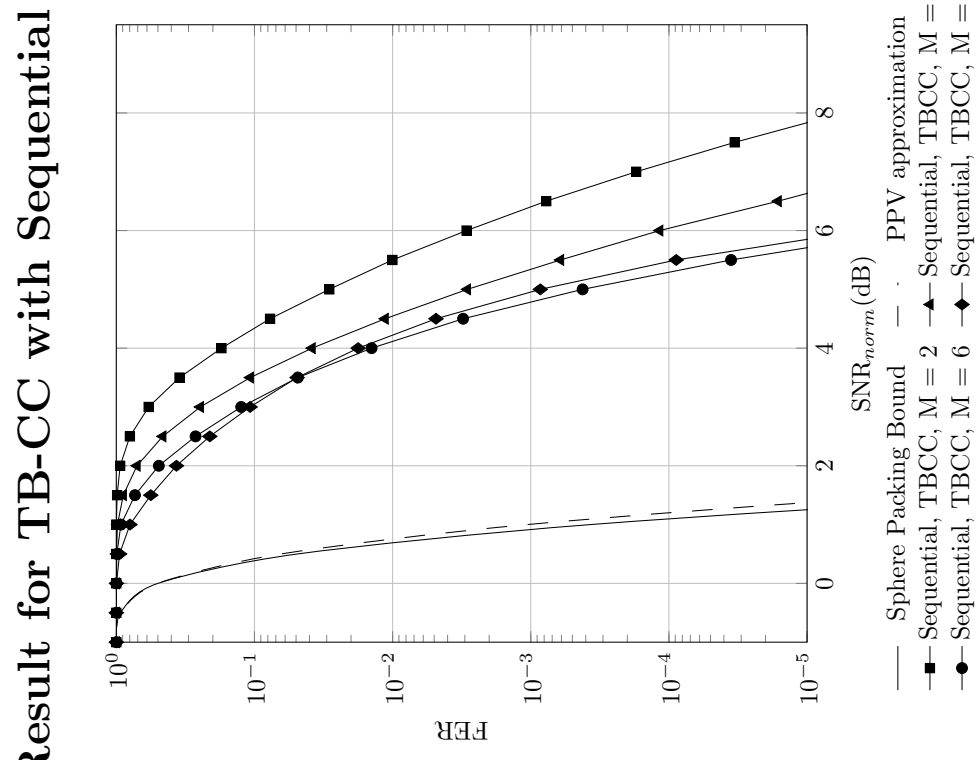

Figure C.45: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/2$, $N = 1024$

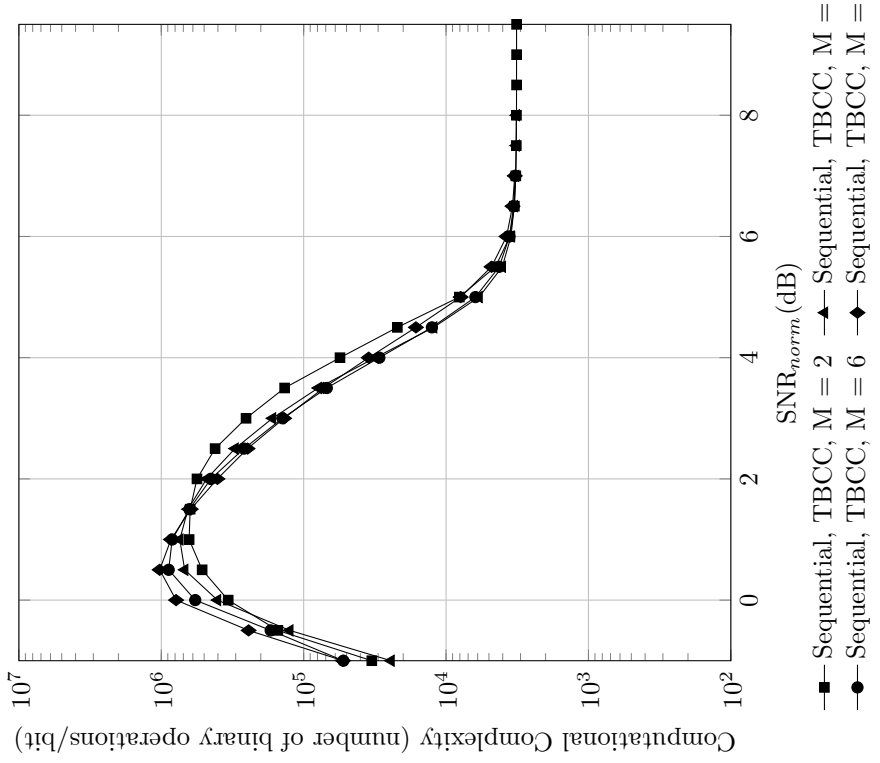

Figure C.46: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/2$, $N = 1024$

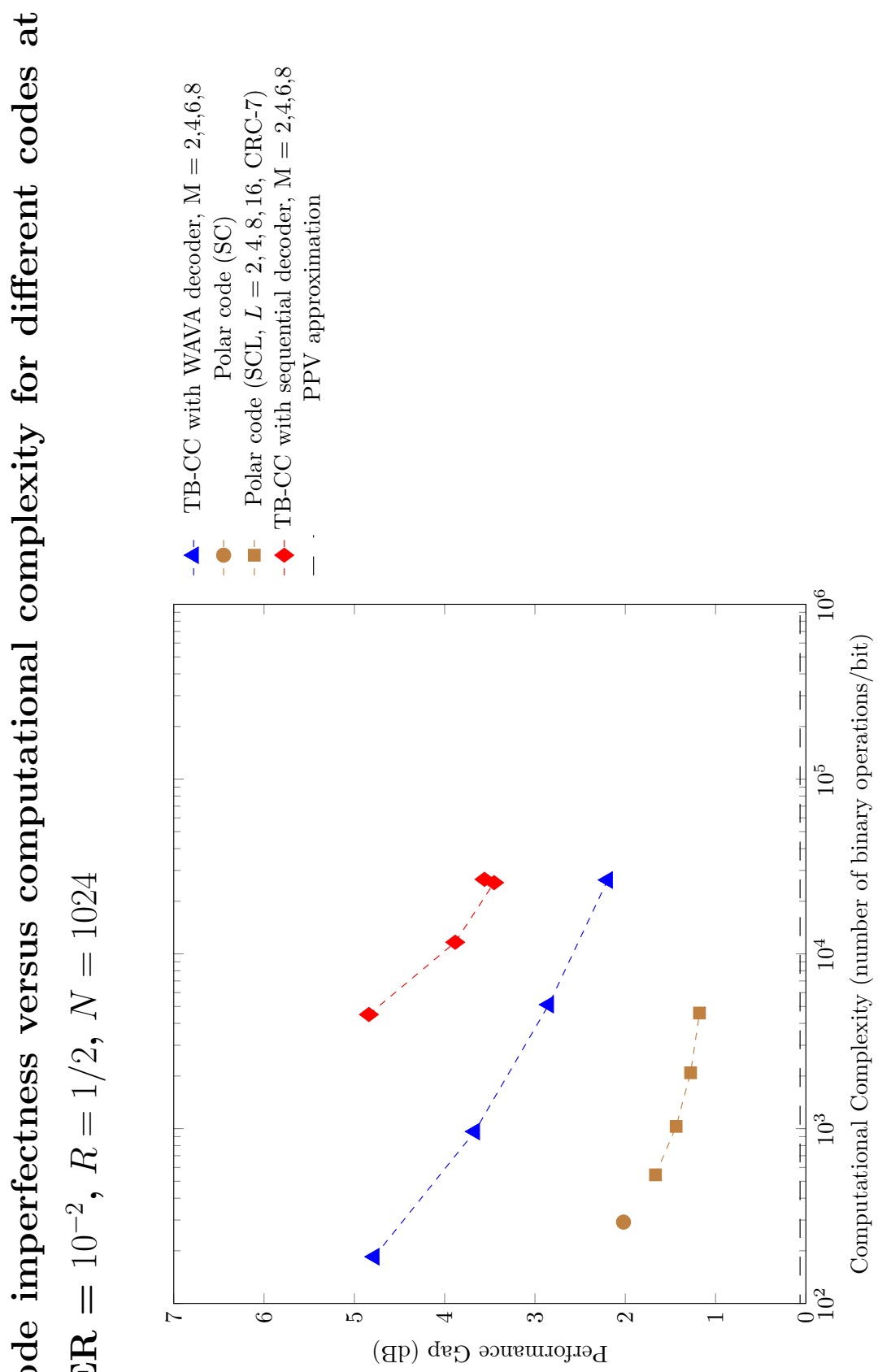

Figure C.47: Code imperfectness versus computational complexity at FER = 10^{-2} for different codes with $R = 1/2$, $N = 1024$

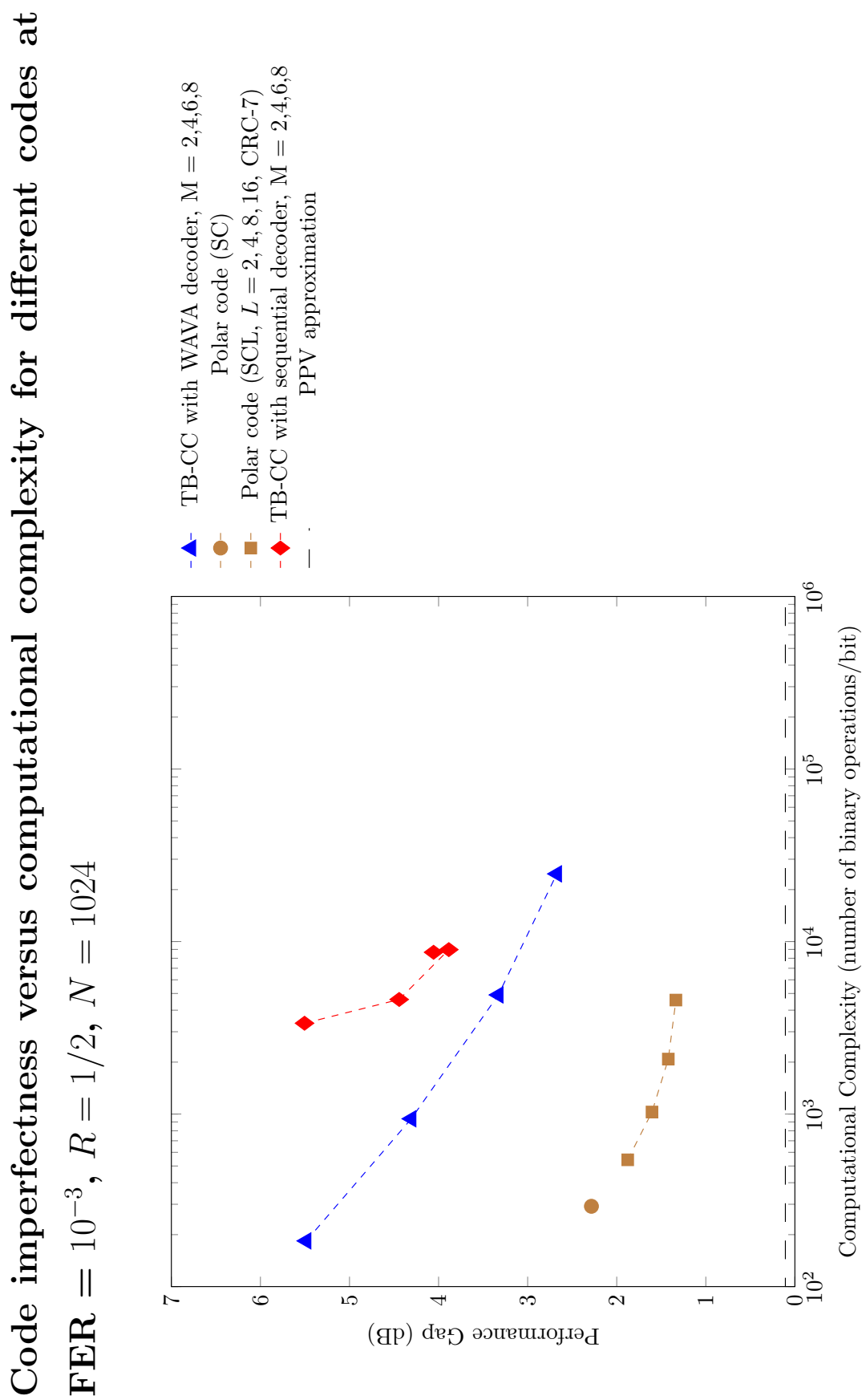

Figure C.48: Code imperfectness versus computational complexity at FER = 10^{-3} for different codes with $R = 1/2$, $N = 1024$

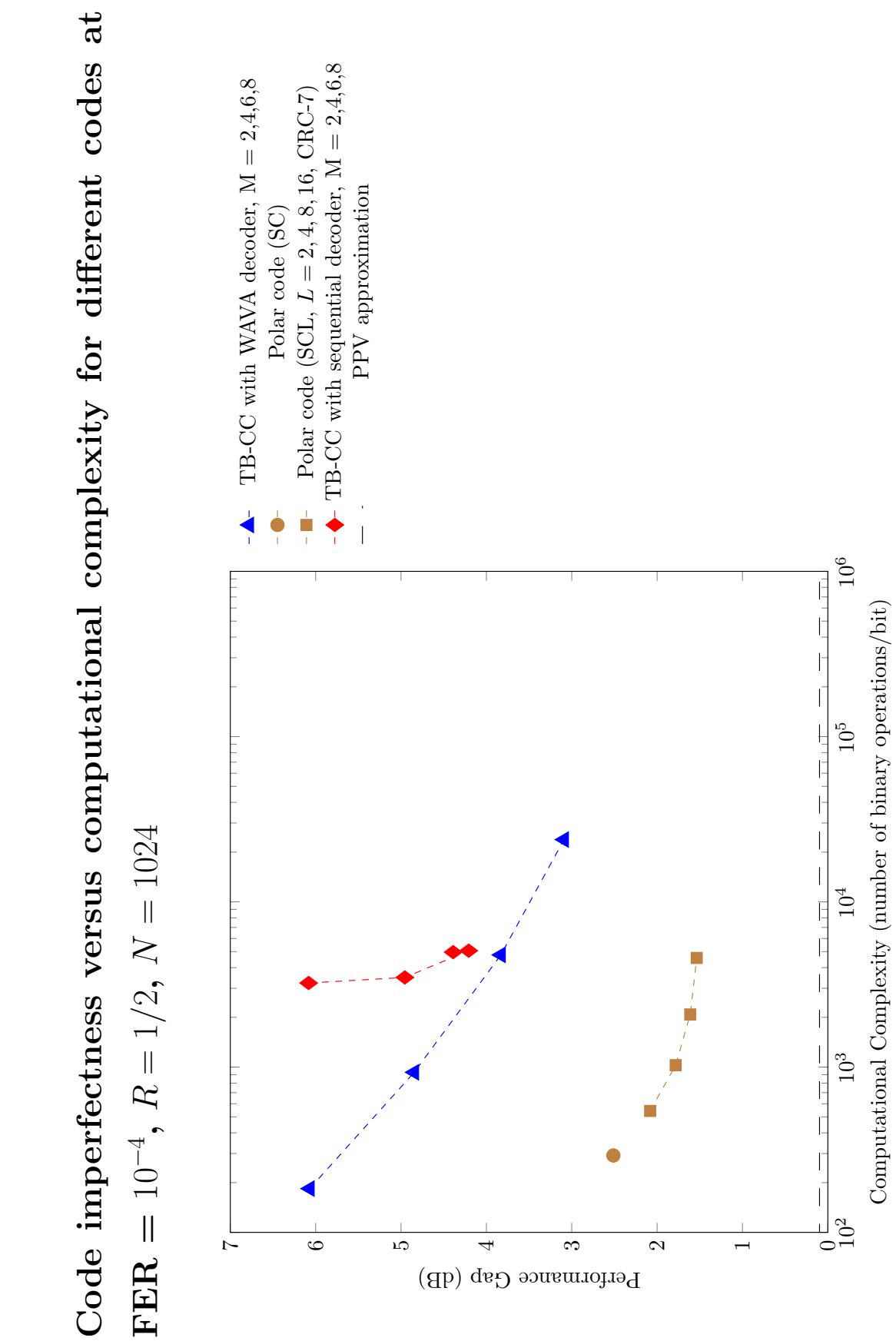

Figure C.49: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 1/2$, $N = 1024$

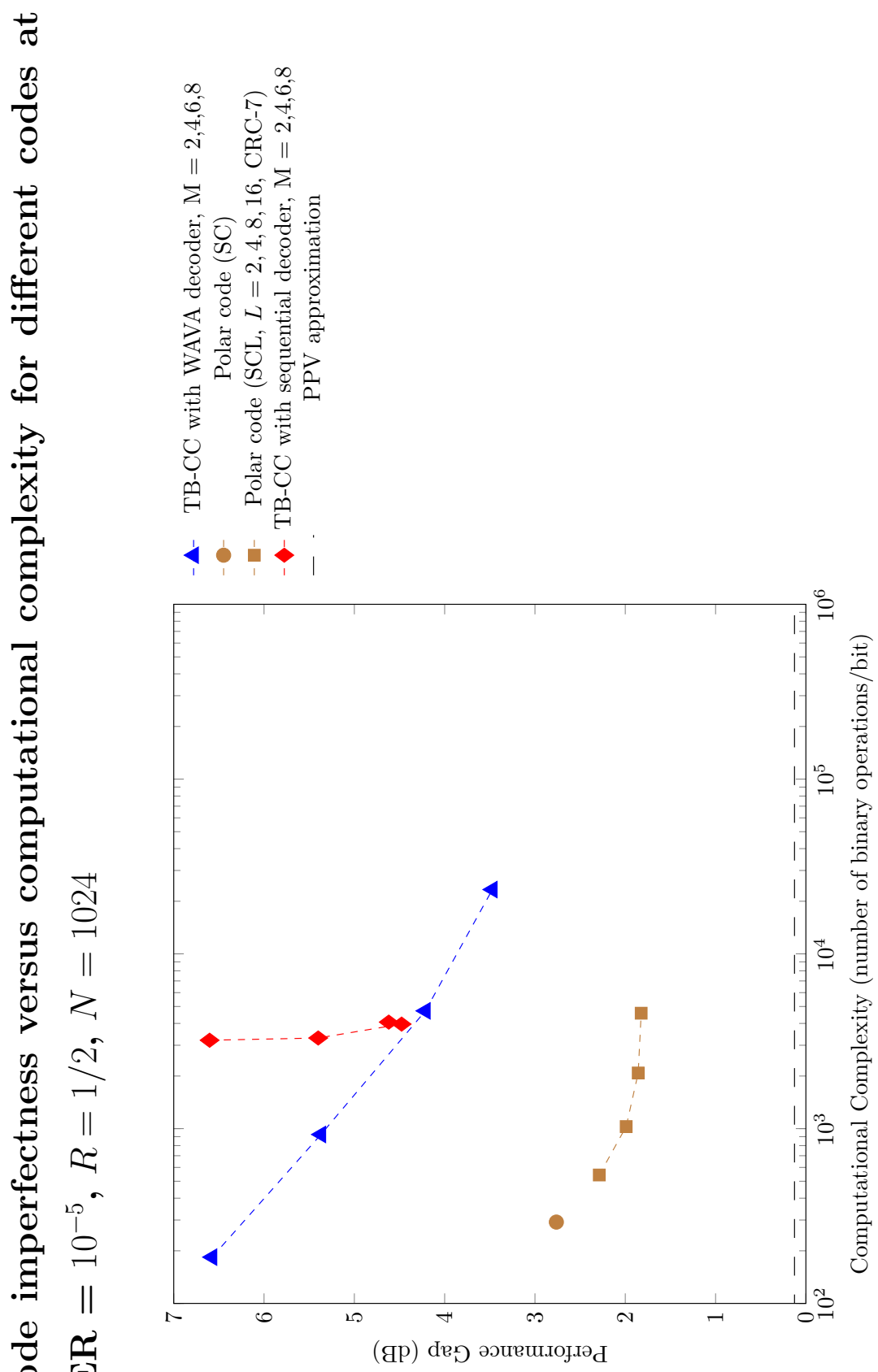

Figure C.50: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 1/2$, $N = 1024$

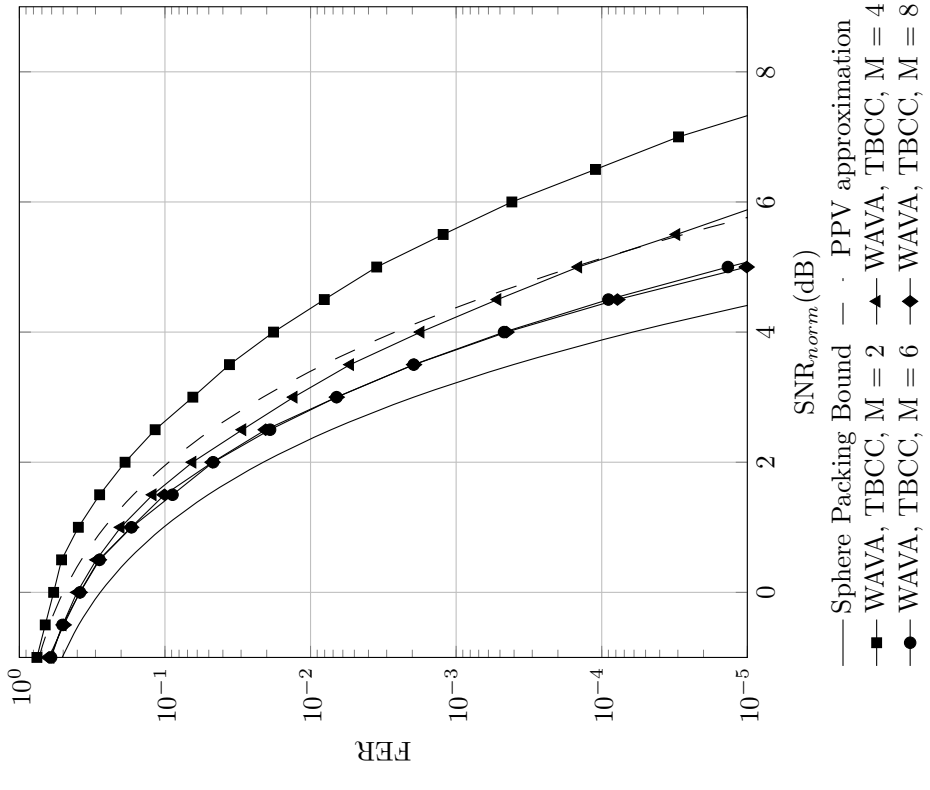

Figure C.51: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/3$, $N = 64$, maximum number of iterations = 4.

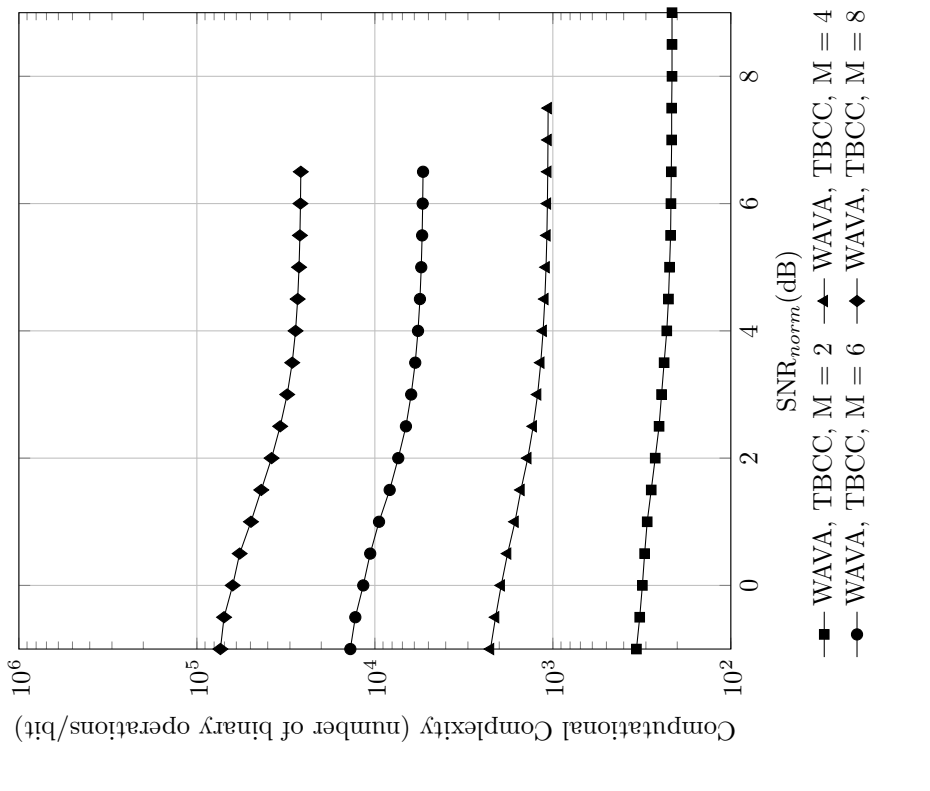

Figure C.52: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/3$, $N = 64$, maximum number of iterations = 4

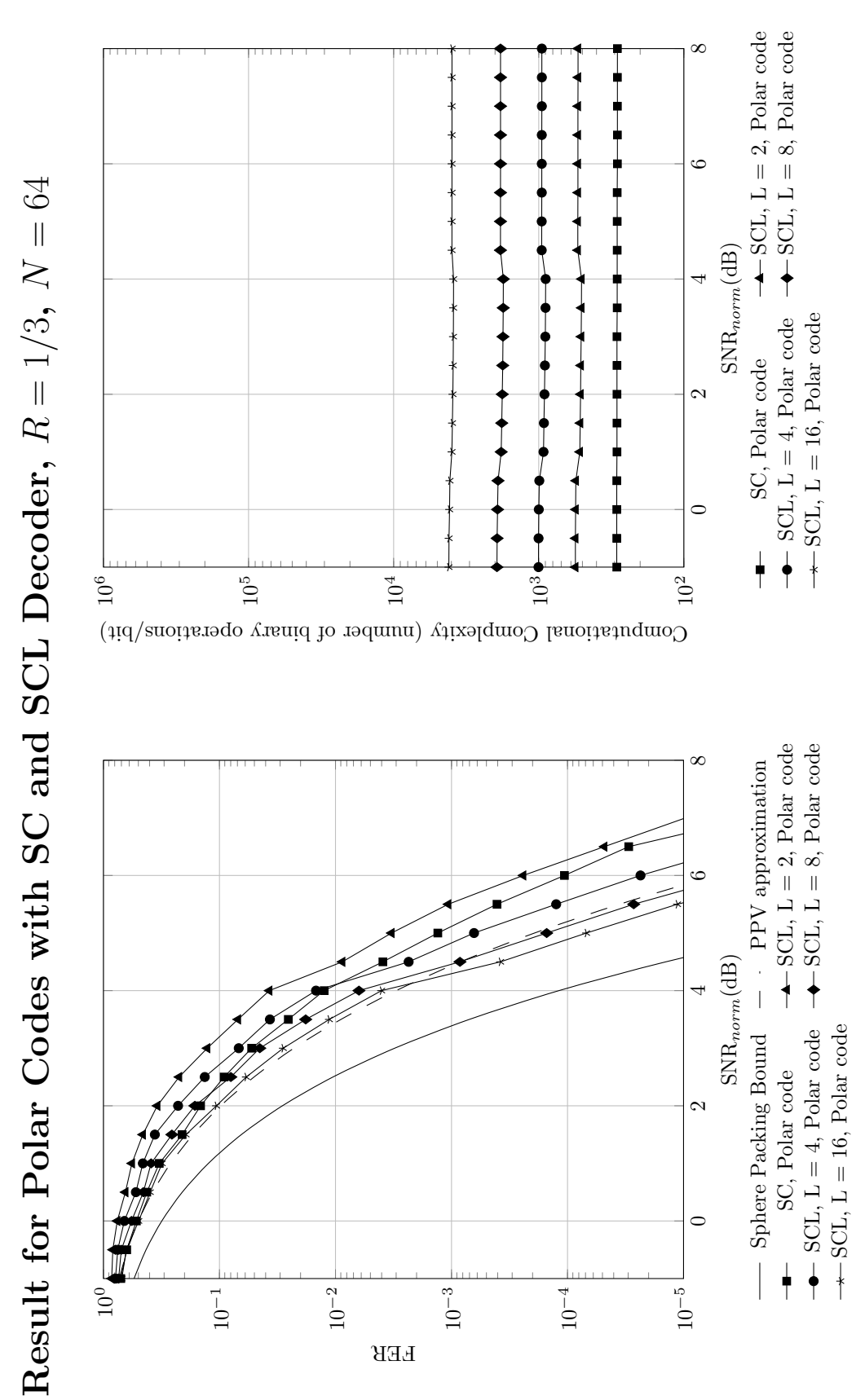

Figure C.53: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/3$, $N = 64$

Figure C.54: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/3$, $N = 64$

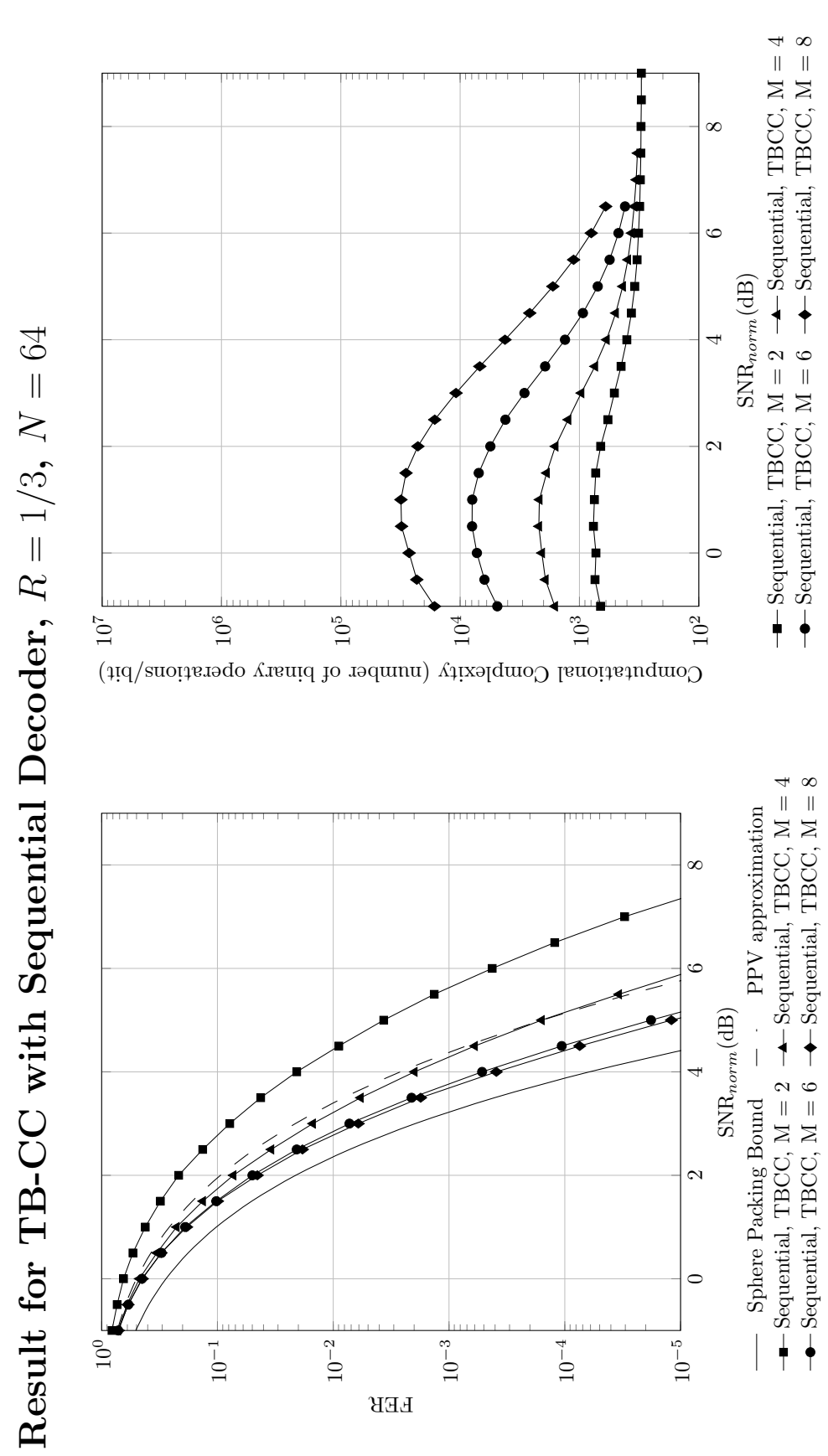

Figure C.55: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/3$, $N = 64$

Figure C.56: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/3$, $N = 64$

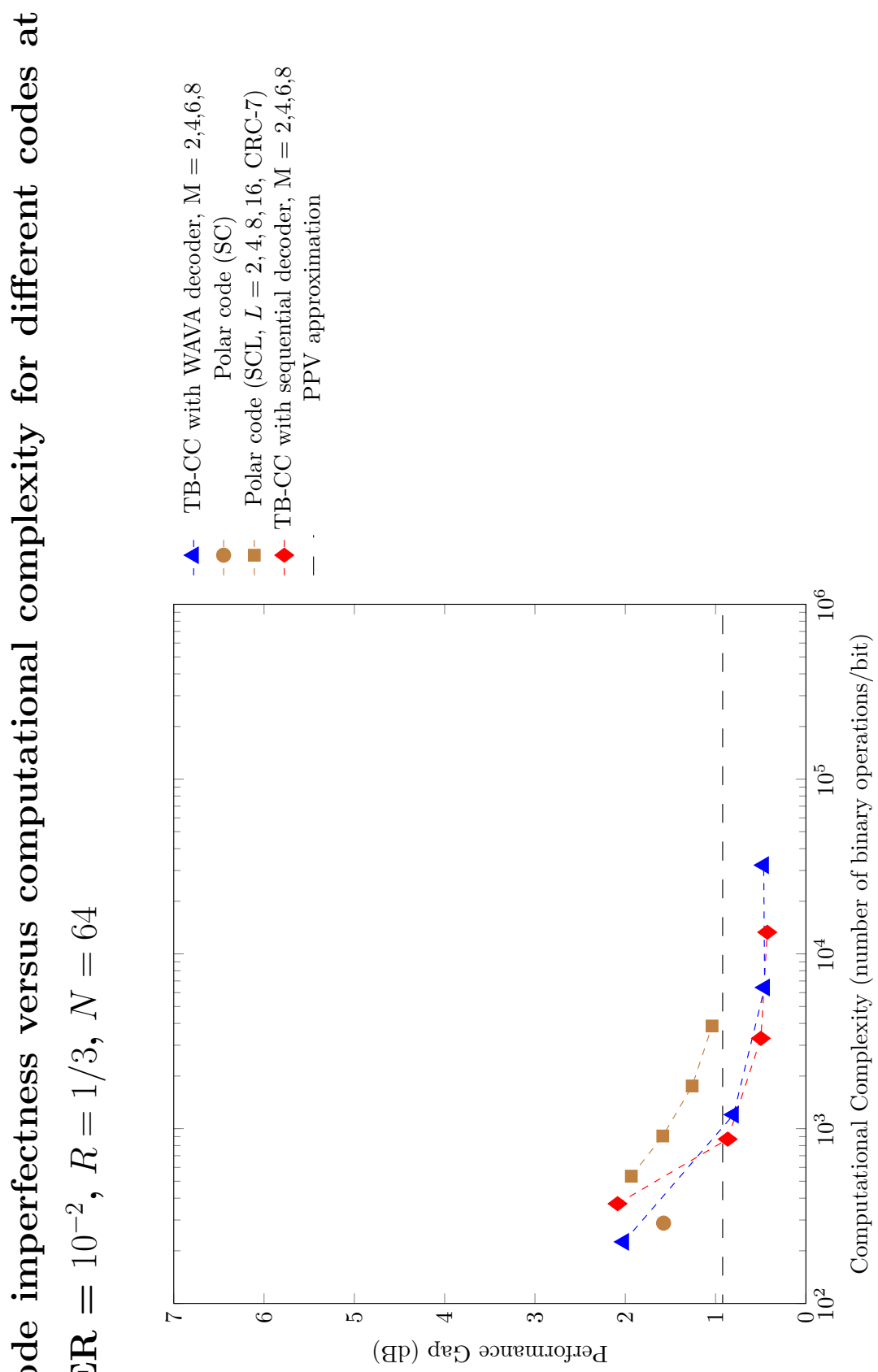

Figure C.57: Code imperfectness versus computational complexity at FER = 10^{-2} for different codes with $R = 1/3$, $N = 64$

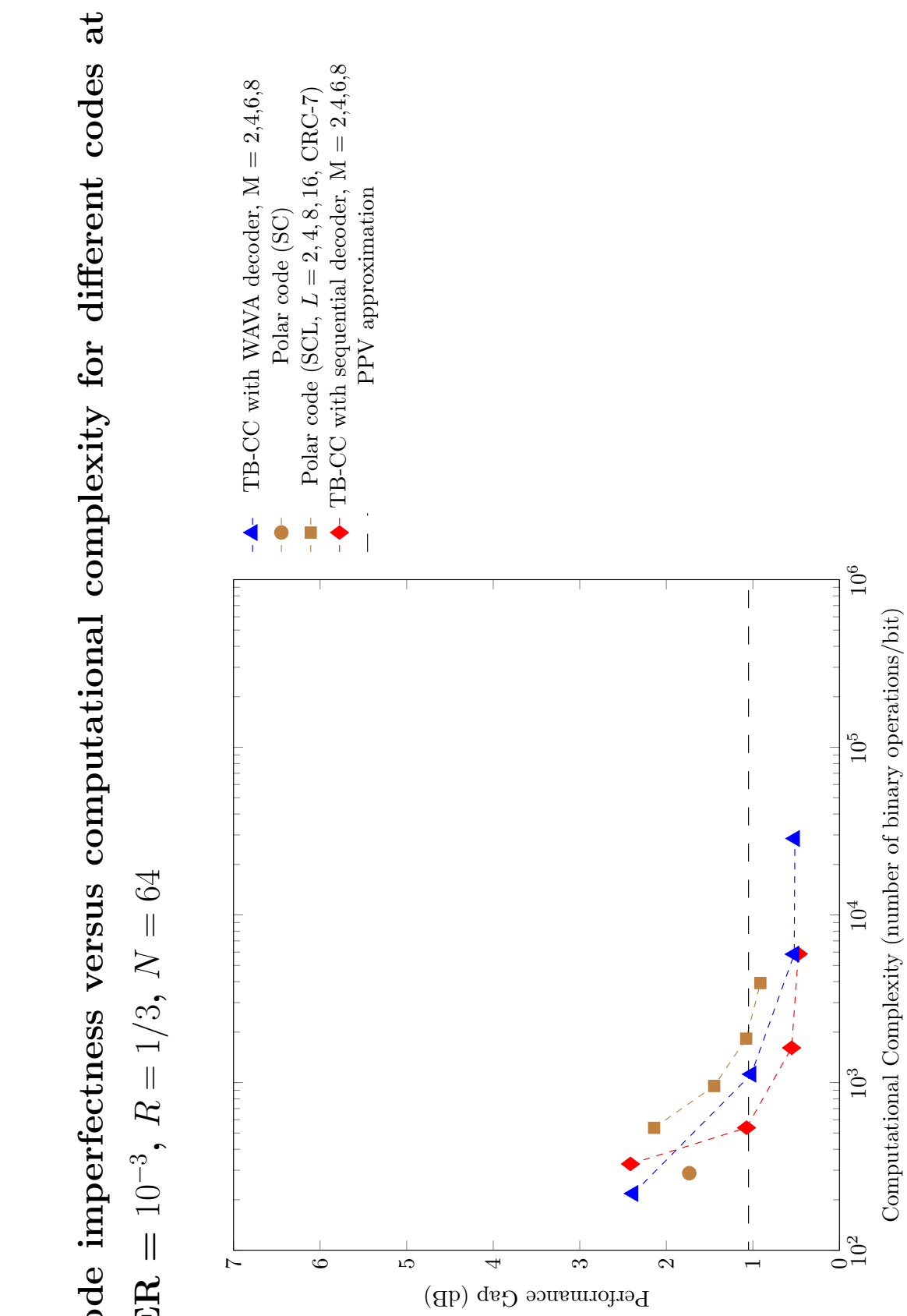

Figure C.58: Code imperfectness versus computational complexity at FER = 10^{-3} for different codes with $R = 1/3$, $N = 64$

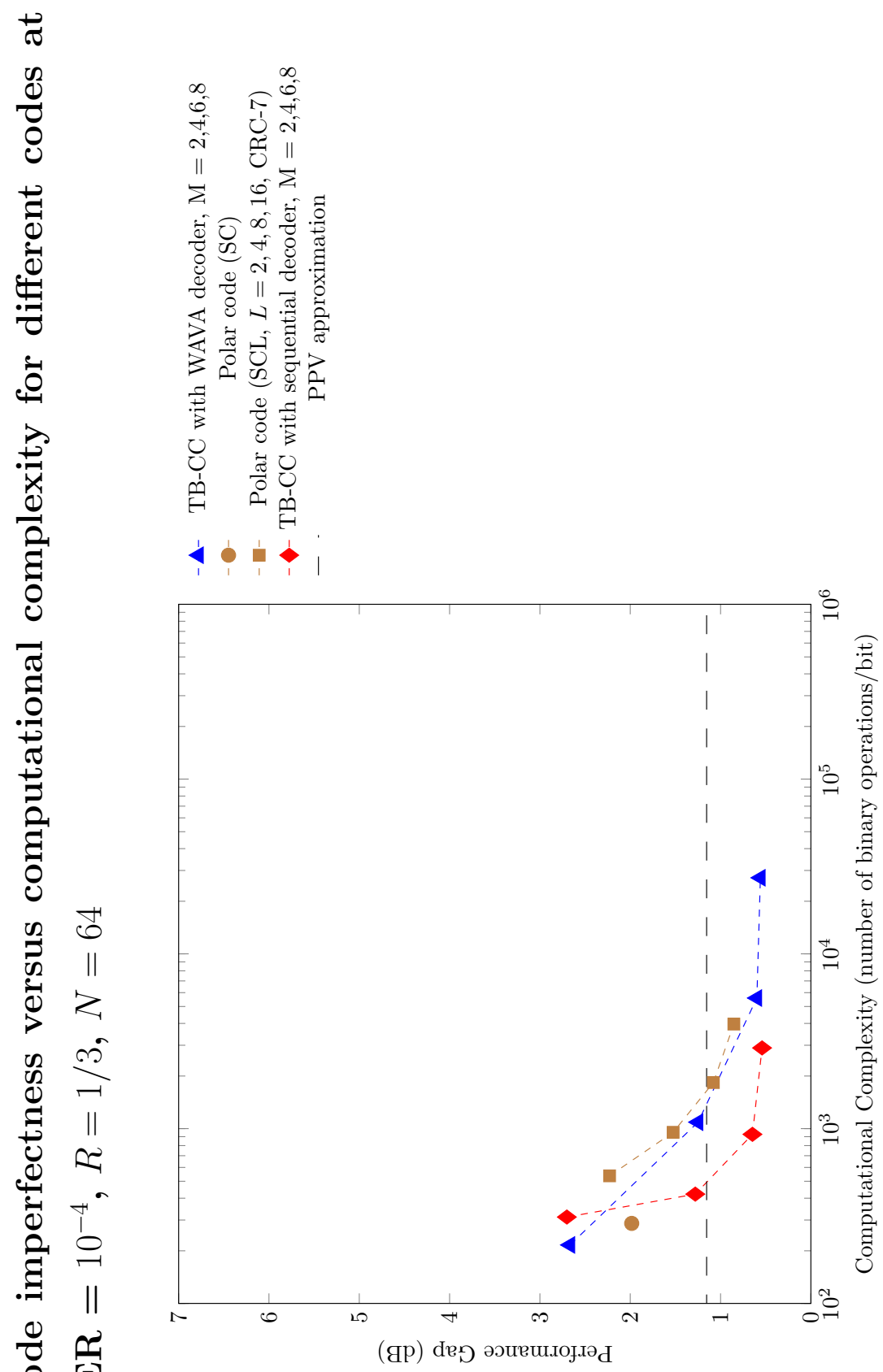

Figure C.59: Code imperfectness versus computational complexity at FER = 10^{-4} for different codes with $R = 1/3$, $N = 64$

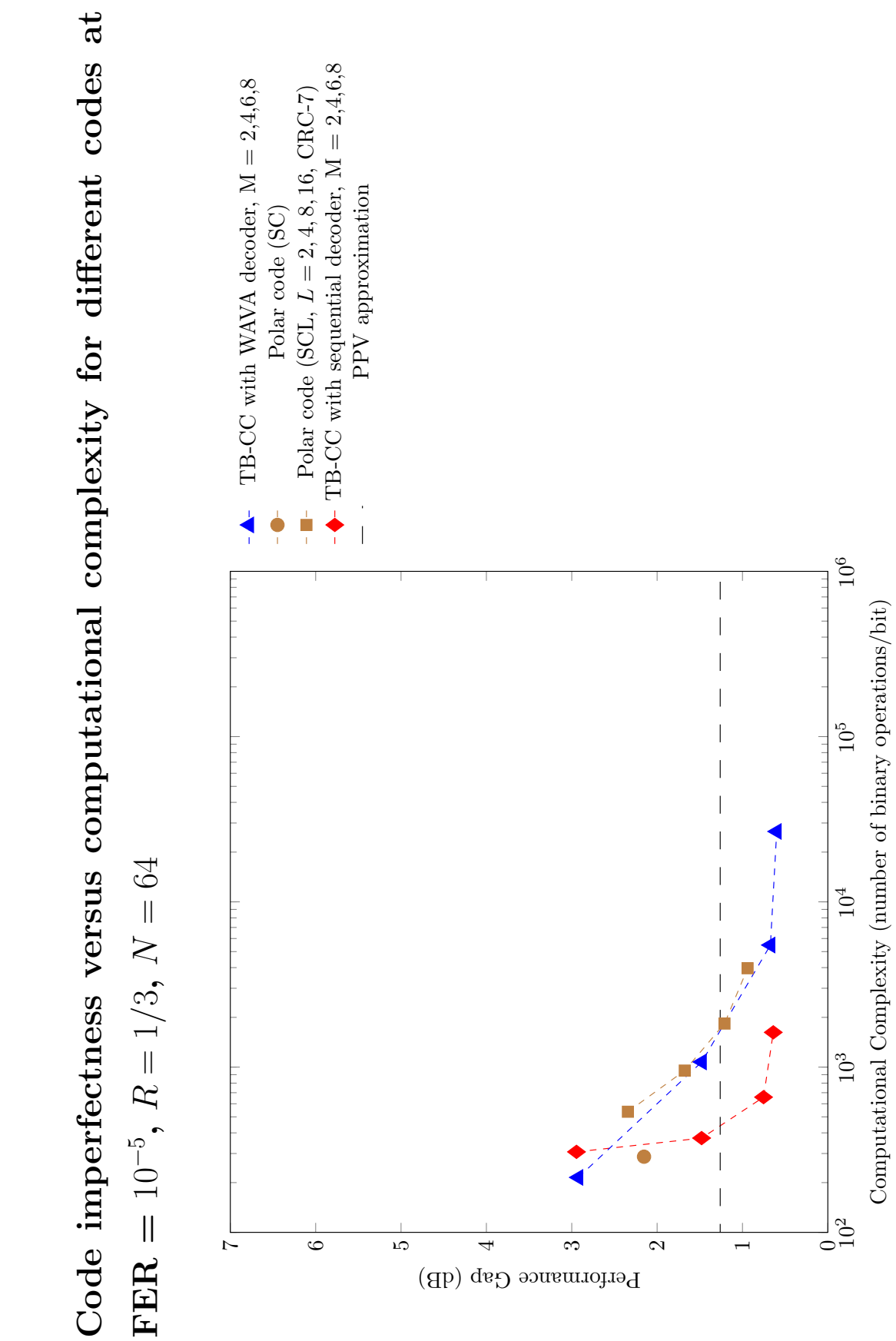

Figure C.60: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 1/3$, $N = 64$

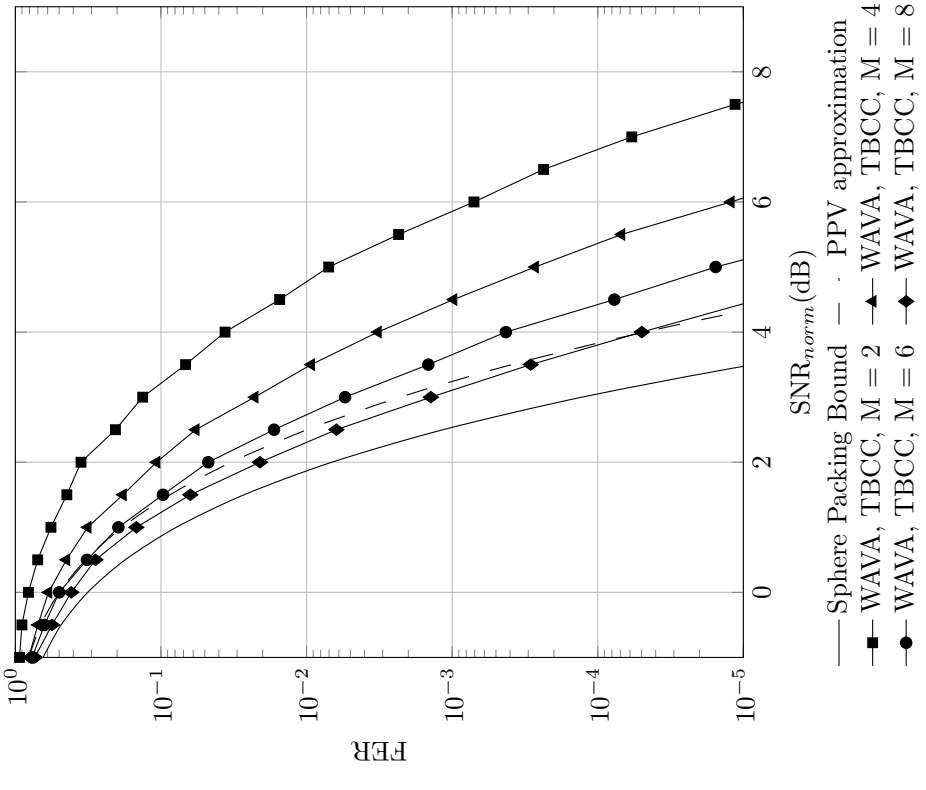

Figure C.61: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/3$, $N = 128$, maximum number of iterations = 4.

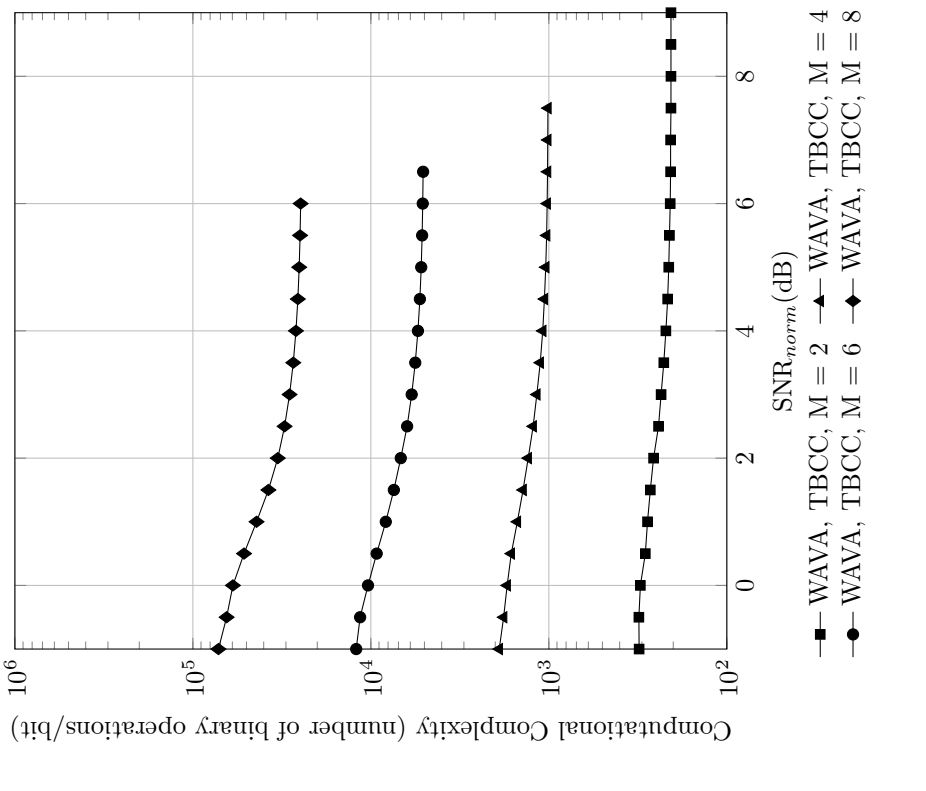

Figure C.62: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/3$, $N = 128$, maximum number of iterations = 4

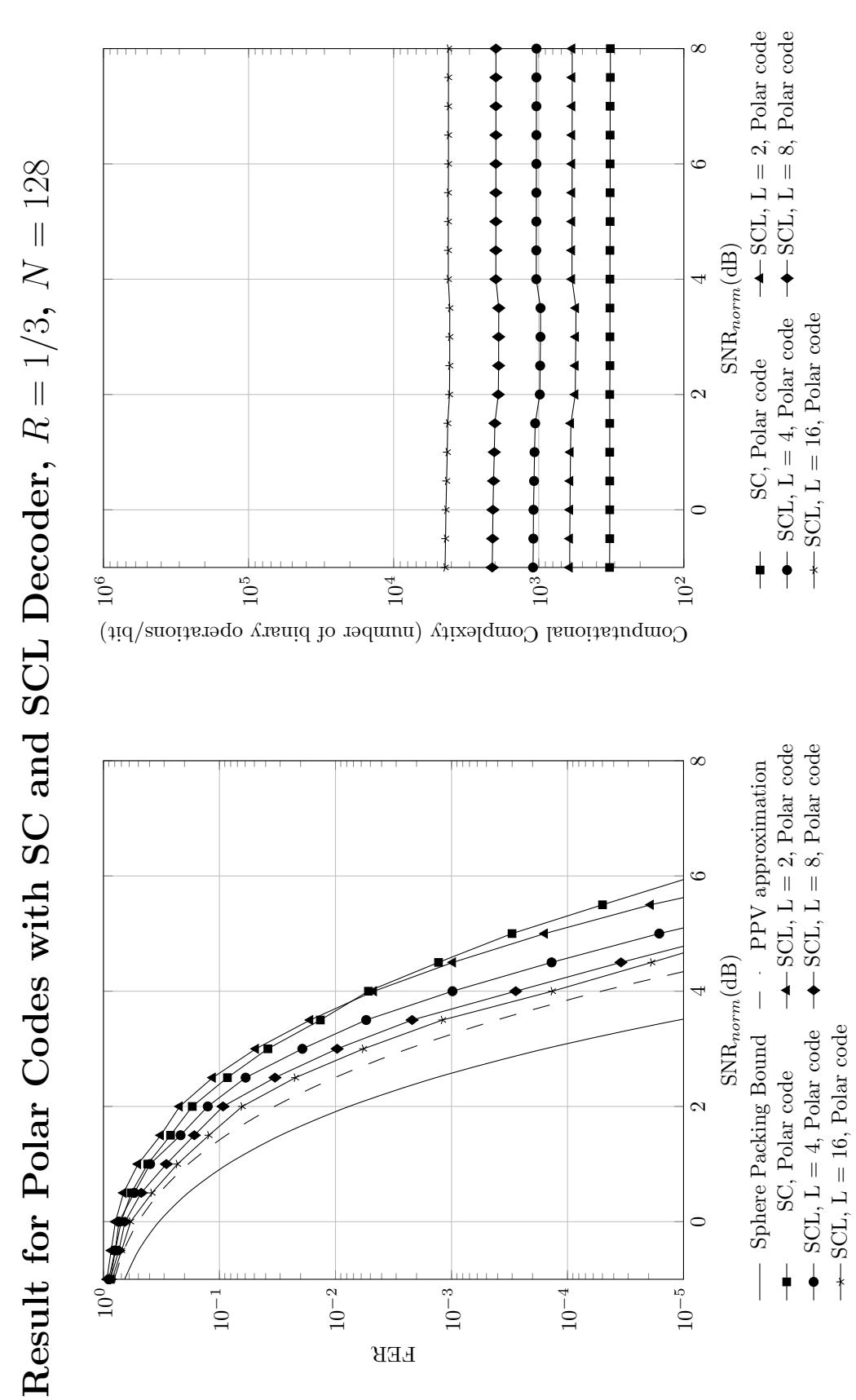

Figure C.63: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/3$, $N = 128$

Figure C.64: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/3$, $N = 128$

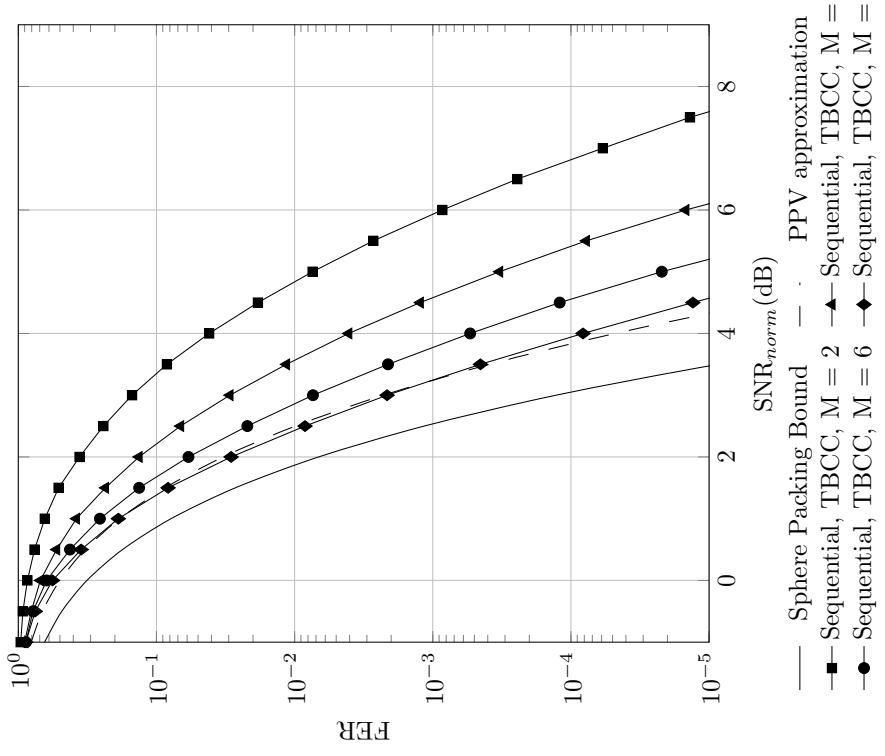

Figure C.65: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/3$, $N = 128$

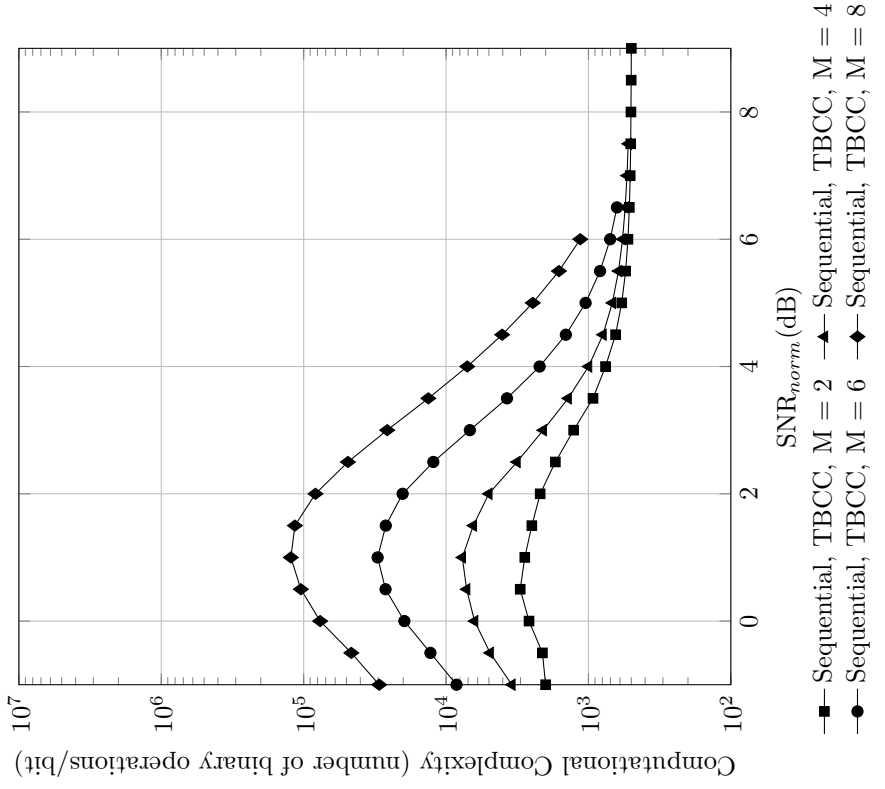

Figure C.66: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/3$, $N = 128$

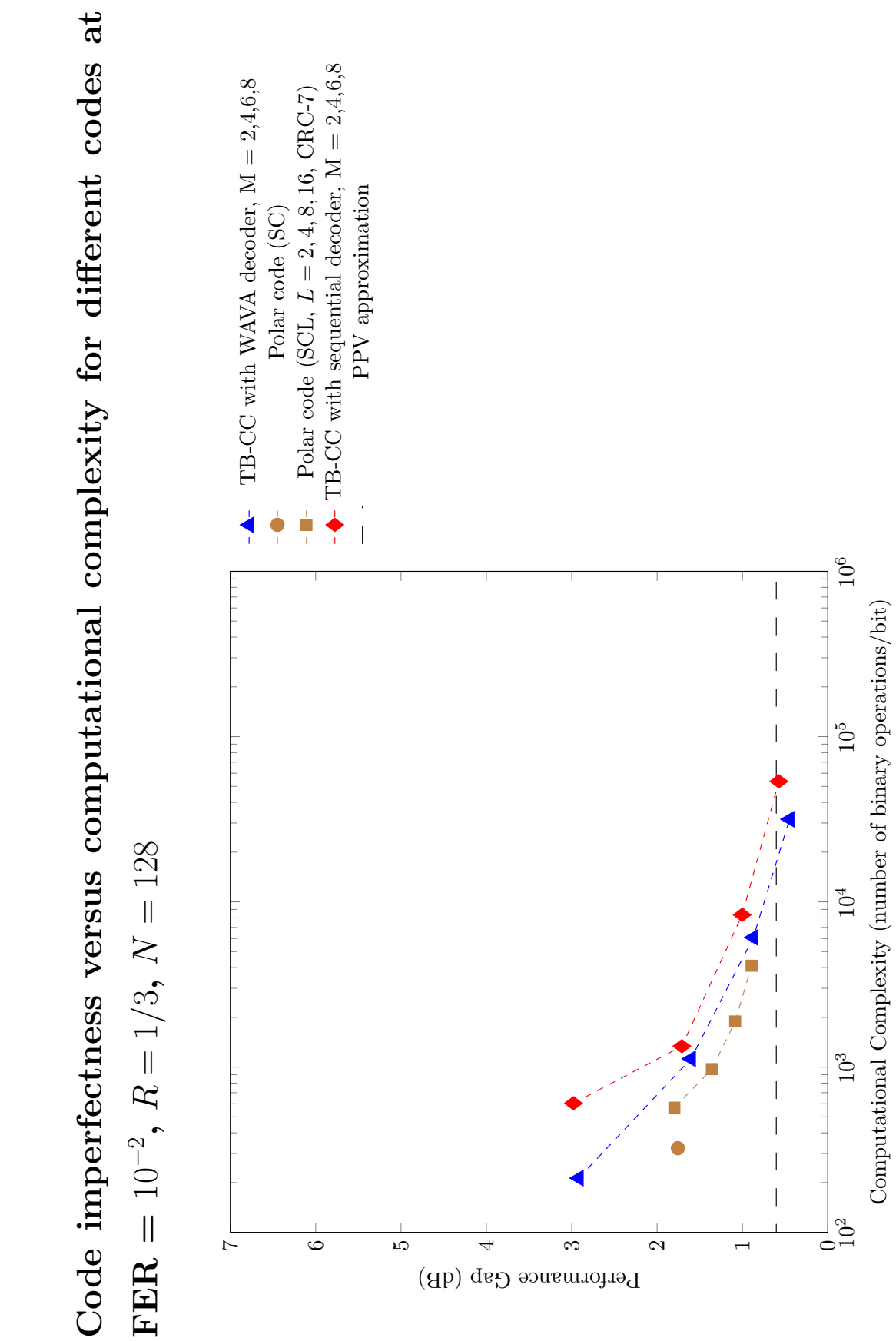

Figure C.67: Code imperfectness versus computational complexity at FER = 10^{-2} for different codes with $R = 1/3$, $N = 128$

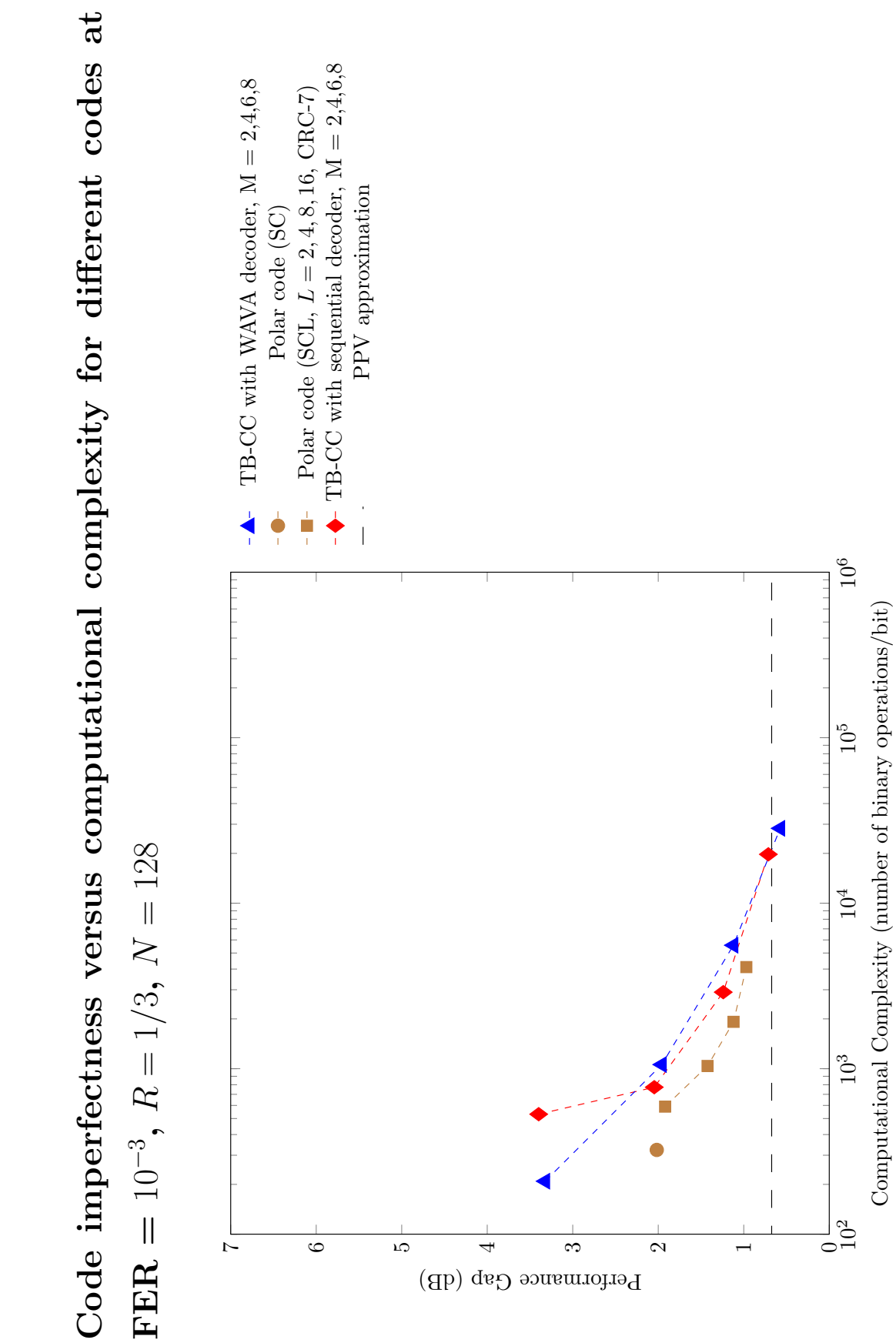

Figure C.68: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 1/3$, $N = 128$

Code imperfectness versus computational complexity for different codes at FER $= 10^{-4}$, $R = 1/3$, $N = 128$

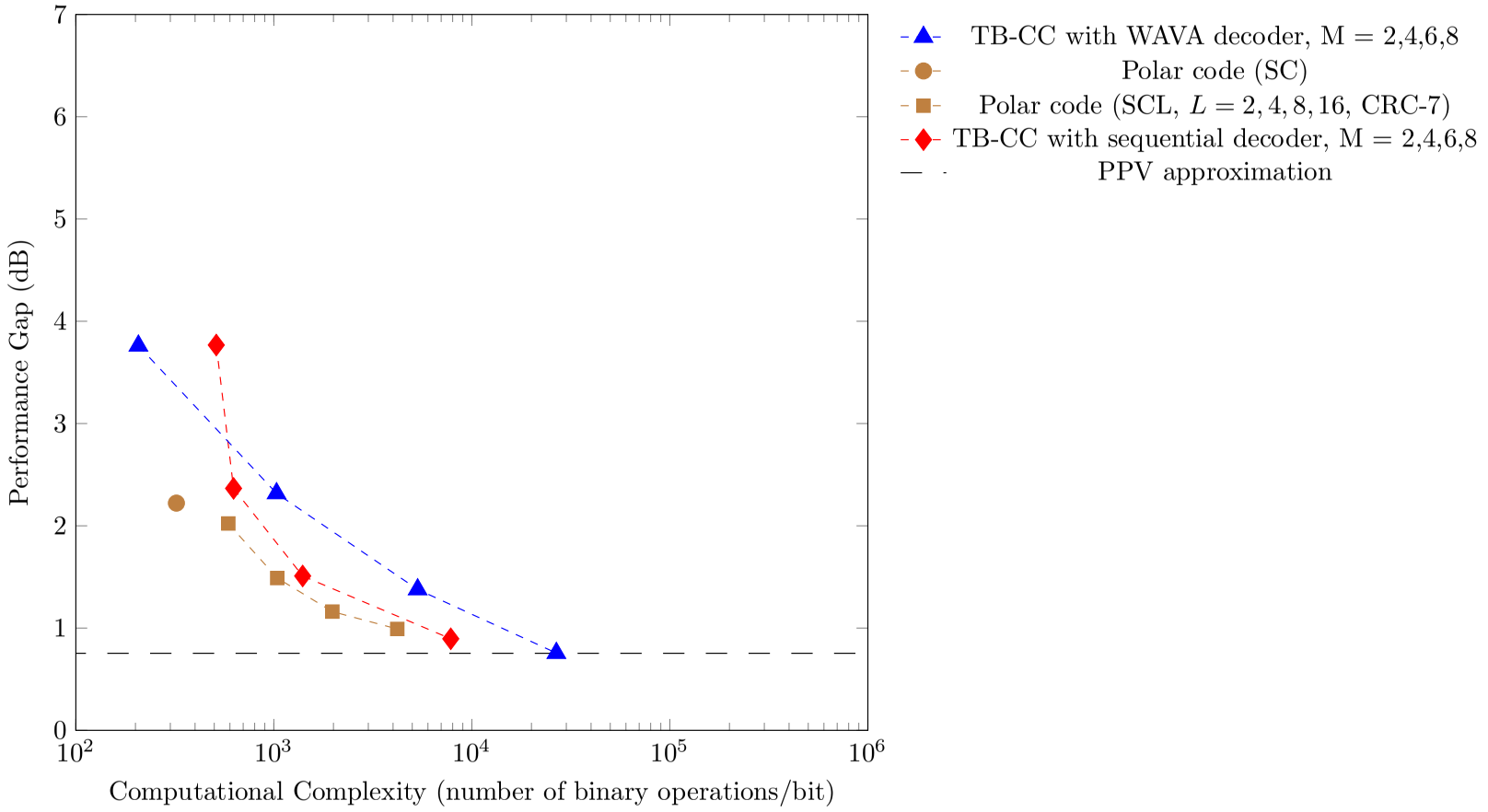

Figure C.69: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 1/3$, $N = 128$

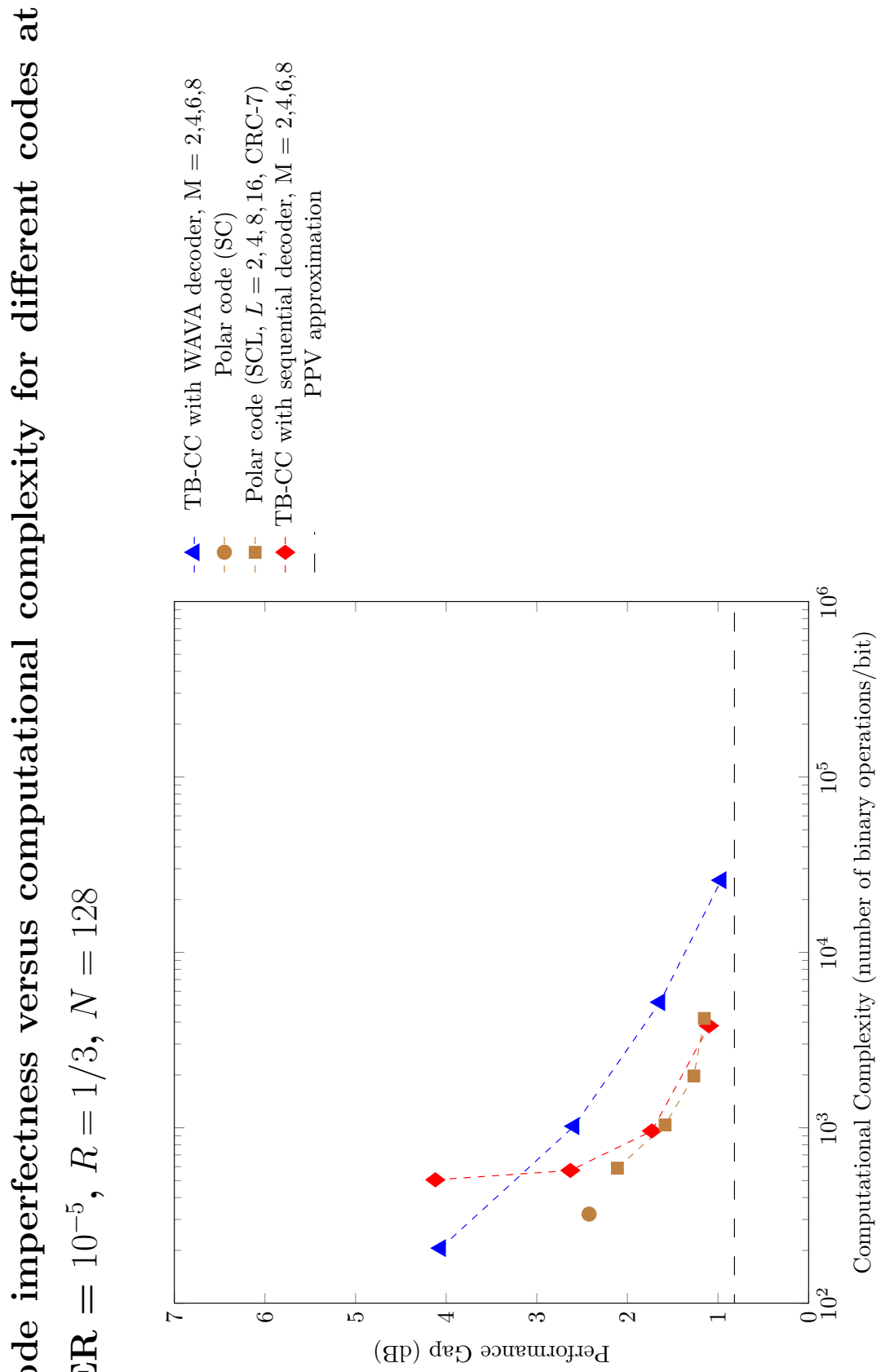

Figure C.70: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 1/3$, $N = 128$

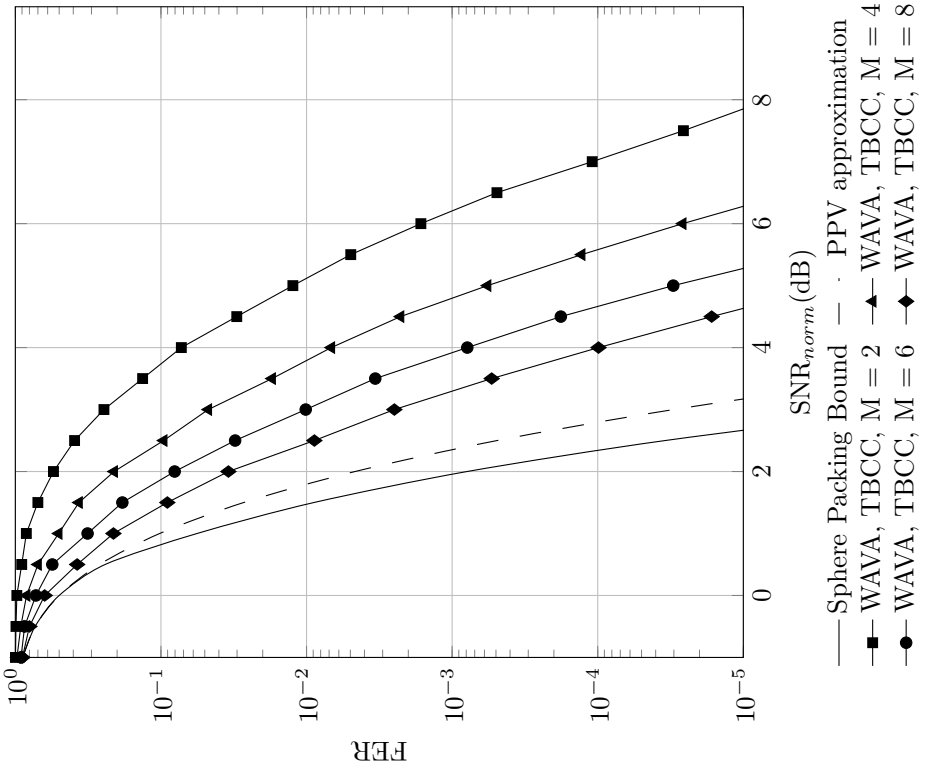

Figure C.71: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/3$, $N = 256$, maximum number of iterations = 4.

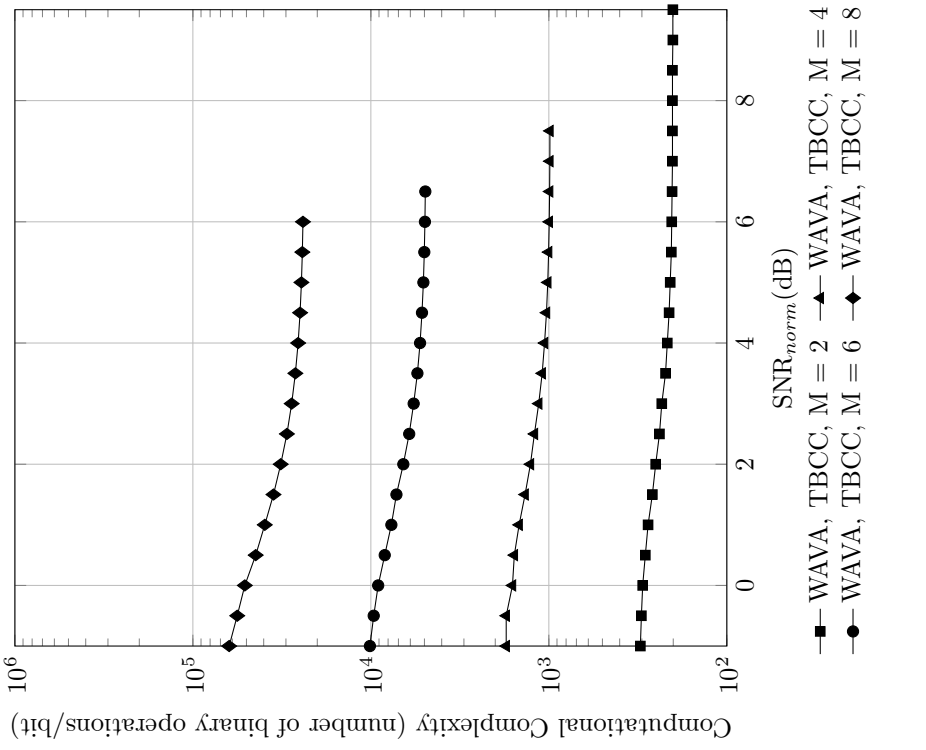

Figure C.72: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/3$, $N = 256$, maximum number of iterations = 4

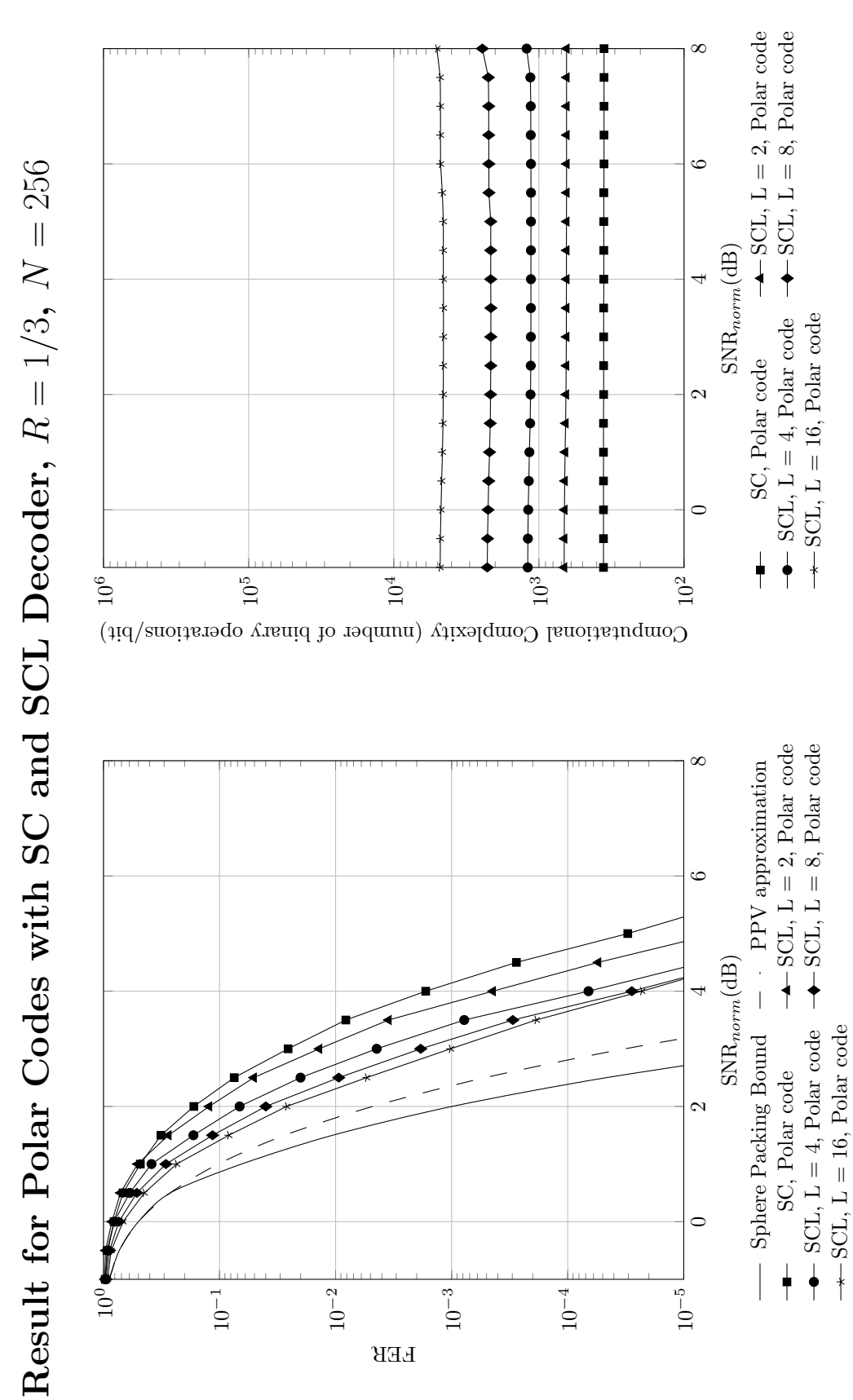

Figure C.73: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/3$, $N = 256$

Figure C.74: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/3$, $N = 256$

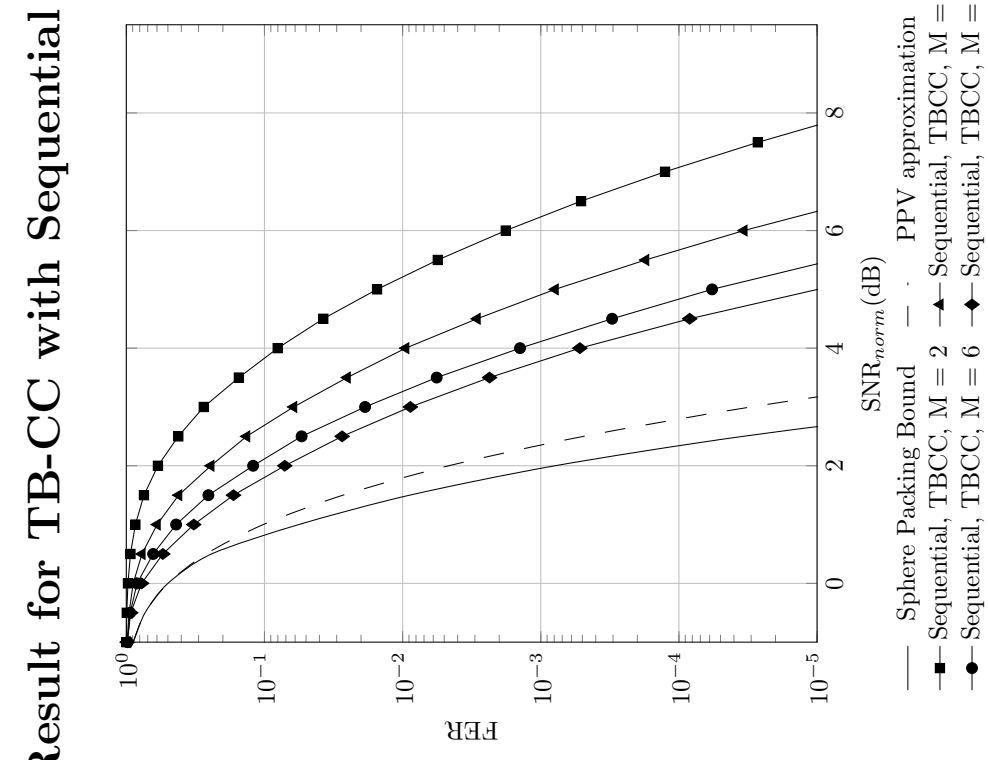

Figure C.75: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/3$, $N = 256$

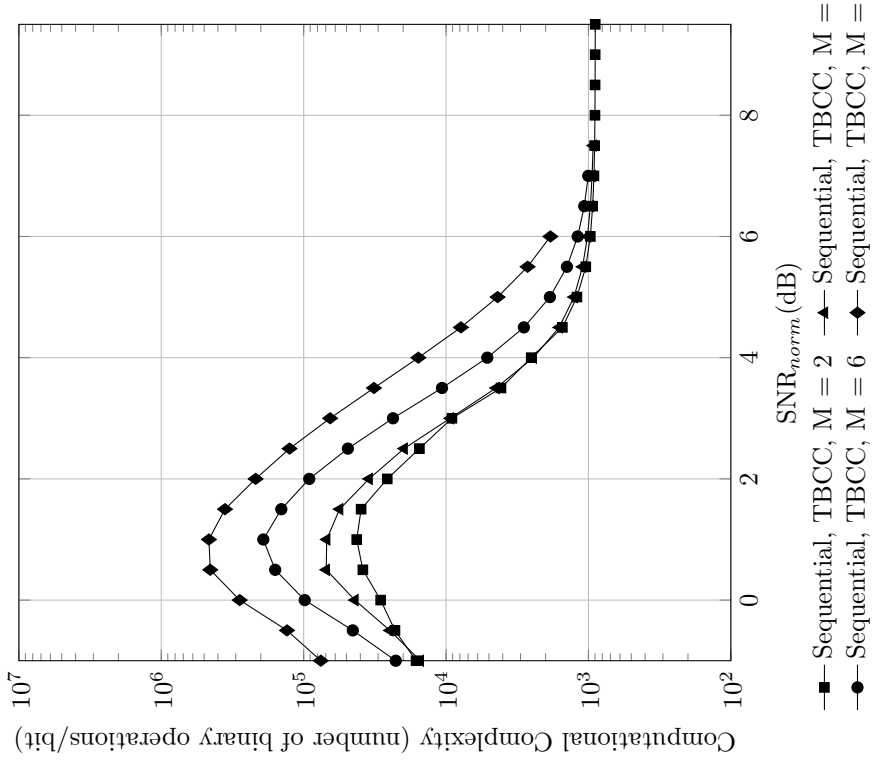

Figure C.76: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/3$, $N = 256$

Code imperfectness versus computational complexity for different codes at FER $= 10^{-2}$, $R = 1/3$, $N = 256$

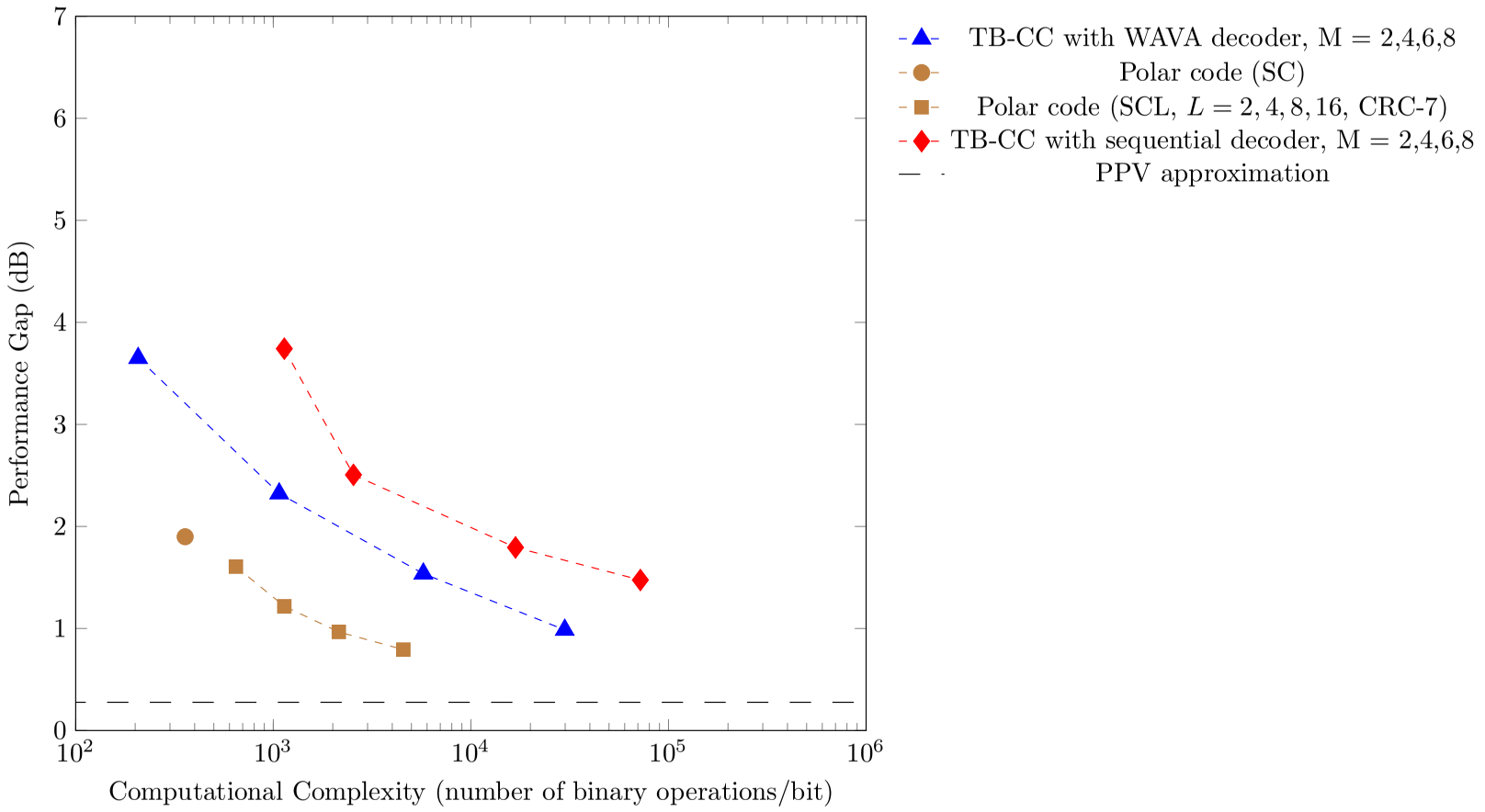

Figure C.77: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 1/3$, $N = 256$

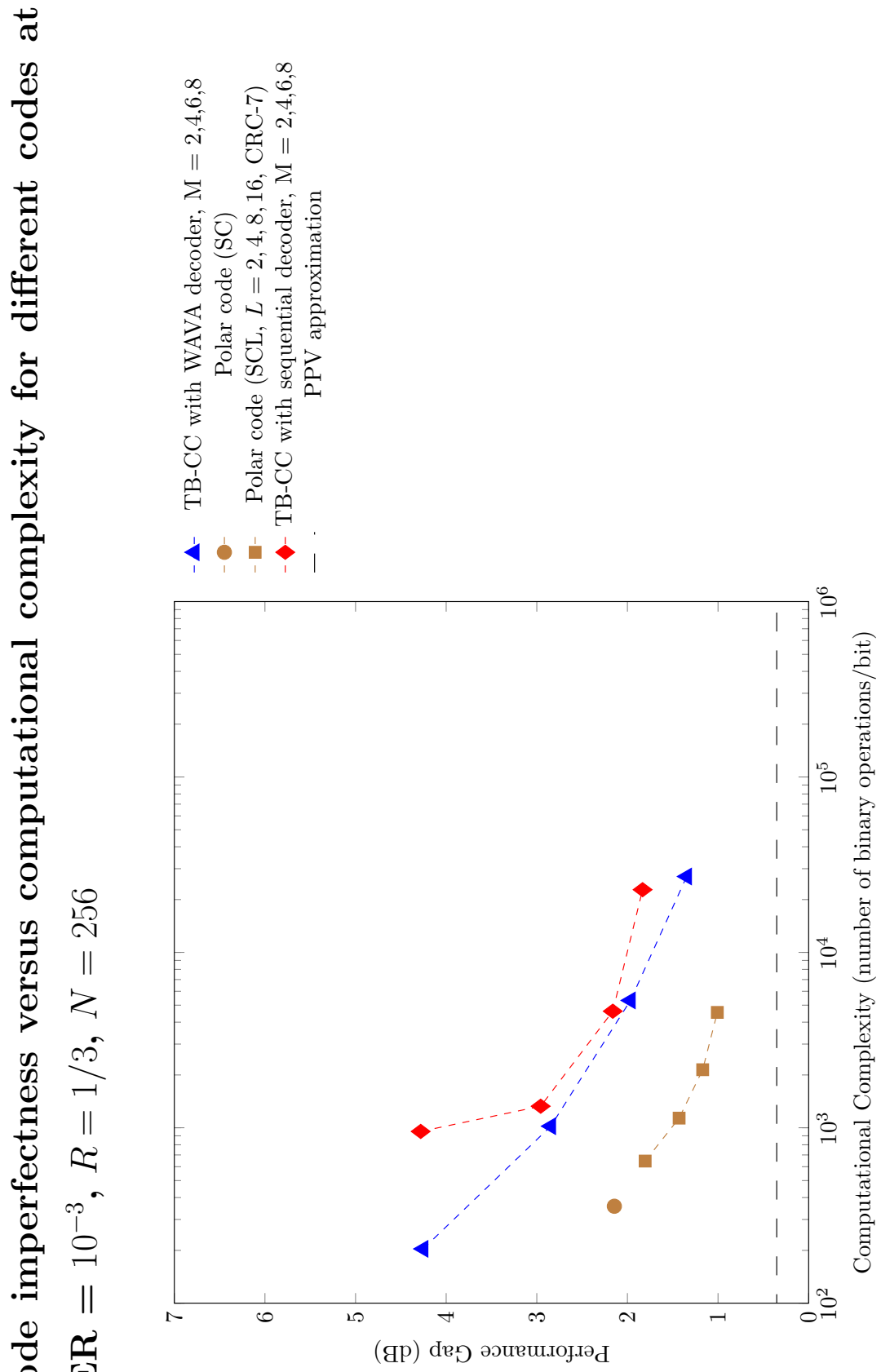

Figure C.78: Code imperfectness versus computational complexity at FER = 10^{-3} for different codes with $R = 1/3$, $N = 256$

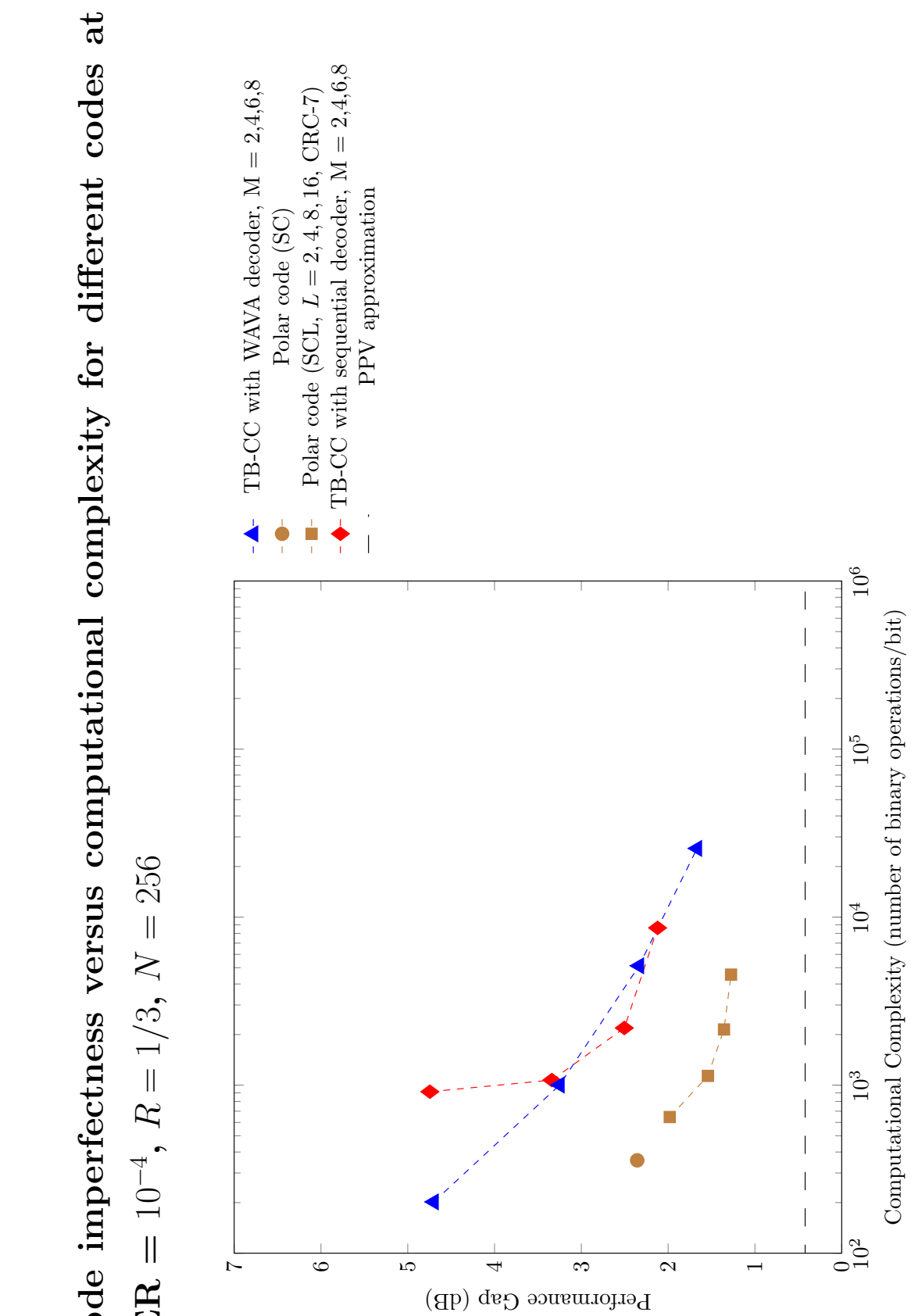

Figure C.79: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 1/3$, $N = 256$

Code imperfectness versus computational complexity for different codes at FER $= 10^{-5}$, $R = 1/3$, $N = 256$

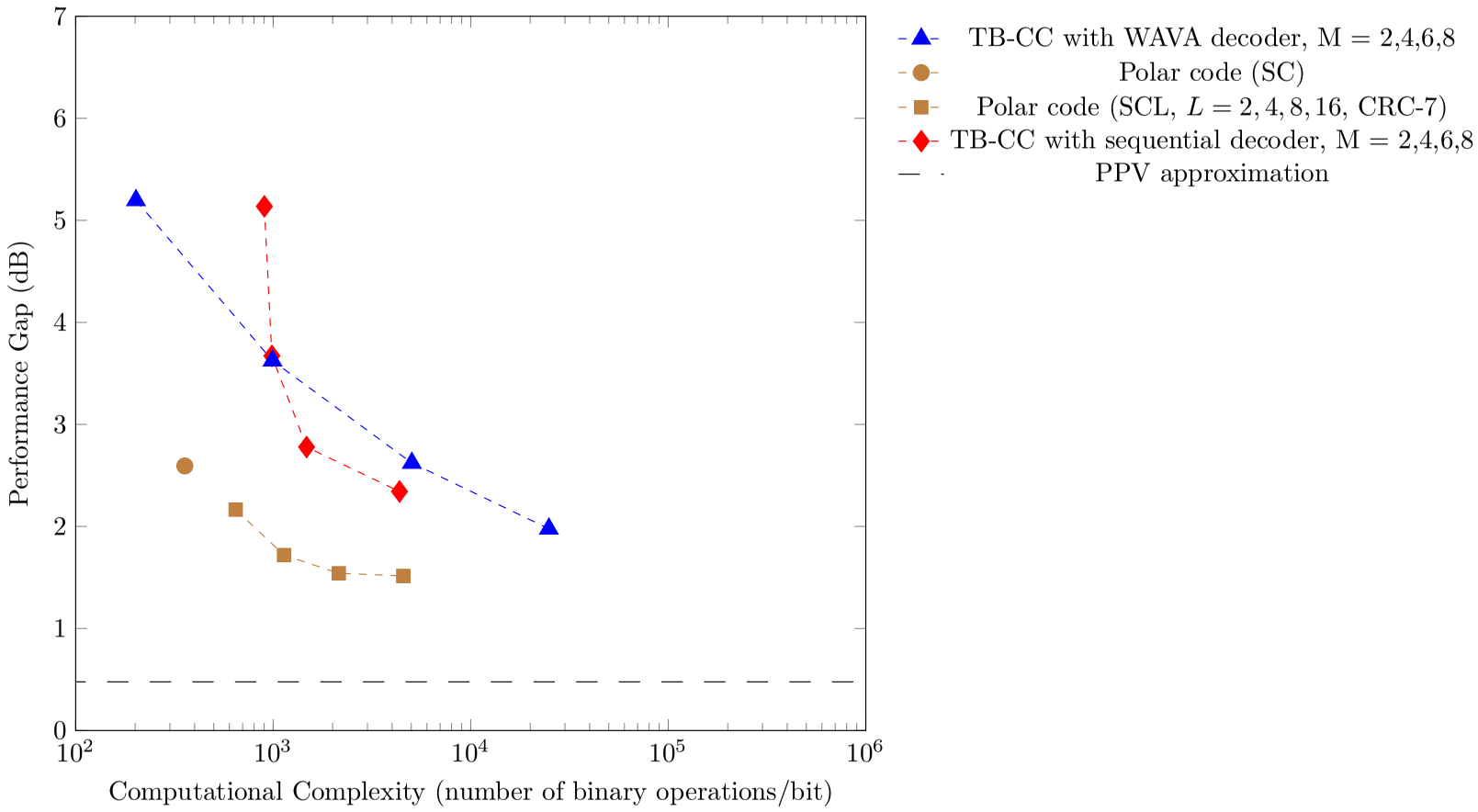

Figure C.80: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 1/3$, $N = 256$

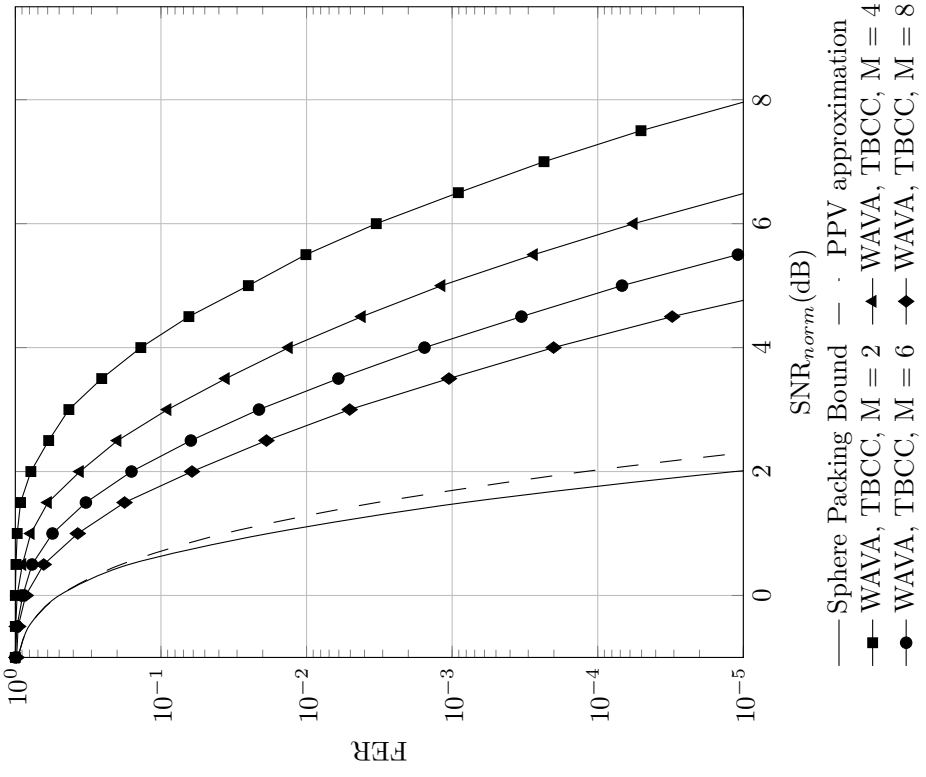

Figure C.81: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/3$, $N = 512$, maximum number of iterations = 4.

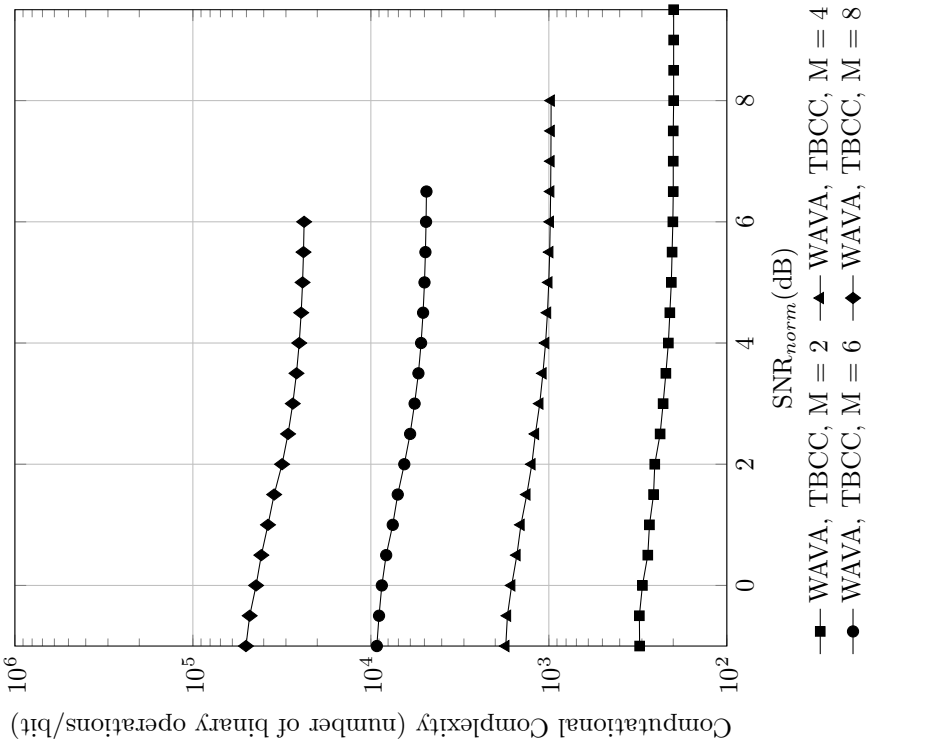

Figure C.82: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/3$, $N = 512$, maximum number of iterations = 4

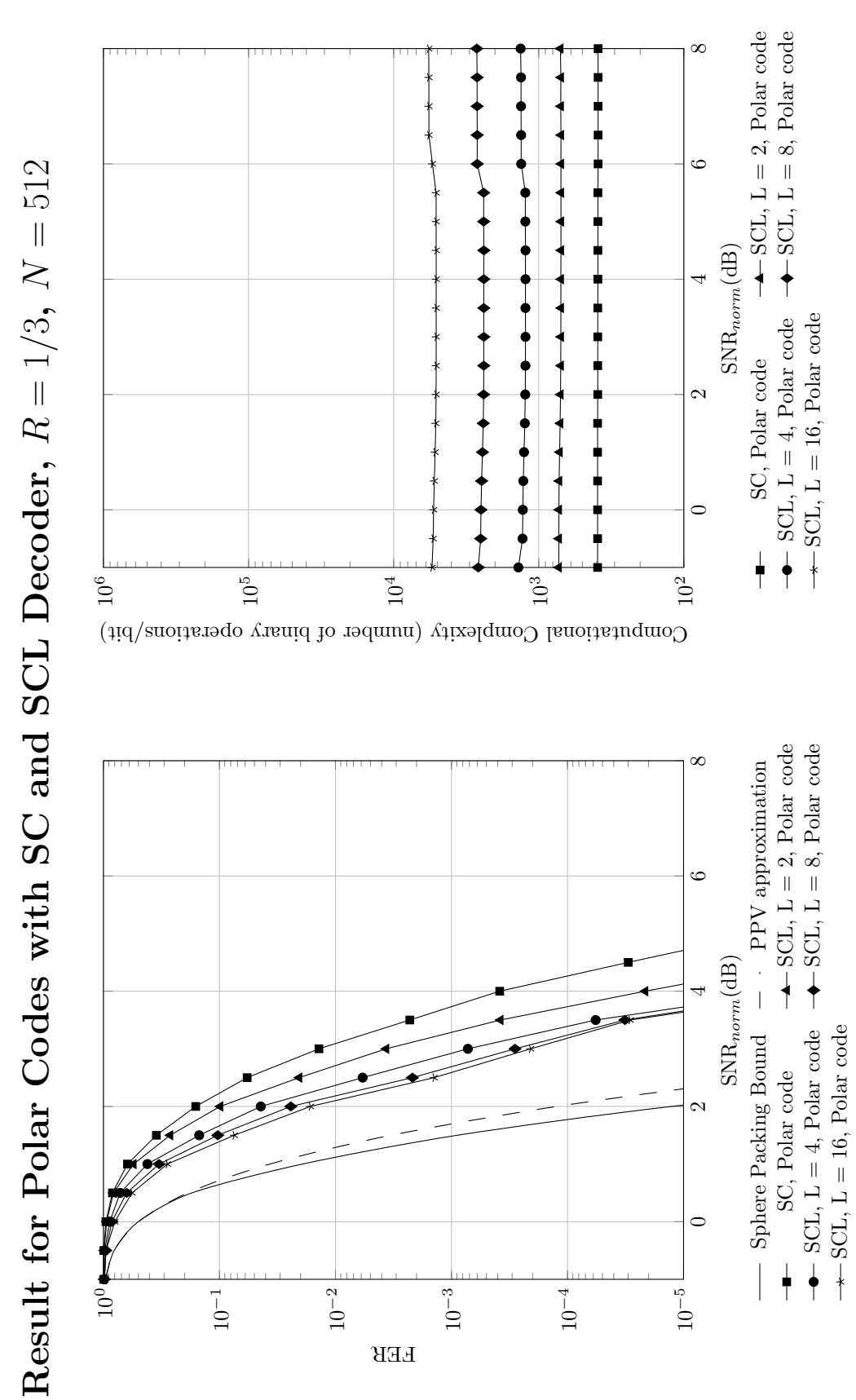

Figure C.83: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/3$, $N = 512$

Figure C.84: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/3$, $N = 512$

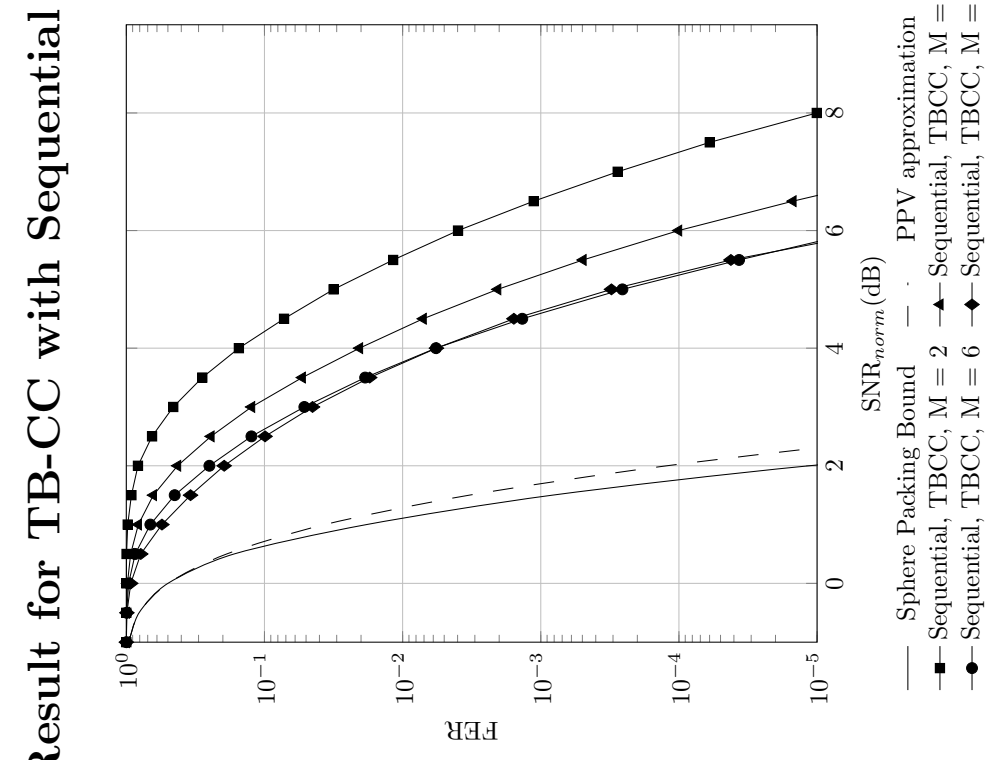

Figure C.85: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/3$, $N = 512$

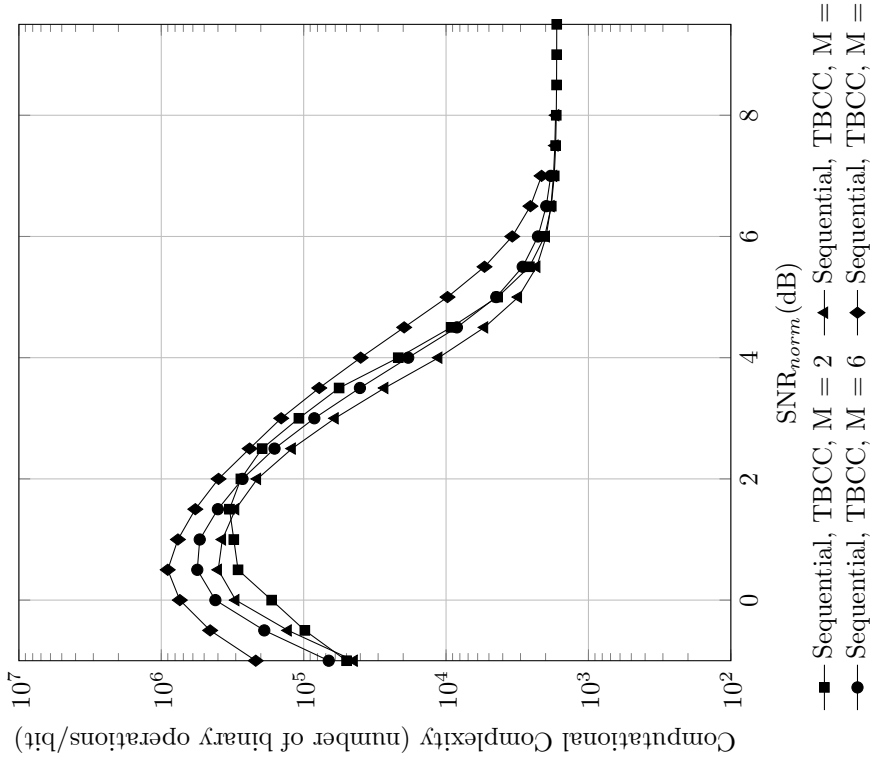

Figure C.86: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/3$, $N = 512$

Code imperfectness versus computational complexity for different codes at FER $= 10^{-2}$, $R = 1/3$, $N = 512$

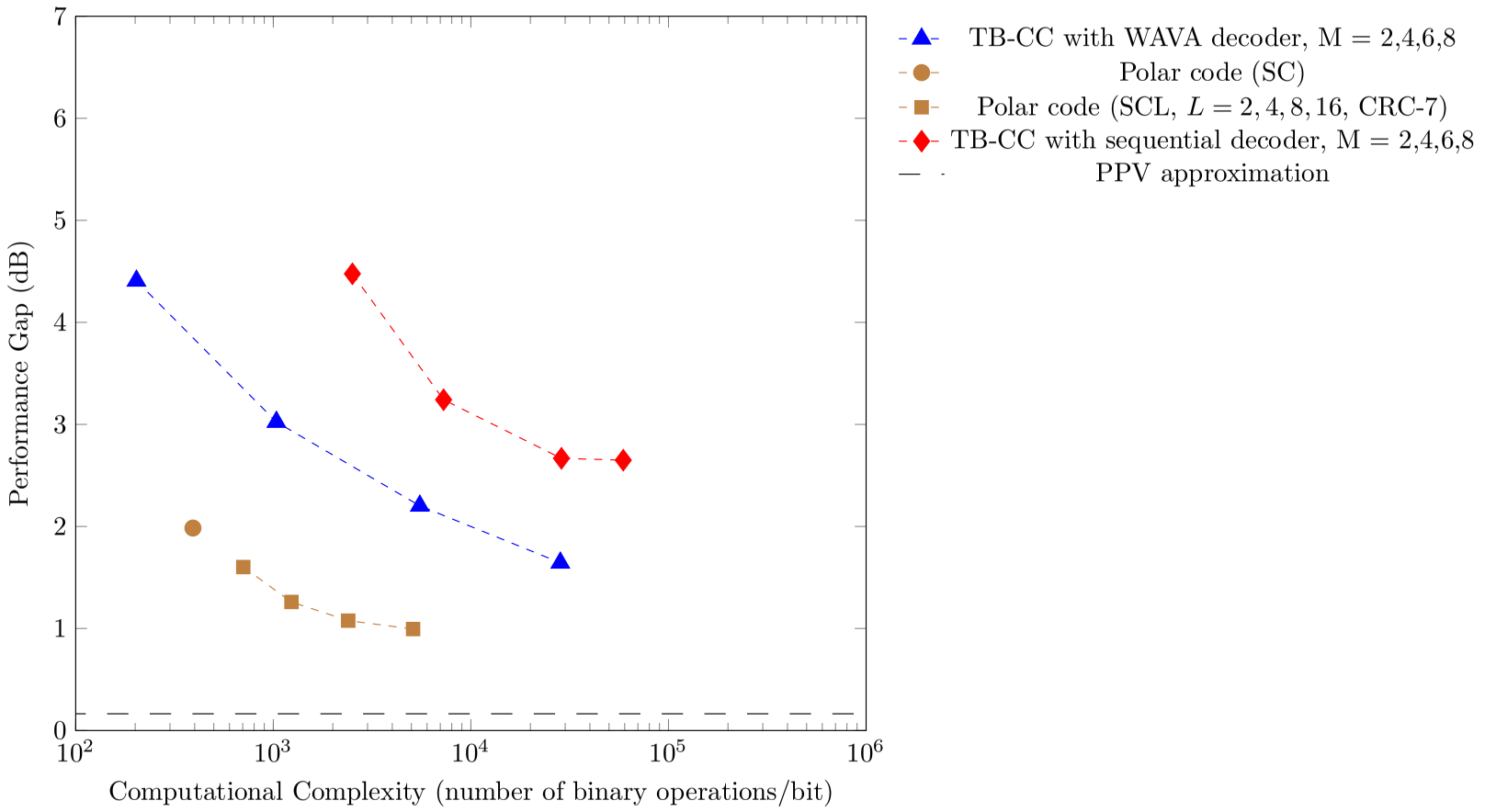

Figure C.87: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 1/3$, $N = 512$

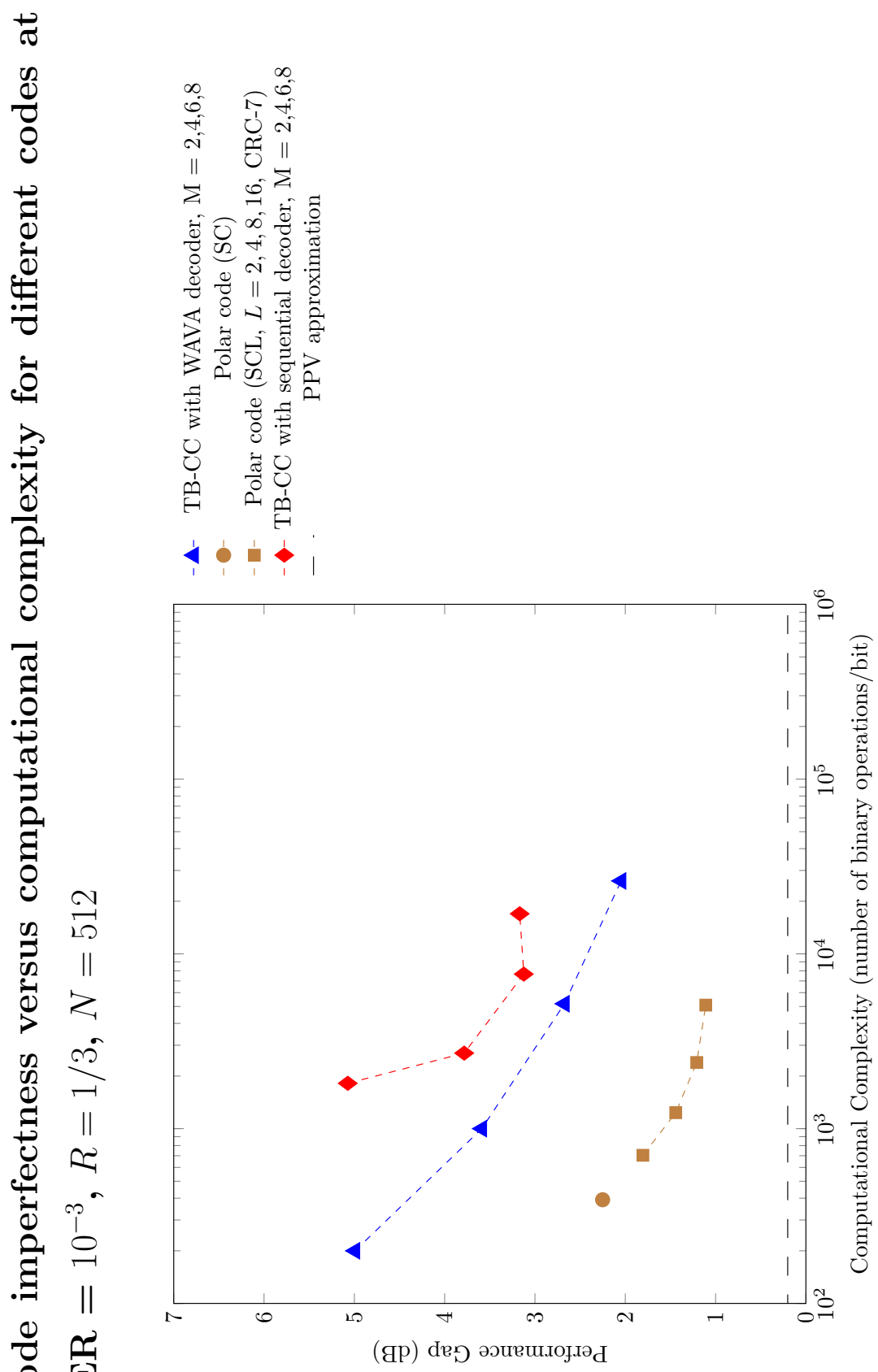

Figure C.88: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 1/3$, $N = 512$

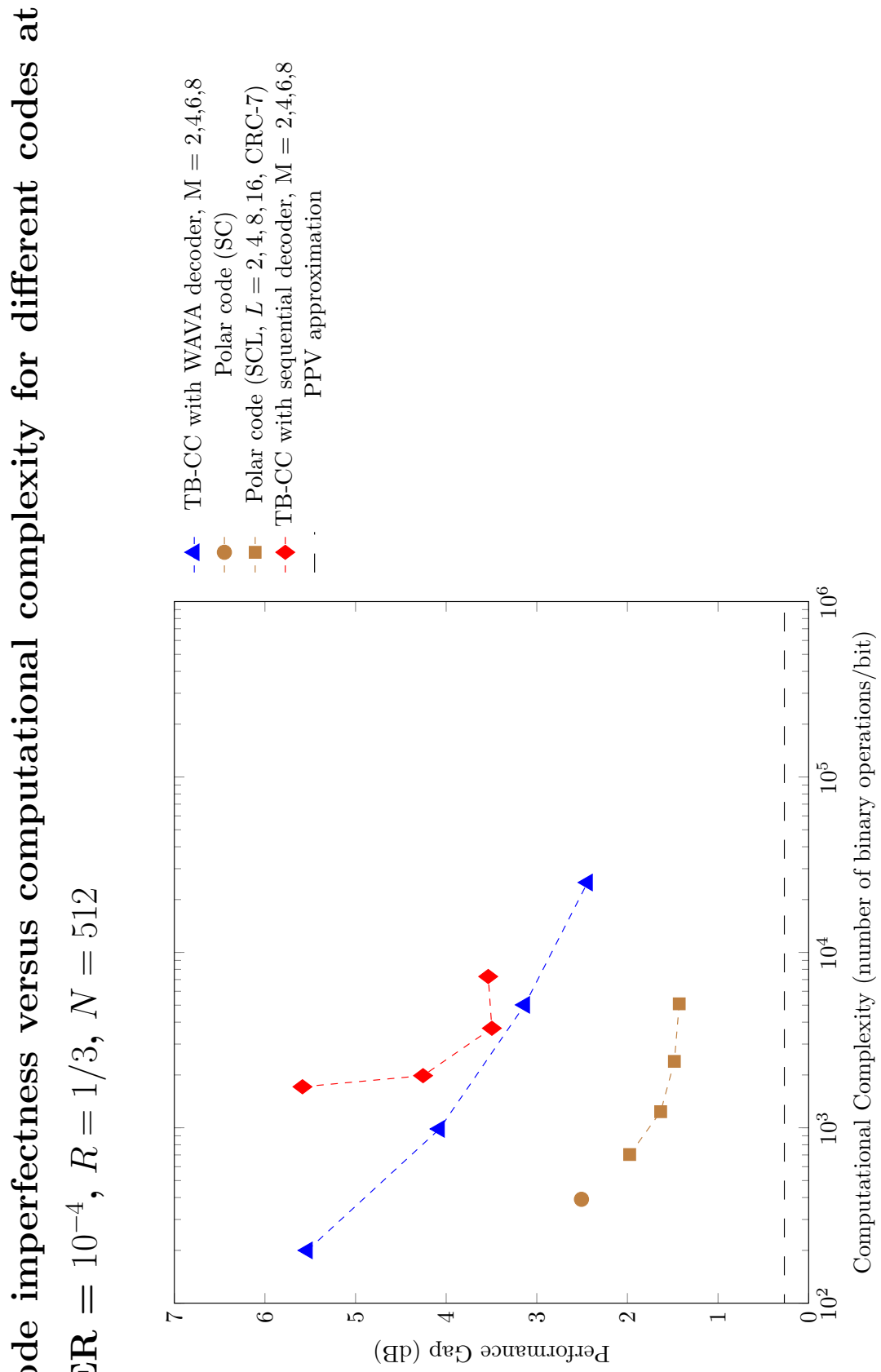

Figure C.89: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 1/3$, $N = 512$

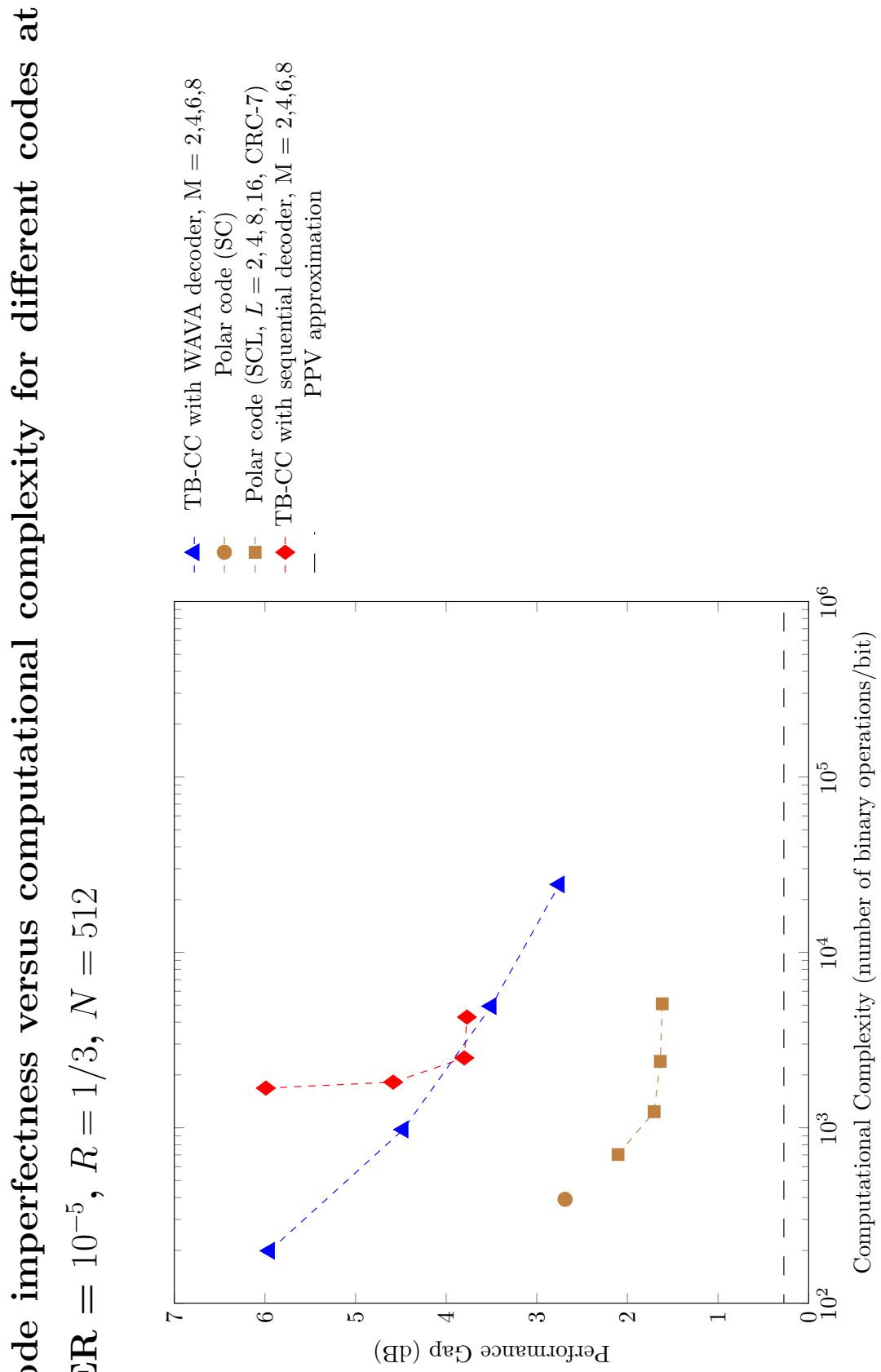

Figure C.90: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 1/3$, $N = 512$

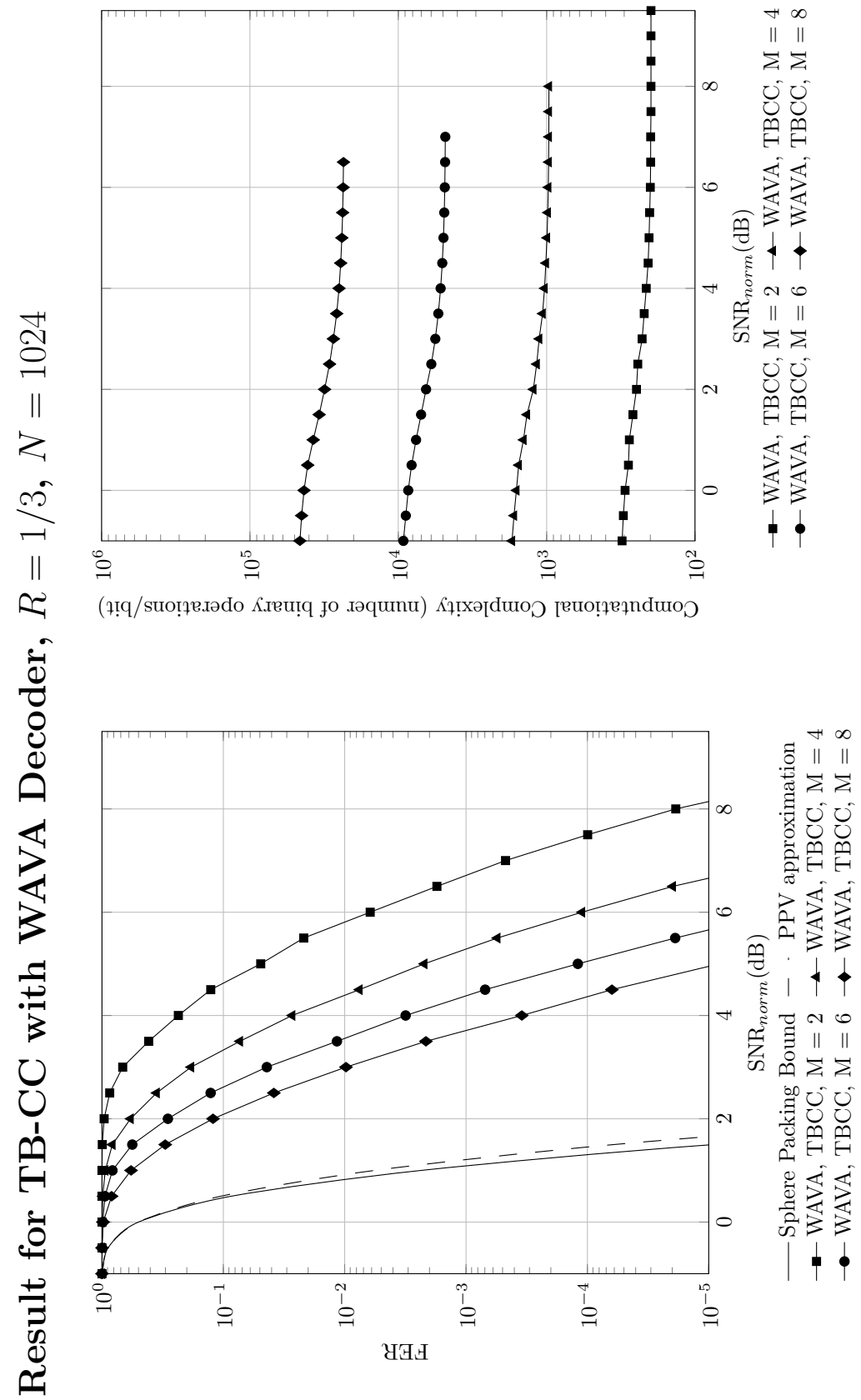

Figure C.91: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/3$, $N = 1024$, maximum number of iterations = 4.

Figure C.92: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/3$, $N = 1024$, maximum number of iterations = 4

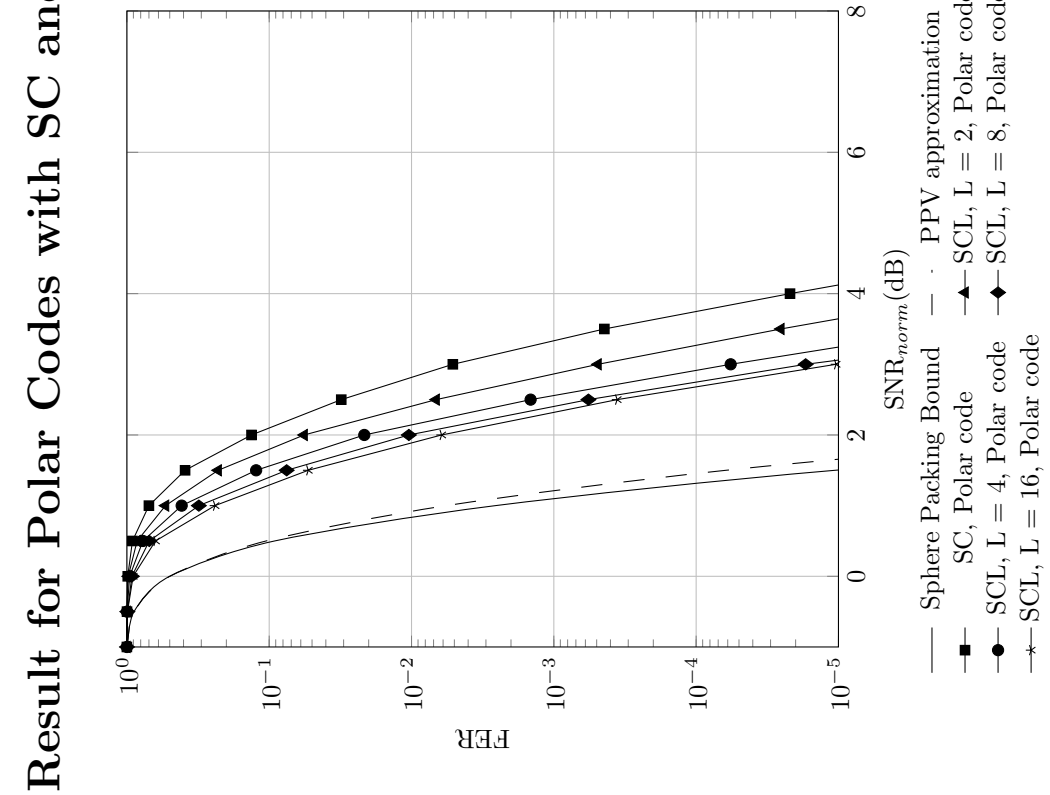

Figure C.93: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/3$, $N = 1024$

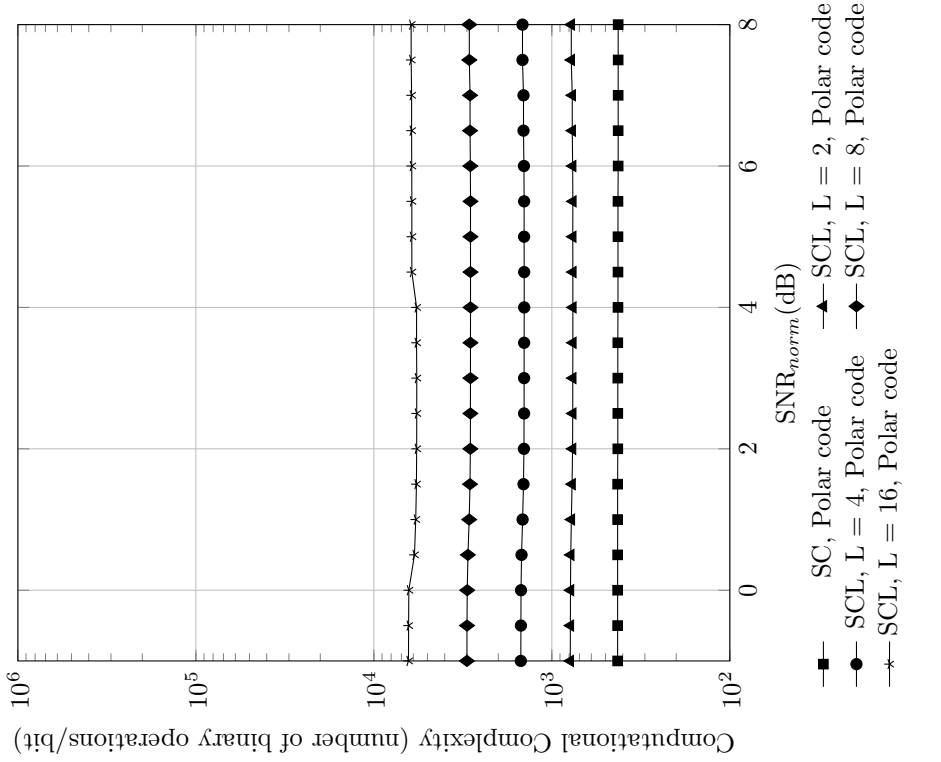

Figure C.94: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/3$, $N = 1024$

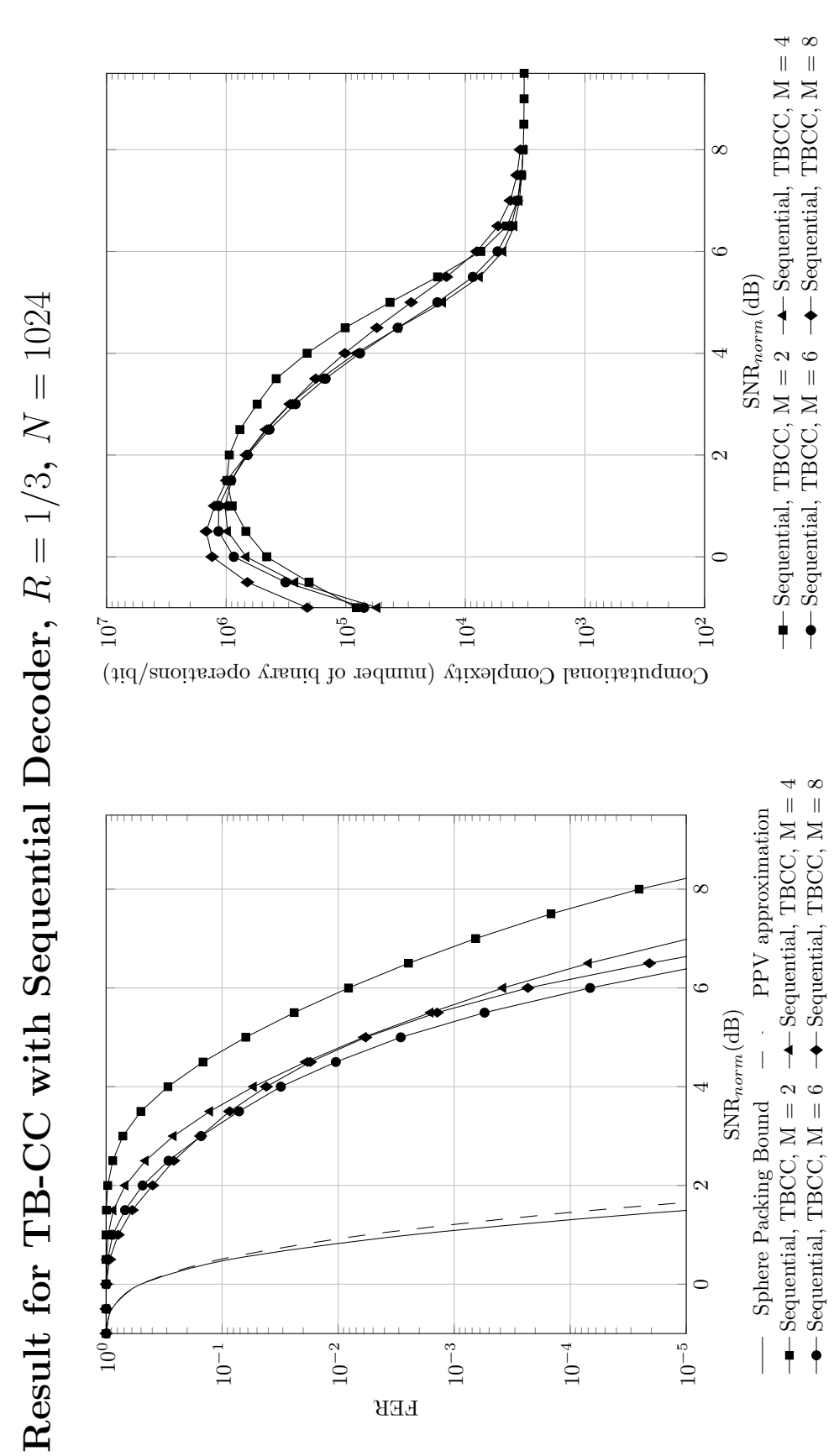

Figure C.95: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/3$, $N = 1024$

Figure C.96: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/3$, $N = 1024$

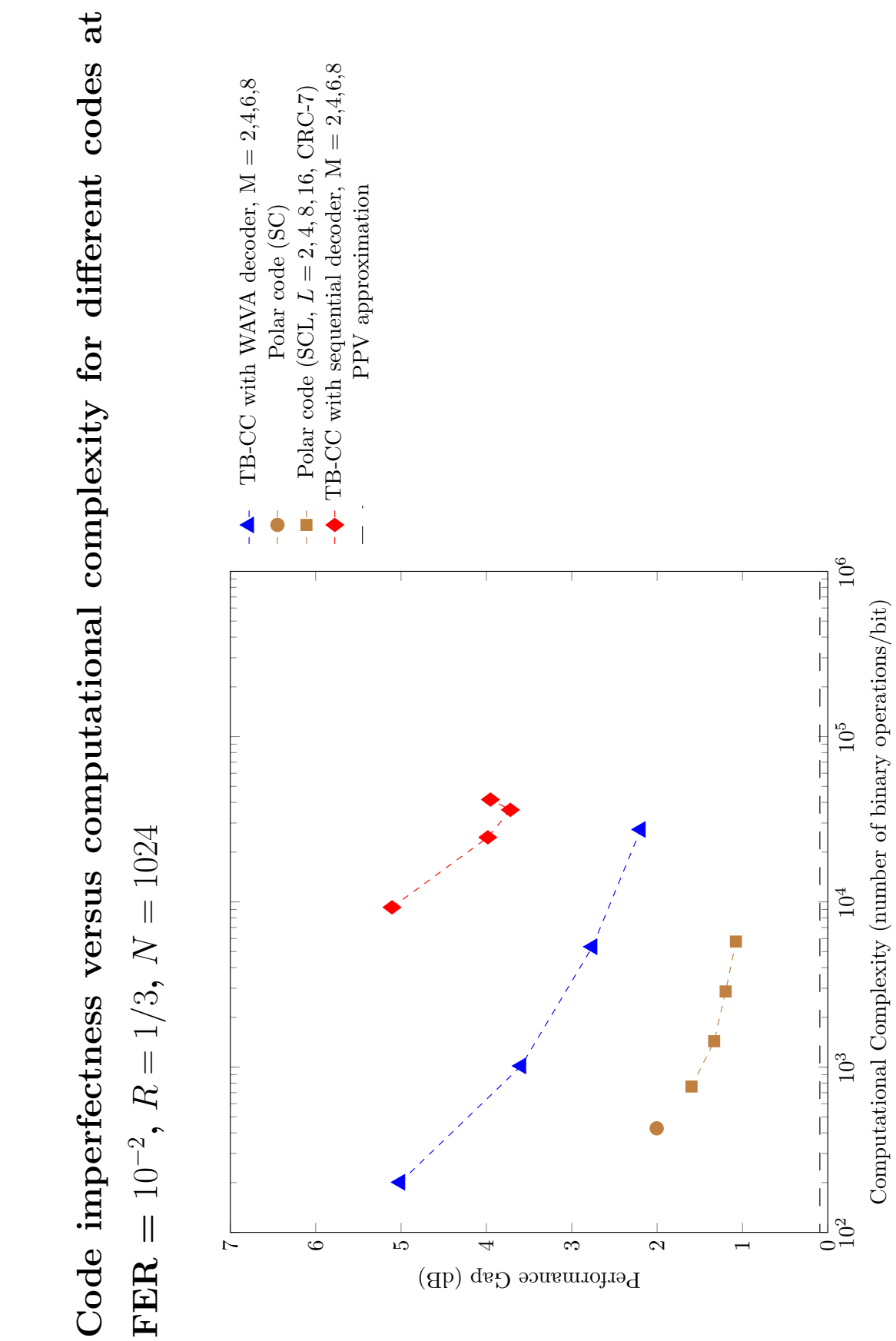

Figure C.97: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 1/3$, $N = 1024$

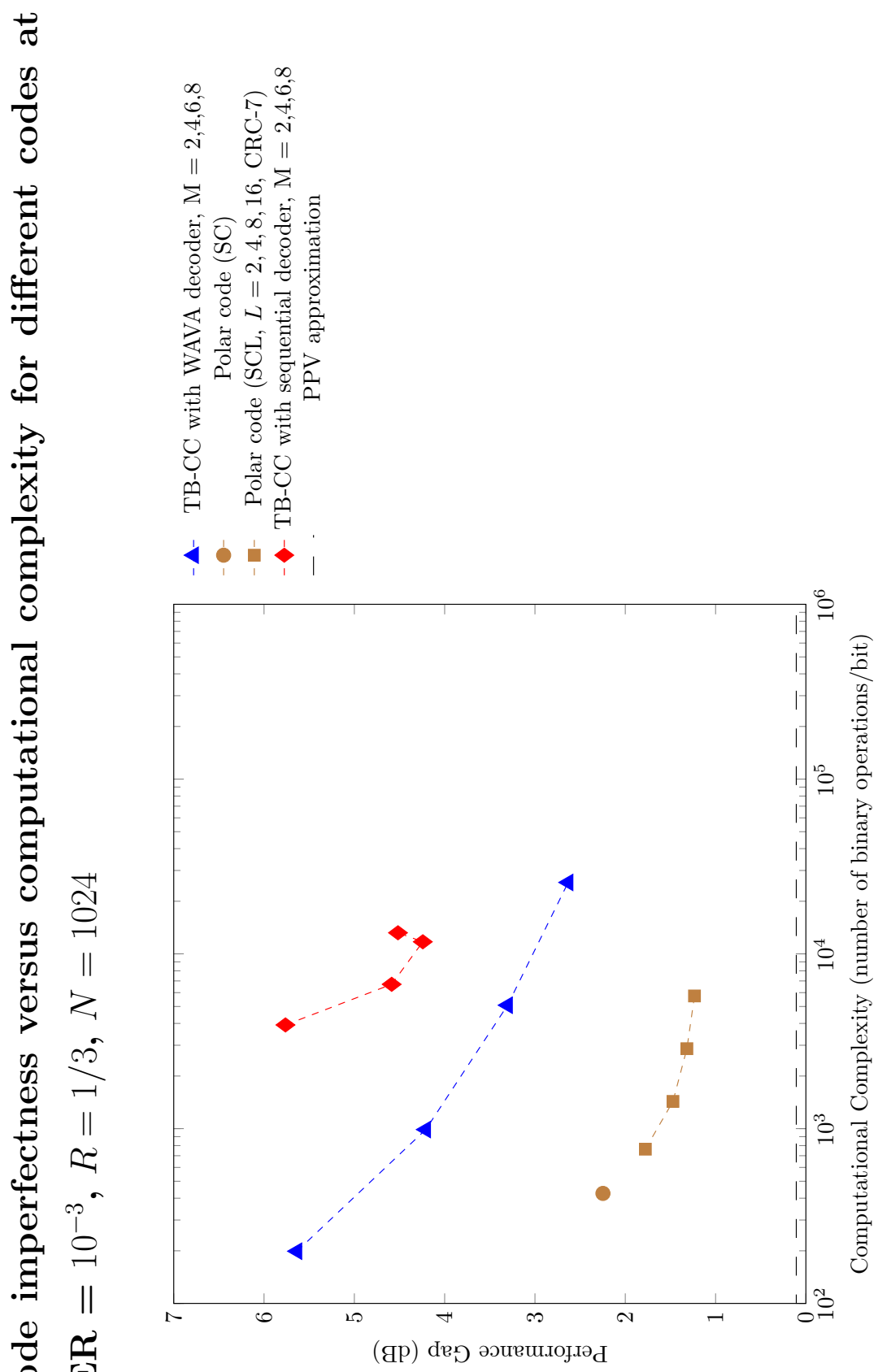

Figure C.98: Code imperfectness versus computational complexity at FER = 10^{-3} for different codes with $R = 1/3$, $N = 1024$

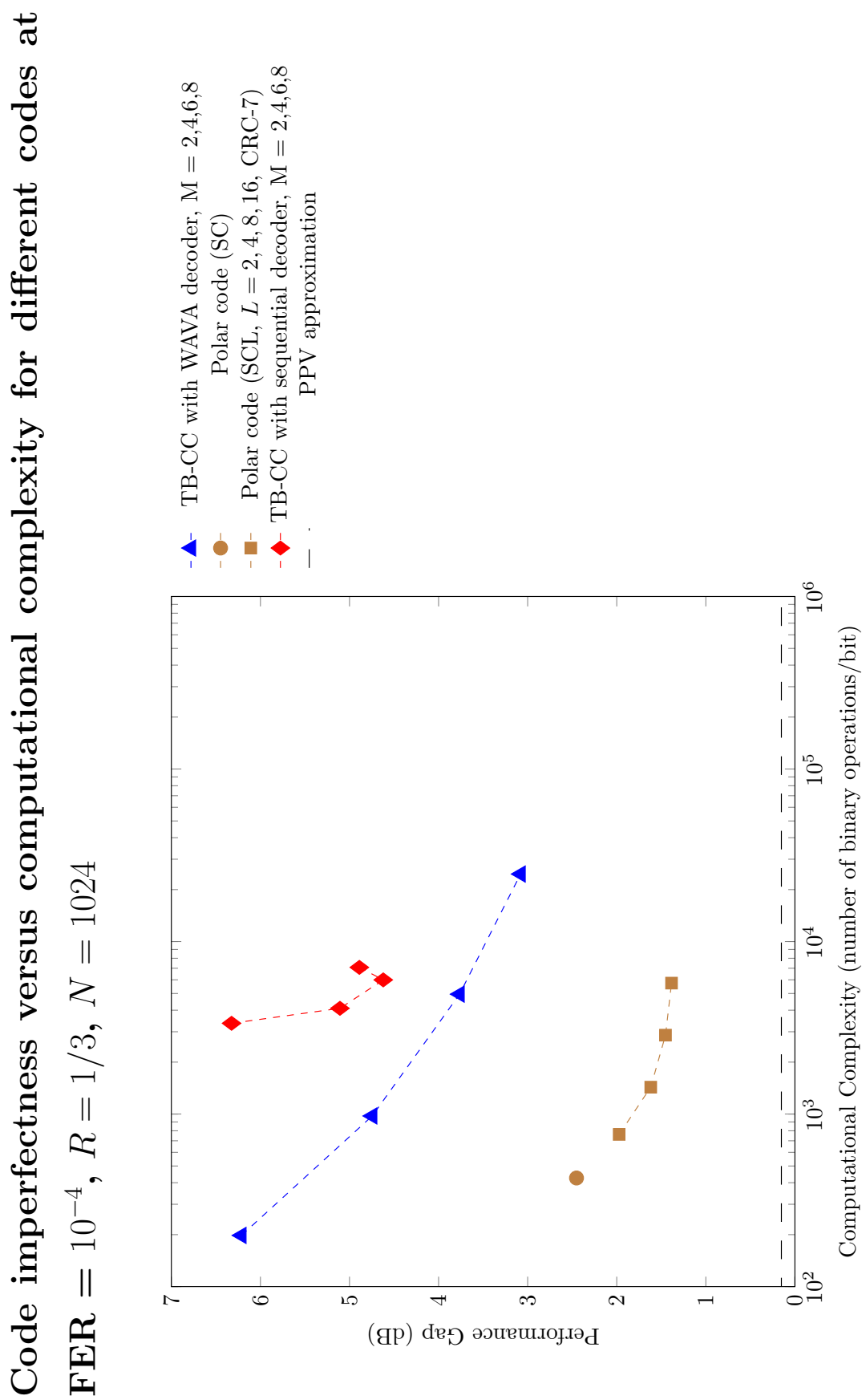

Figure C.99: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 1/3$, $N = 1024$

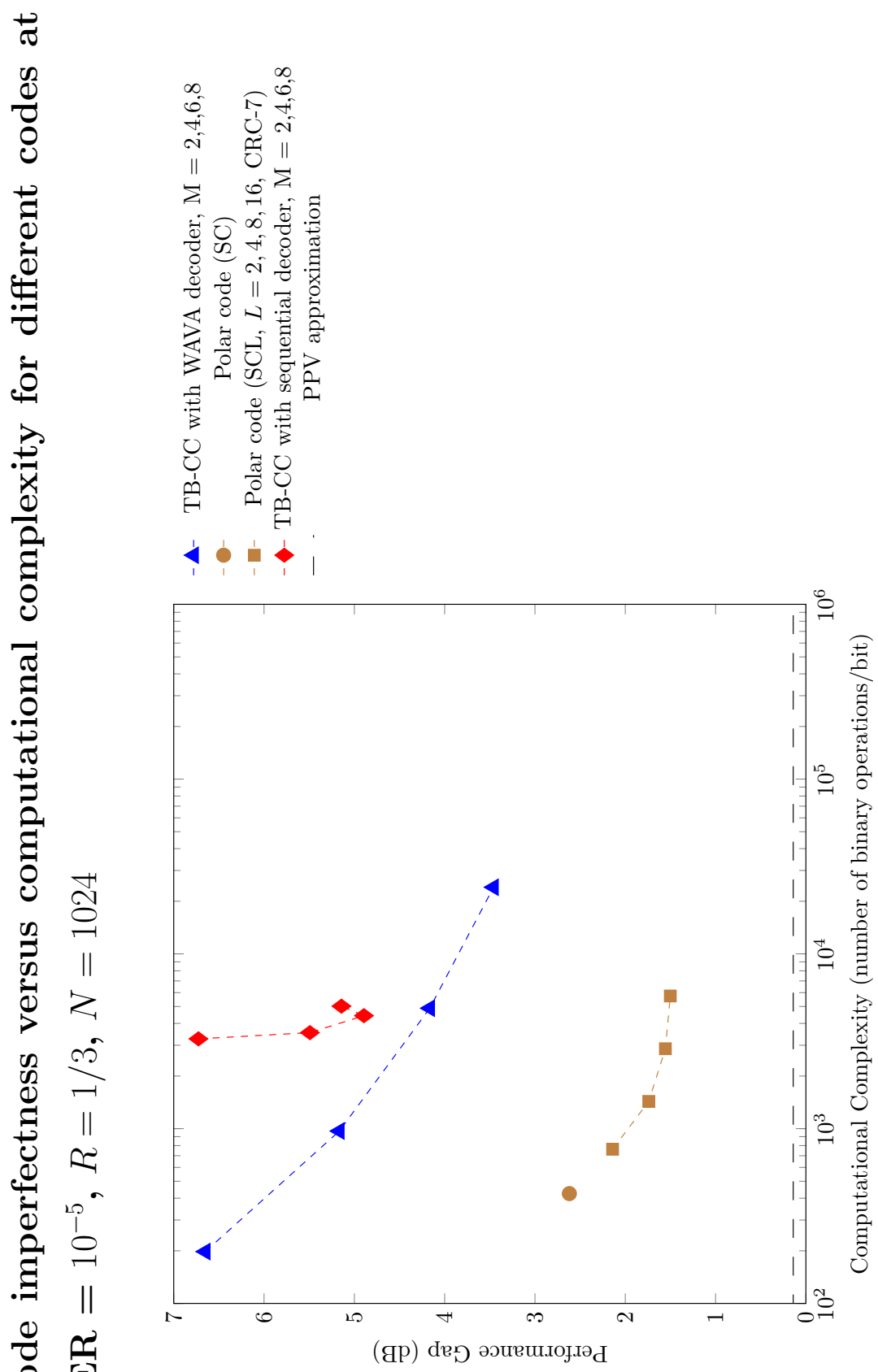

Figure C.100: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 1/3$, $N = 1024$

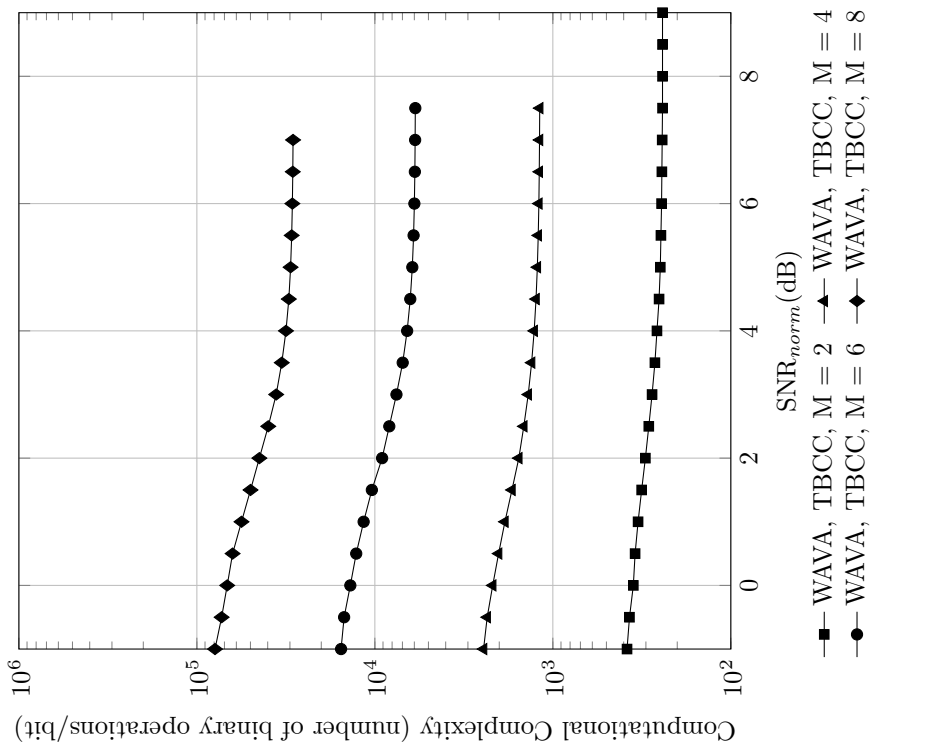

Figure C.102: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/4$, $N = 64$, maximum number of iterations = 4

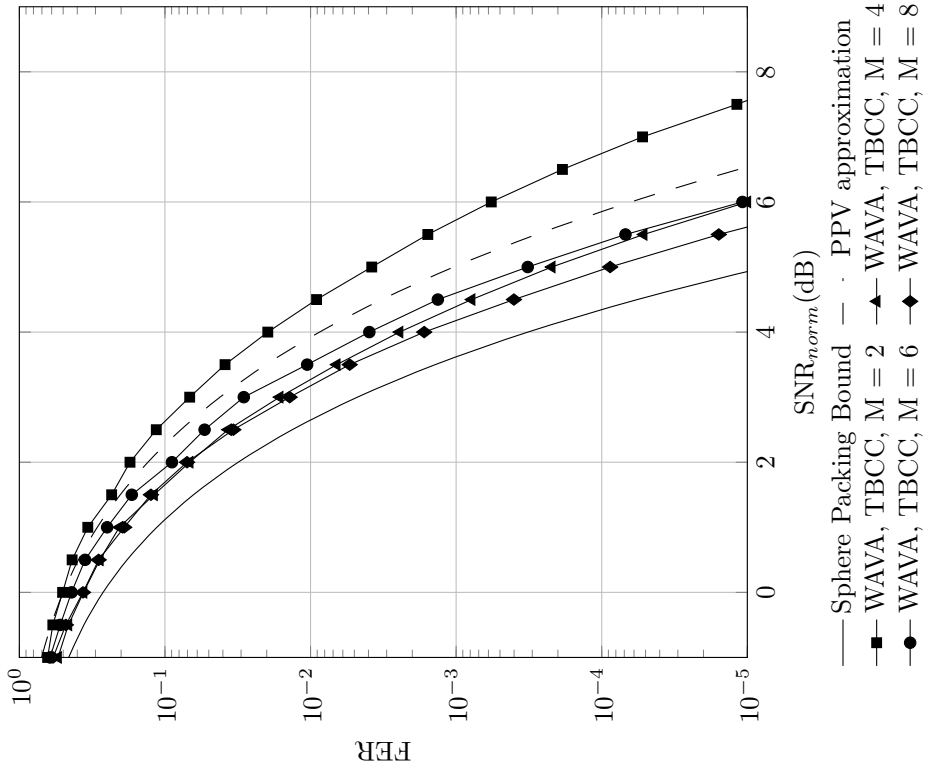

Figure C.101: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/4$, $N = 64$, maximum number of iterations = 4.

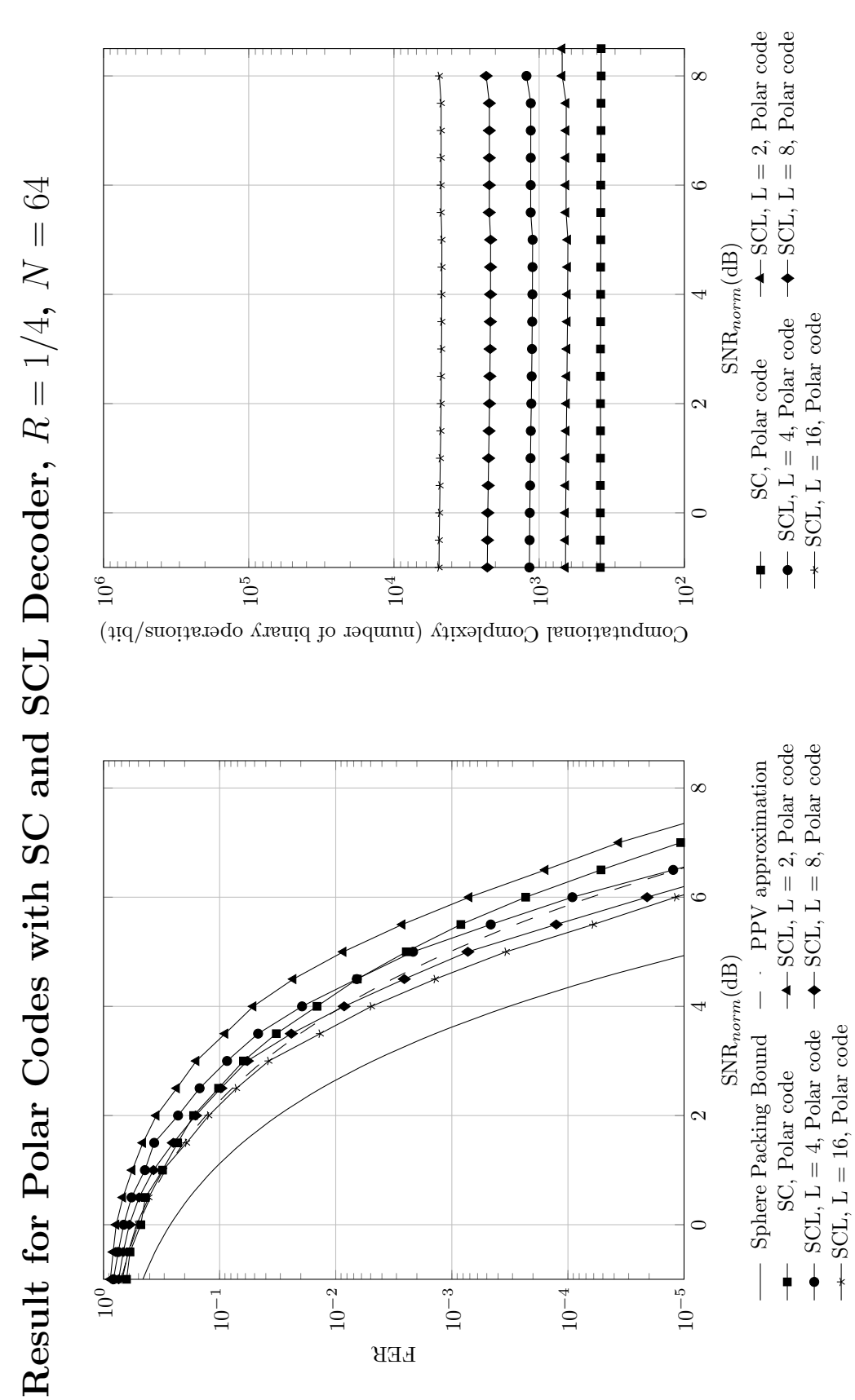

Figure C.103: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/4$, $N = 64$

Figure C.104: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/4$, $N = 64$

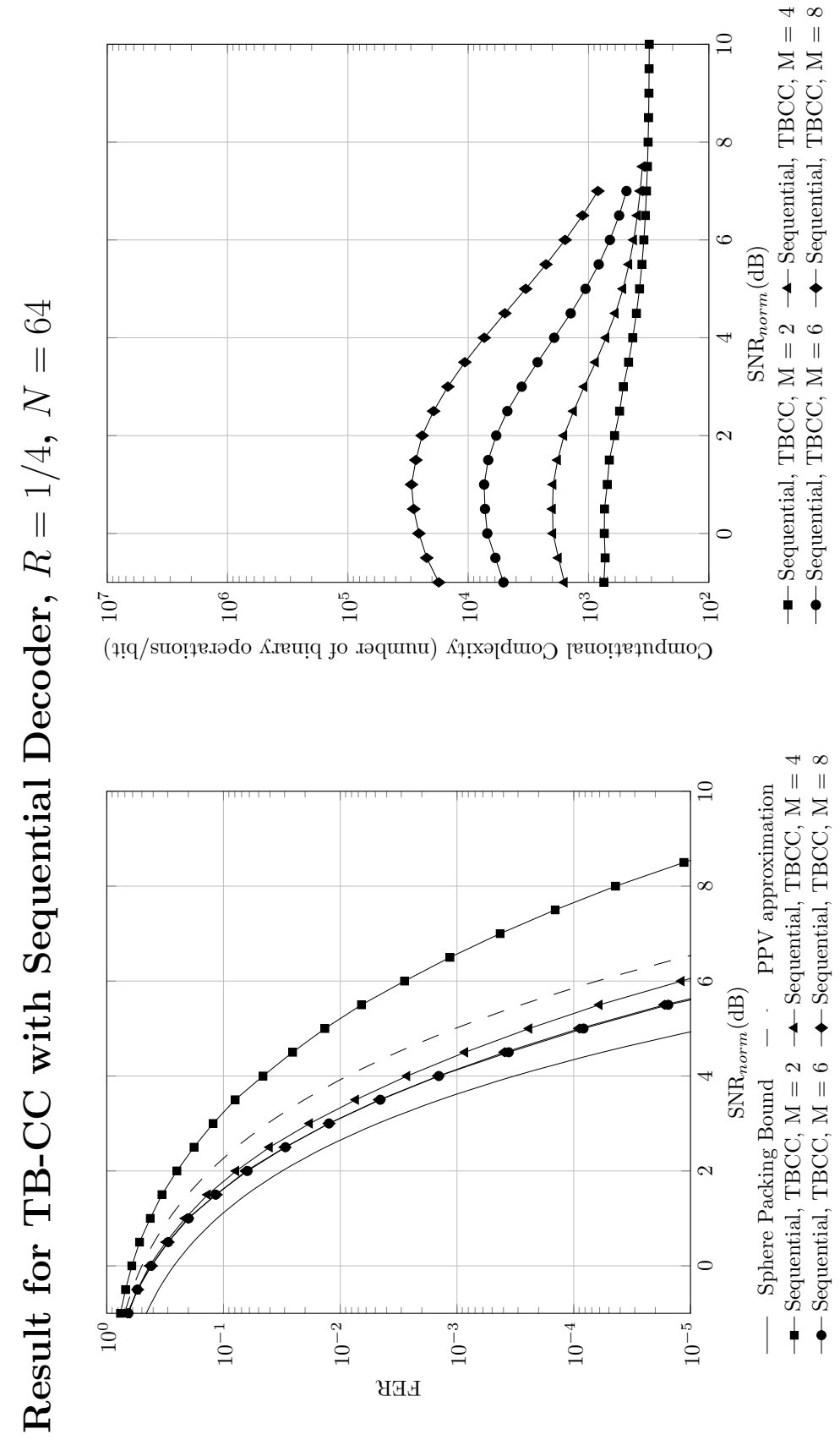

Figure C.105: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/4$, $N = 64$

Figure C.106: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/4$, $N = 64$

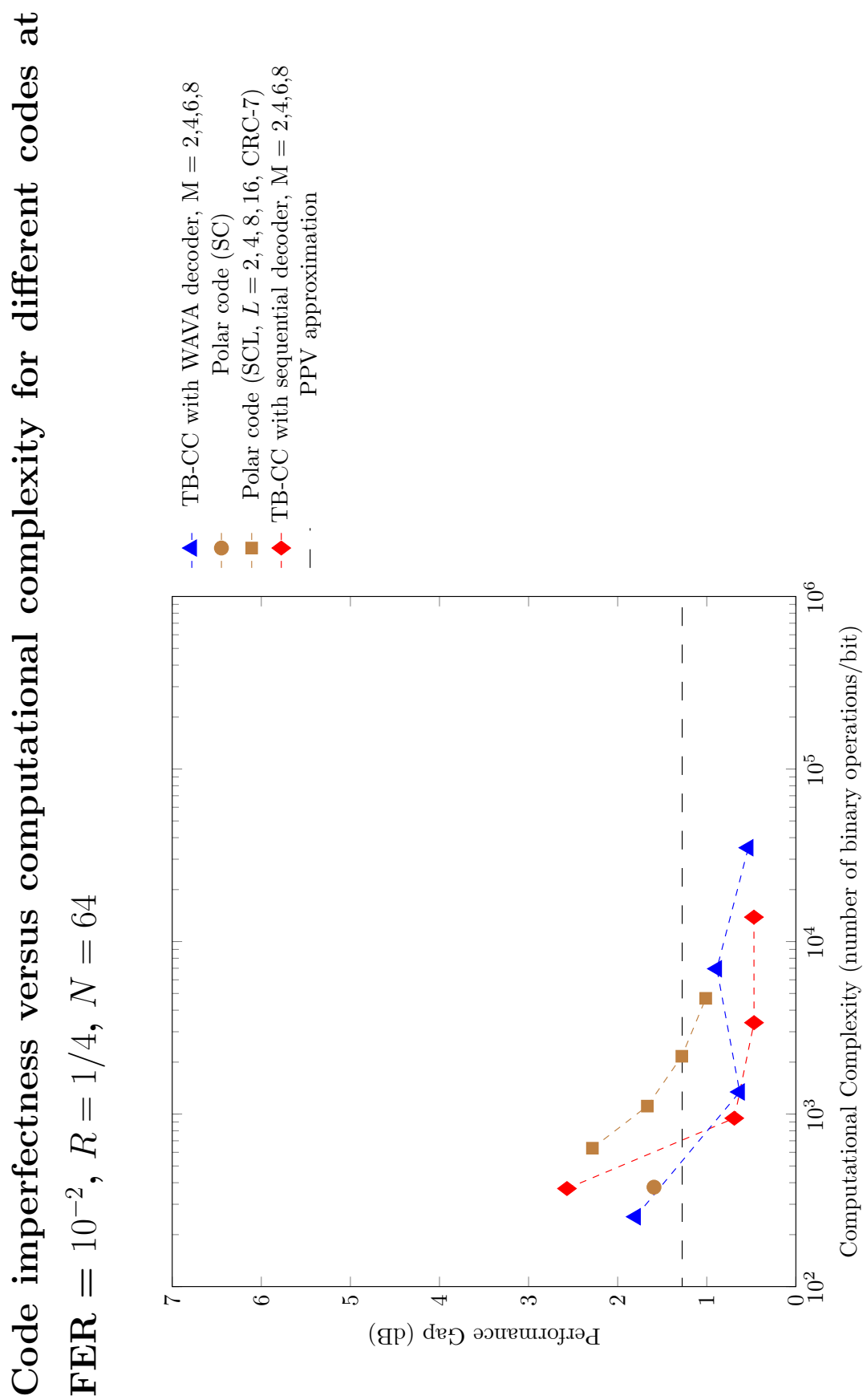

Figure C.107: Code imperfectness versus computational complexity at FER = 10^{-2} for different codes with $R = 1/4$, $N = 64$

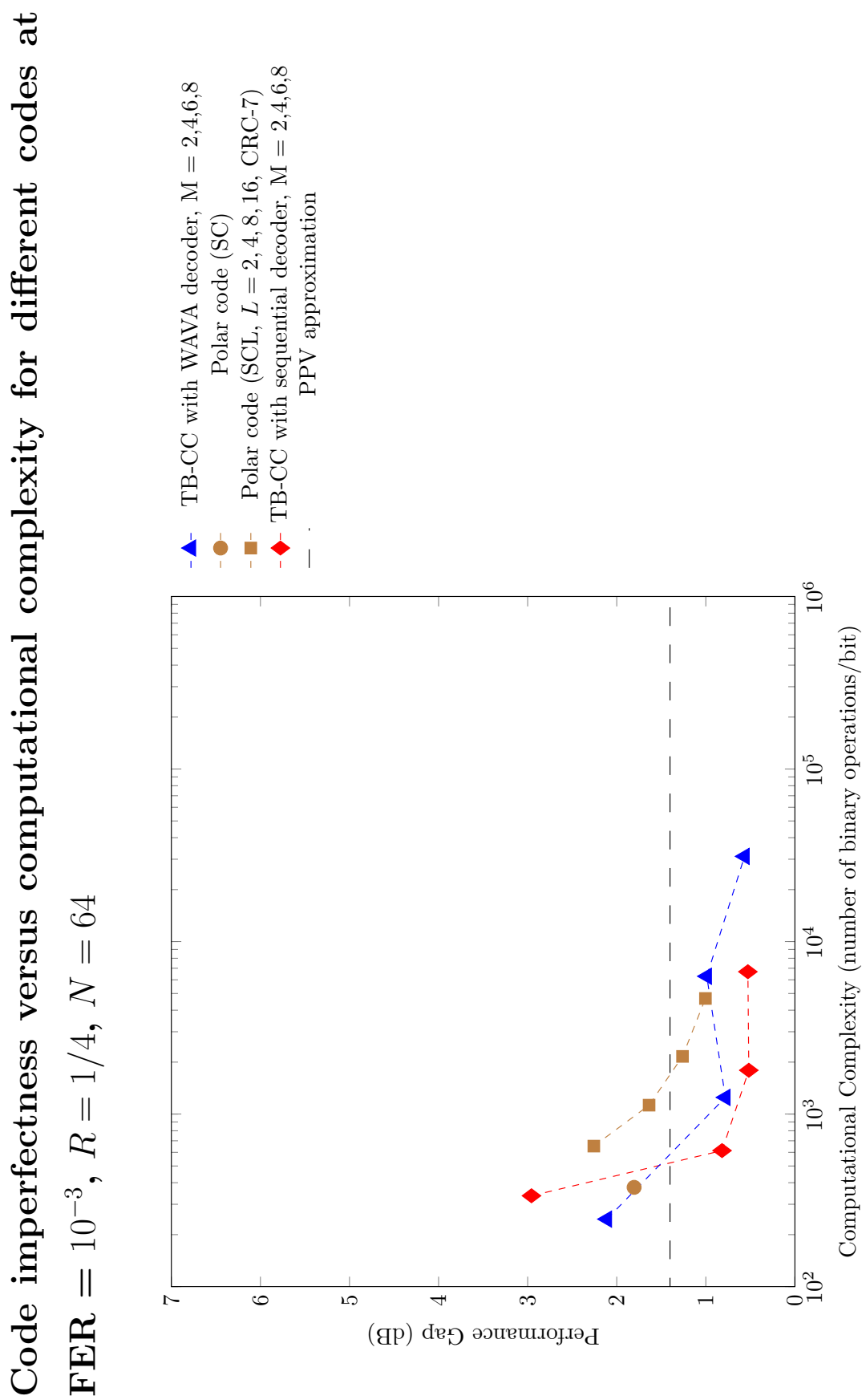

Figure C.108: Code imperfectness versus computational complexity at FER = 10^{-3} for different codes with $R = 1/4$, $N = 64$

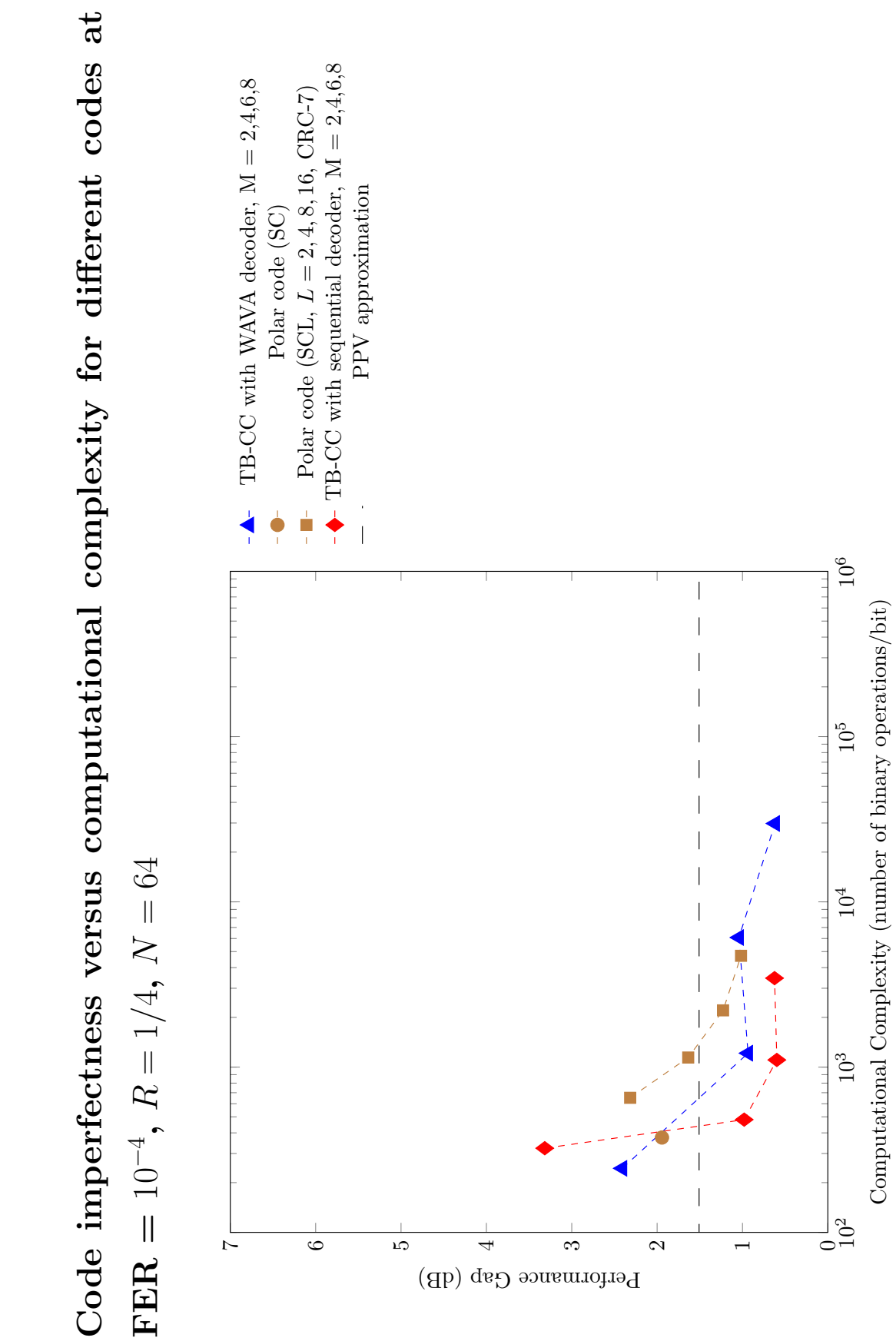

Figure C.109: Code imperfectness versus computational complexity at FER = 10^{-4} for different codes with $R = 1/4$, $N = 64$

Code imperfectness versus computational complexity for different codes at FER $= 10^{-5}$, $R = 1/4$, $N = 64$

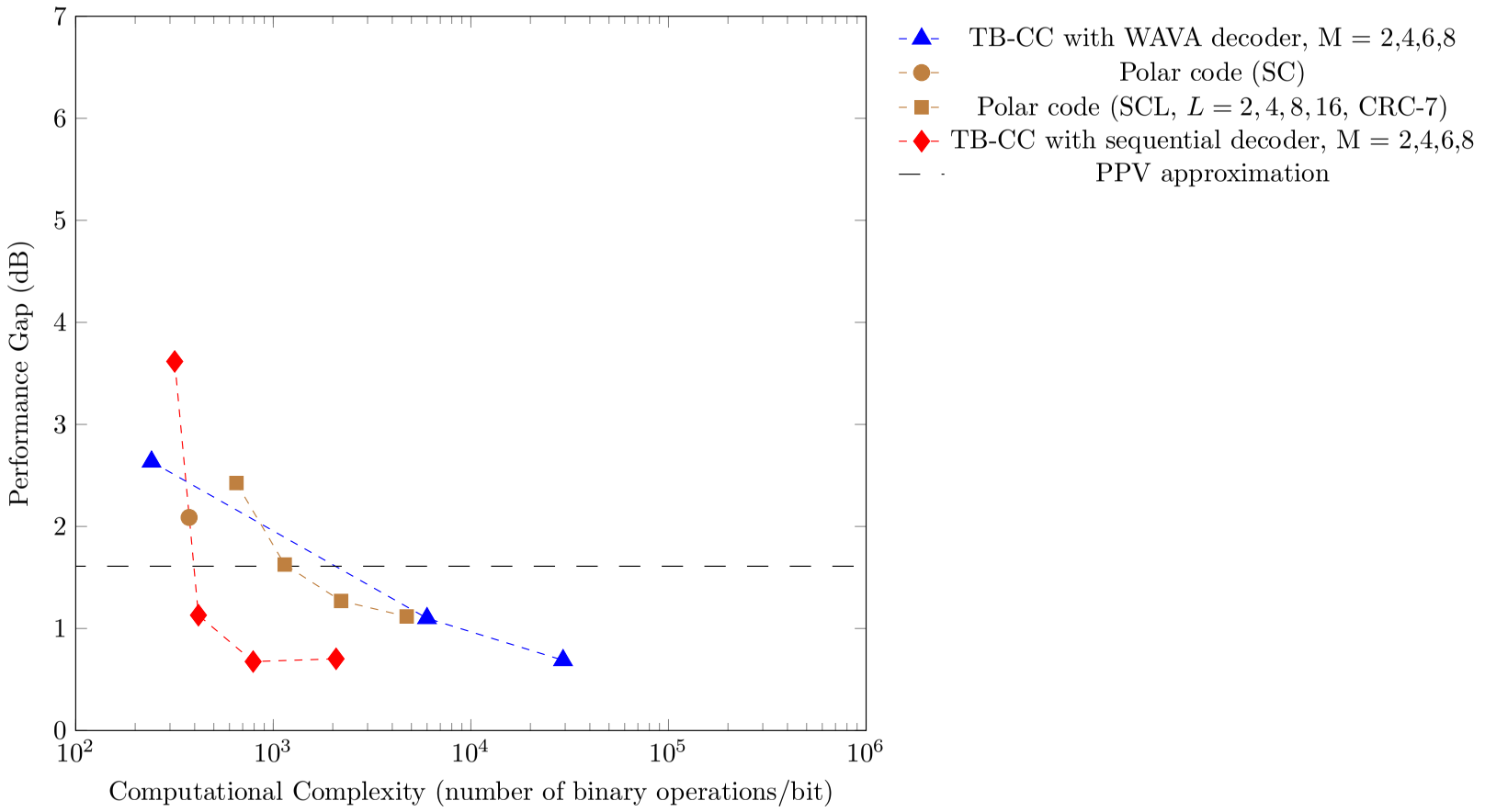

Figure C.110: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 1/4$, $N = 64$

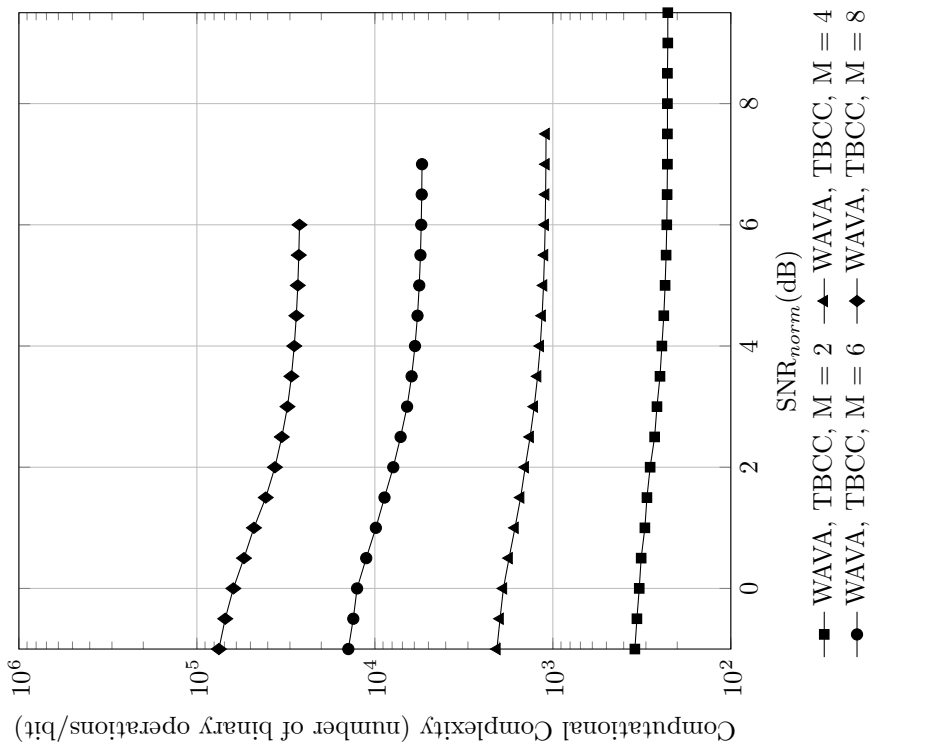

Figure C.112: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/4$, $N = 128$, maximum number of iterations = 4

Result for TB-CC with WAVA Decoder, $R = 1/4$, $N = 128$

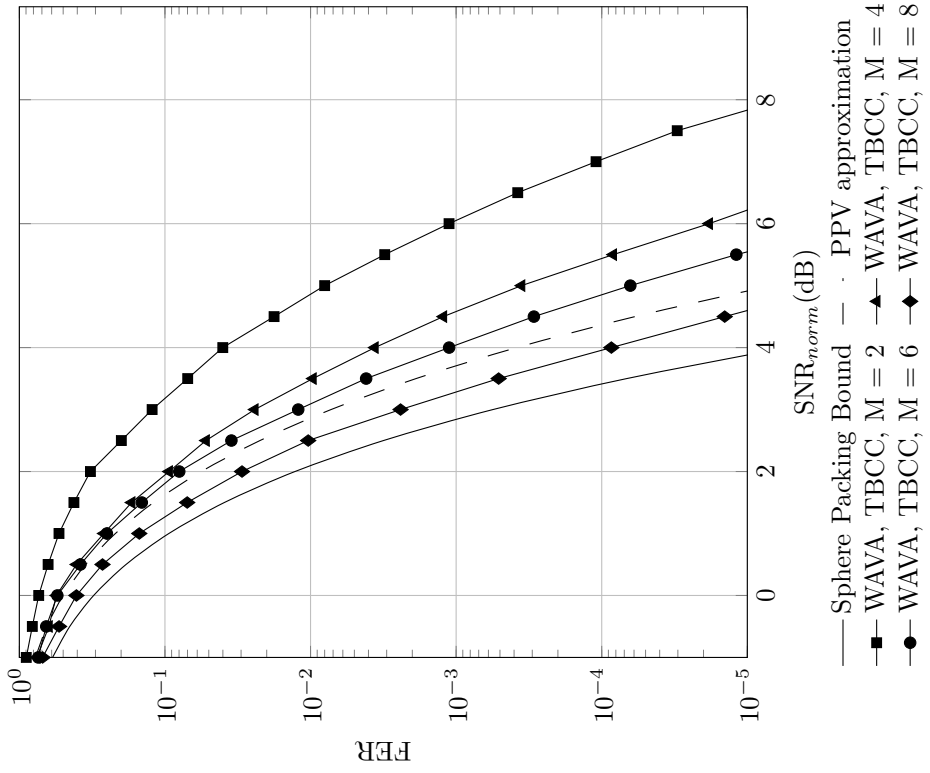

Figure C.111: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/4$, $N = 128$, maximum number of iterations = 4.

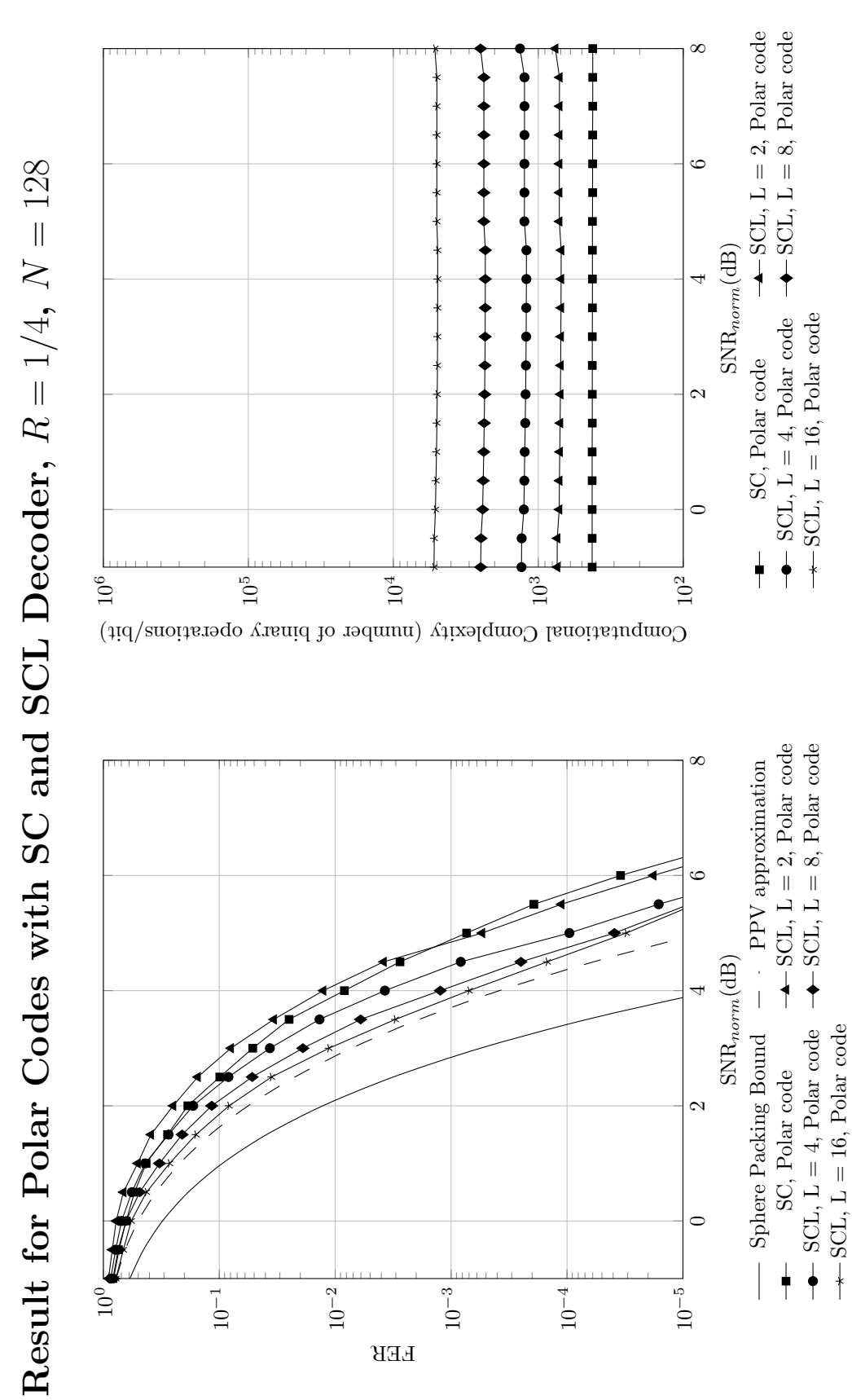

Figure C.113: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/4$, $N = 128$

Figure C.114: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/4$, $N = 128$

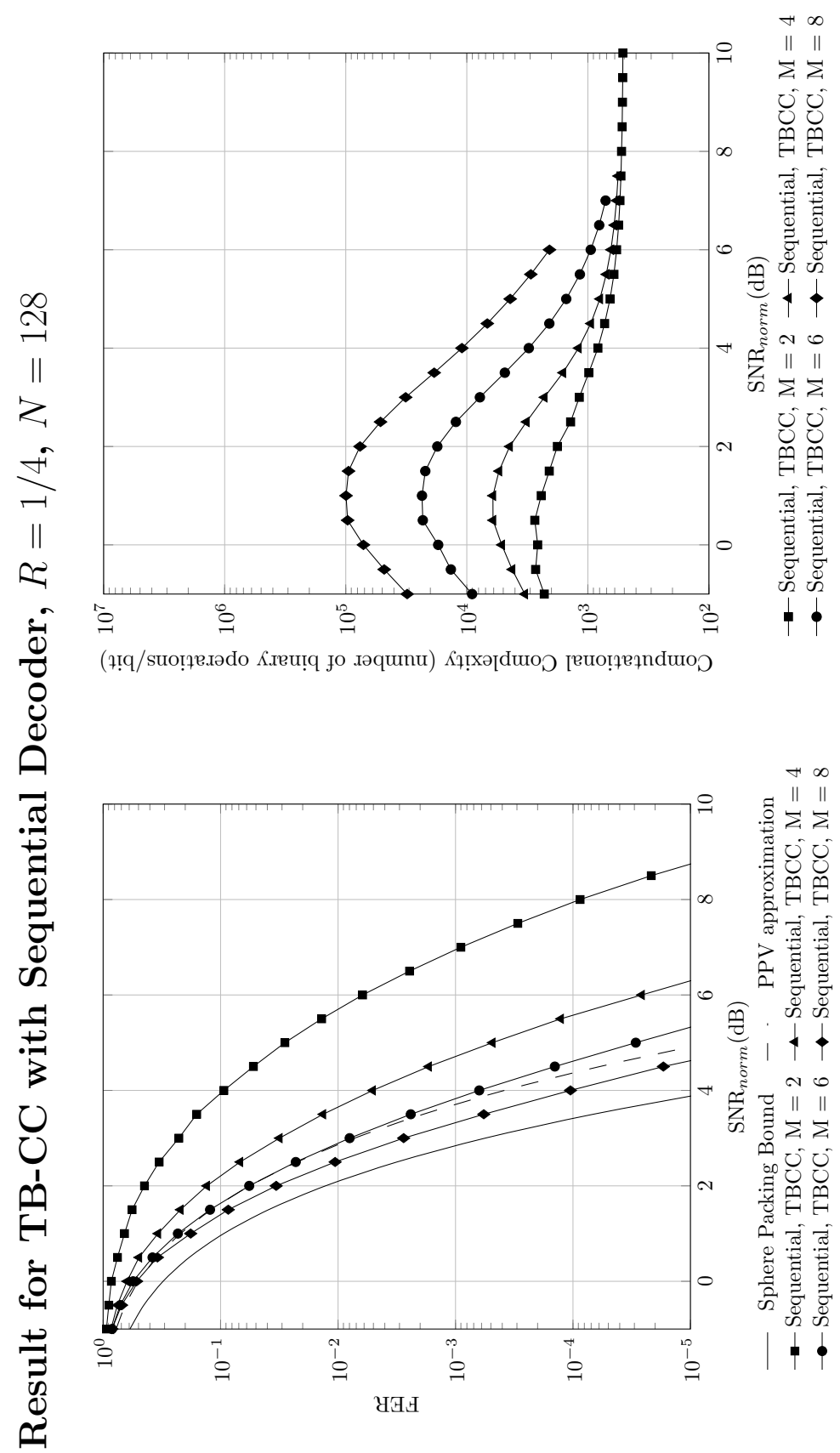

Figure C.115: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/4$, $N = 128$

Figure C.116: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/4$, $N = 128$

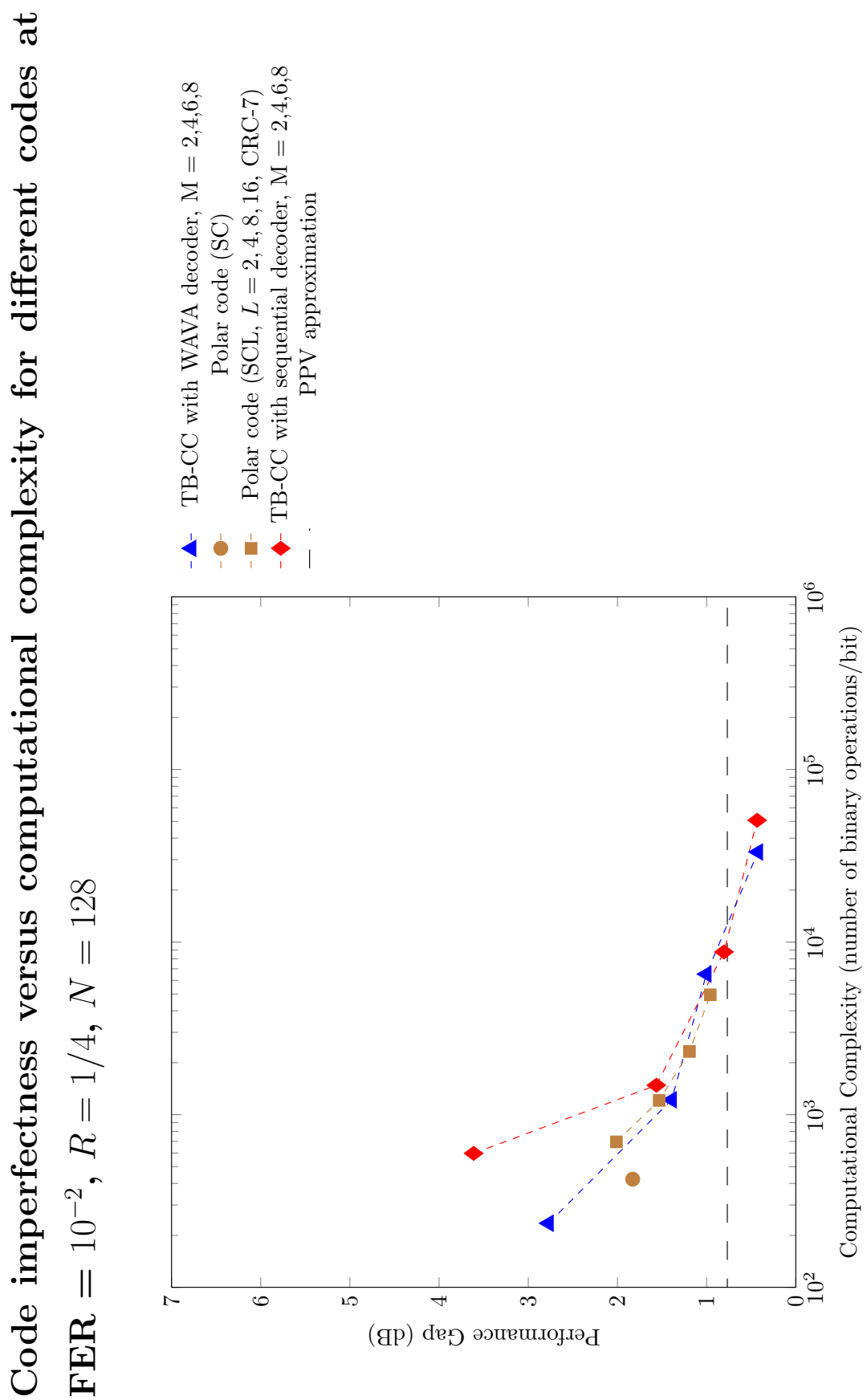

Figure C.117: Code imperfectness versus computational complexity at FER = 10^{-2} for different codes with $R = 1/4$, $N = 128$

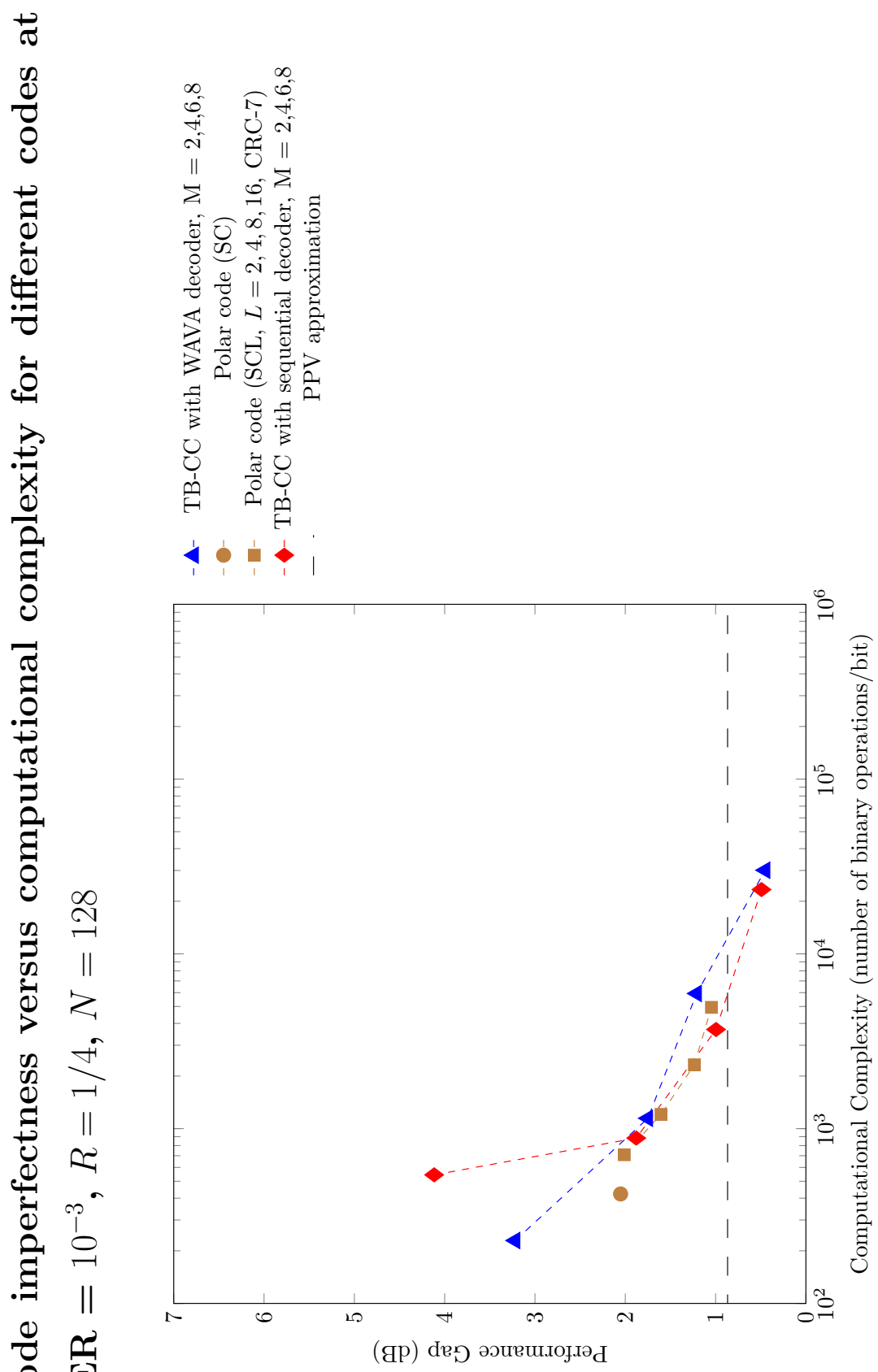

Figure C.118: Code imperfectness versus computational complexity at FER = 10^{-3} for different codes with $R = 1/4$, $N = 128$

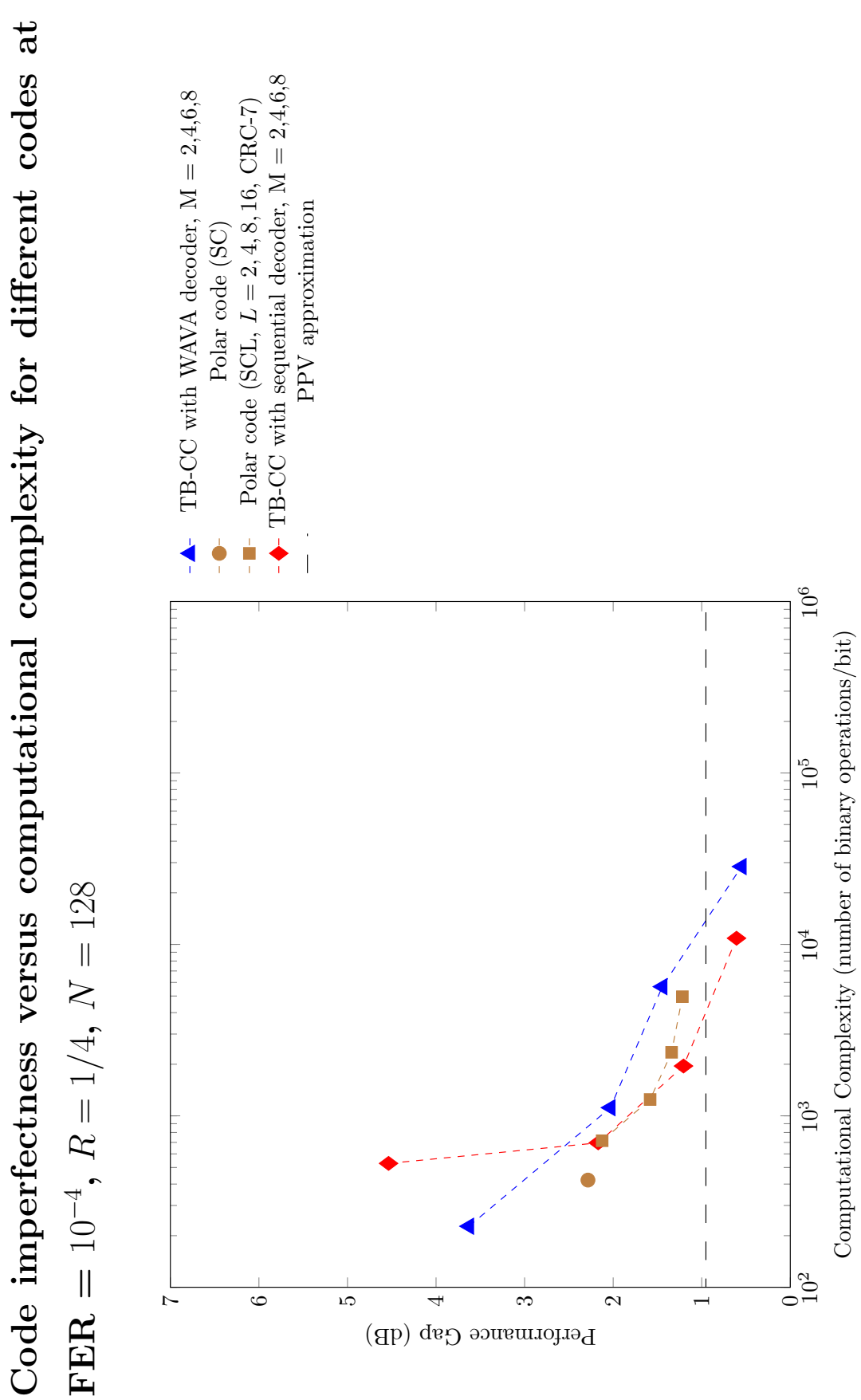

Figure C.119: Code imperfectness versus computational complexity at FER = 10^{-4} for different codes with $R = 1/4$, $N = 128$

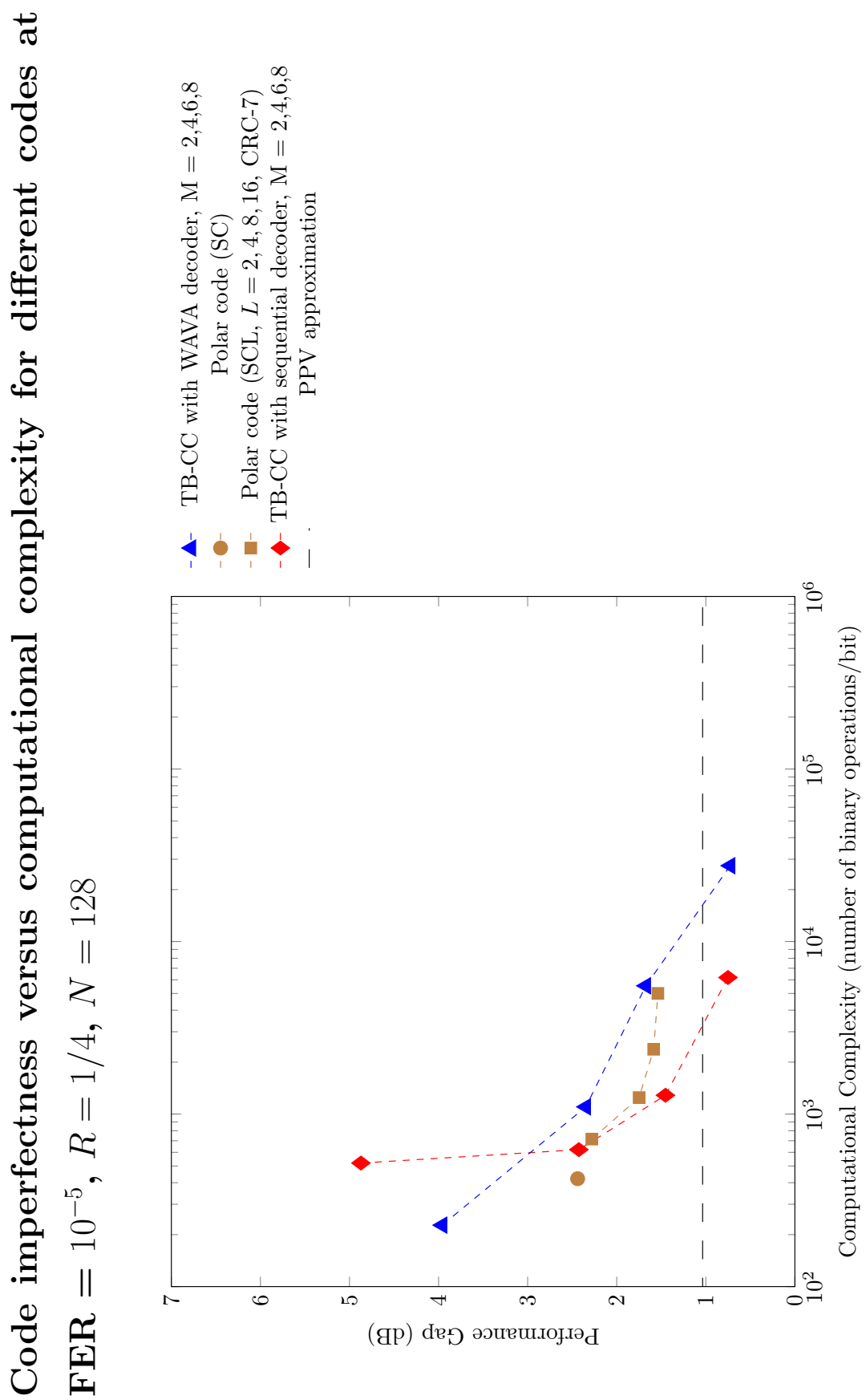

Figure C.120: Code imperfectness versus computational complexity at FER = 10^{-5} for different codes with $R = 1/4$, $N = 128$

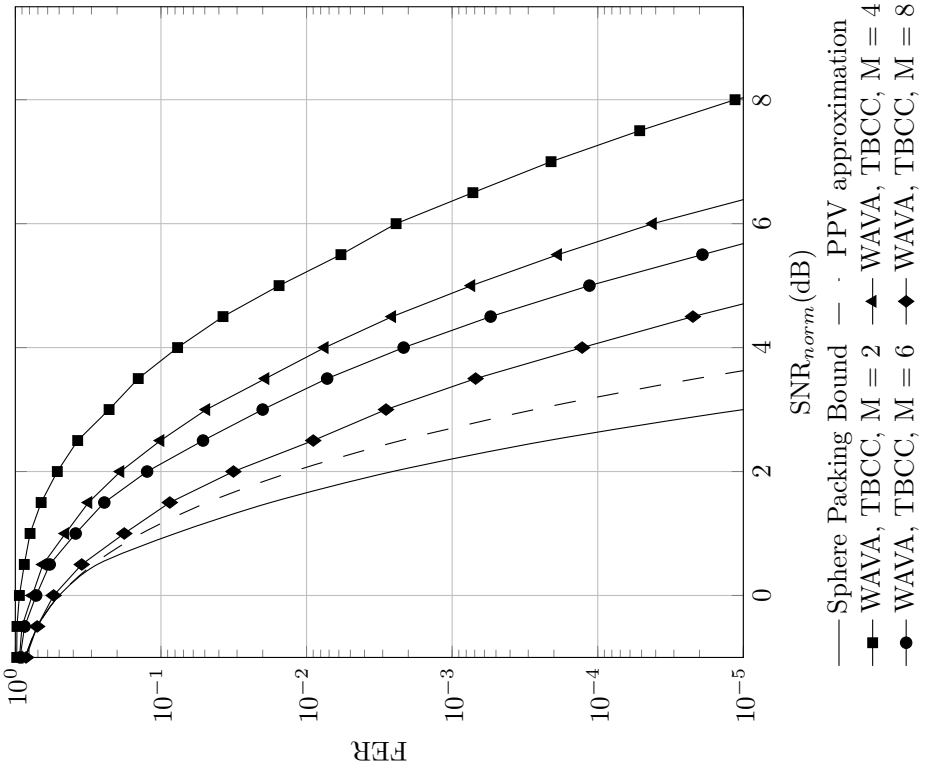

Figure C.121: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/4$, $N = 256$, maximum number of iterations = 4.

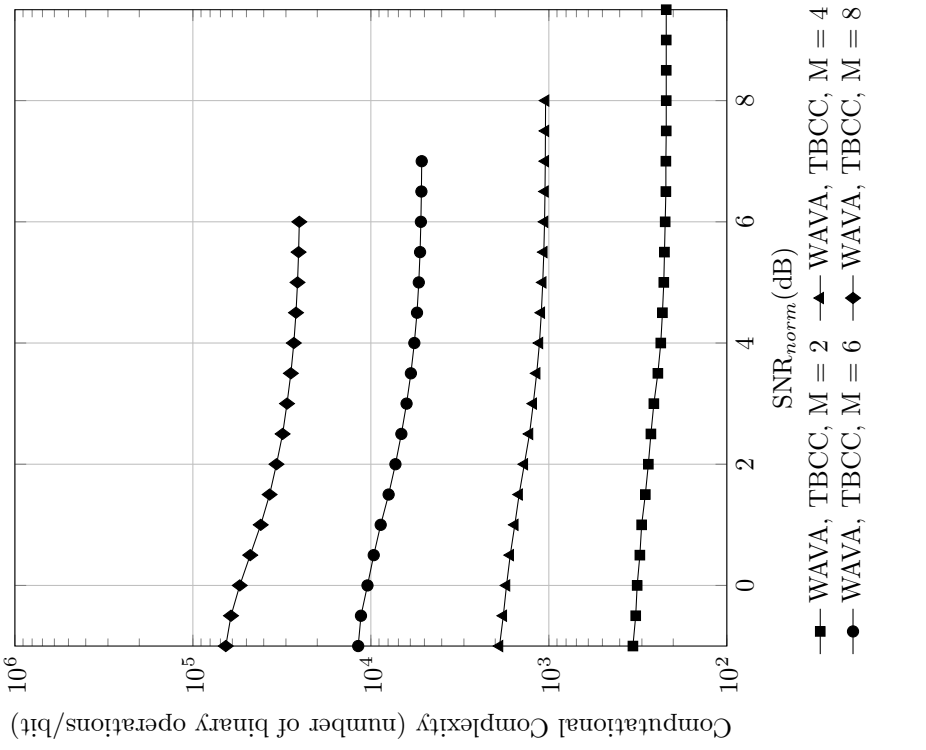

Figure C.122: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/4$, $N = 256$, maximum number of iterations = 4

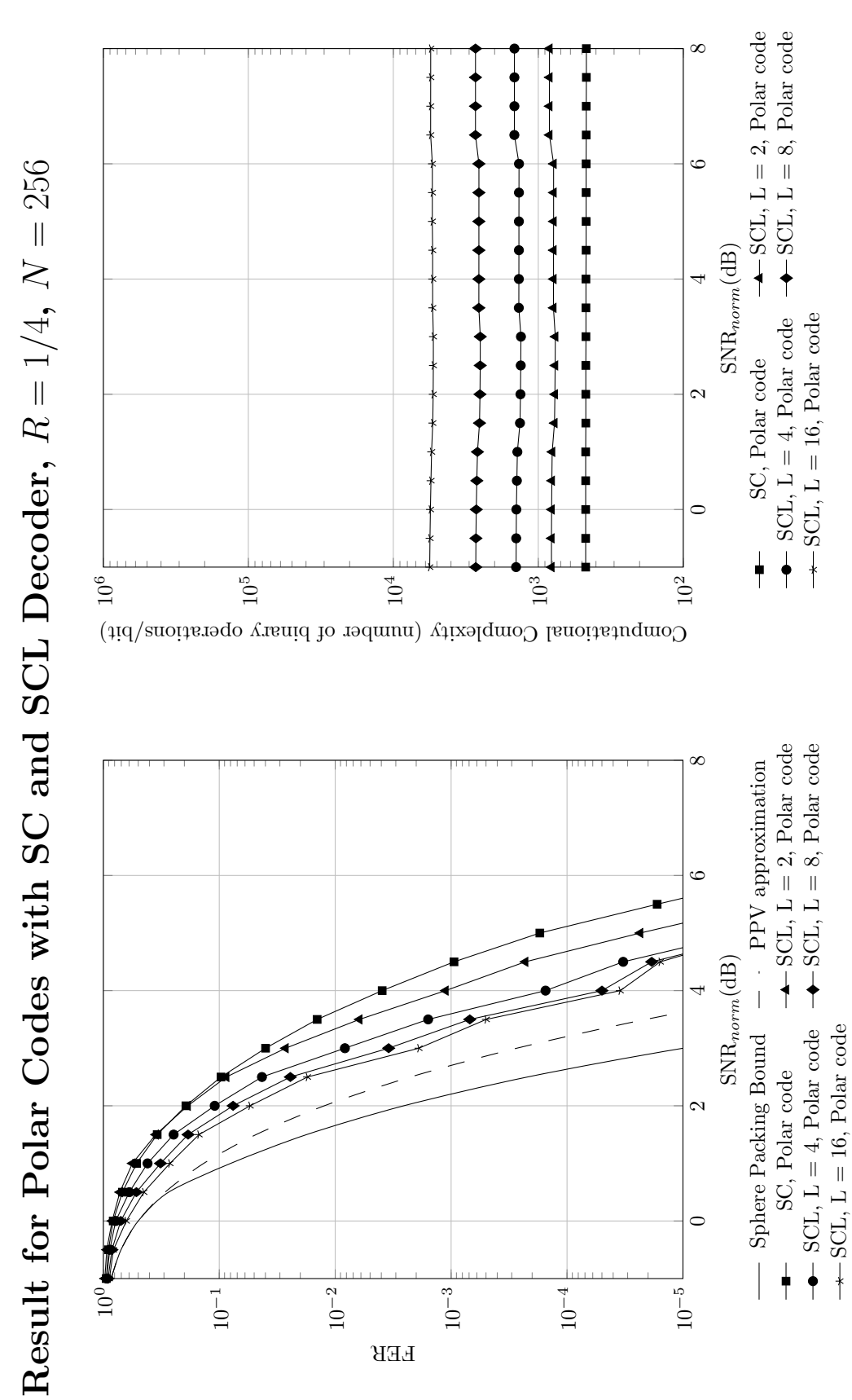

Figure C.123: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/4$, $N = 256$

Figure C.124: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/4$, $N = 256$

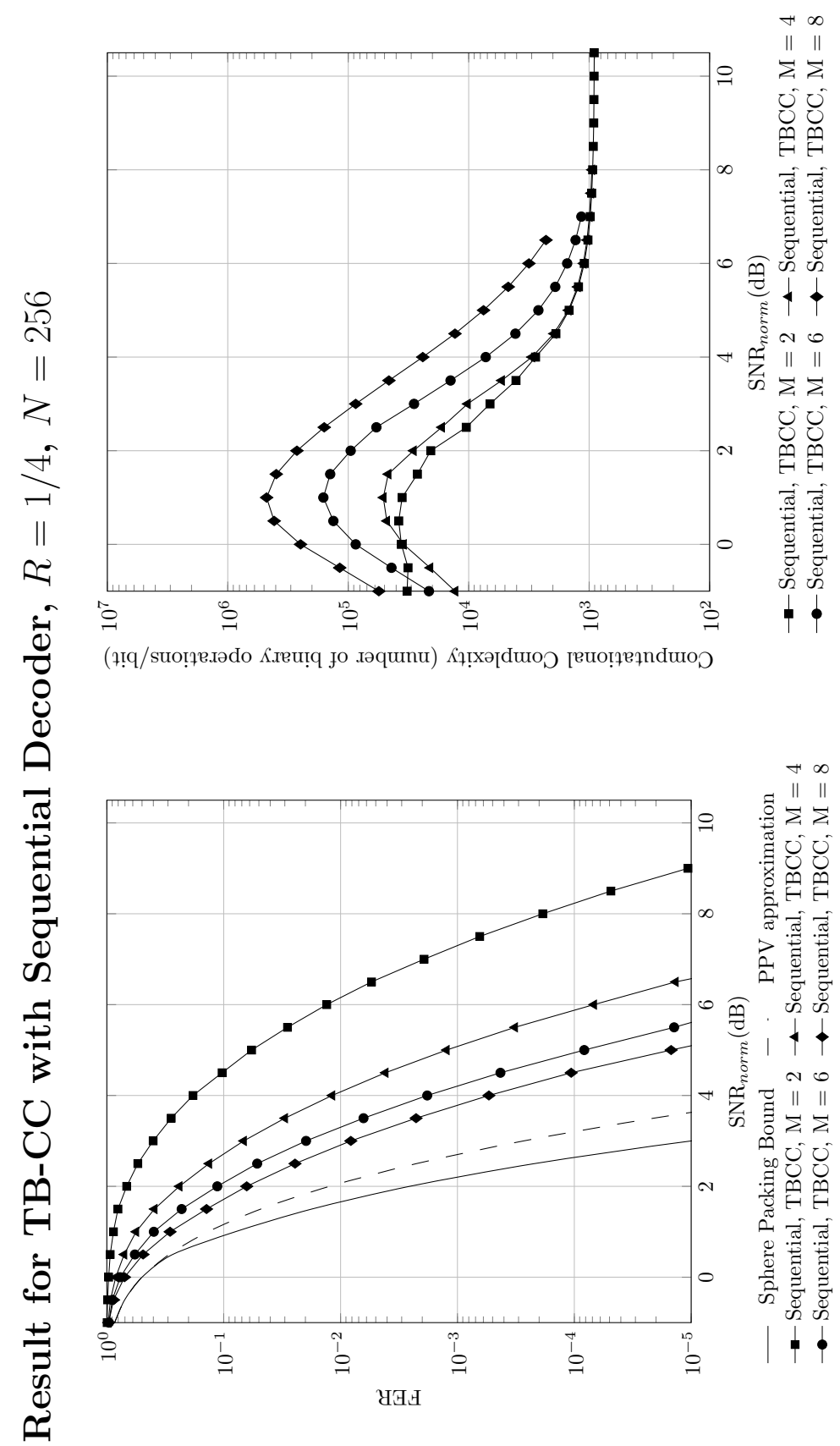

Figure C.125: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/4$, $N = 256$

Figure C.126: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/4$, $N = 256$

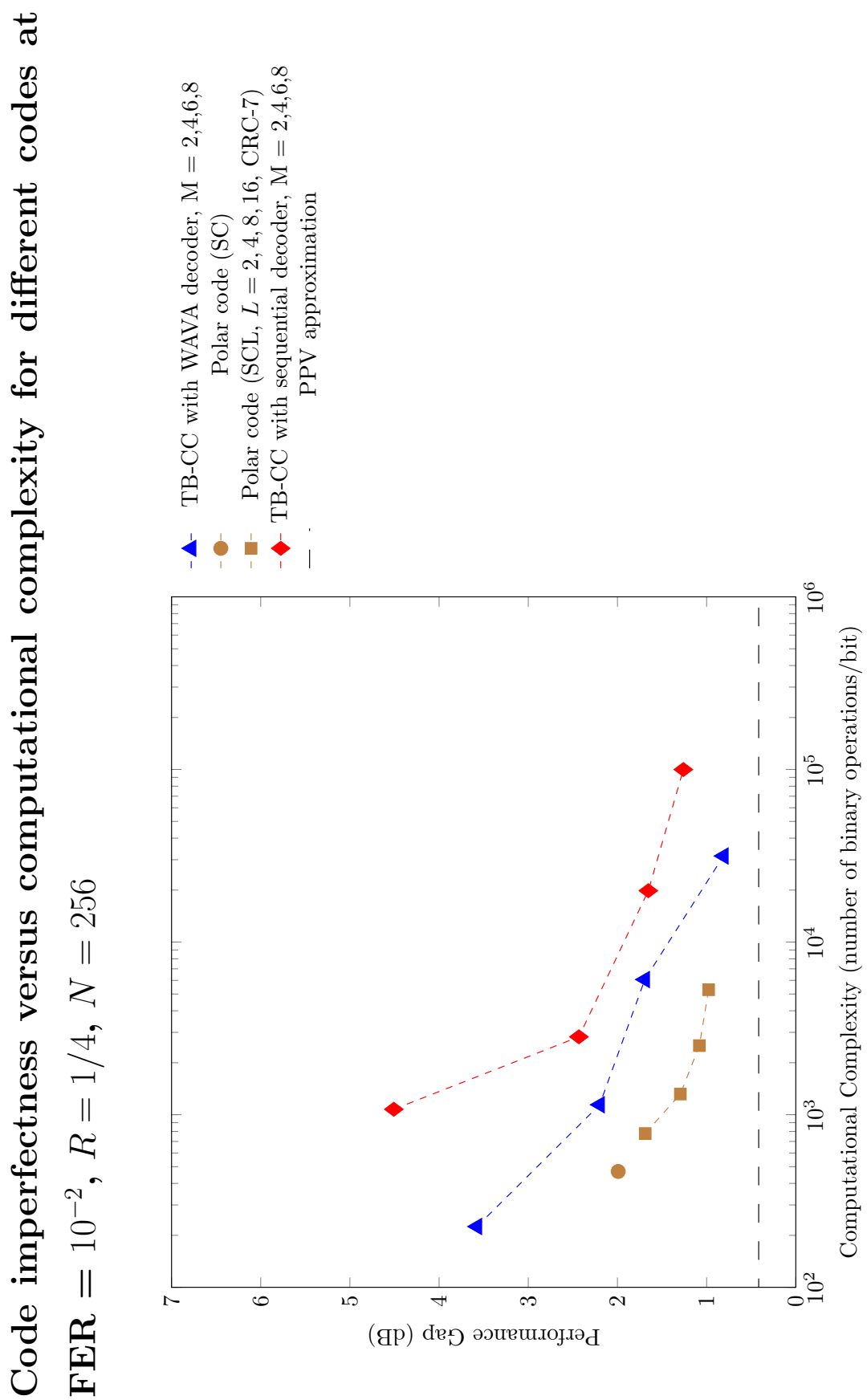

Figure C.127: Code imperfectness versus computational complexity at FER = 10^{-2} for different codes with $R = 1/4$, $N = 256$

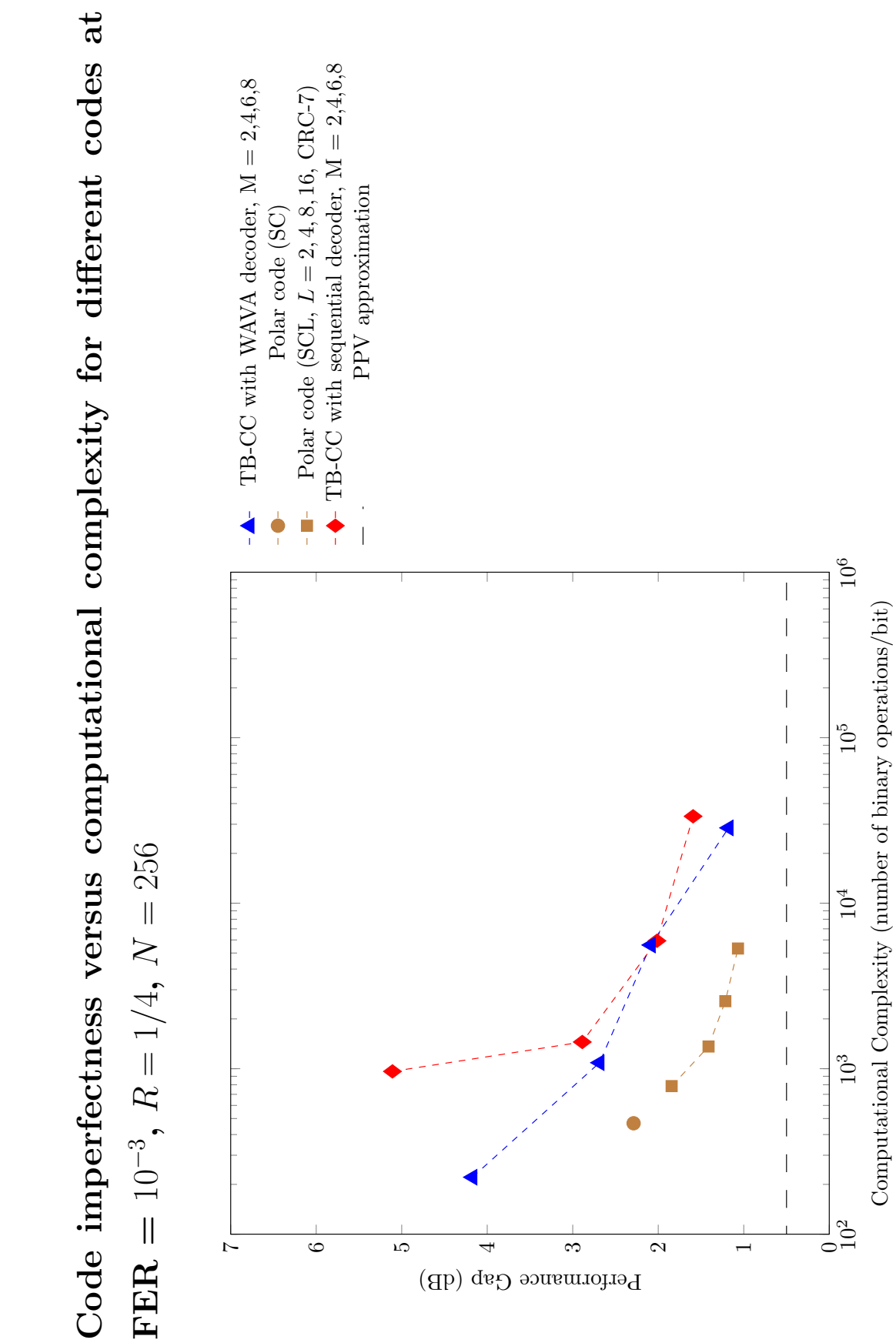

Figure C.128: Code imperfectness versus computational complexity at FER = 10^{-3} for different codes with $R = 1/4$, $N = 256$

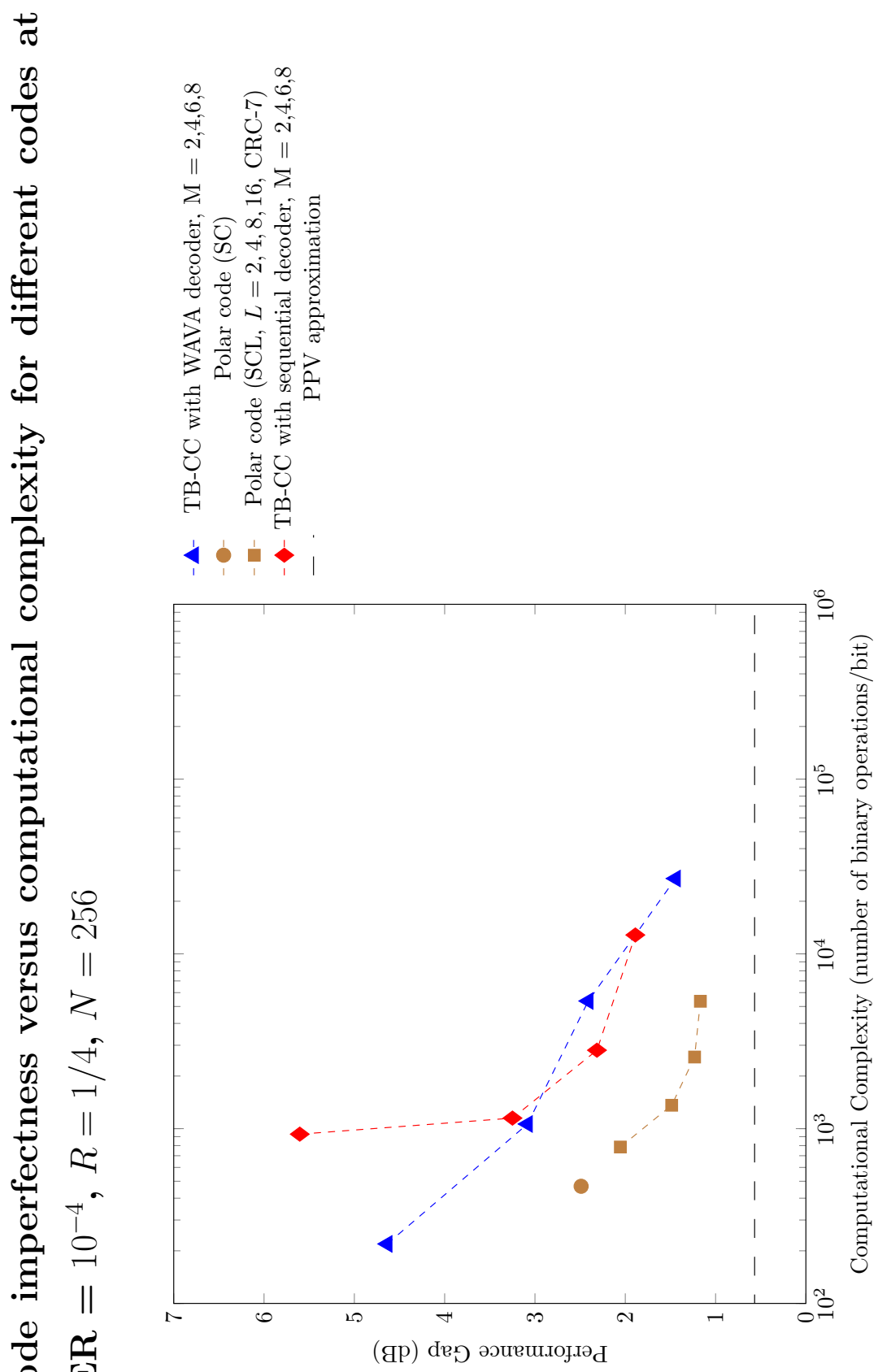

Figure C.129: Code imperfectness versus computational complexity at FER = 10^{-4} for different codes with $R = 1/4$, $N = 256$

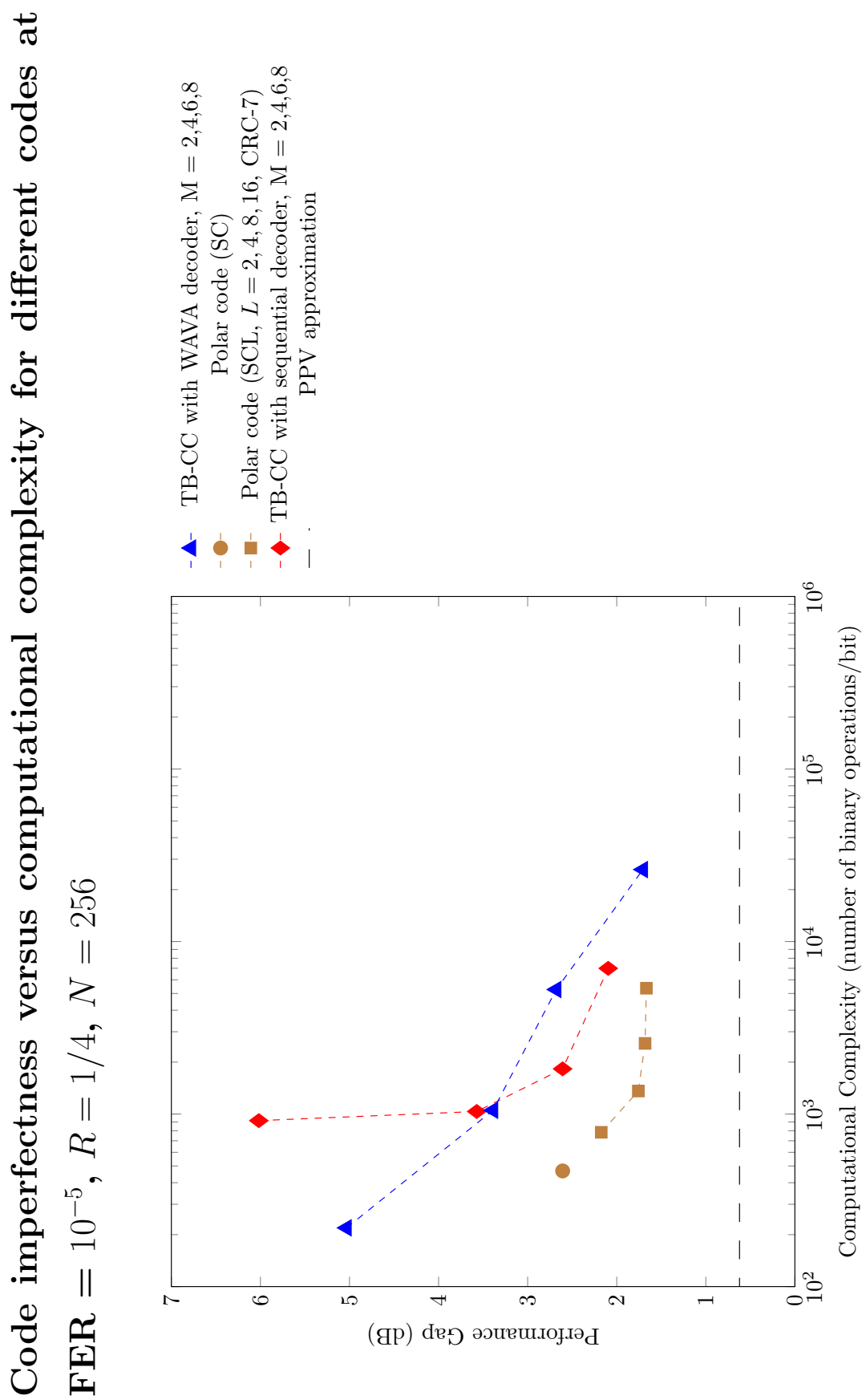

Figure C.130: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 1/4$, $N = 256$

Result for TB-CC with WAVA Decoder, $R = 1/4$, $N = 512$

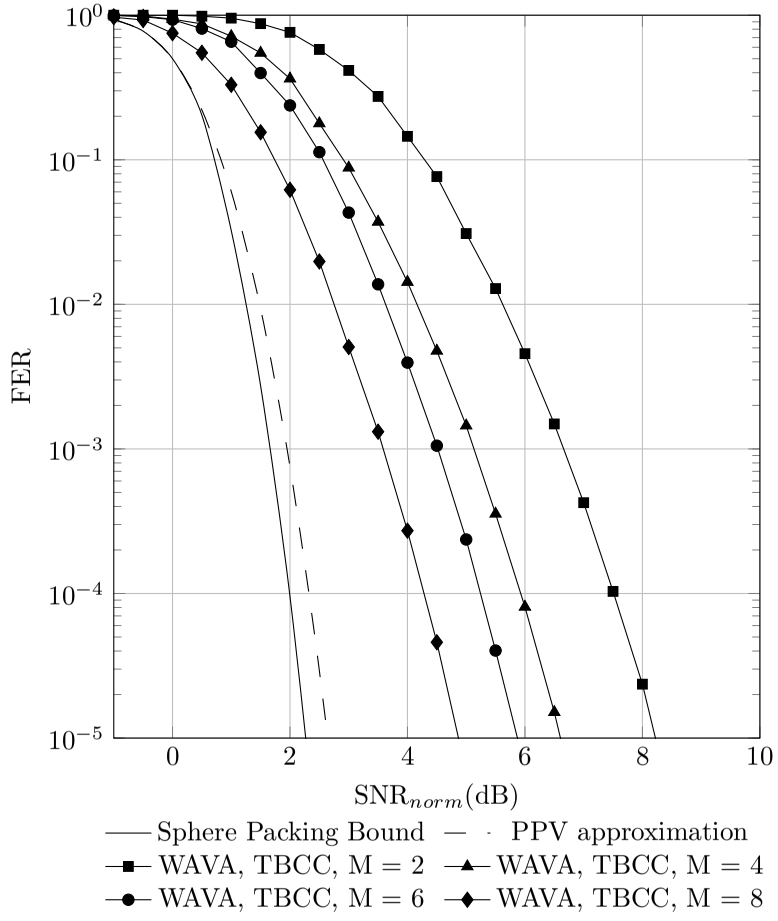

Figure C.131: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/4$, $N = 512$, maximum number of iterations = 4.

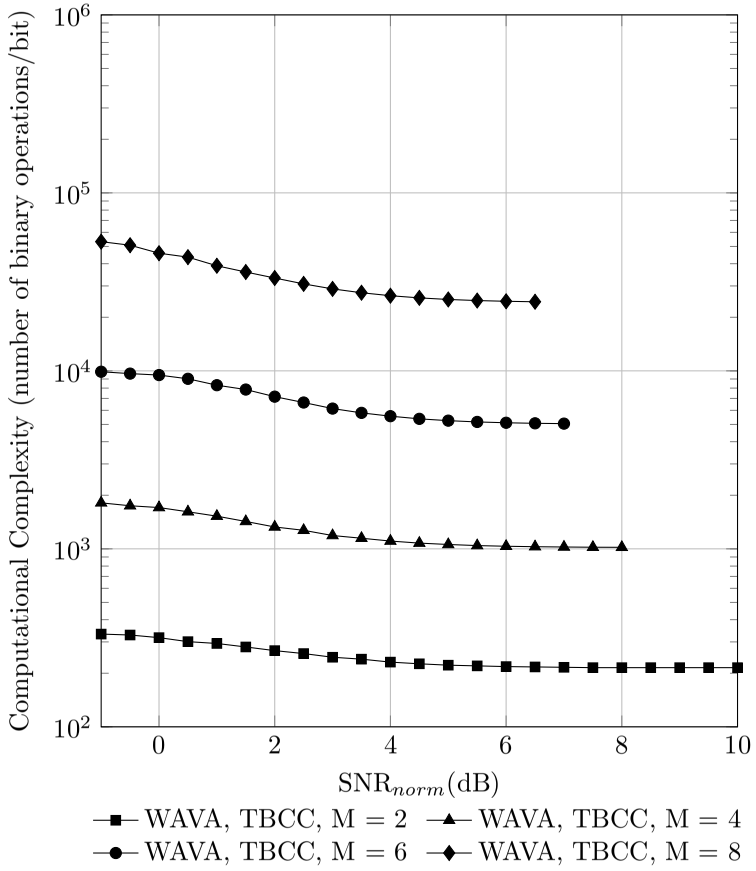

Figure C.132: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/4$, $N = 512$, maximum number of iterations = 4

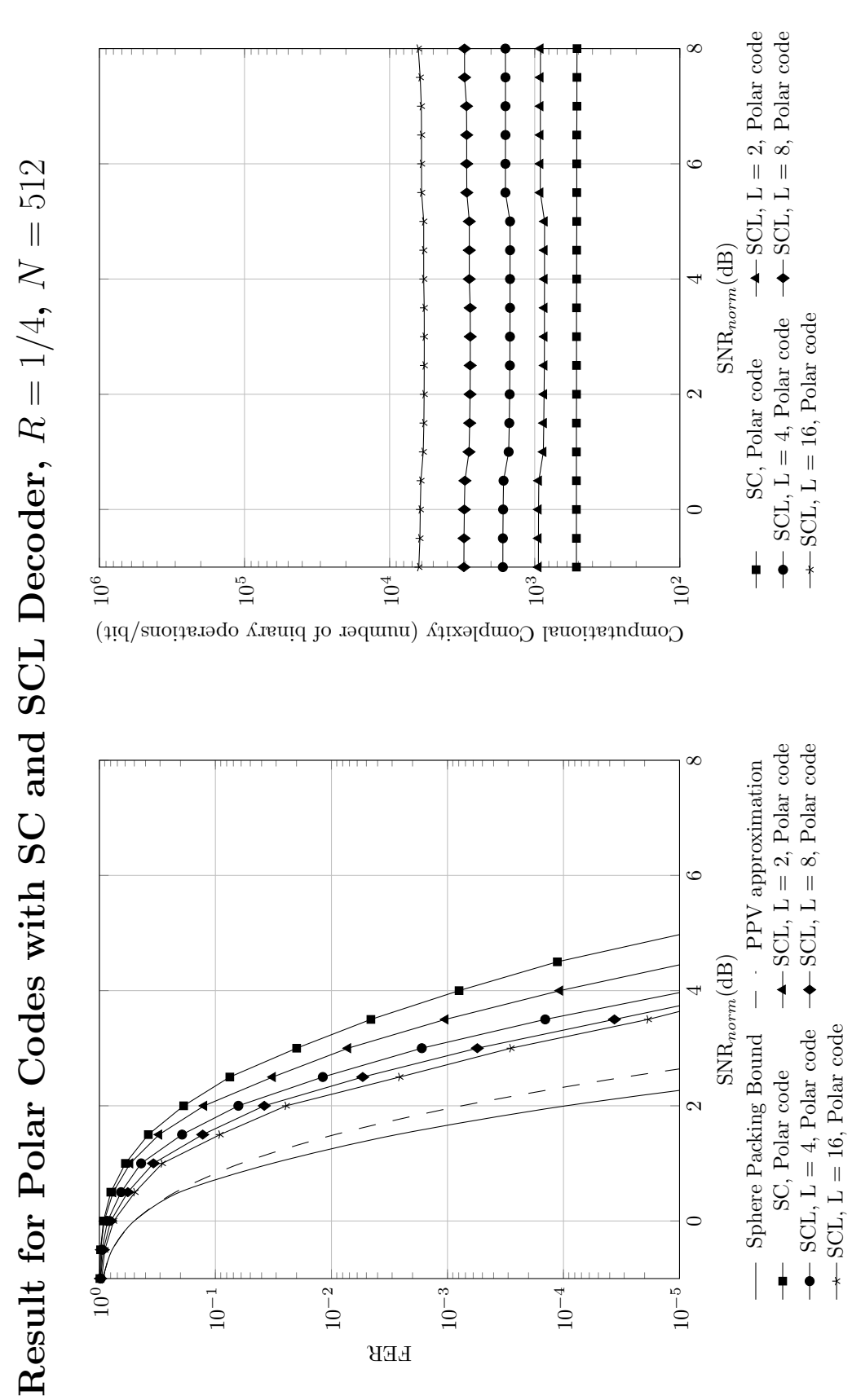

Figure C.133: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/4$, $N = 512$

Figure C.134: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/4$, $N = 512$

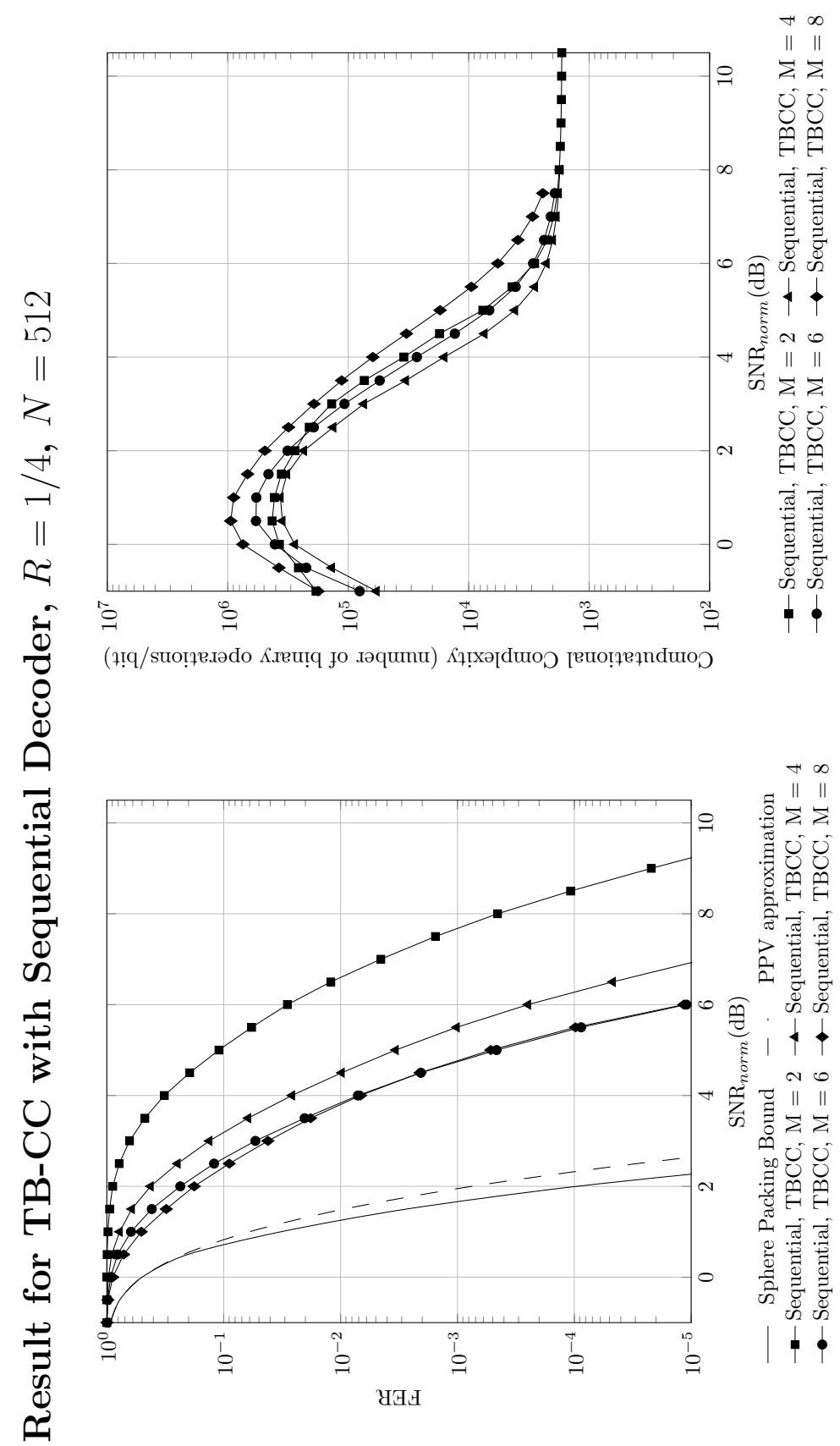

Figure C.135: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/4$, $N = 512$

Figure C.136: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/4$, $N = 512$

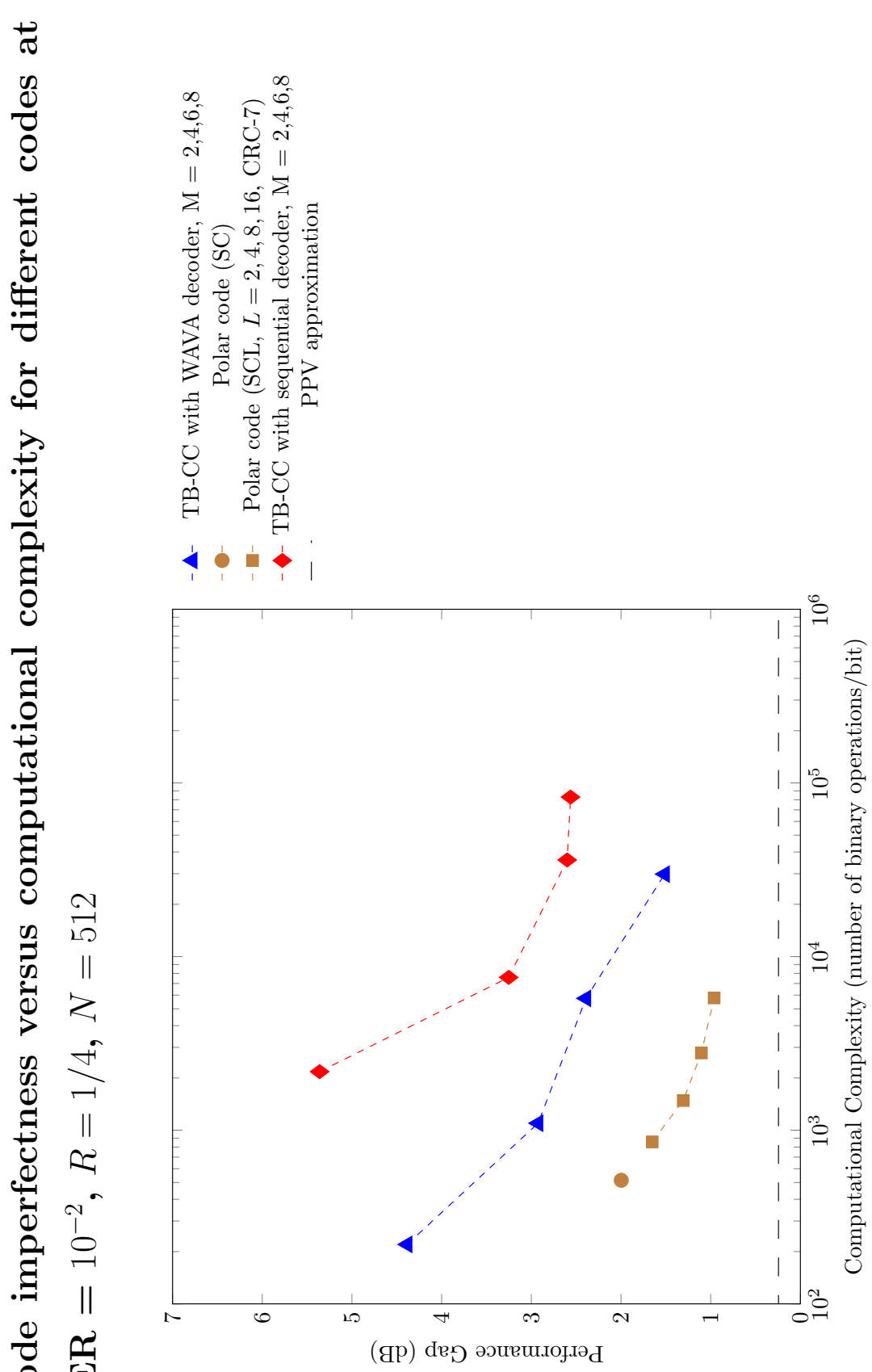

Figure C.137: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 1/4$, $N = 512$

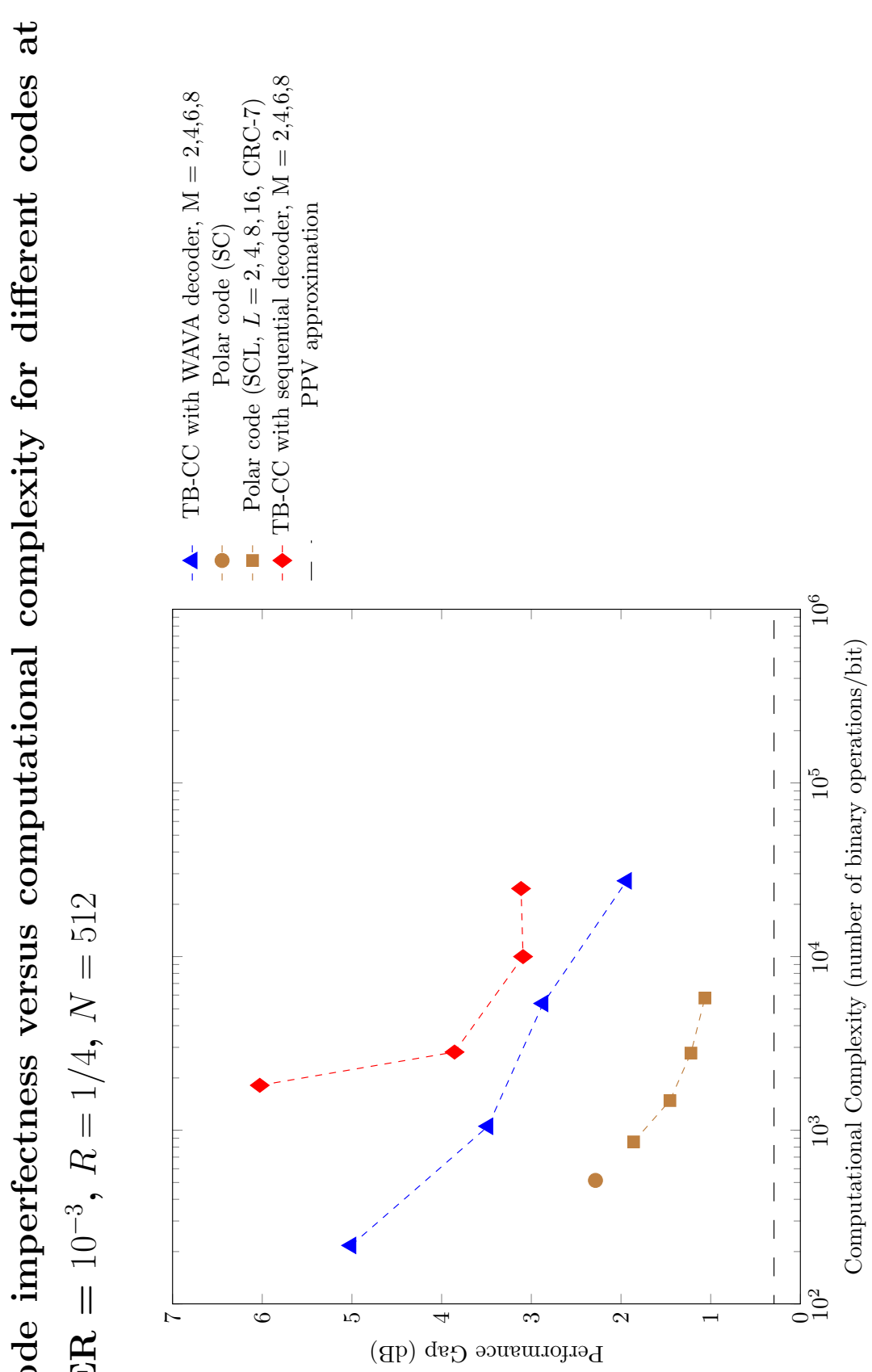

Figure C.138: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 1/4$, $N = 512$

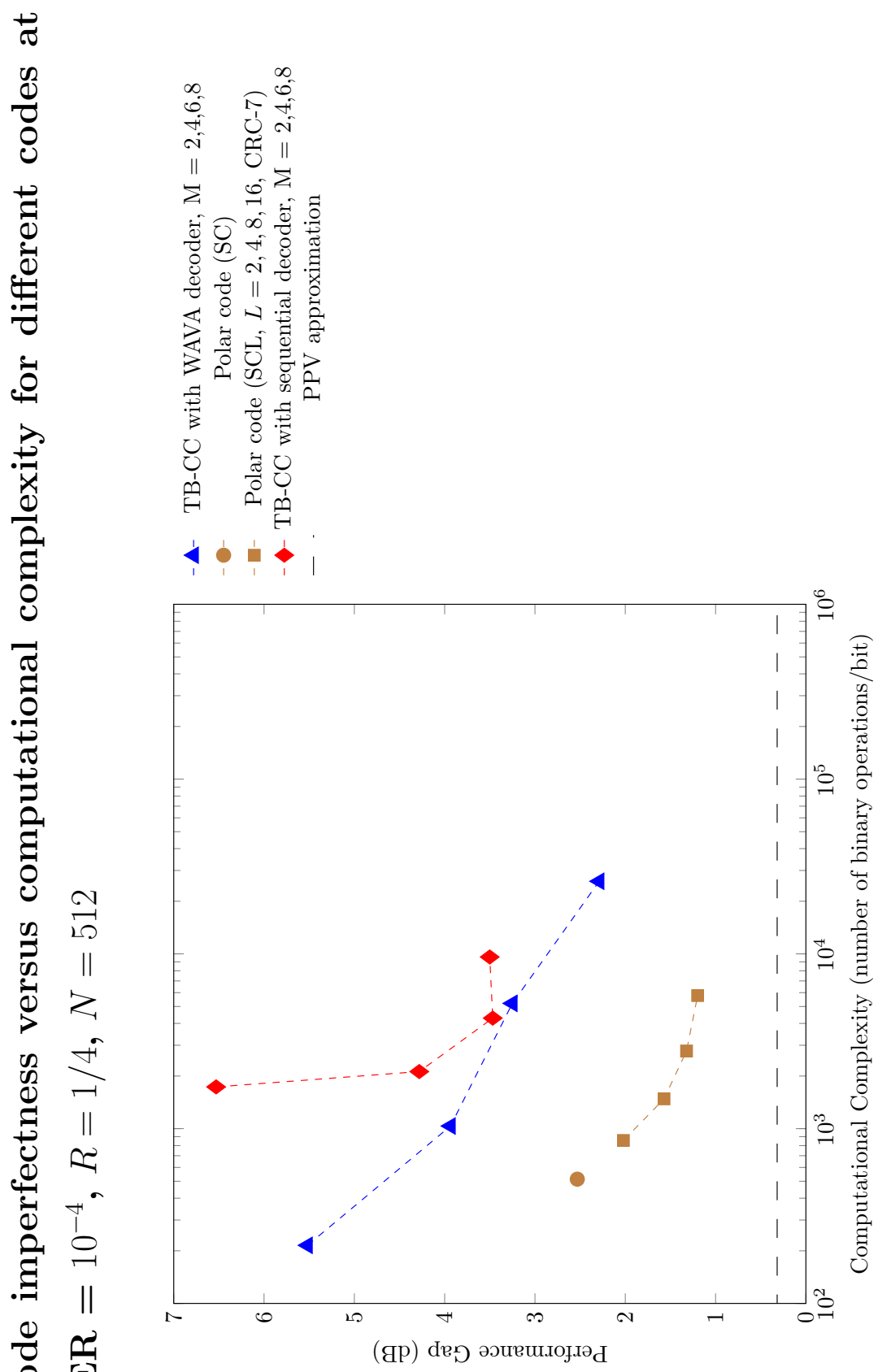

Figure C.139: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 1/4$, $N = 512$

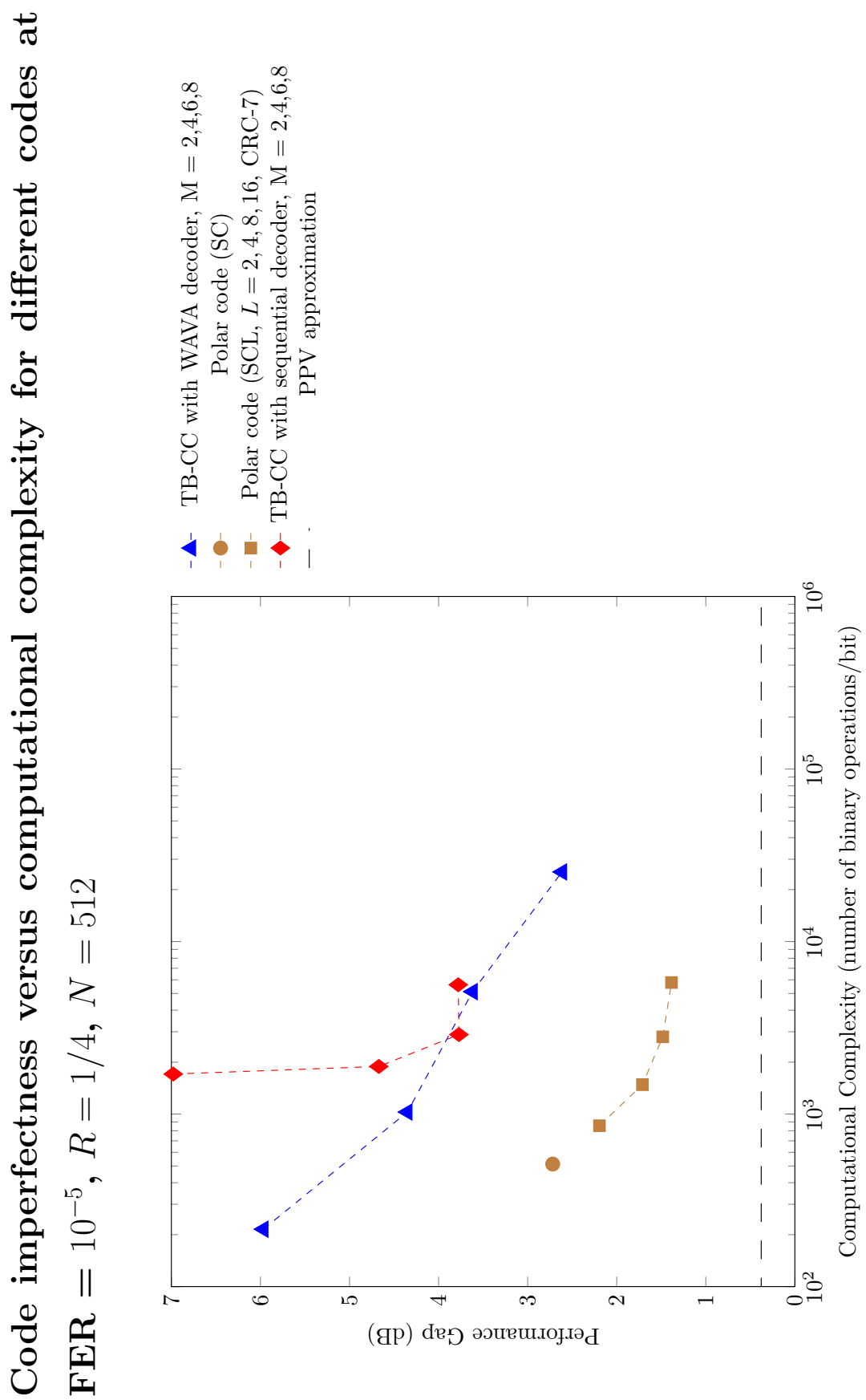

Figure C.140: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 1/4$, $N = 512$

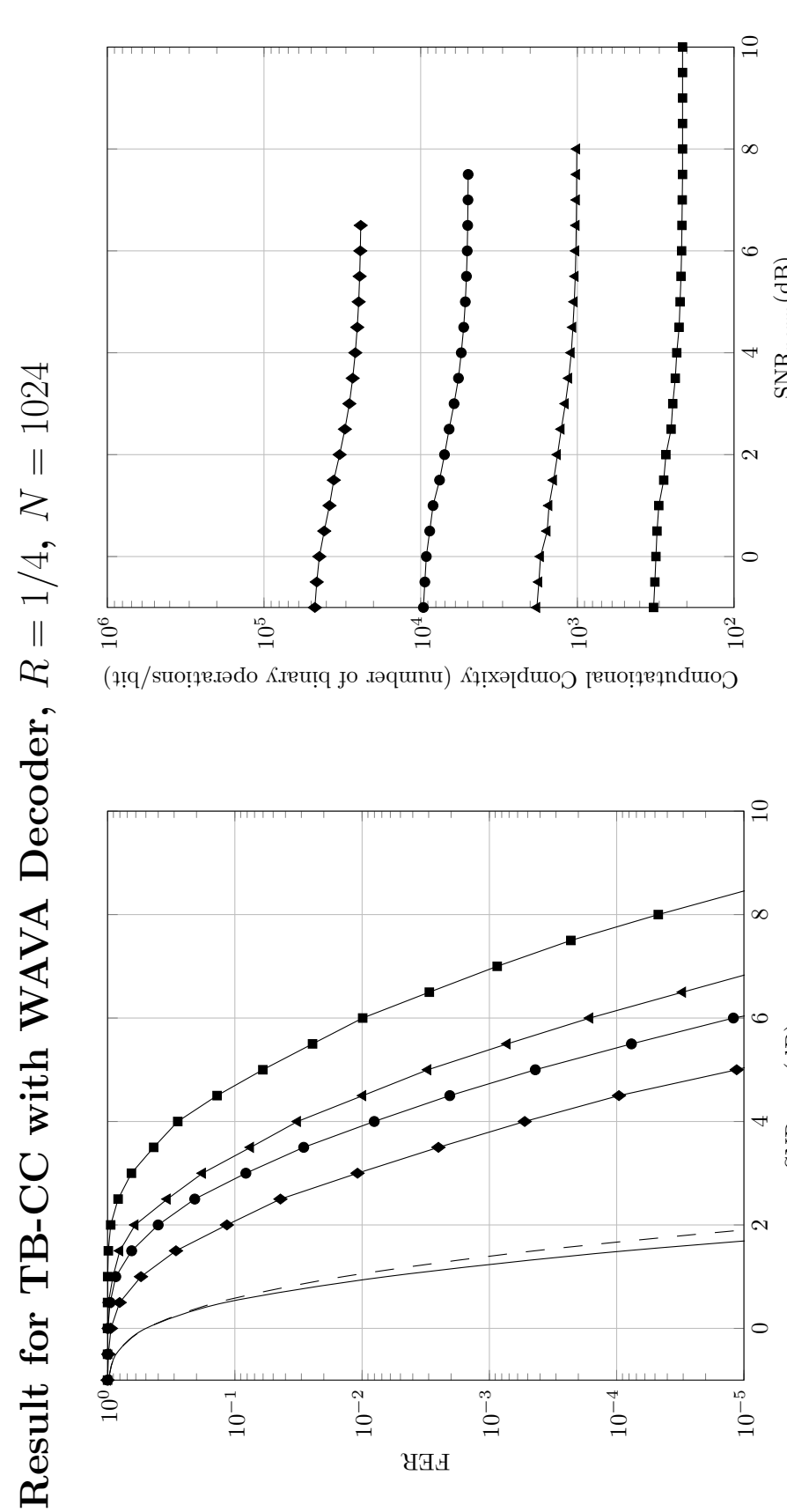

Figure C.141: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 1/4$, $N = 1024$, maximum number of iterations = 4.

Figure C.142: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 1/4$, $N = 1024$, maximum number of iterations = 4

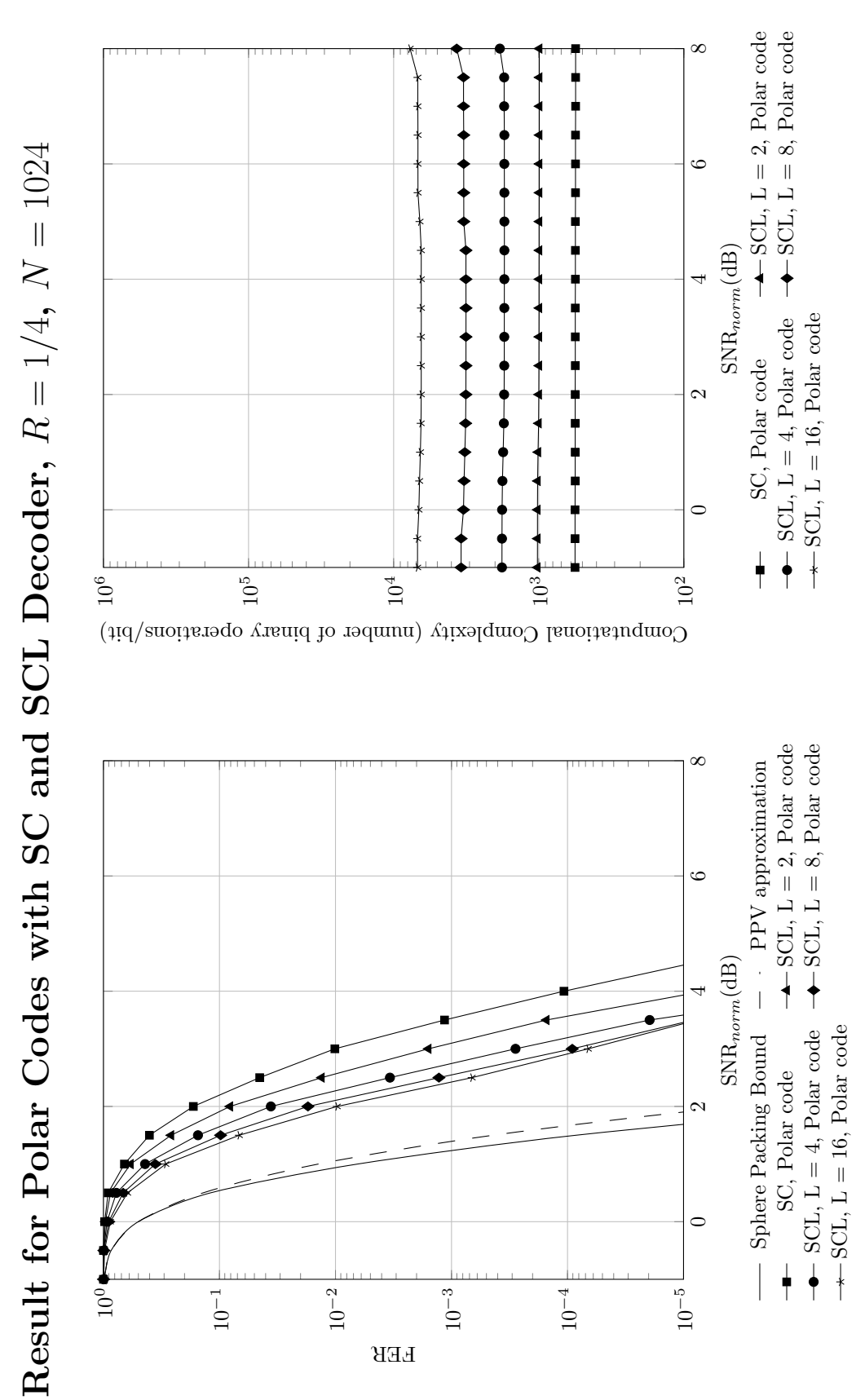

Figure C.143: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/4$, $N = 1024$

Figure C.144: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 1/4$, $N = 1024$

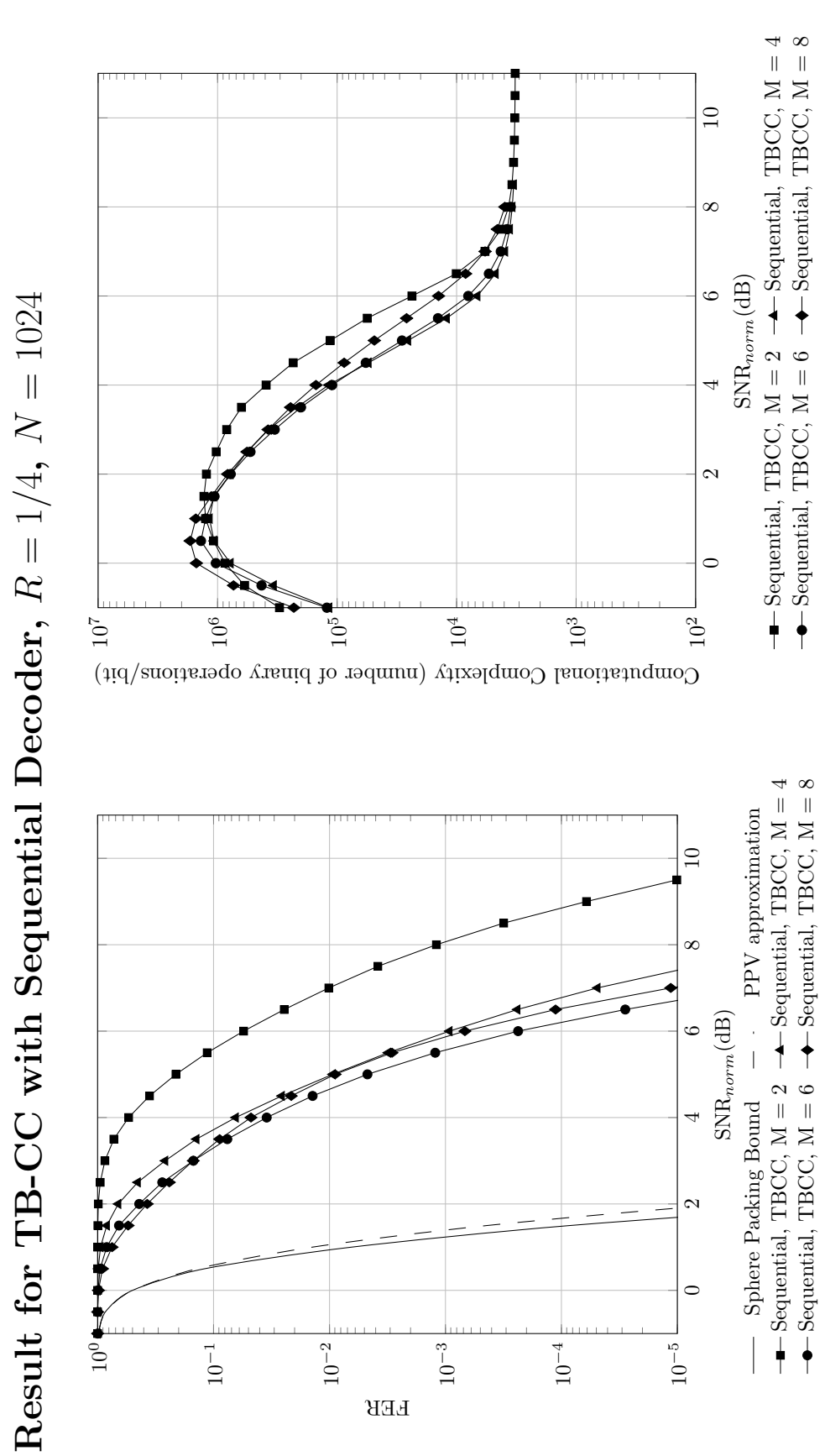

Figure C.145: Frame error rate versus normalized SNR for TB-CC with sequential decoder, $R = 1/4$, $N = 1024$

Figure C.146: Computational complexity versus normalized SNR for TB-CC with sequential decoder, $R = 1/4$, $N = 1024$

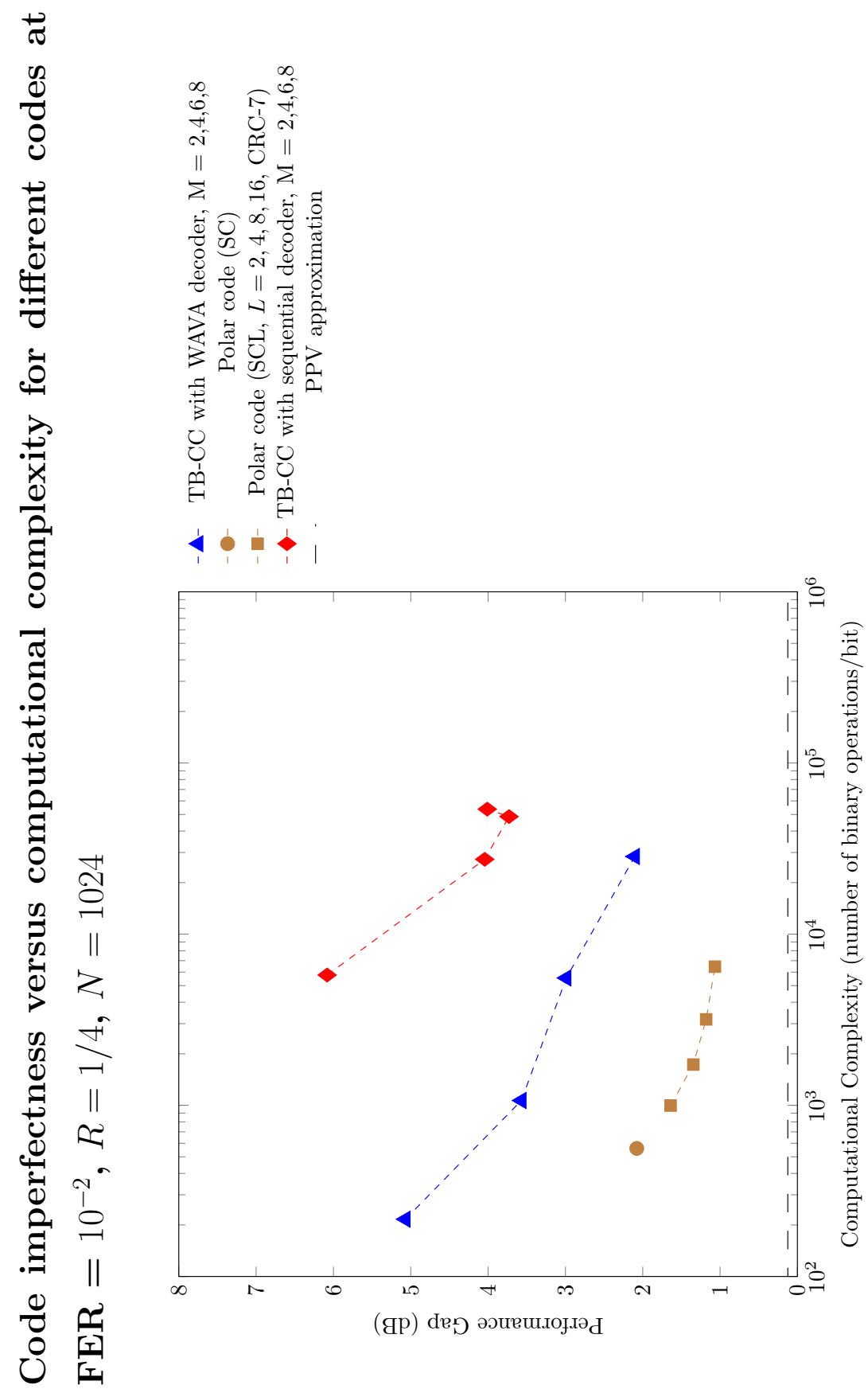

Figure C.147: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 1/4$, $N = 1024$

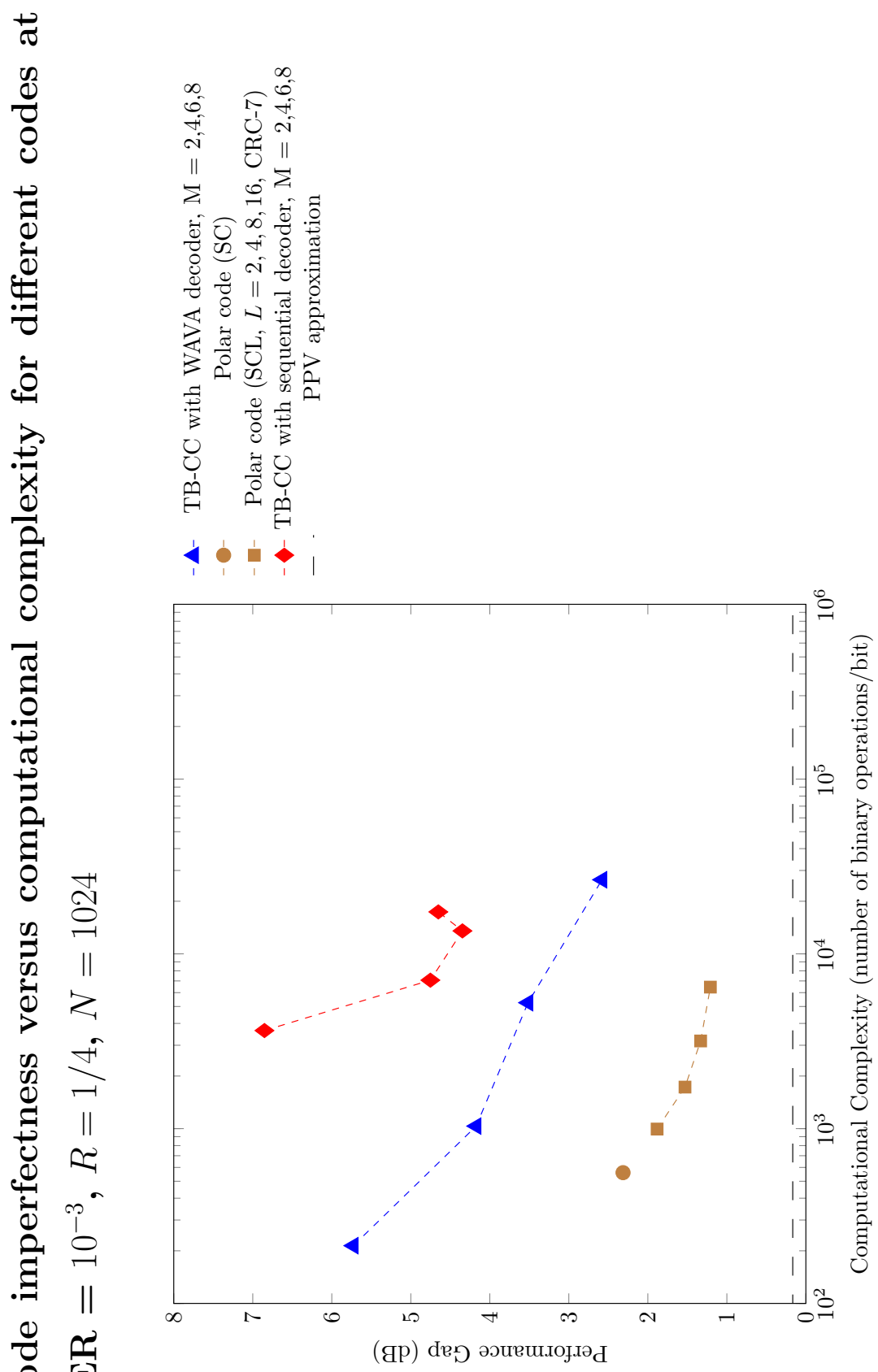

Figure C.148: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 1/4$, $N = 1024$

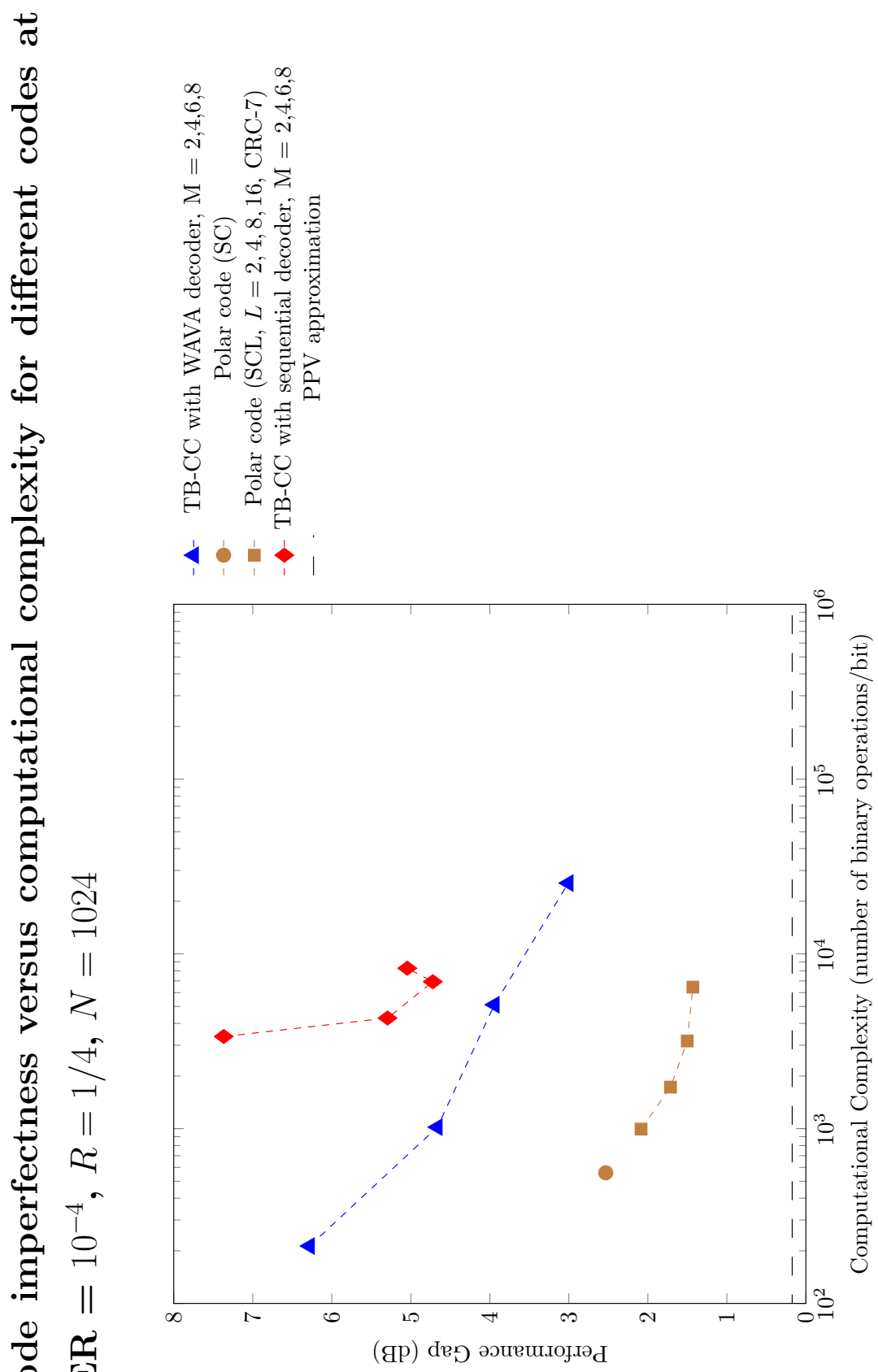

Figure C.149: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 1/4$, $N = 1024$

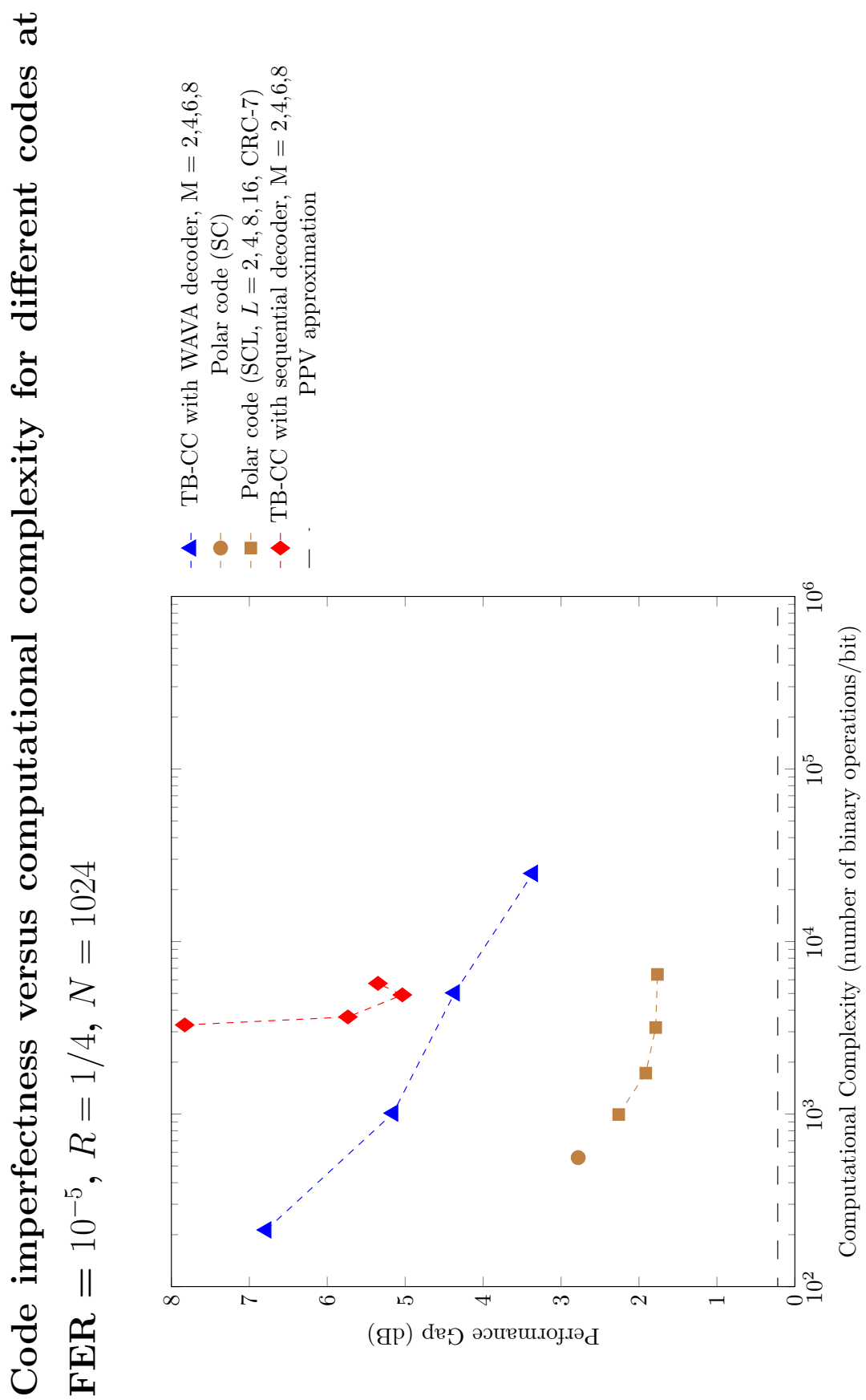

Figure C.150: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 1/4$, $N = 1024$

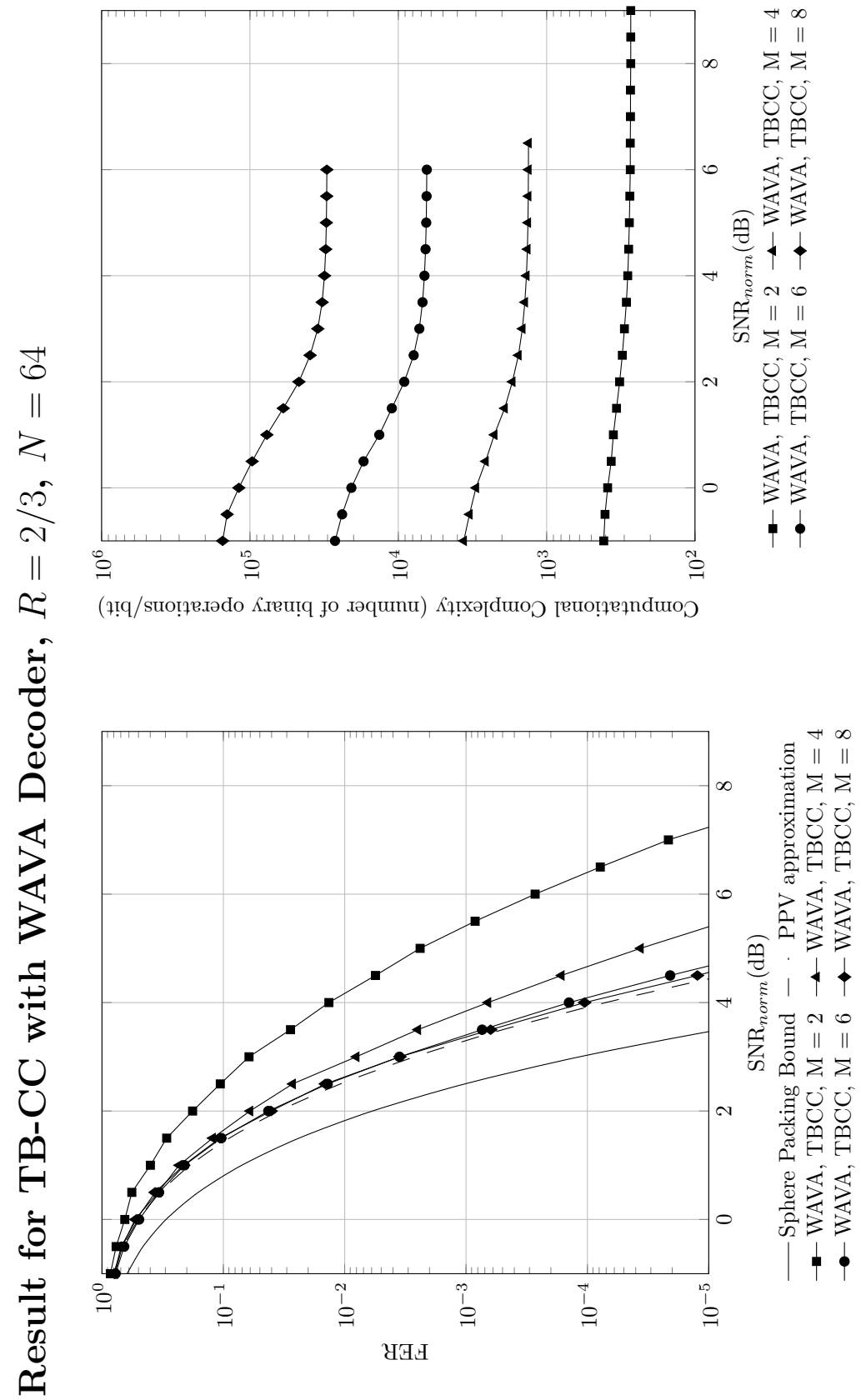

Figure C.151: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 2/3$, $N = 64$, maximum number of iterations = 4.

Figure C.152: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 2/3$, $N = 64$, maximum number of iterations = 4

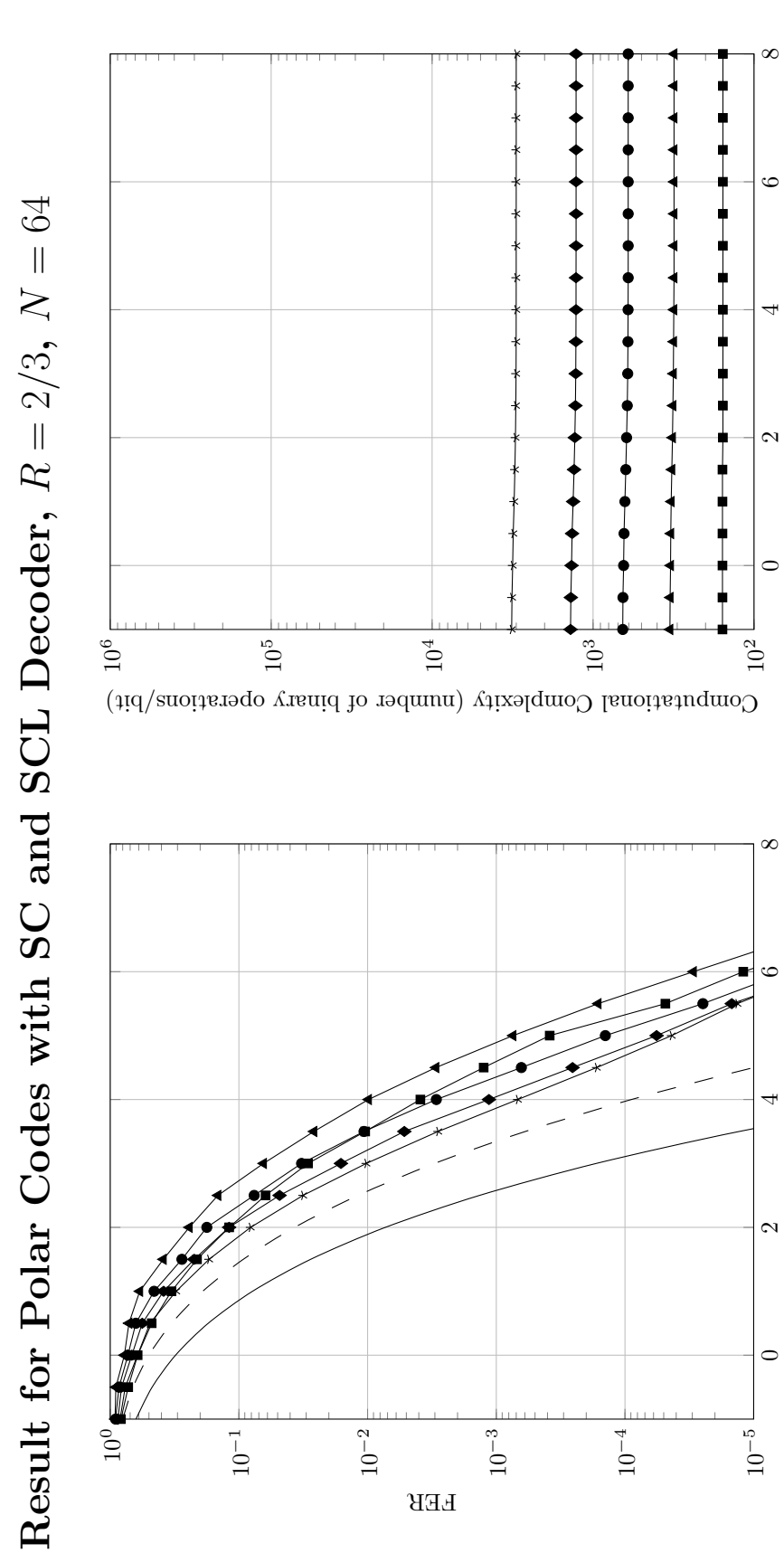

Figure C.153: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 2/3$, $N = 64$

Figure C.154: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 2/3$, $N = 64$

Result for TB-CC with Sequential Decoder, $R = 2/3$, $N = 64$

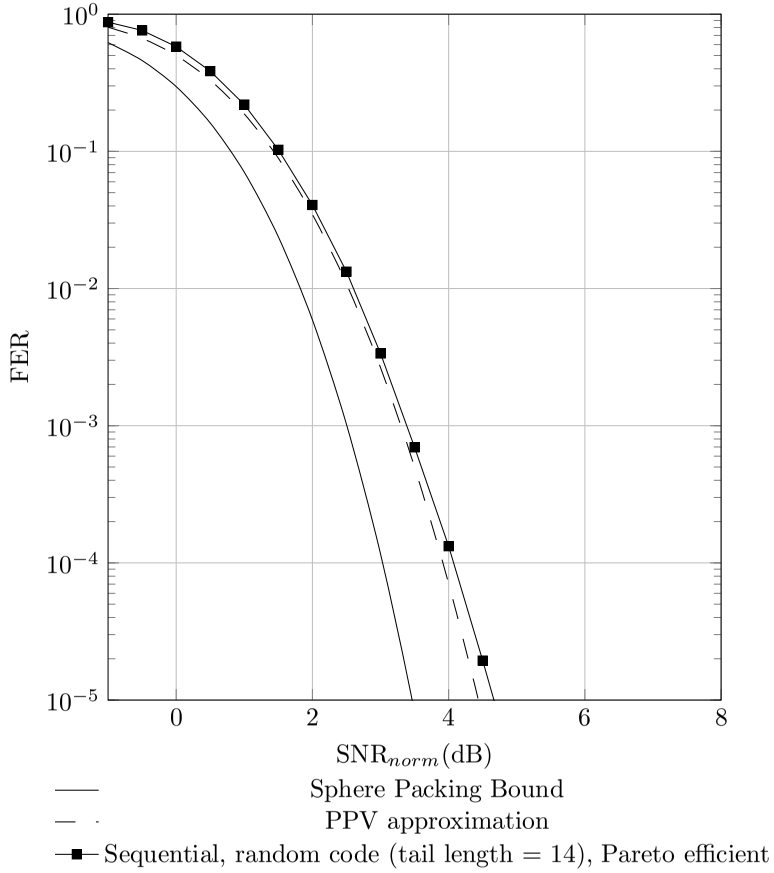

Figure C.155: Frame error rate versus normalized SNR for random code with sequential decoder, $R = 2/3$, $N = 64$

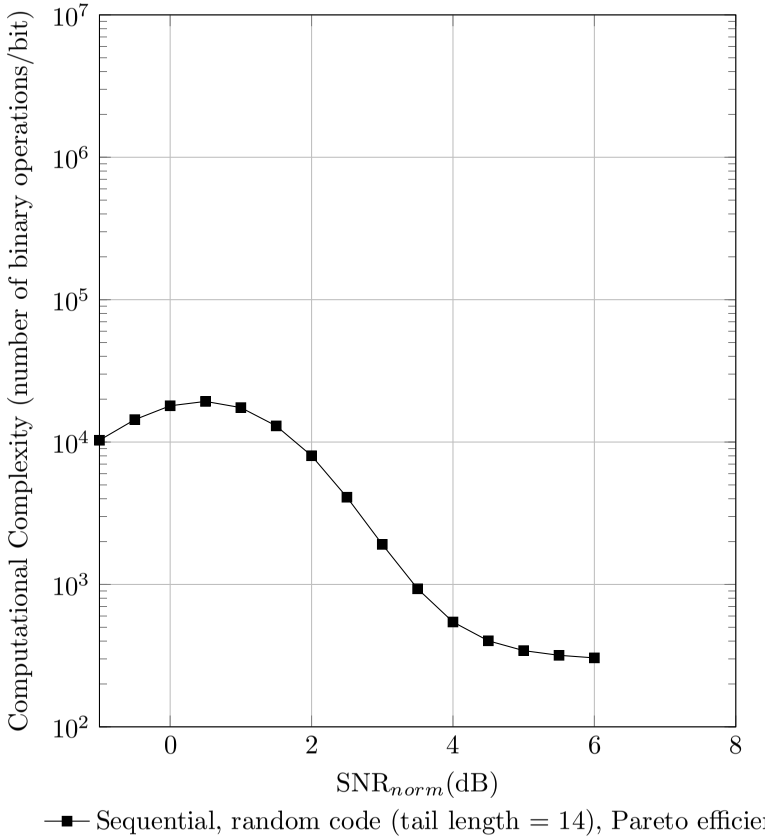

Figure C.156: Computational complexity versus normalized SNR for random code with sequential decoder, $R = 2/3$, $N = 64$

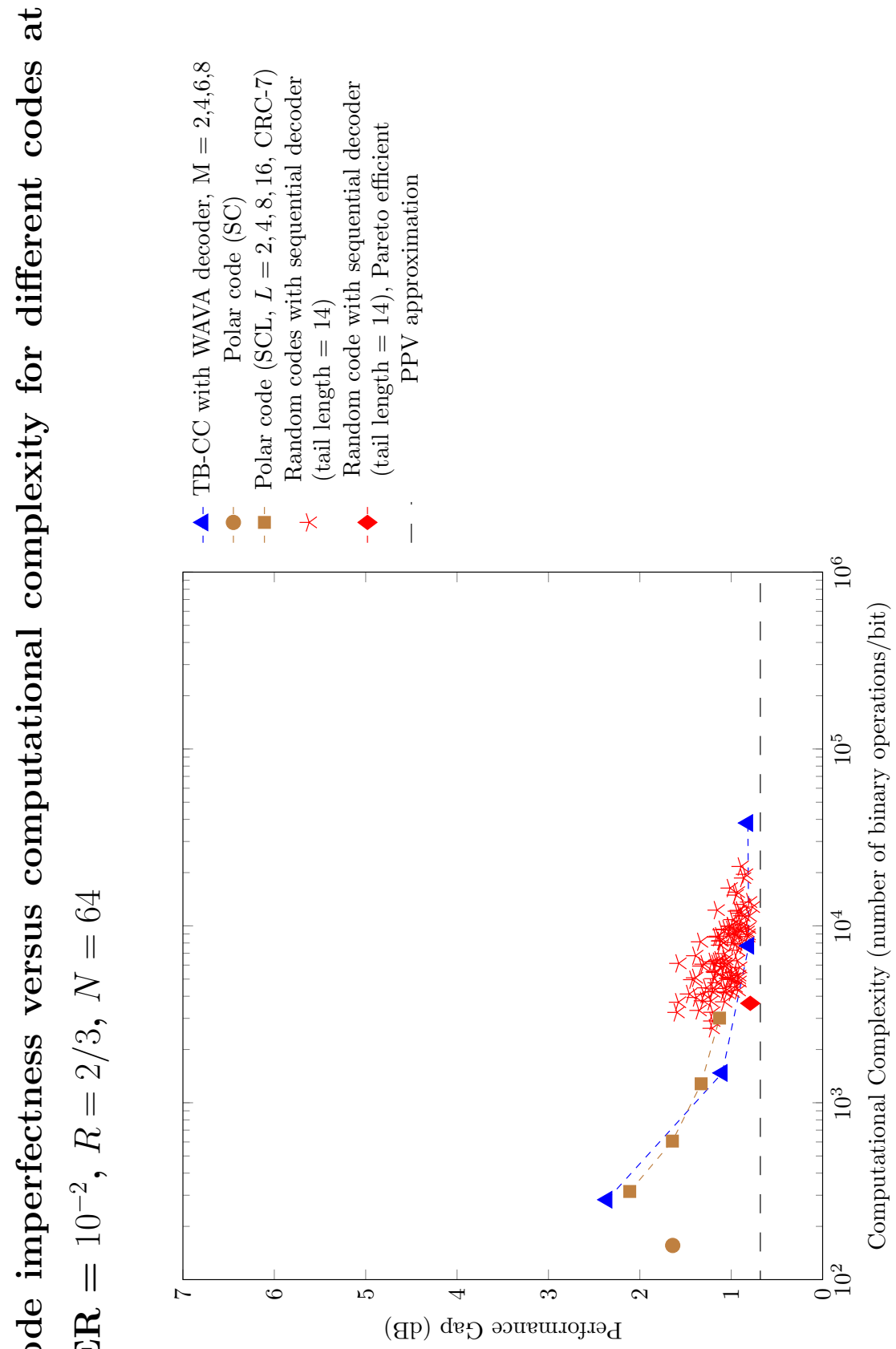

Figure C.157: Code imperfectness versus computational complexity at FER = 10^{-2} for different codes with $R = 2/3$, $N = 64$

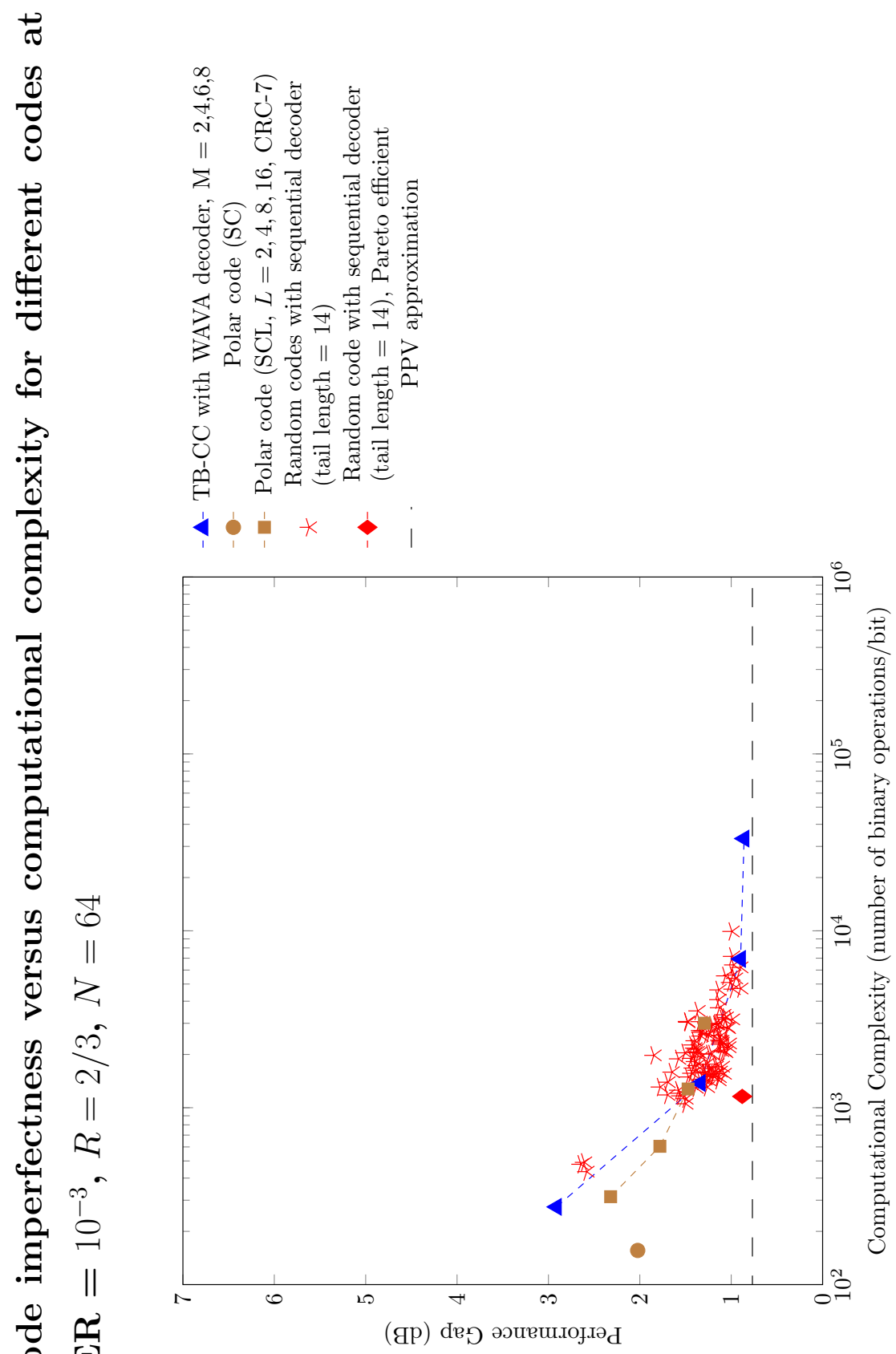

Figure C.158: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 2/3$, $N = 64$

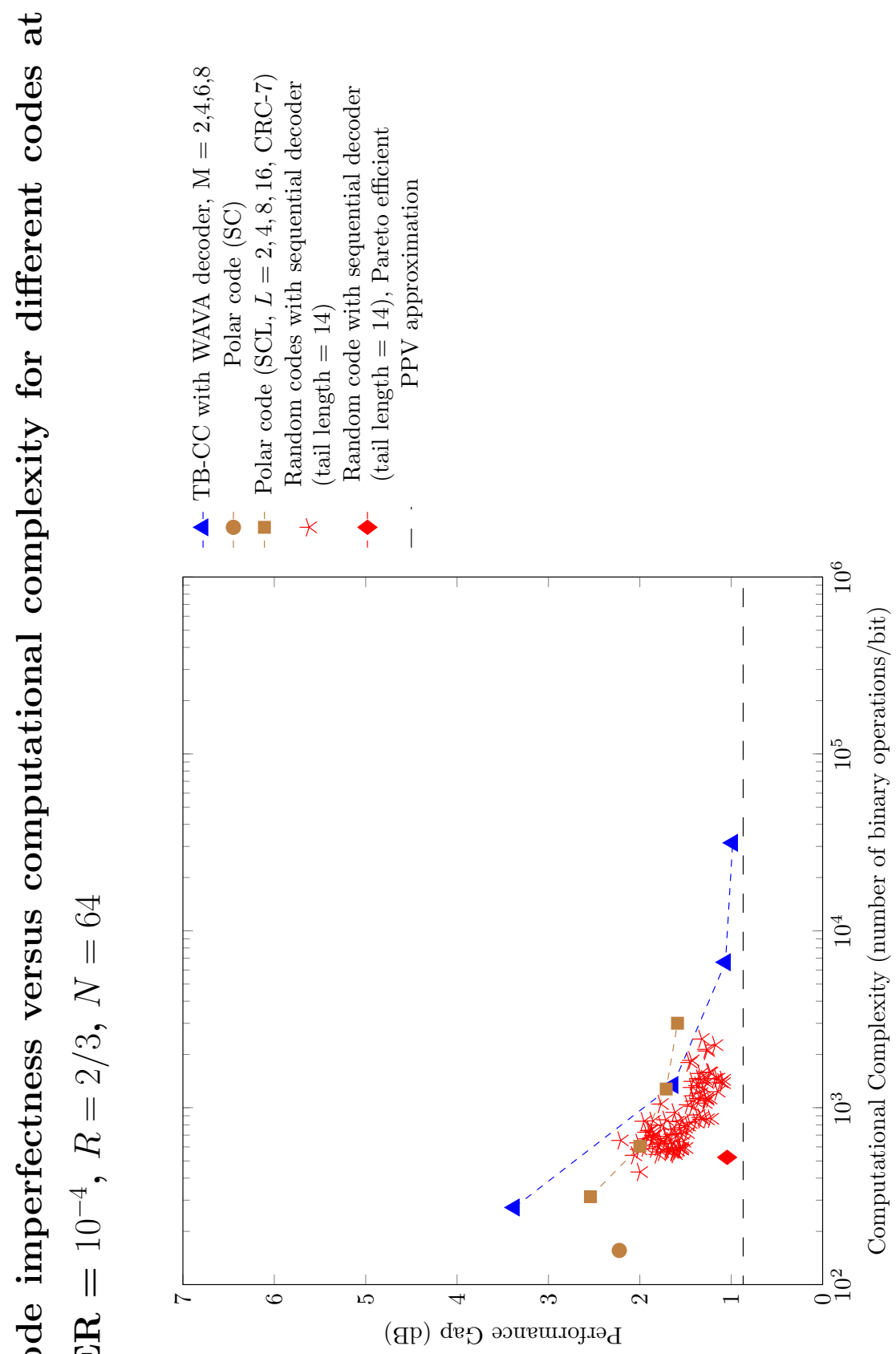

Figure C.159: Code imperfectness versus computational complexity at FER = 10^{-4} for different codes with $R = 2/3$, $N = 64$

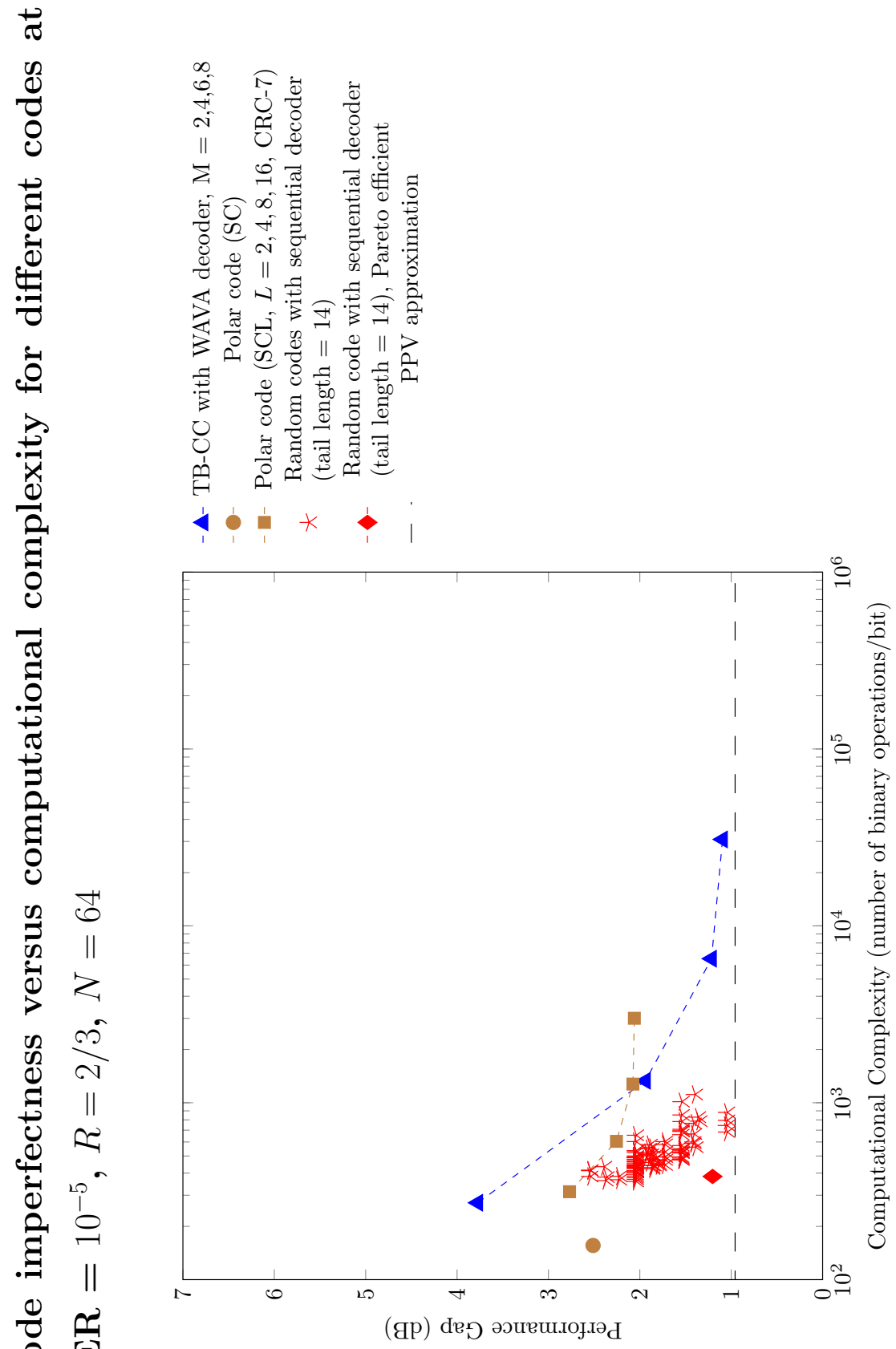

Figure C.160: Code imperfectness versus computational complexity at FER = 10^{-5} for different codes with $R = 2/3$, $N = 64$

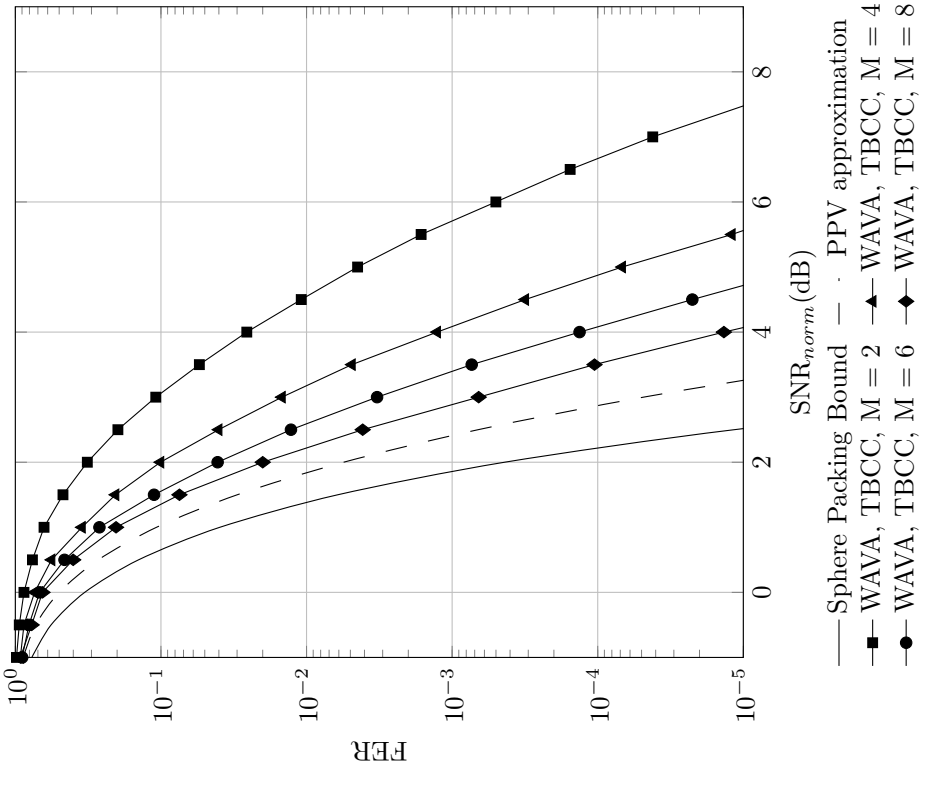

Figure C.161: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 2/3$, $N = 128$, maximum number of iterations = 4.

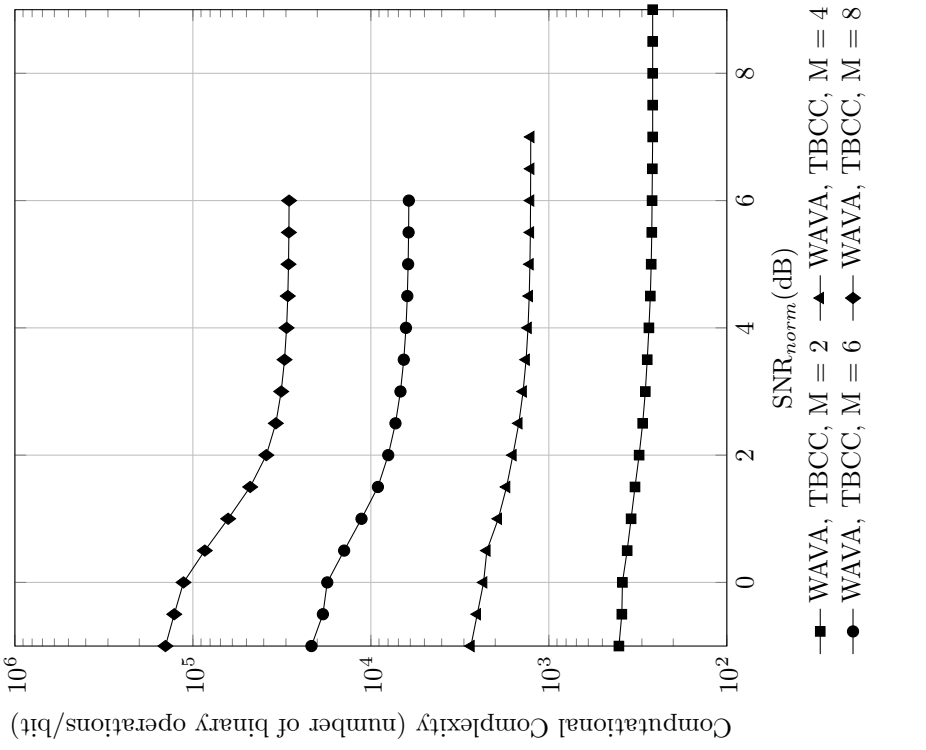

Figure C.162: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 2/3$, $N = 128$, maximum number of iterations = 4

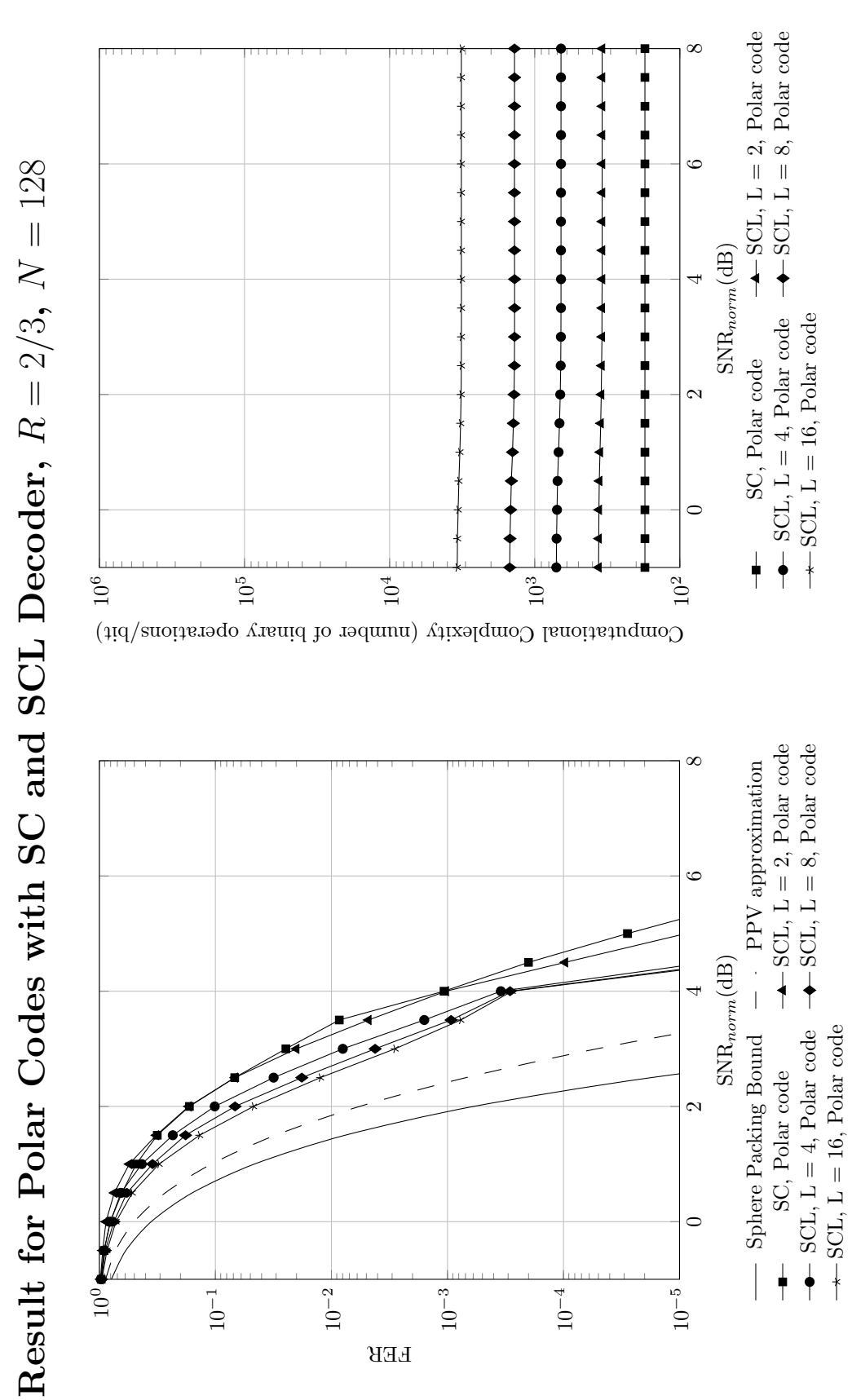

Figure C.163: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 2/3$, $N = 128$

Figure C.164: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 2/3$, $N = 128$

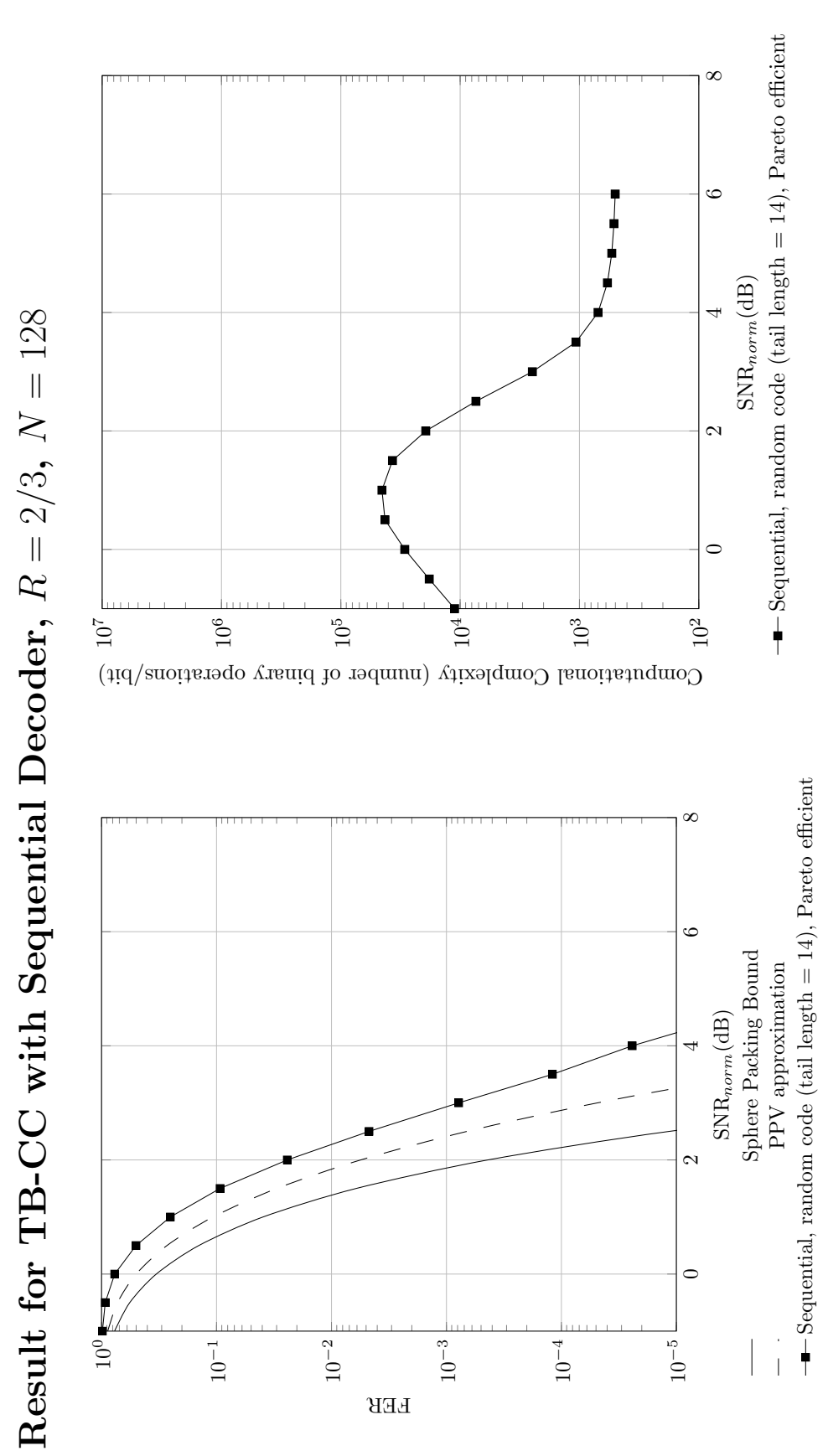

Figure C.165: Frame error rate versus normalized SNR for random code with sequential decoder, $R = 2/3$, $N = 128$

Figure C.166: Computational complexity versus normalized SNR for random code with sequential decoder, $R = 2/3$, $N = 128$

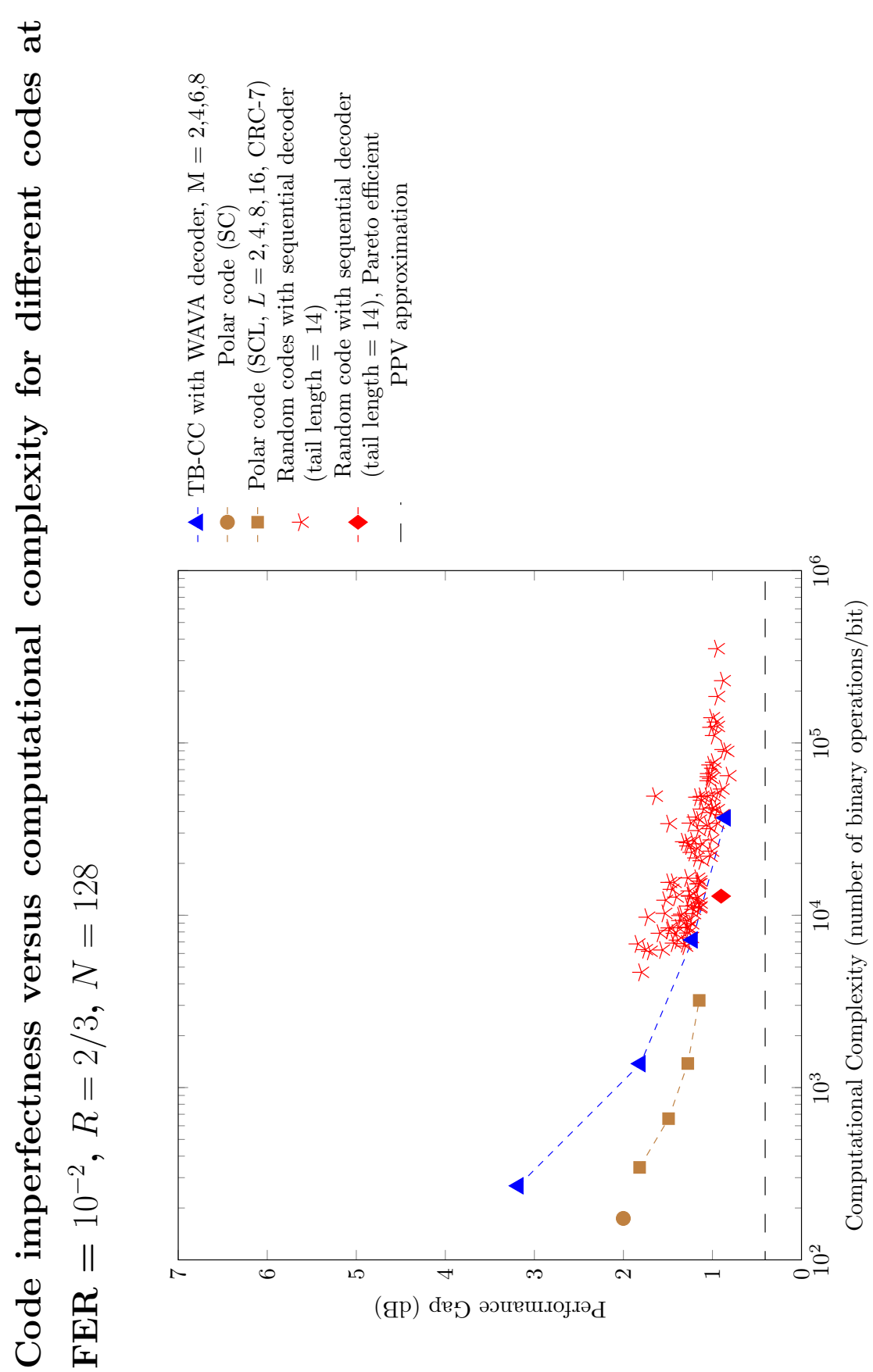

Figure C.167: Code imperfectness versus computational complexity at FER = 10^{-2} for different codes with $R = 2/3$, $N = 128$

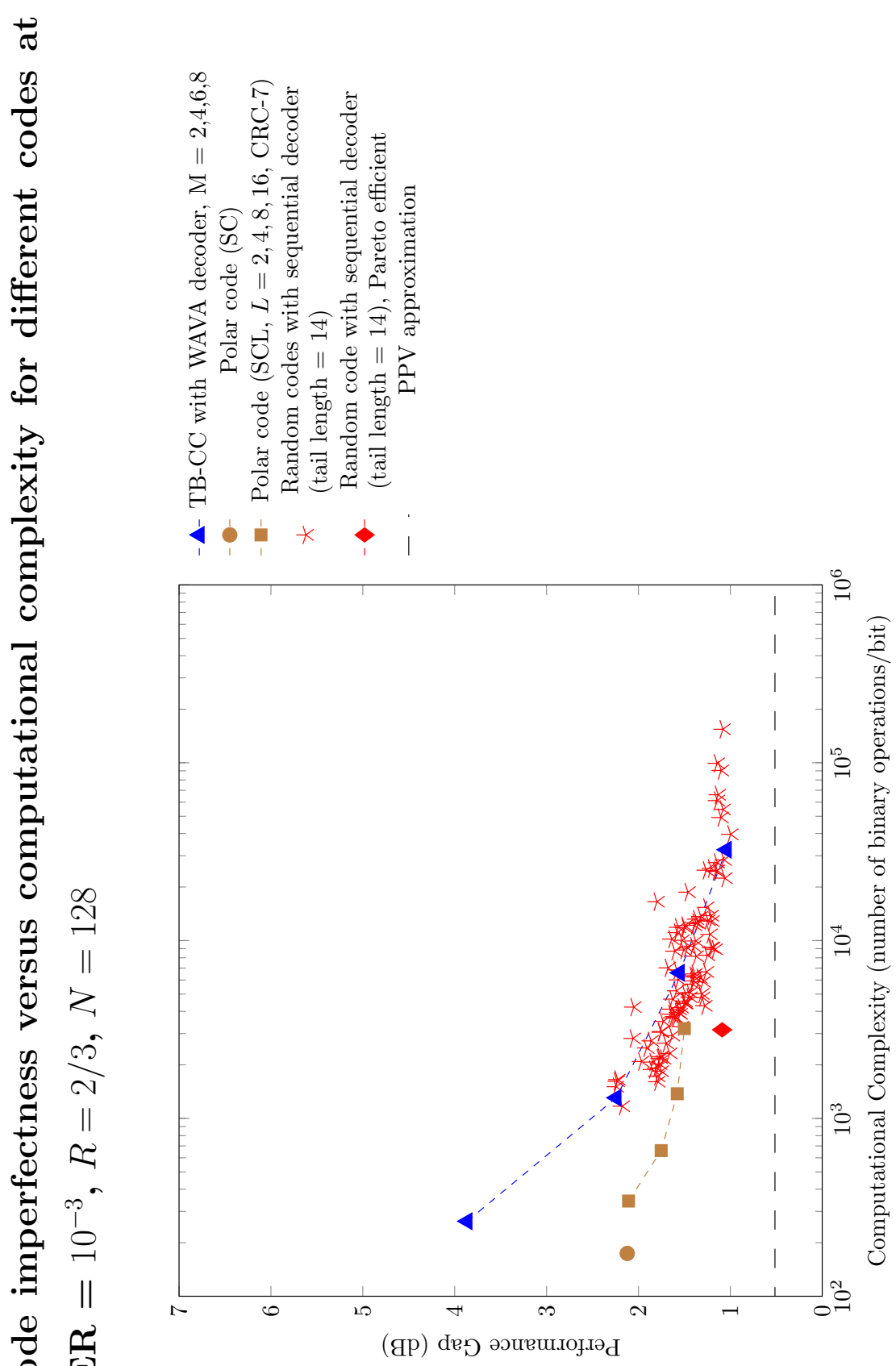

Figure C.168: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 2/3$, $N = 128$

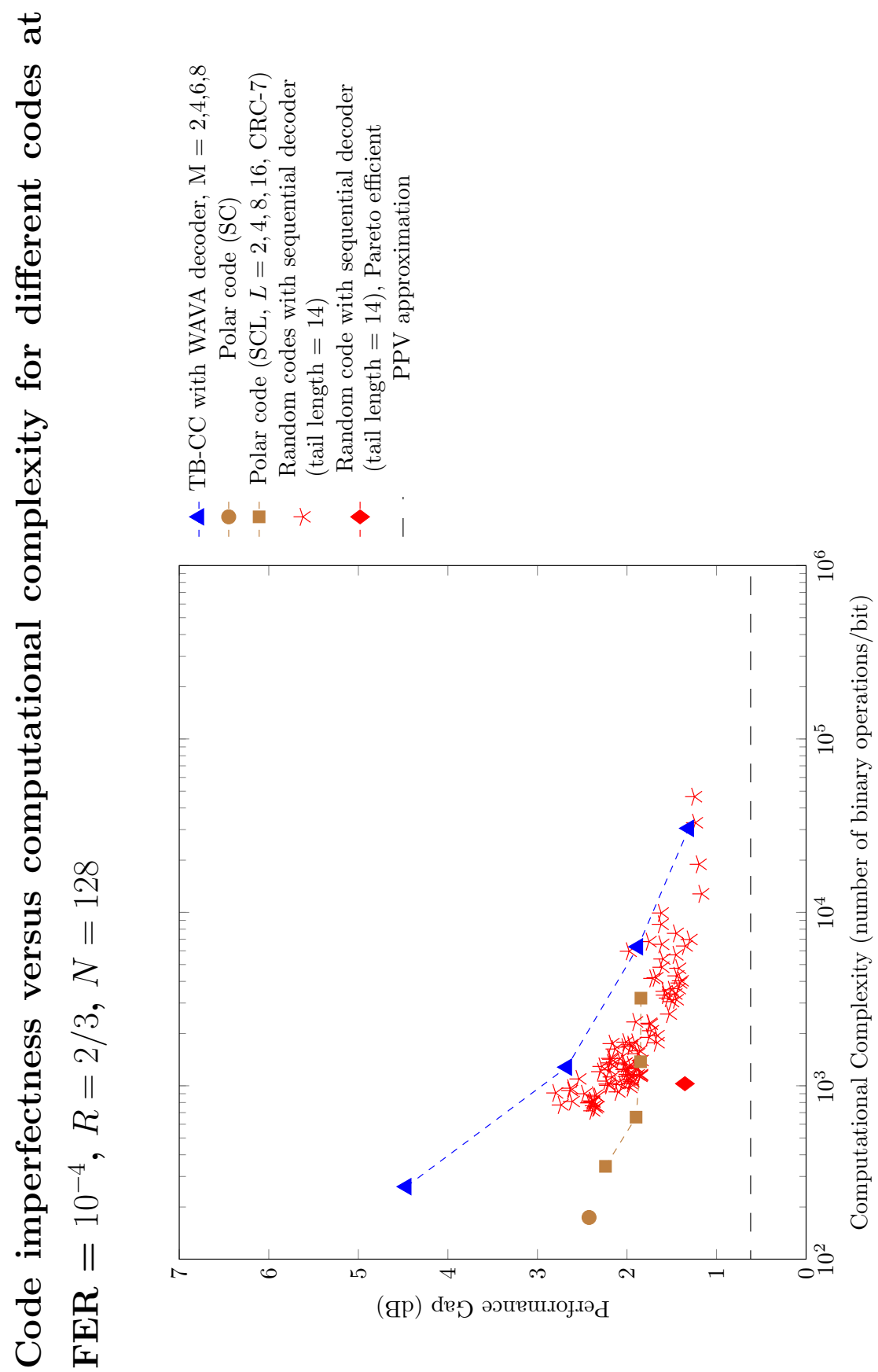

Figure C.169: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 2/3$, $N = 128$

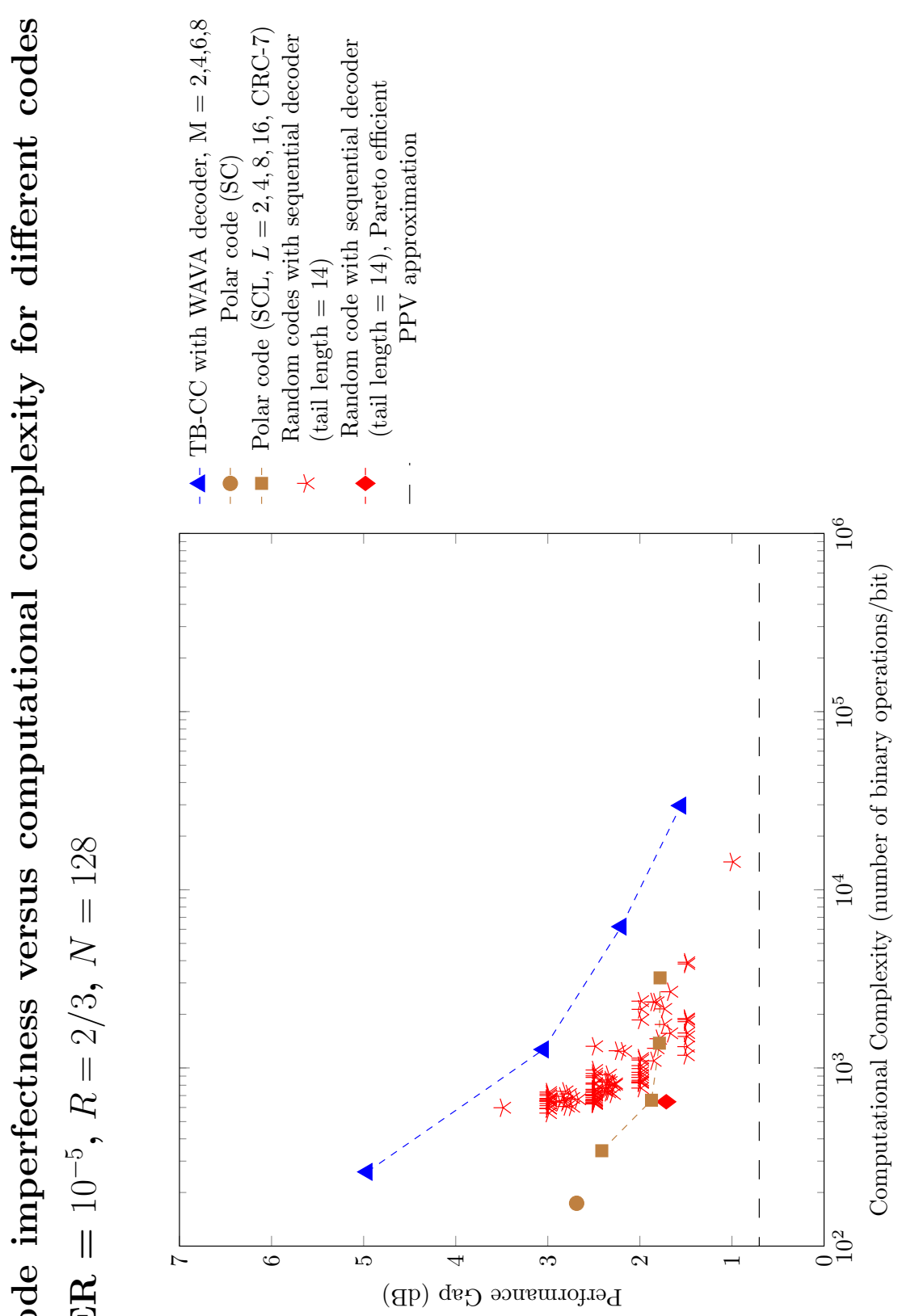

Figure C.170: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 2/3$, $N = 128$

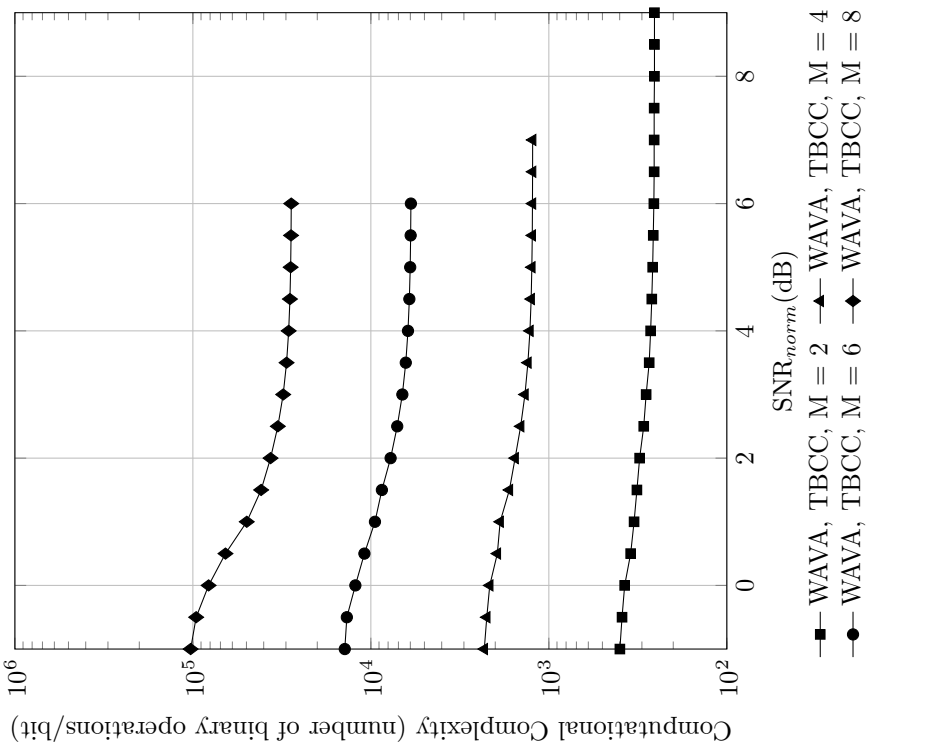

Figure C.172: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 2/3$, $N = 256$, maximum number of iterations = 4

Result for TB-CC with WAVA Decoder, $R = 2/3$, $N = 256$

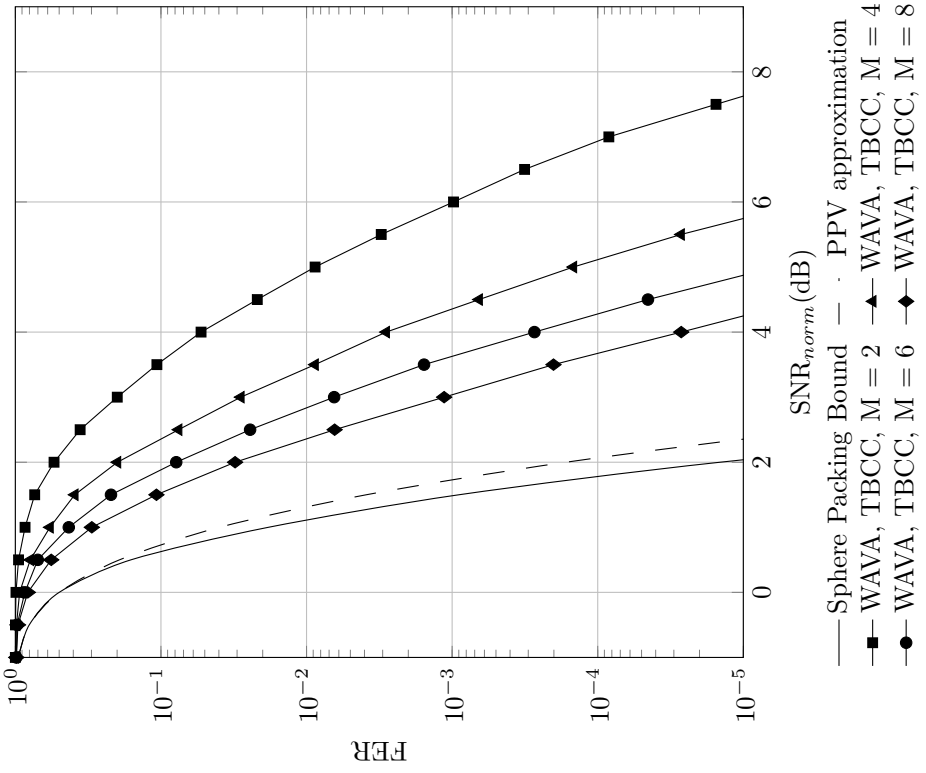

Figure C.171: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 2/3$, $N = 256$, maximum number of iterations = 4.

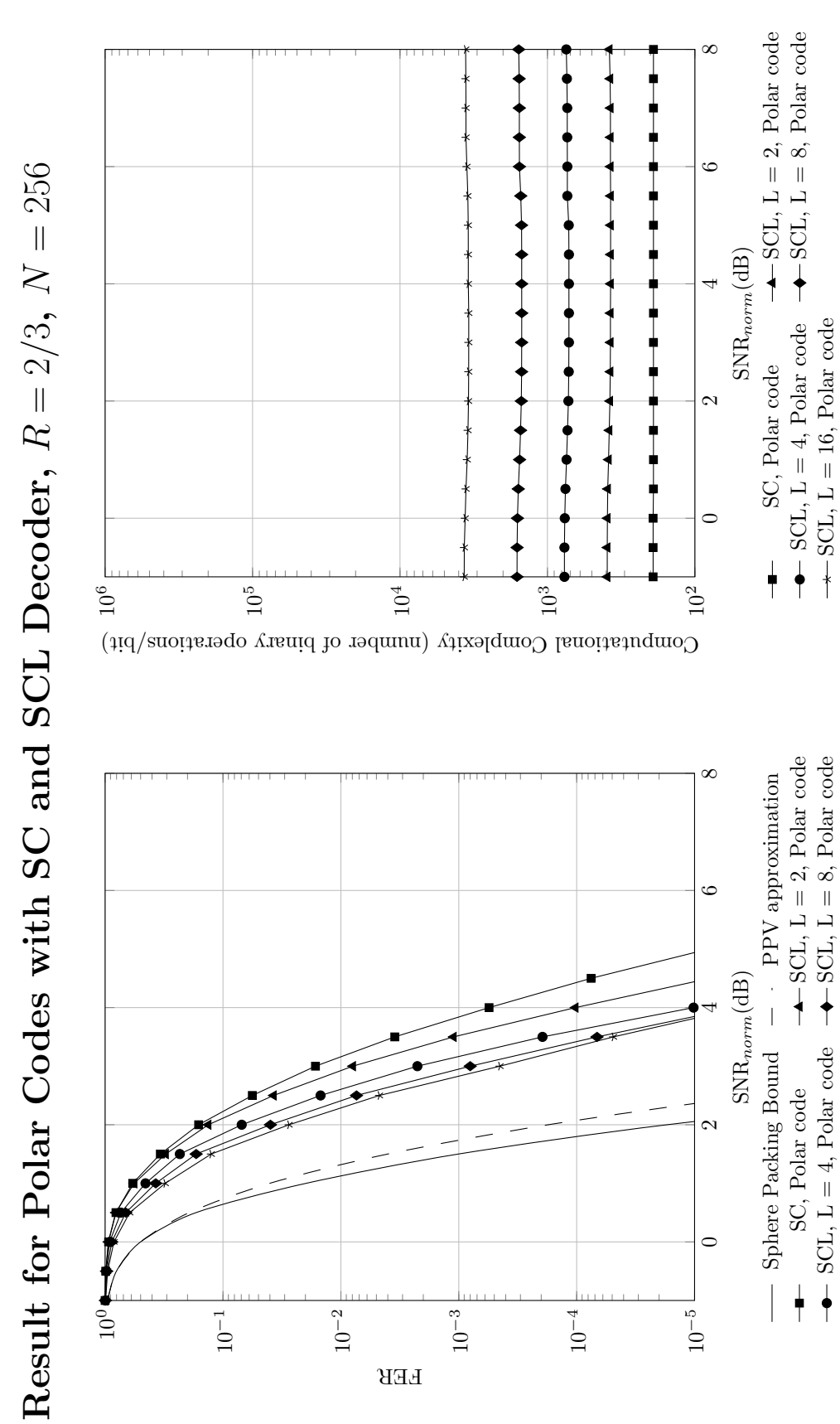

Figure C.173: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 2/3$, $N = 256$

Figure C.174: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 2/3$, $N = 256$

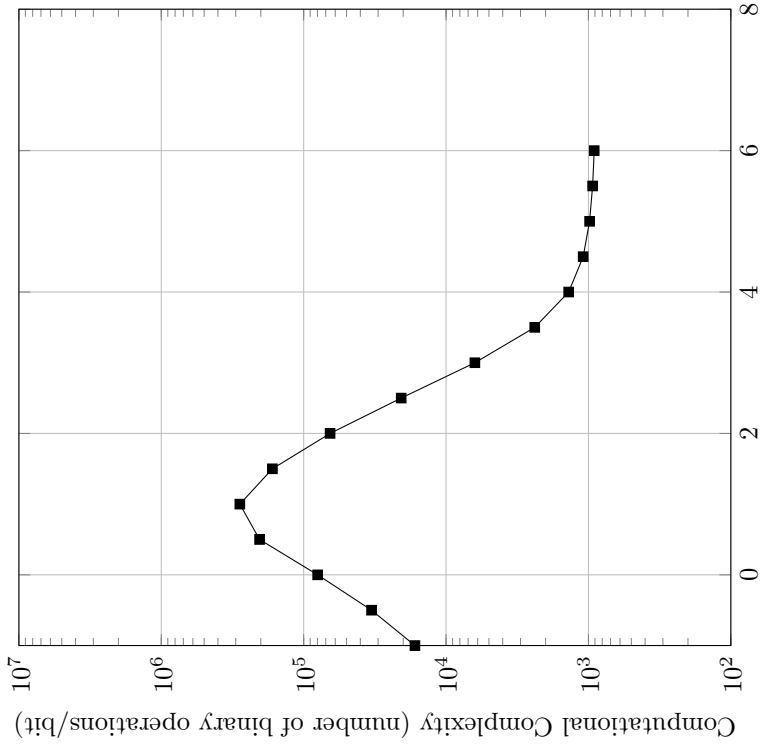

Figure C.176: Computational complexity versus normalized SNR for random code with sequential decoder, $R = 2/3$, $N = 256$

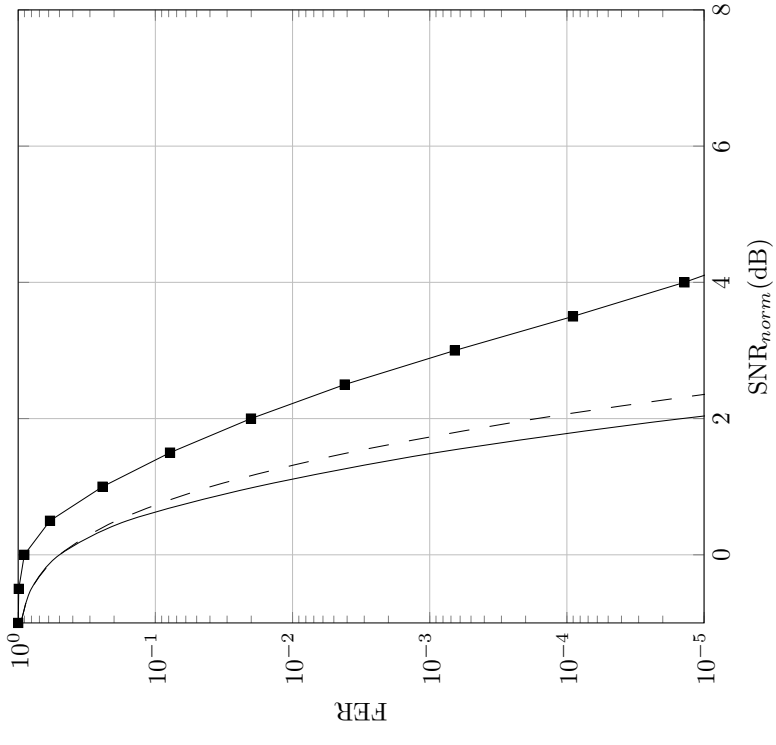

Figure C.175: Frame error rate versus normalized SNR for random code with sequential decoder, $R = 2/3$, $N = 256$

Result for TB-CC with Sequential Decoder, $R = 2/3$, $N = 256$

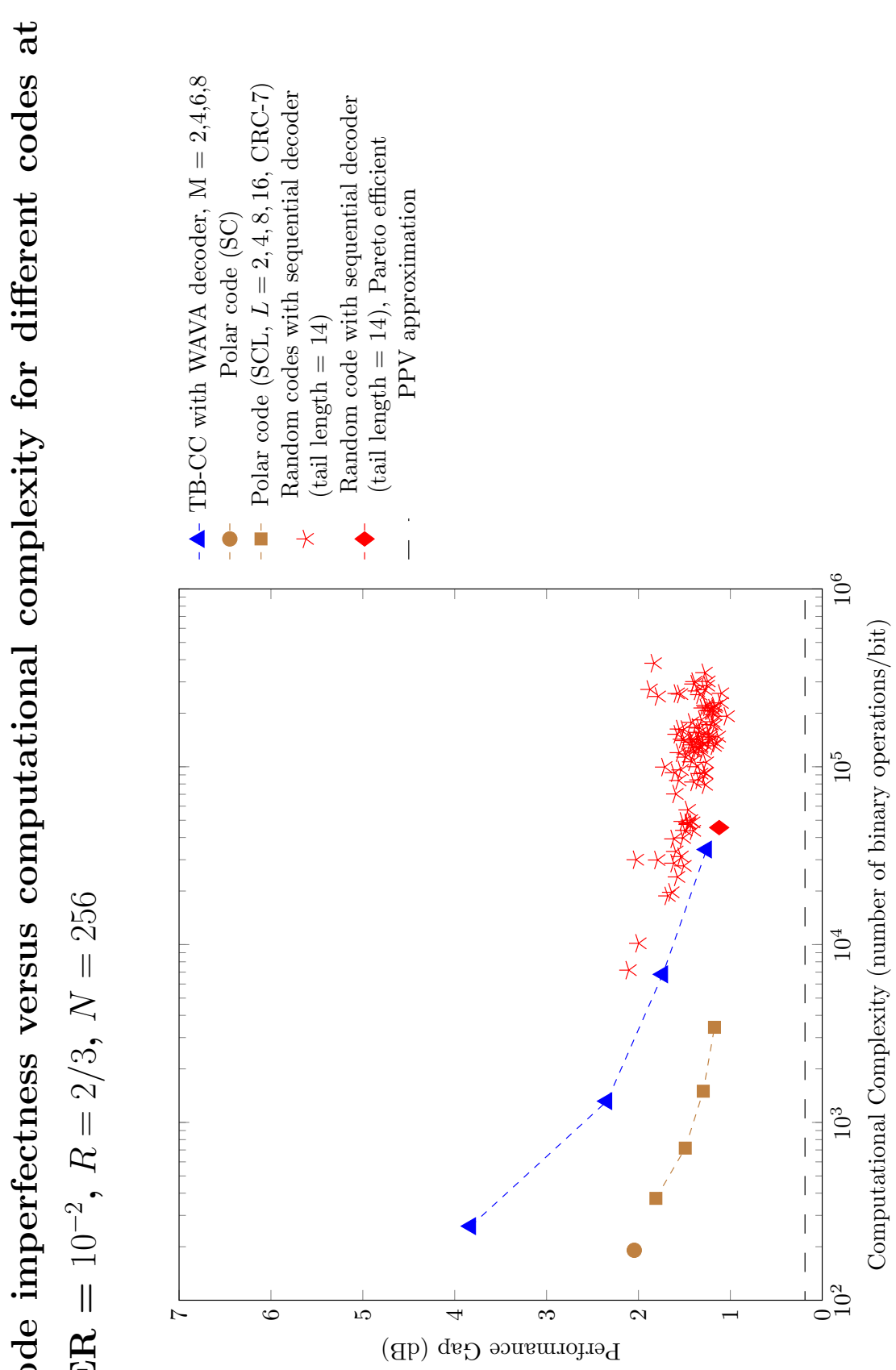

Figure C.177: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 2/3$, $N = 256$

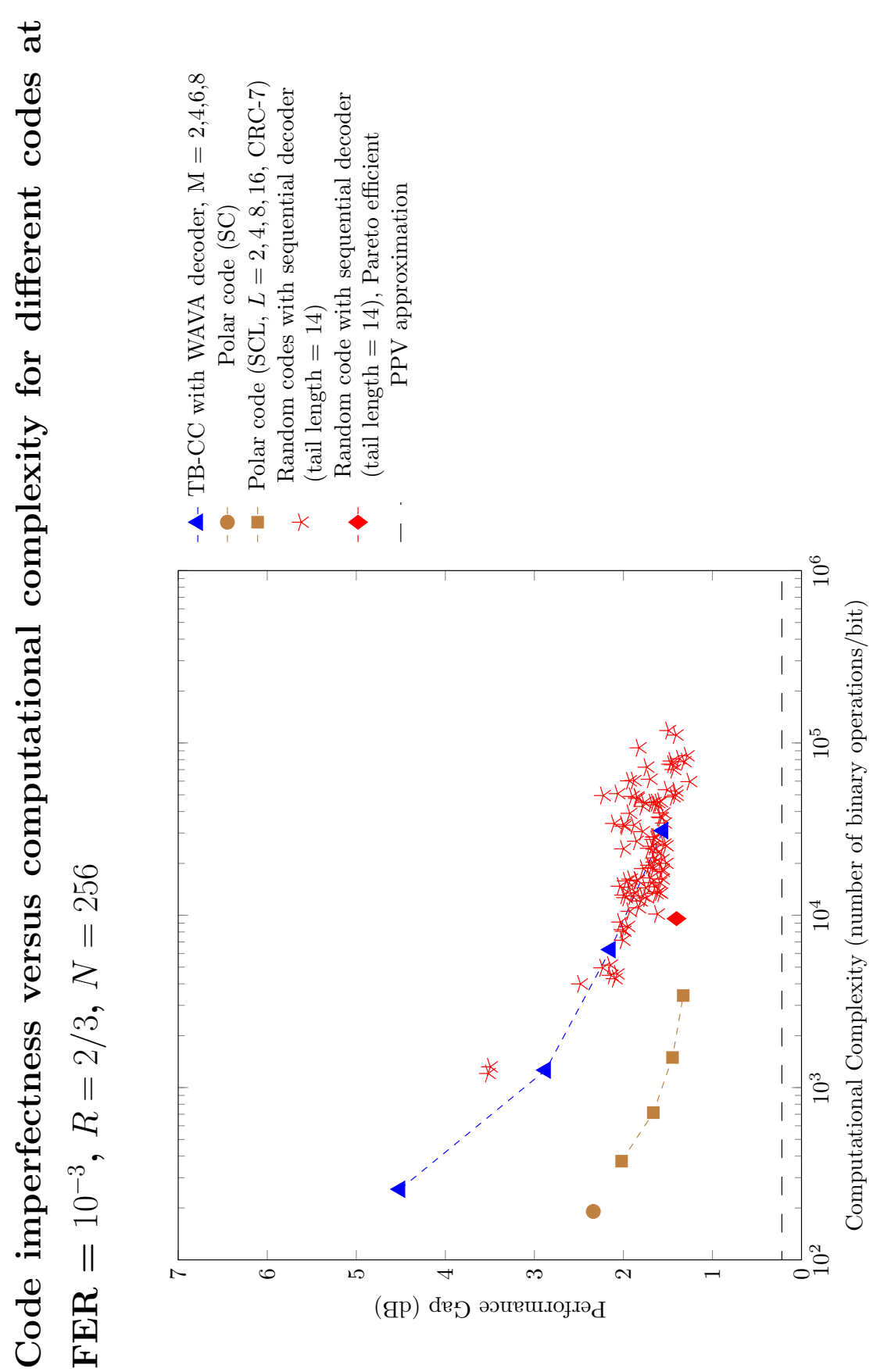

Figure C.178: Code imperfectness versus computational complexity at FER = 10^{-3} for different codes with $R = 2/3$, $N = 256$

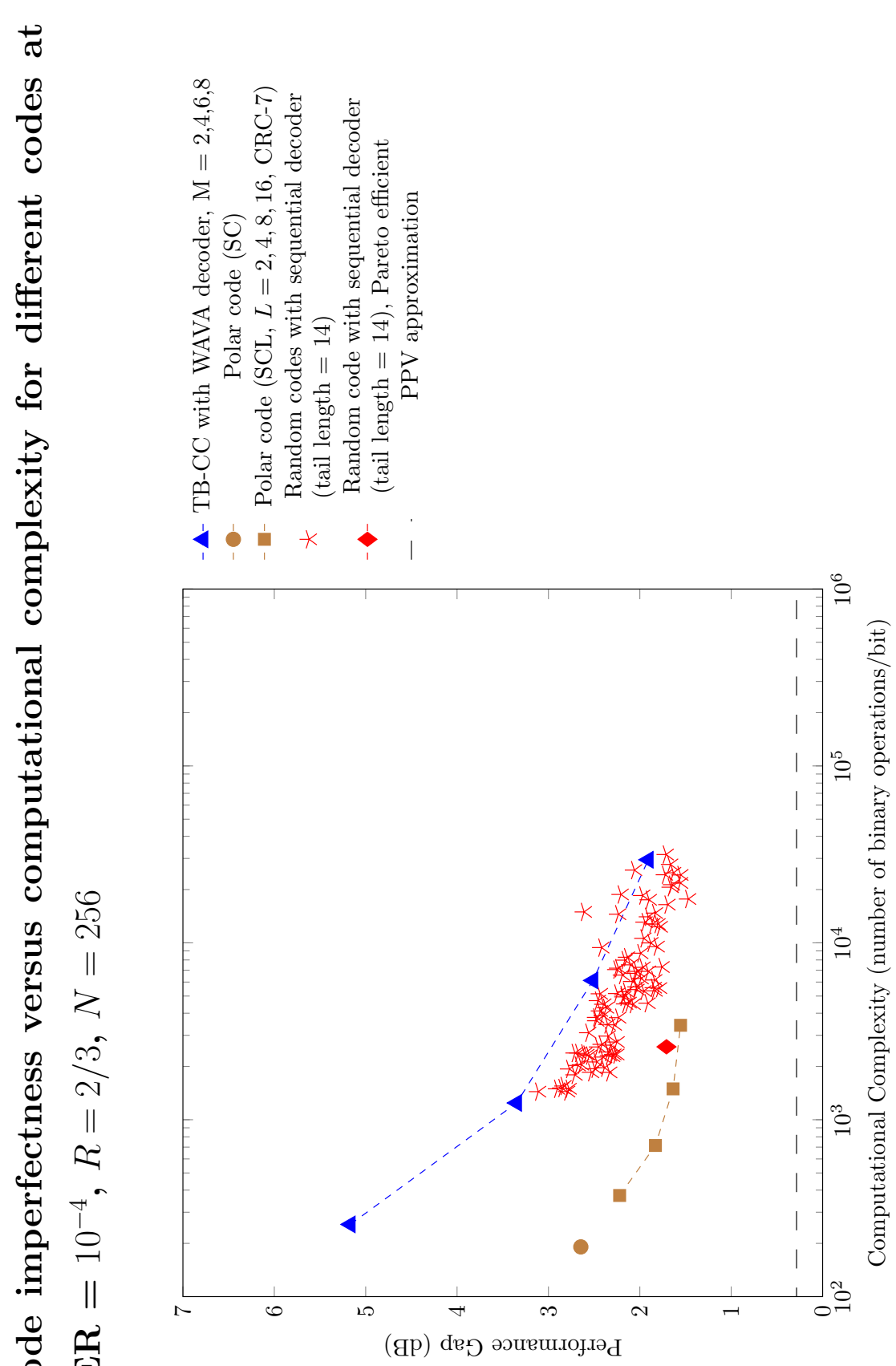

Figure C.179: Code imperfectness versus computational complexity at FER = 10^{-4} for different codes with $R = 2/3$, $N = 256$

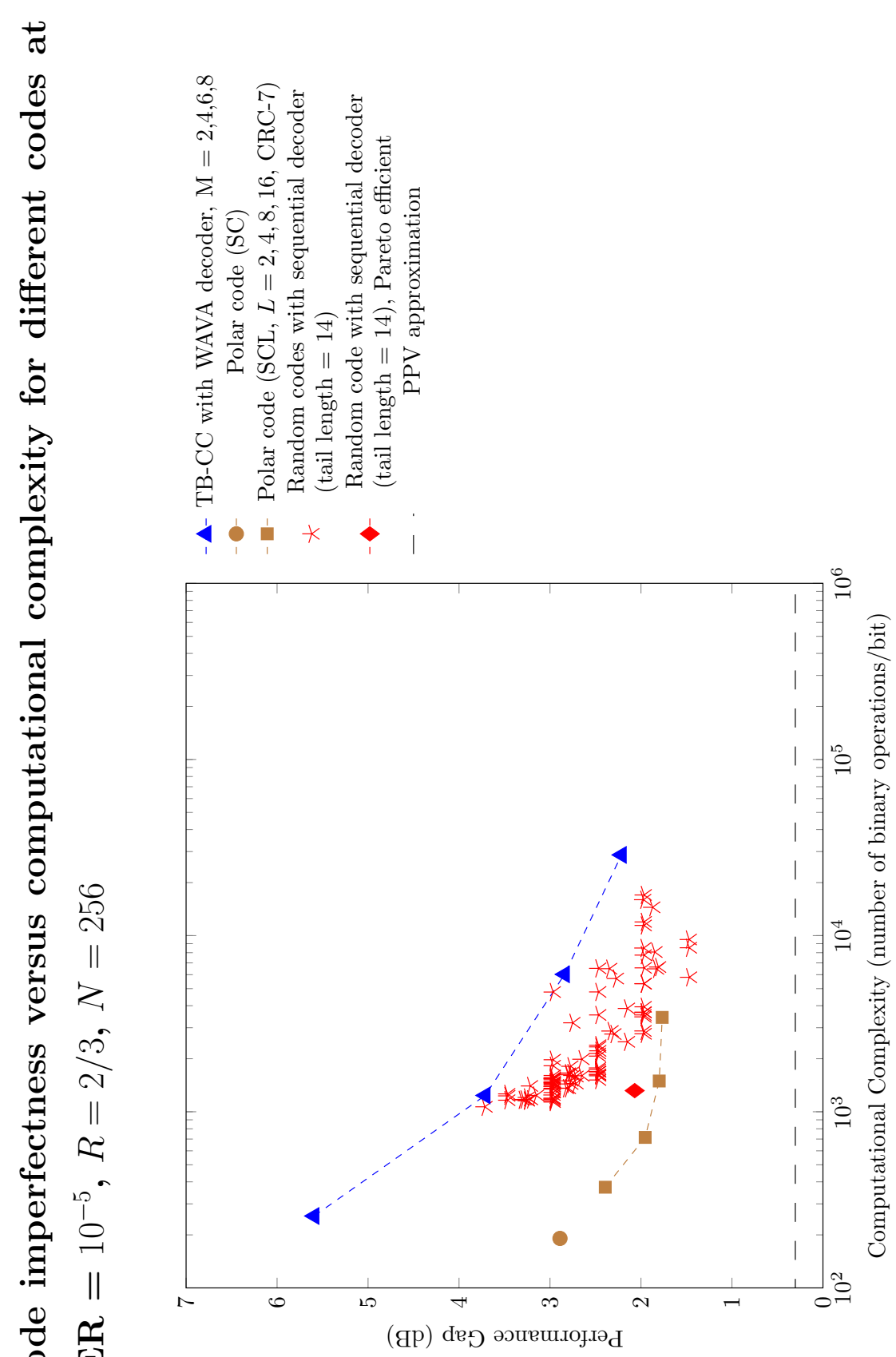

Figure C.180: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 2/3$, $N = 256$

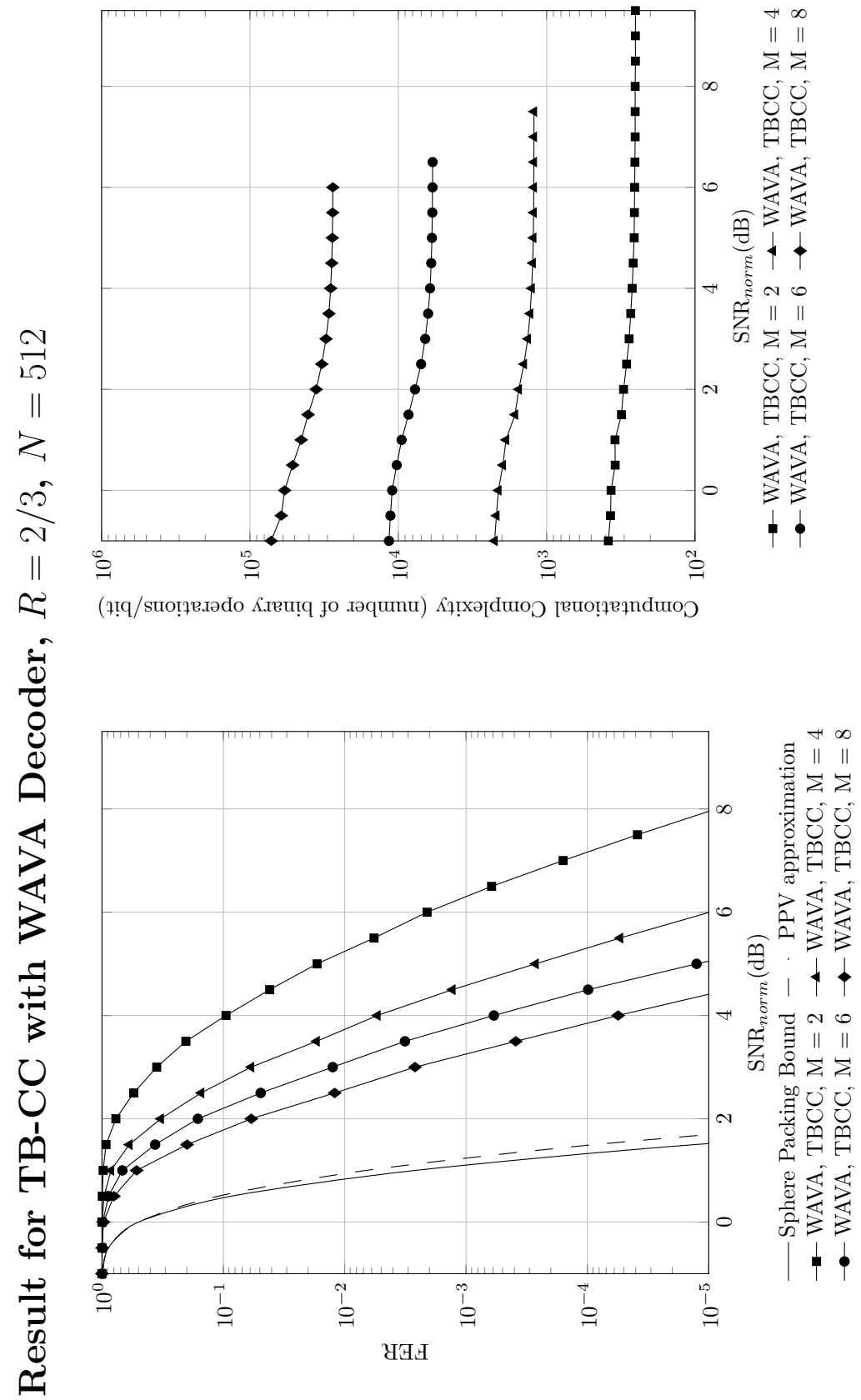

Figure C.181: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 2/3$, $N = 512$, maximum number of iterations = 4.

Figure C.182: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 2/3$, $N = 512$, maximum number of iterations = 4

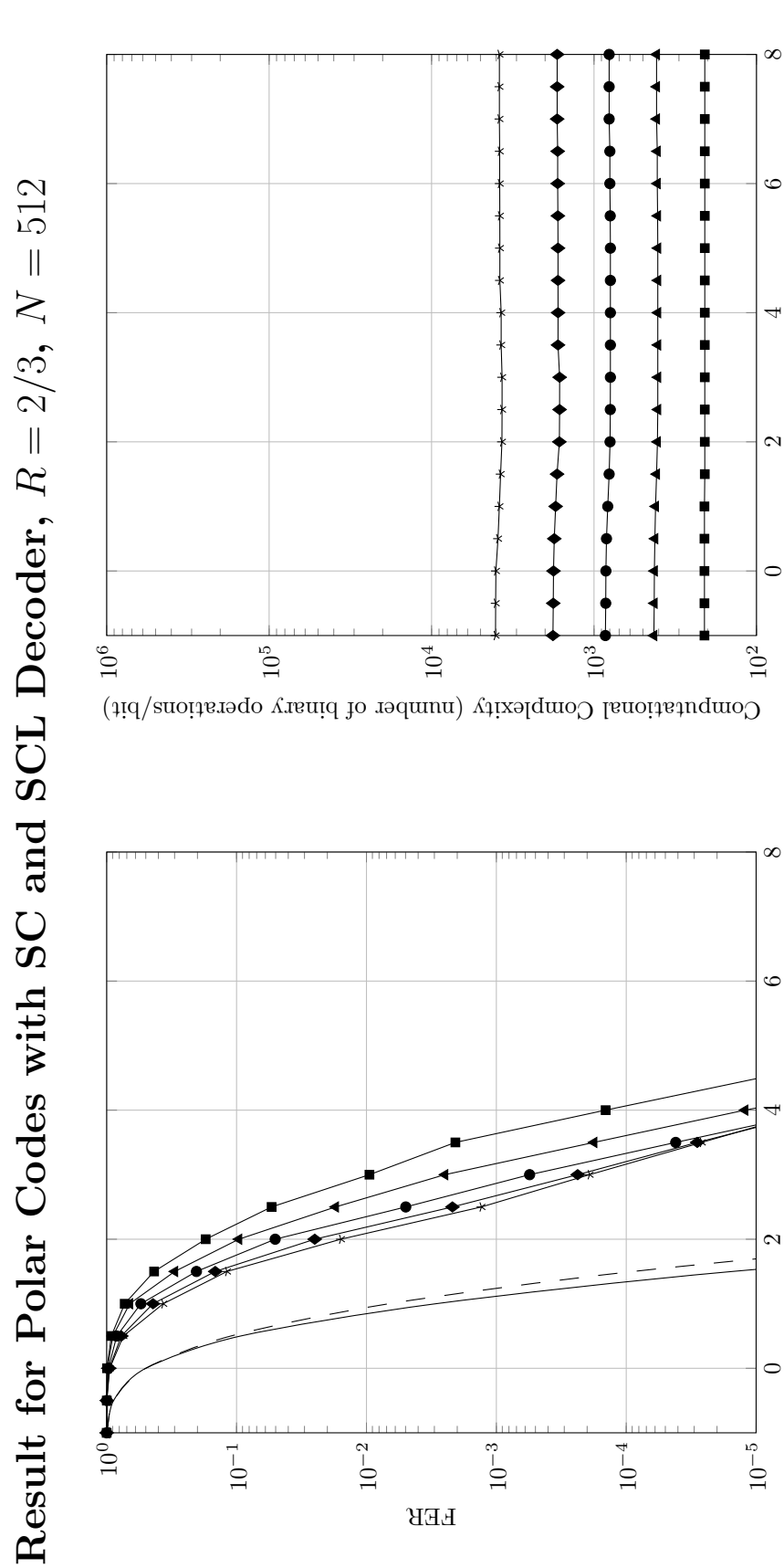

Figure C.183: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 2/3$, $N = 512$

Figure C.184: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 2/3$, $N = 512$

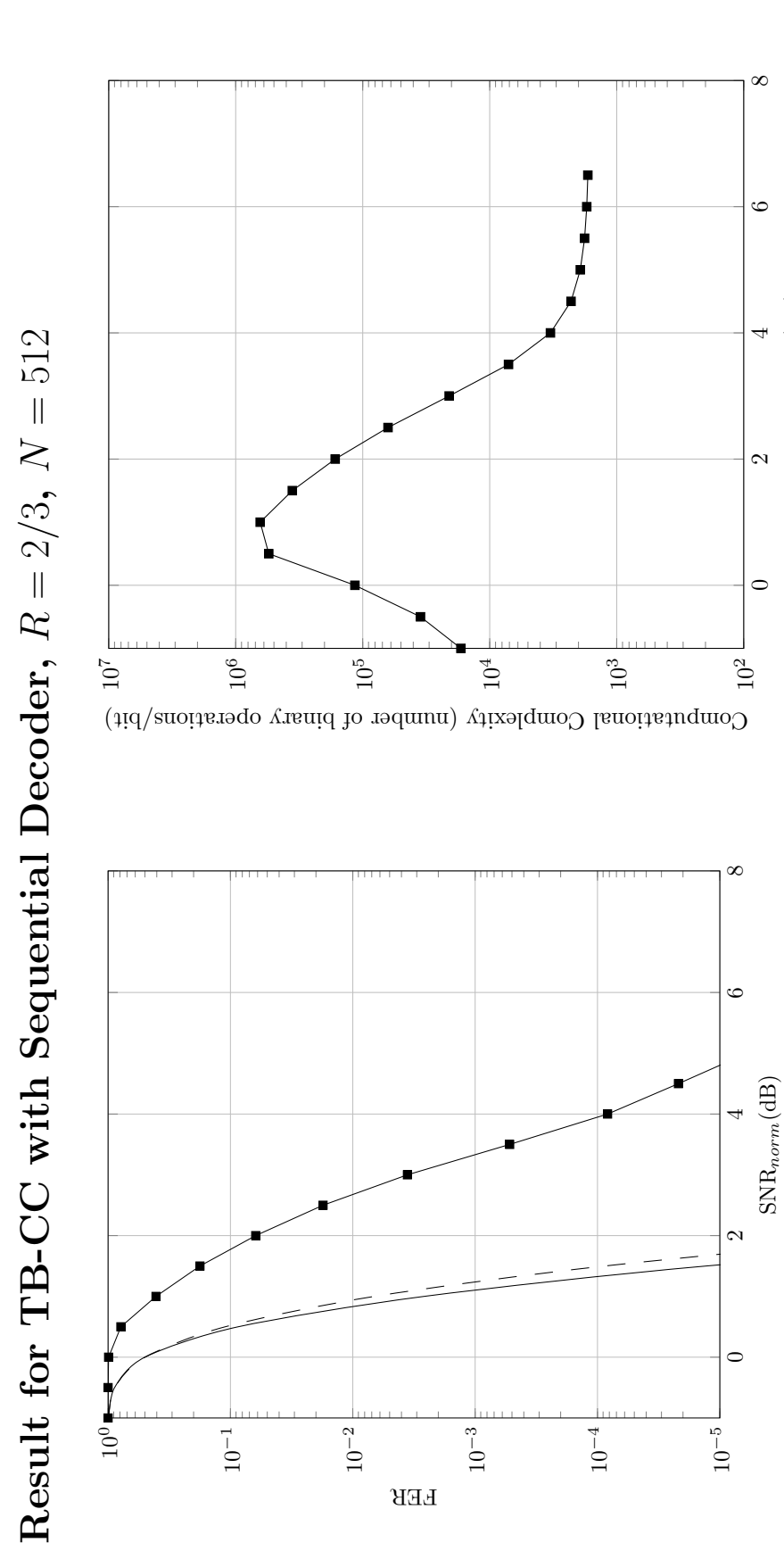

Figure C.185: Frame error rate versus normalized SNR for random code with sequential decoder, $R = 2/3$, $N = 512$

Figure C.186: Computational complexity versus normalized SNR for random code with sequential decoder, $R = 2/3$, $N = 512$

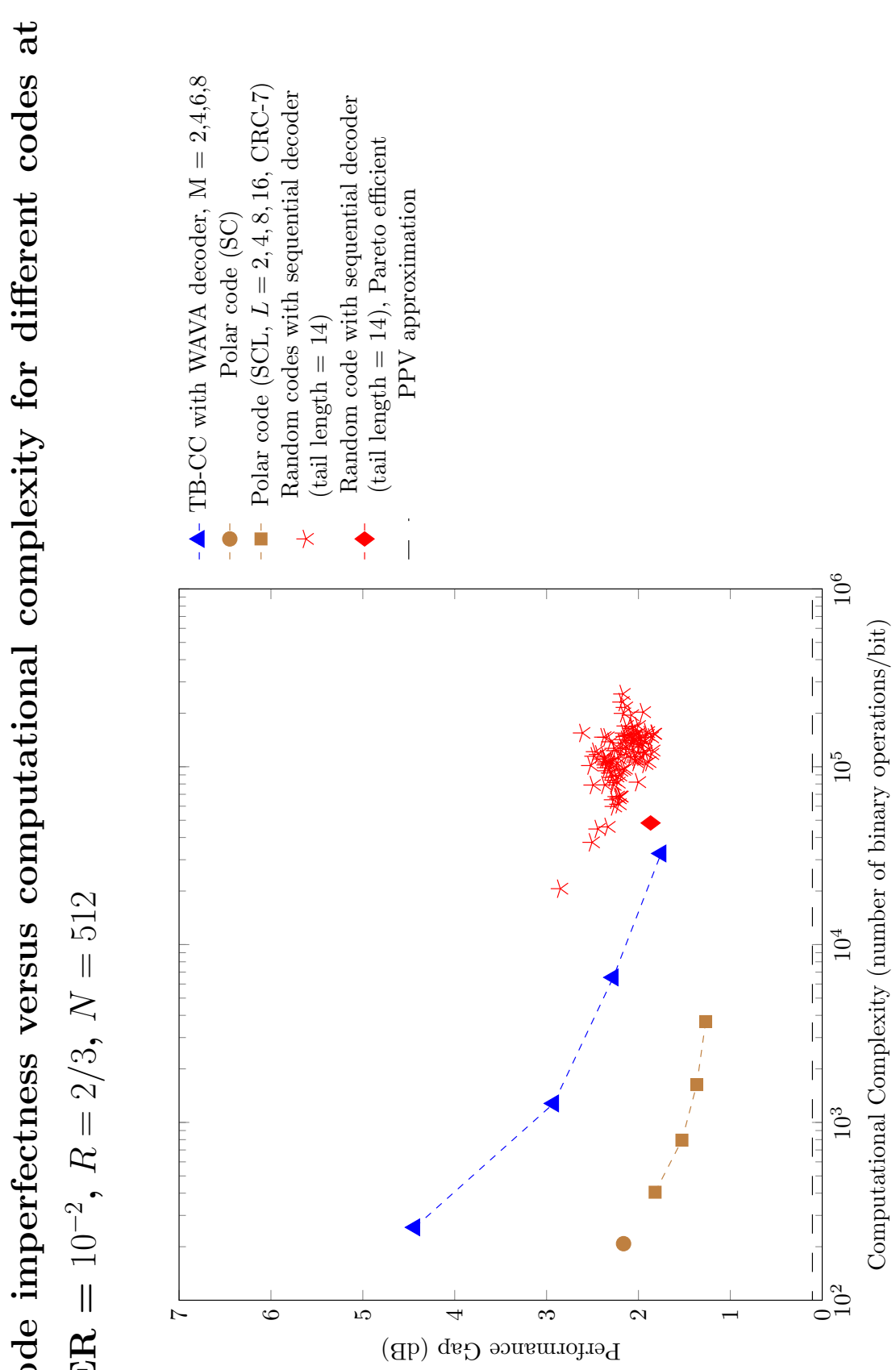

Figure C.187: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 2/3$, $N = 512$

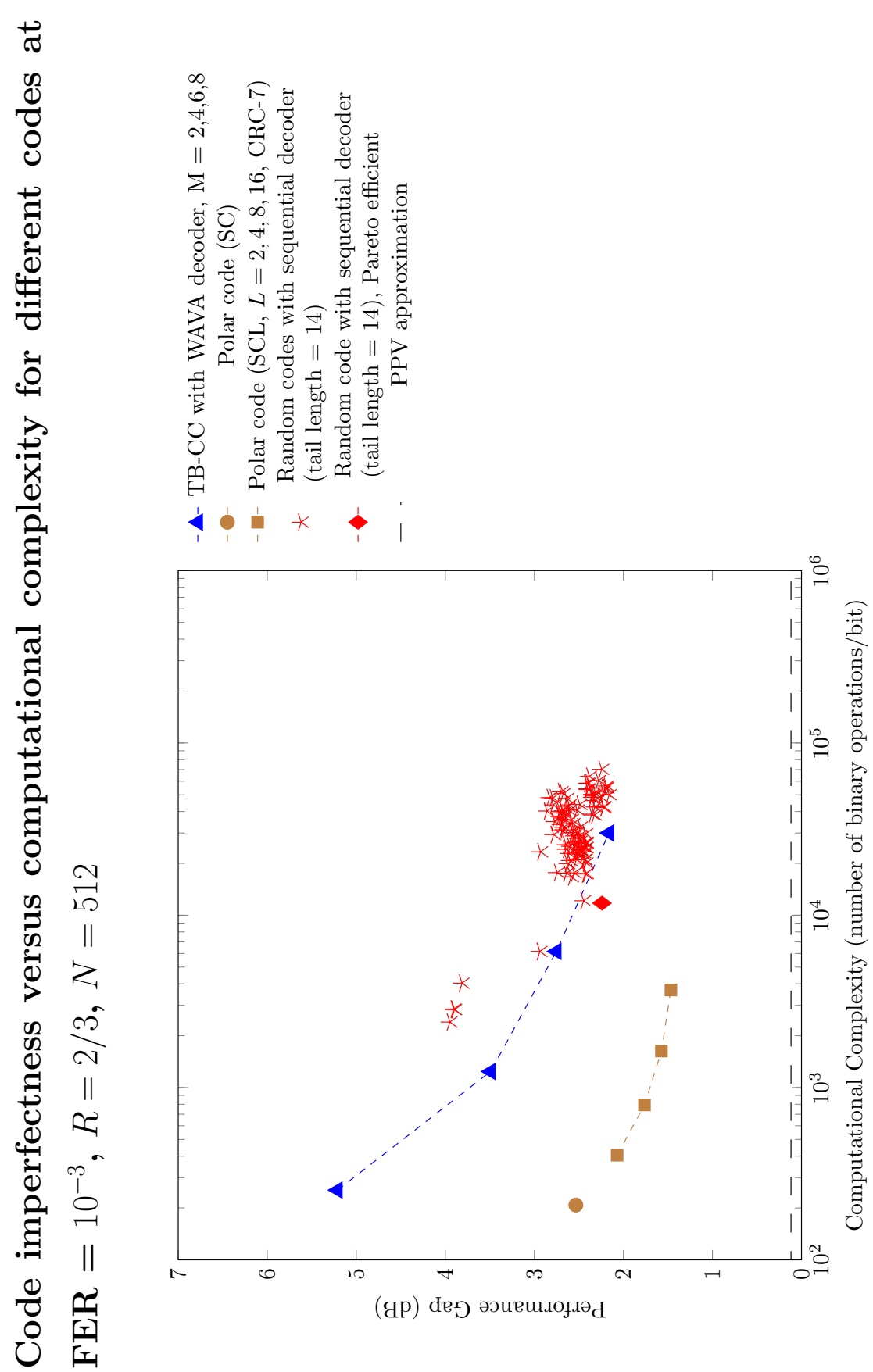

Figure C.188: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 2/3$, $N = 512$

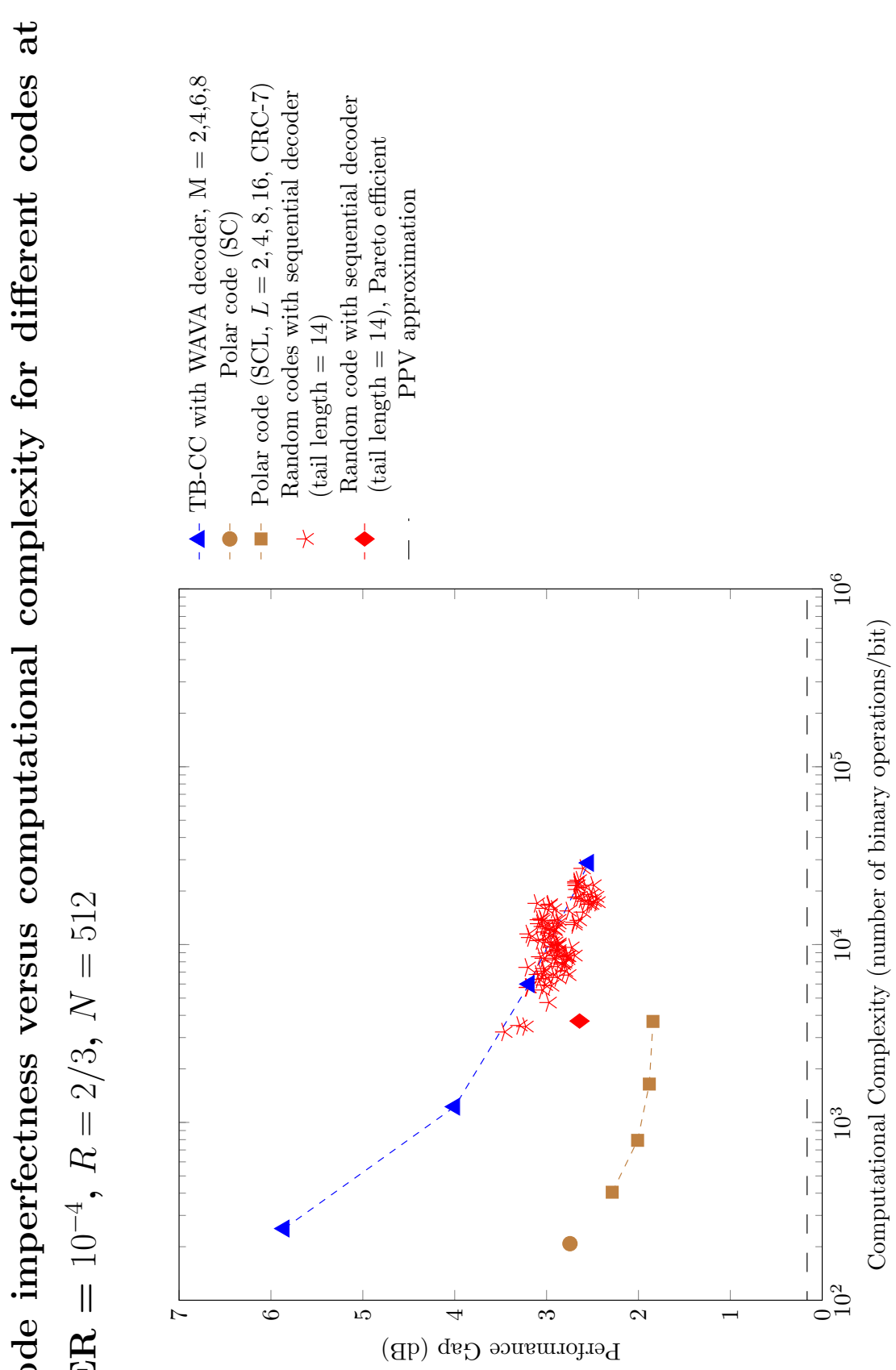

Figure C.189: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 2/3$, $N = 512$

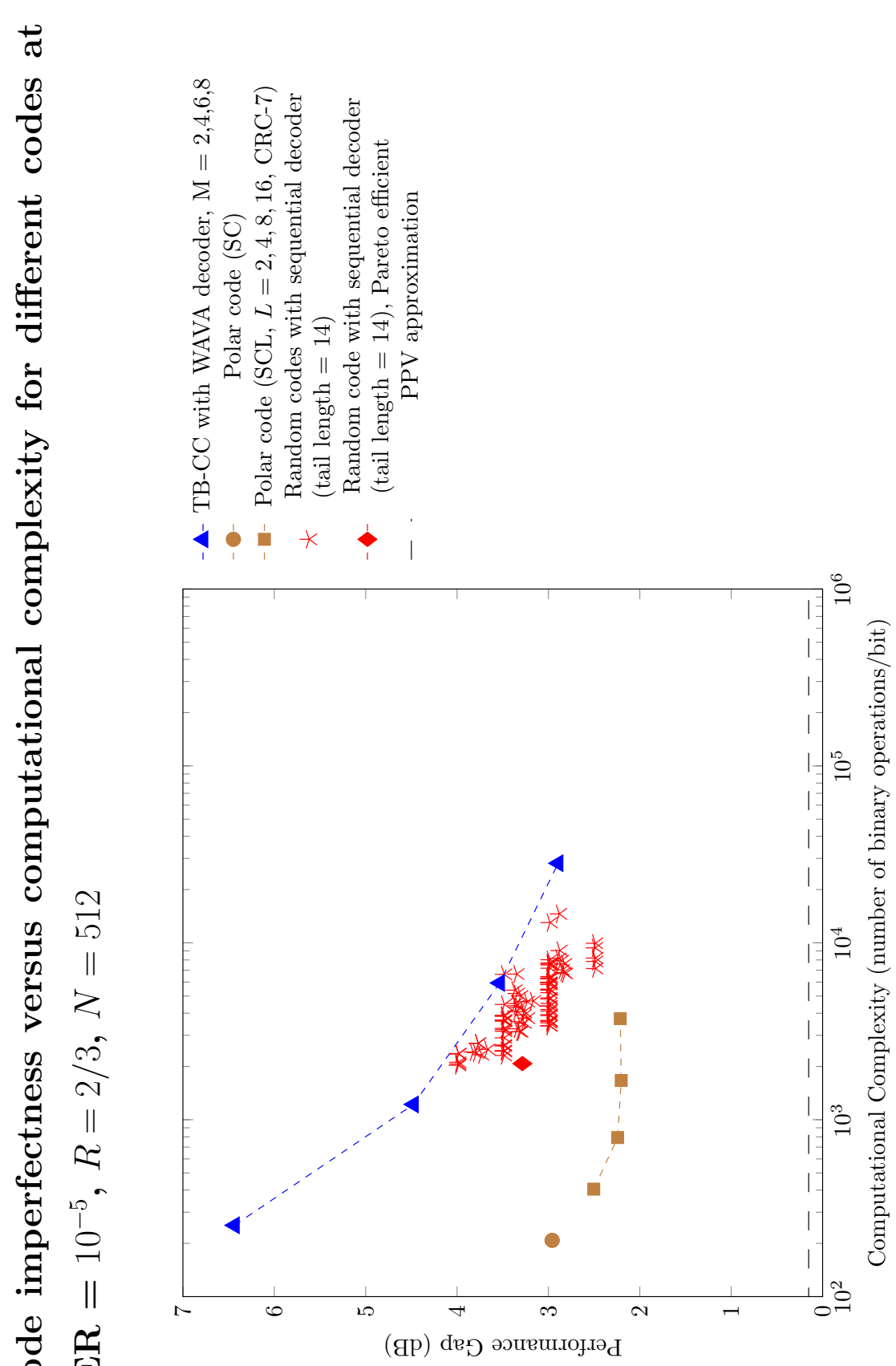

Figure C.190: Code imperfectness versus computational complexity at FER = 10^{-5} for different codes with $R = 2/3$, $N = 512$

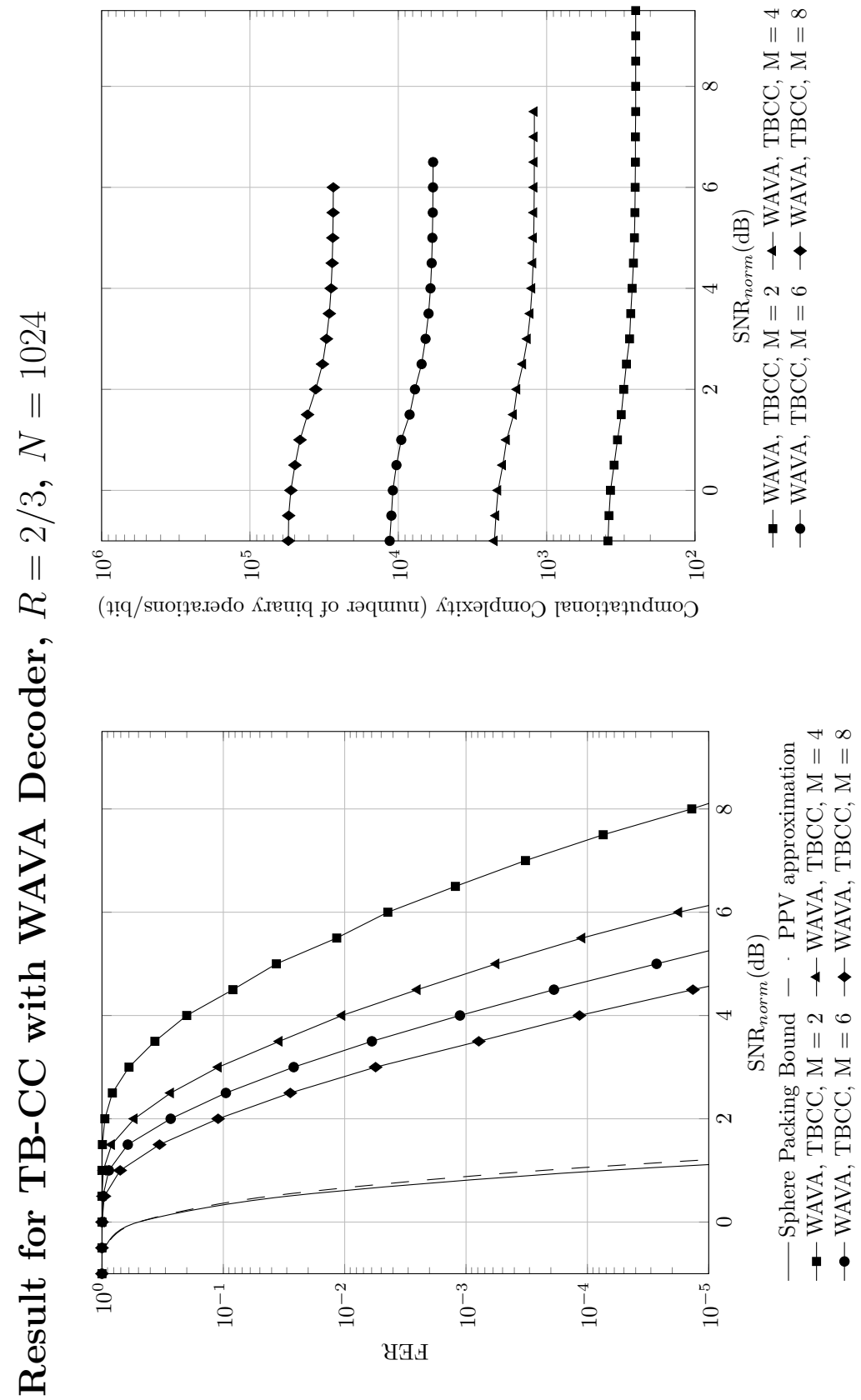

Figure C.191: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 2/3$, $N = 1024$, maximum number of iterations = 4.

Figure C.192: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 2/3$, $N = 1024$, maximum number of iterations = 4

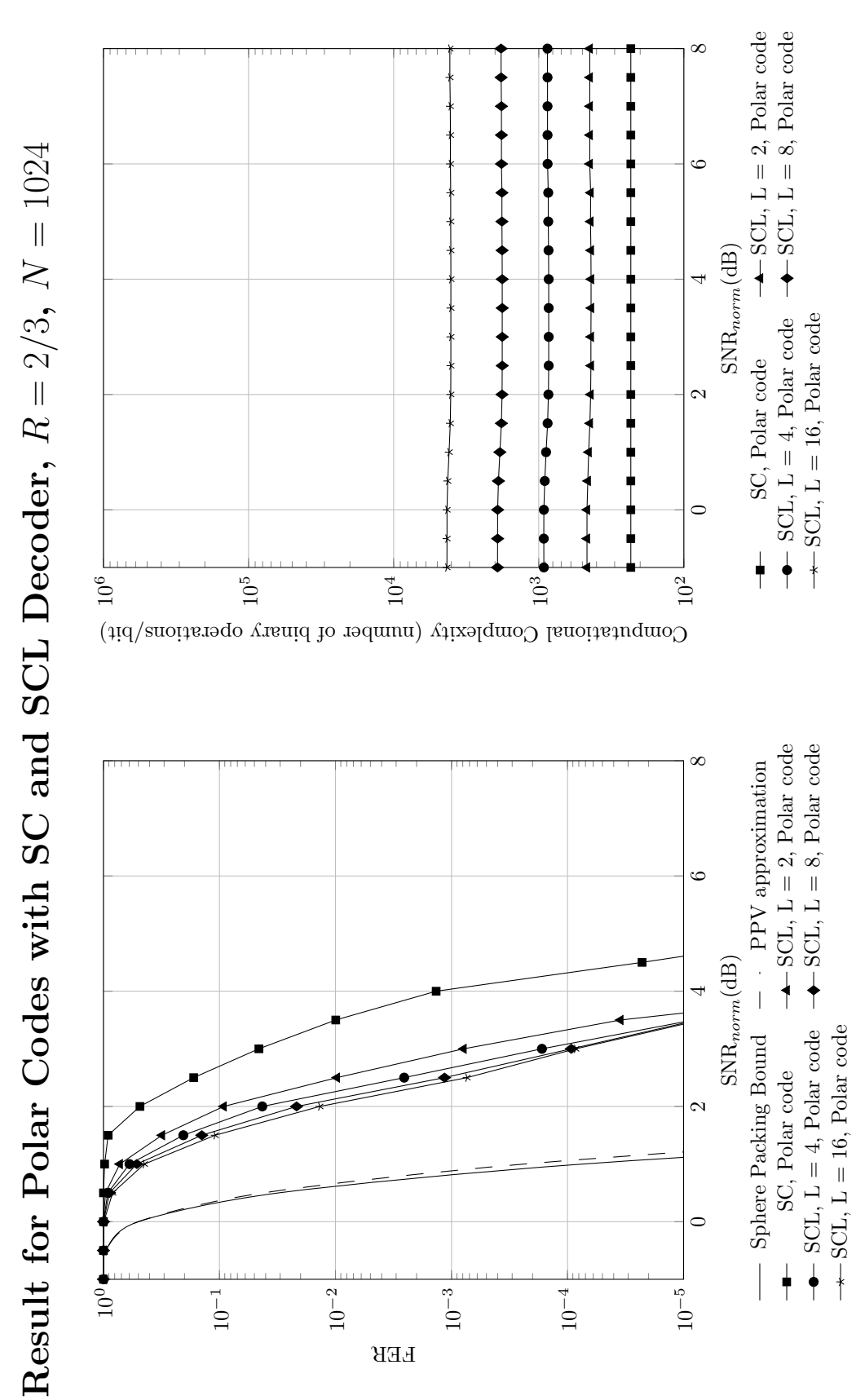

Figure C.193: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 2/3$, $N = 1024$

Figure C.194: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 2/3$, $N = 1024$

Result for TB-CC with Sequential Decoder, $R = 2/3$, $N = 1024$

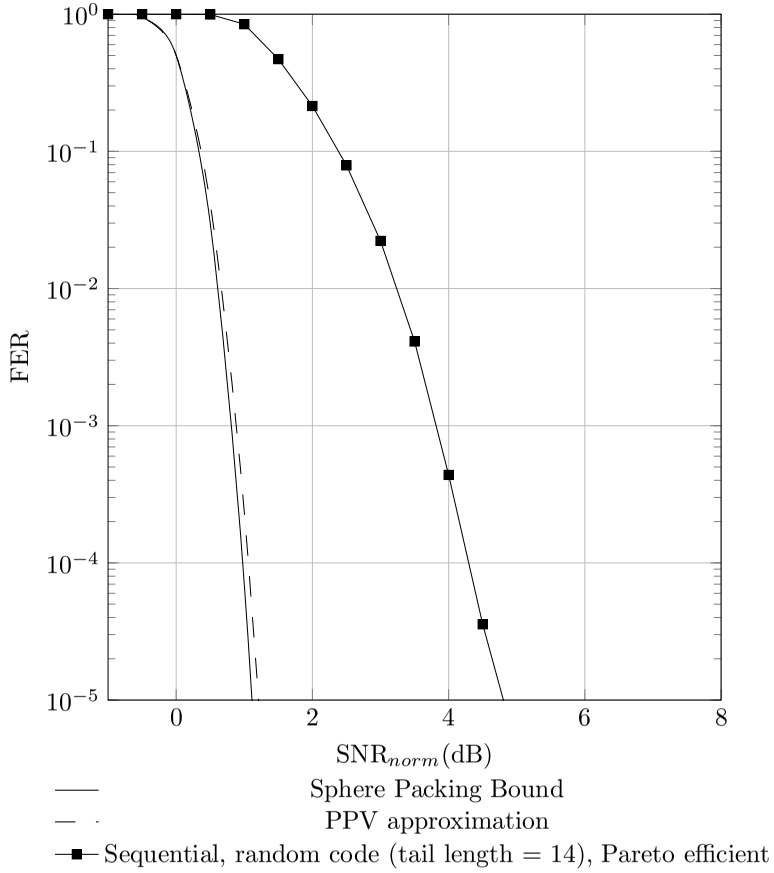

Figure C.195: Frame error rate versus normalized SNR for random code with sequential decoder, $R = 2/3$, $N = 1024$

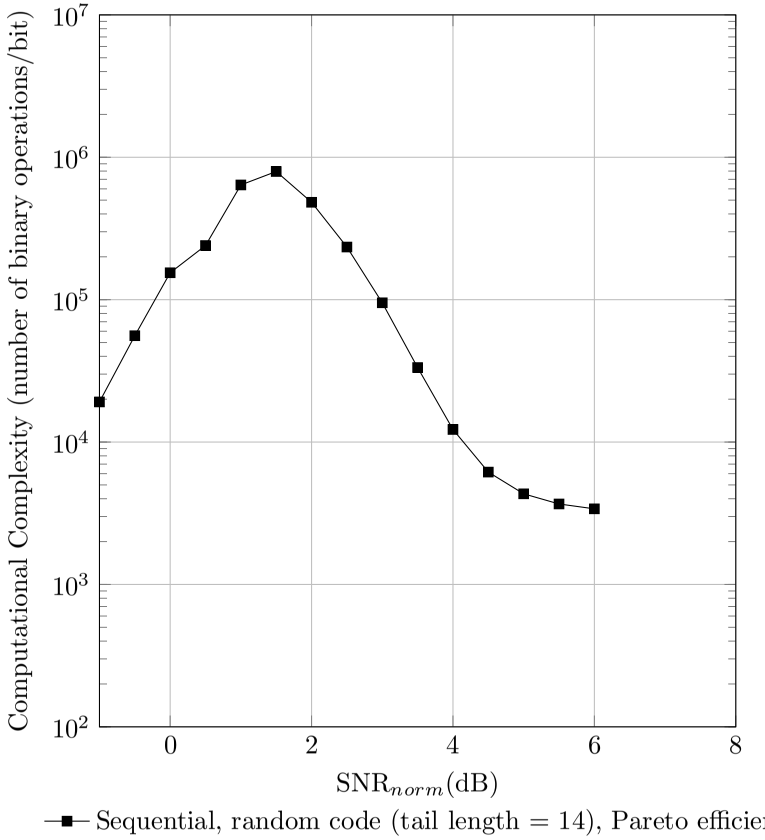

Figure C.196: Computational complexity versus normalized SNR for random code with sequential decoder, $R = 2/3$, $N = 1024$

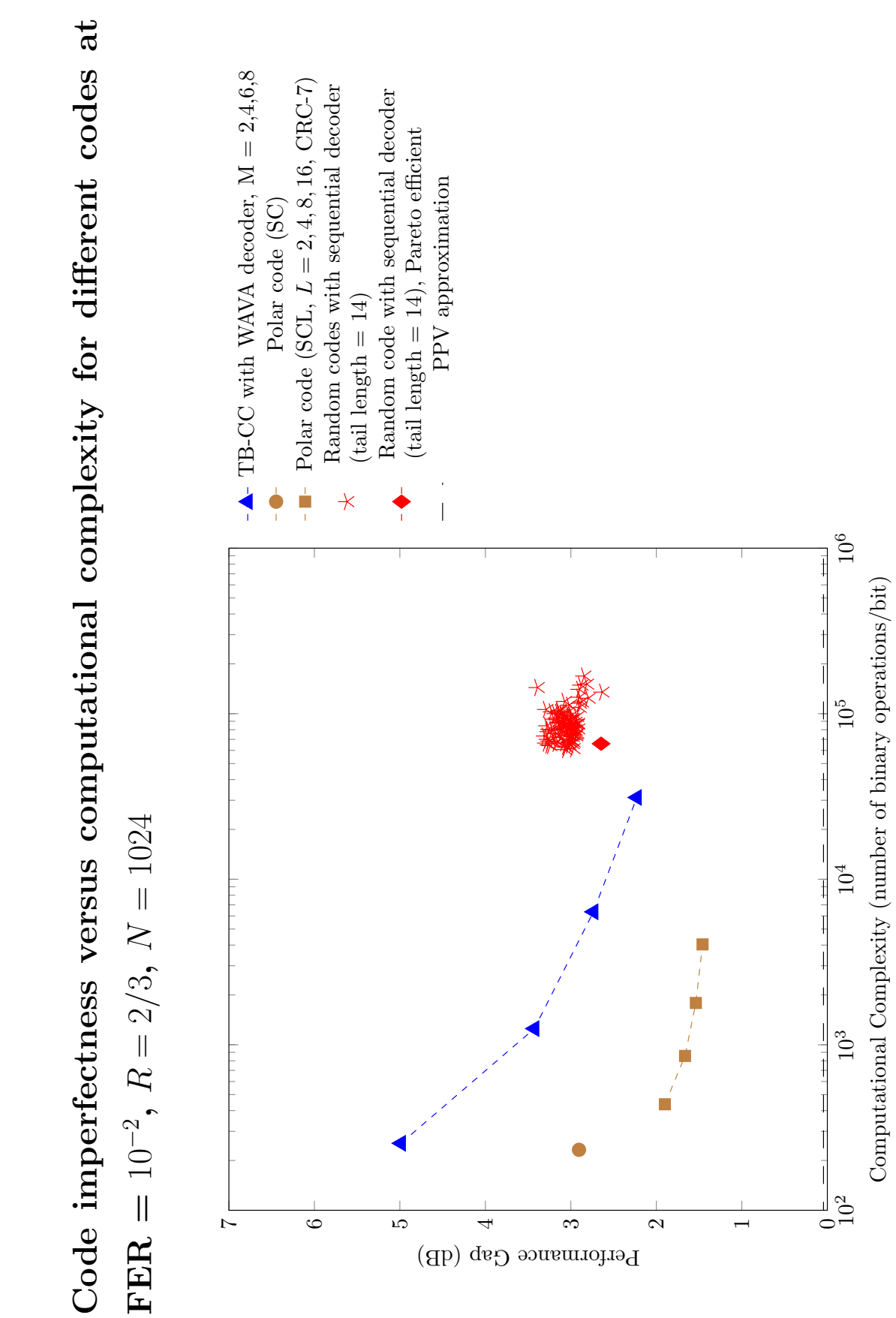

Figure C.197: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 2/3$, $N = 1024$

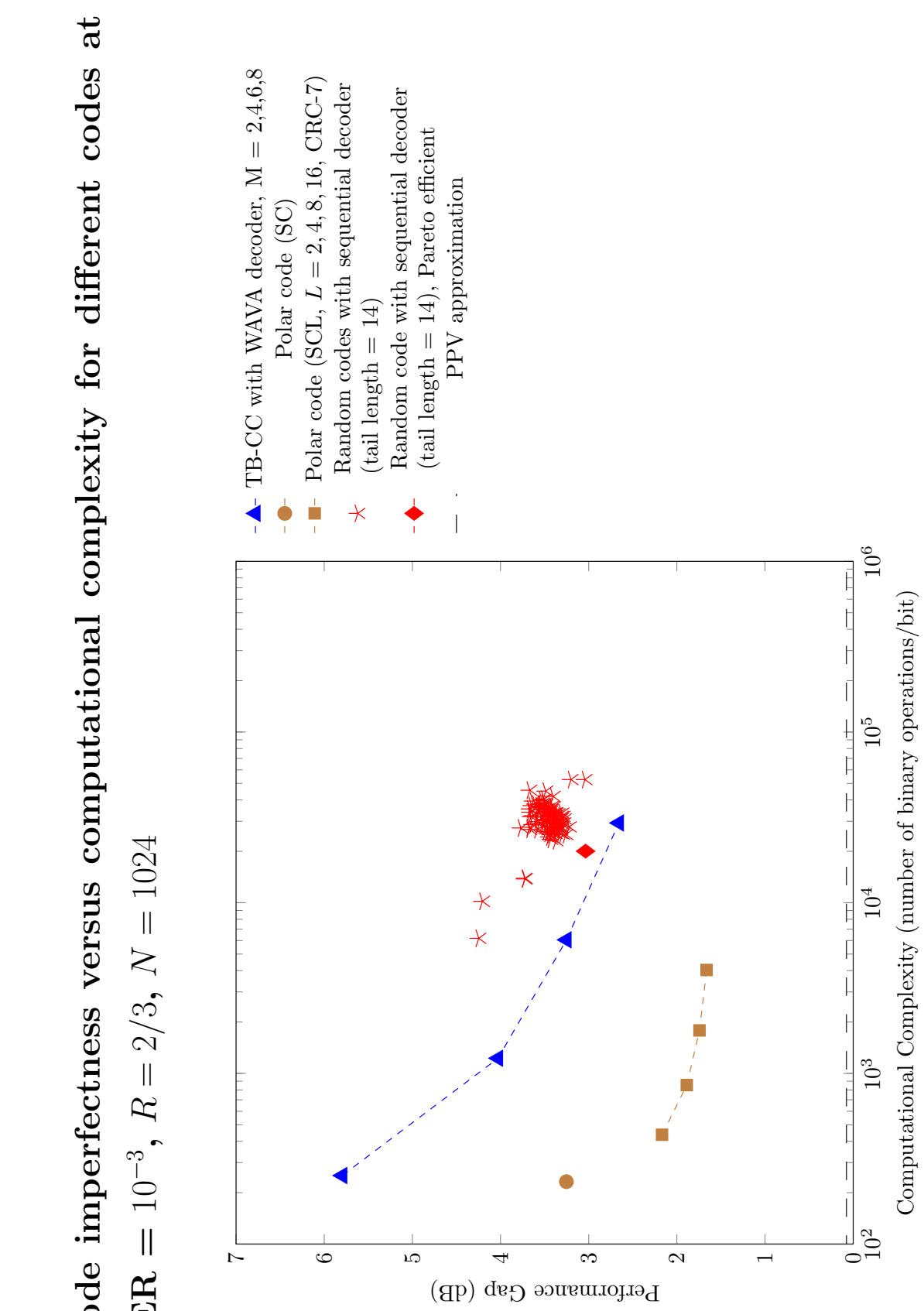

Figure C.198: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 2/3$, $N = 1024$

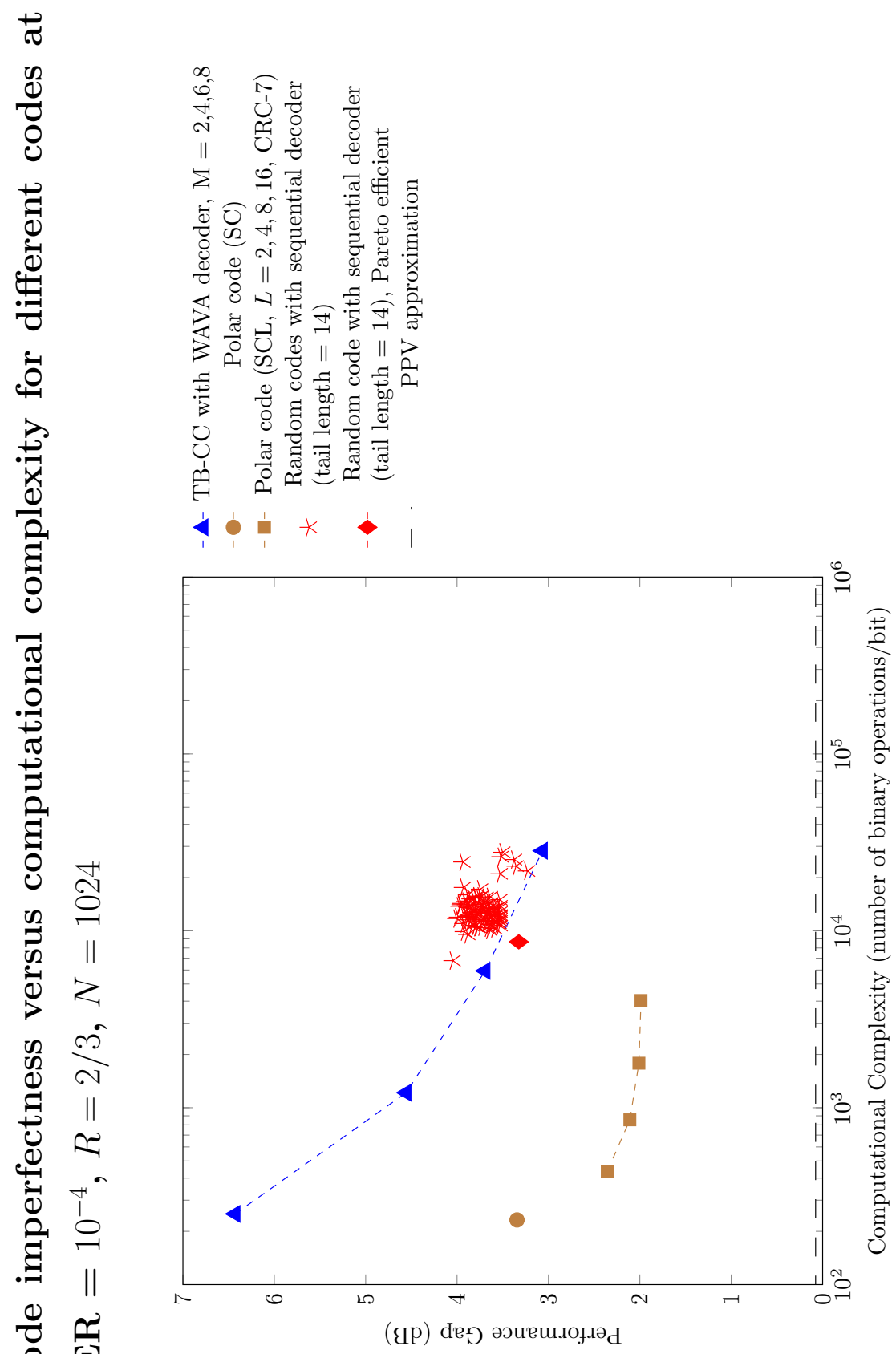

Figure C.199: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 2/3$, $N = 1024$

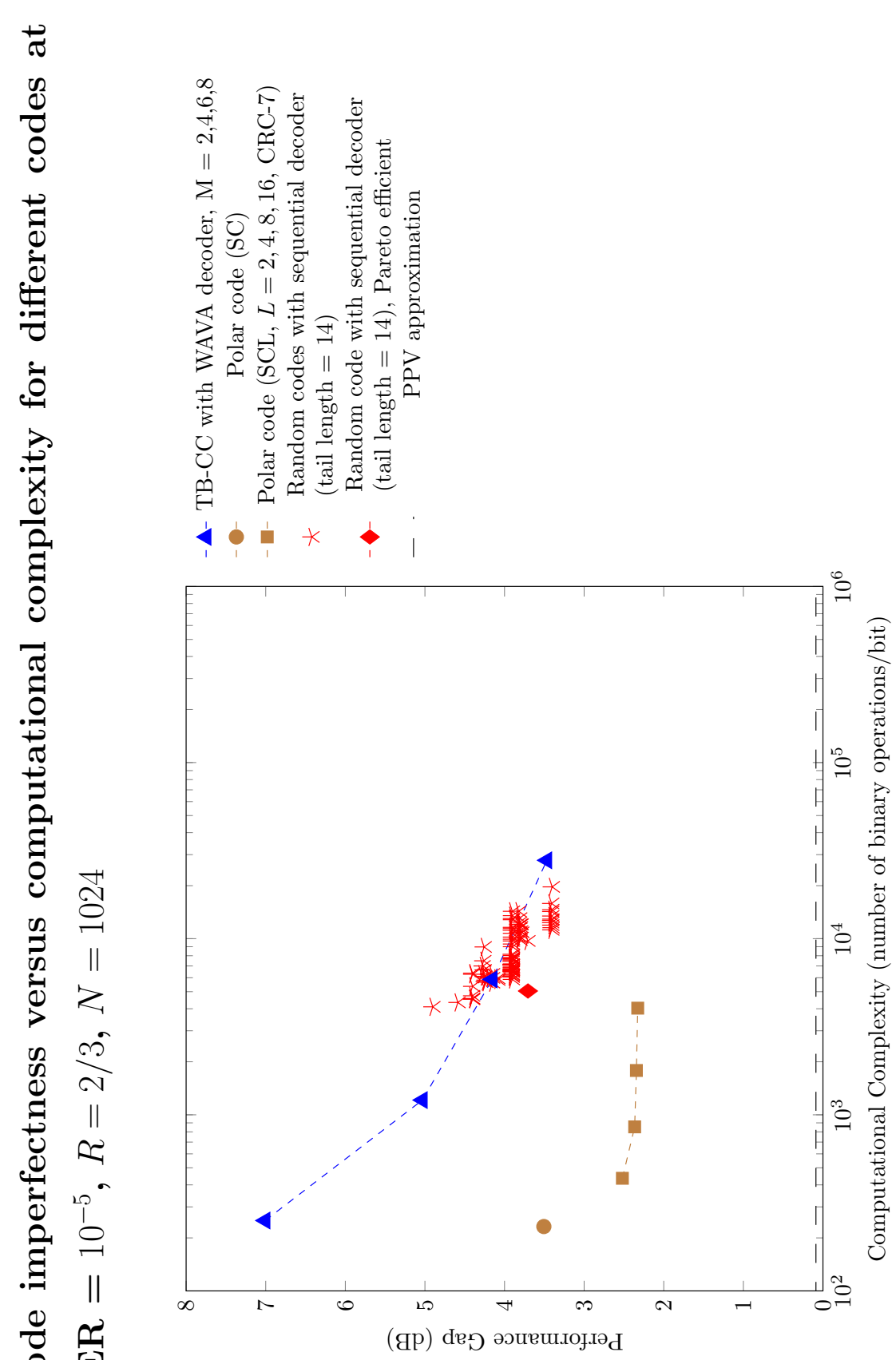

Figure C.200: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 2/3$, $N = 1024$

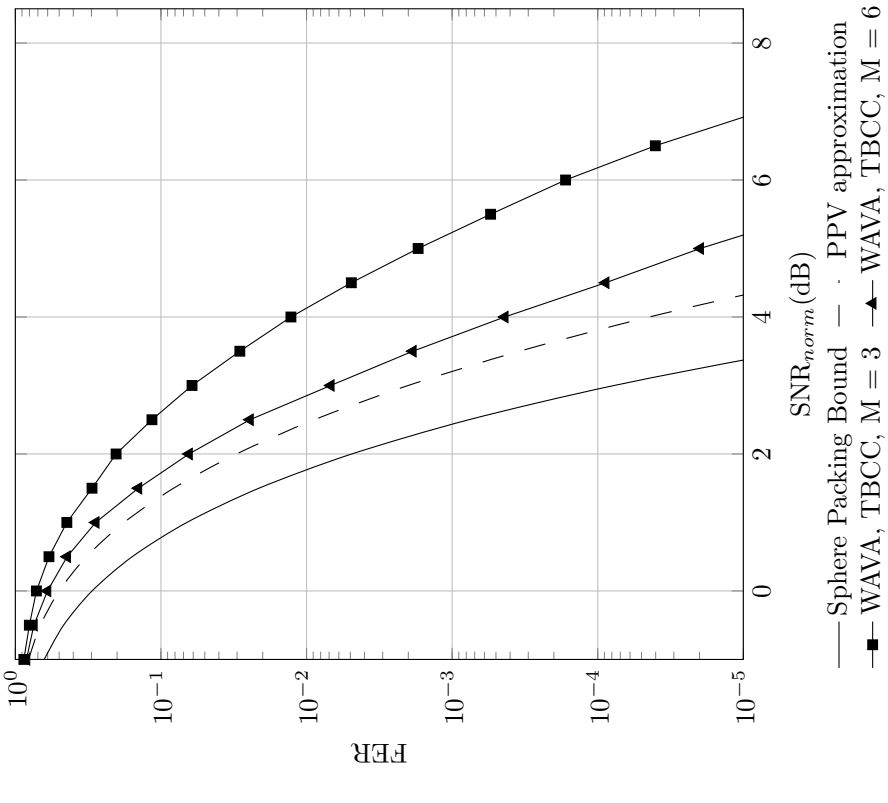

Figure C.201: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 3/4$, $N = 64$, maximum number of iterations = 4.

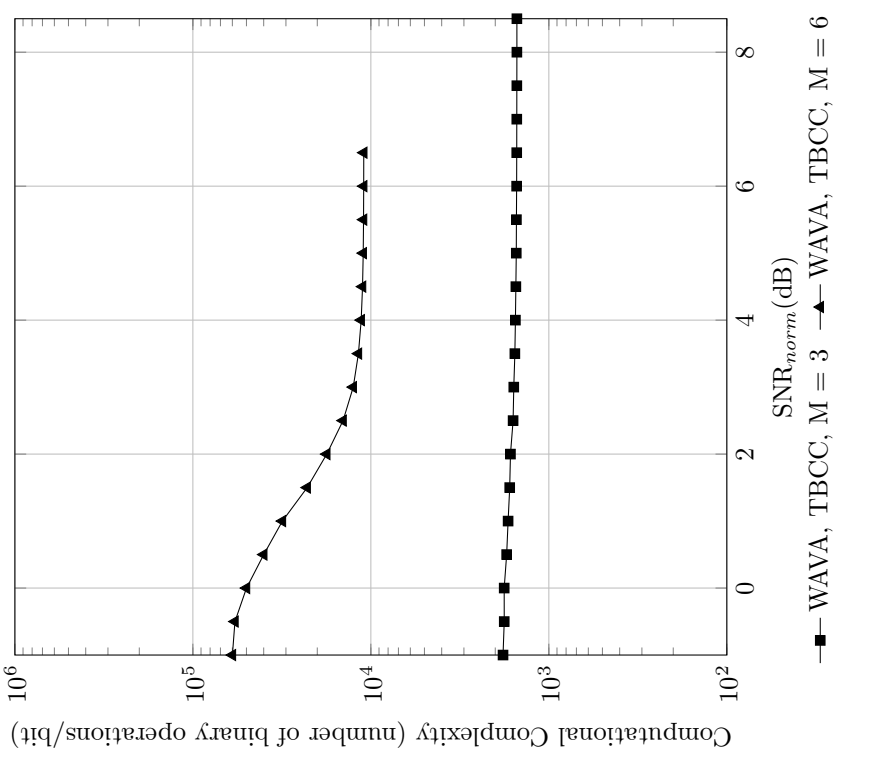

Figure C.202: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 3/4$, $N = 64$, maximum number of iterations = 4

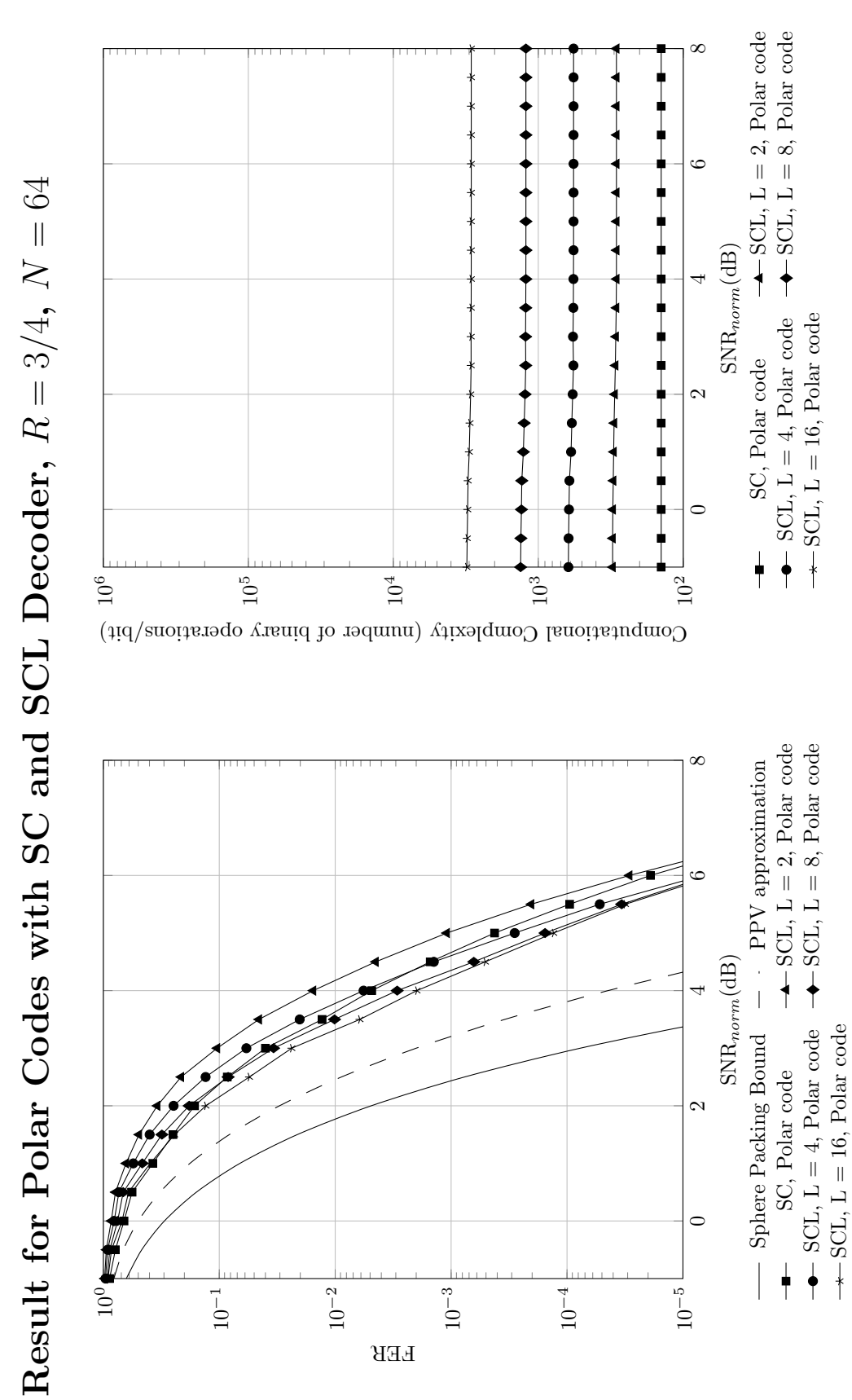

Figure C.203: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 3/4$, $N = 64$

Figure C.204: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 3/4$, $N = 64$

Result for TB-CC with Sequential Decoder, $R = 3/4$, $N = 64$

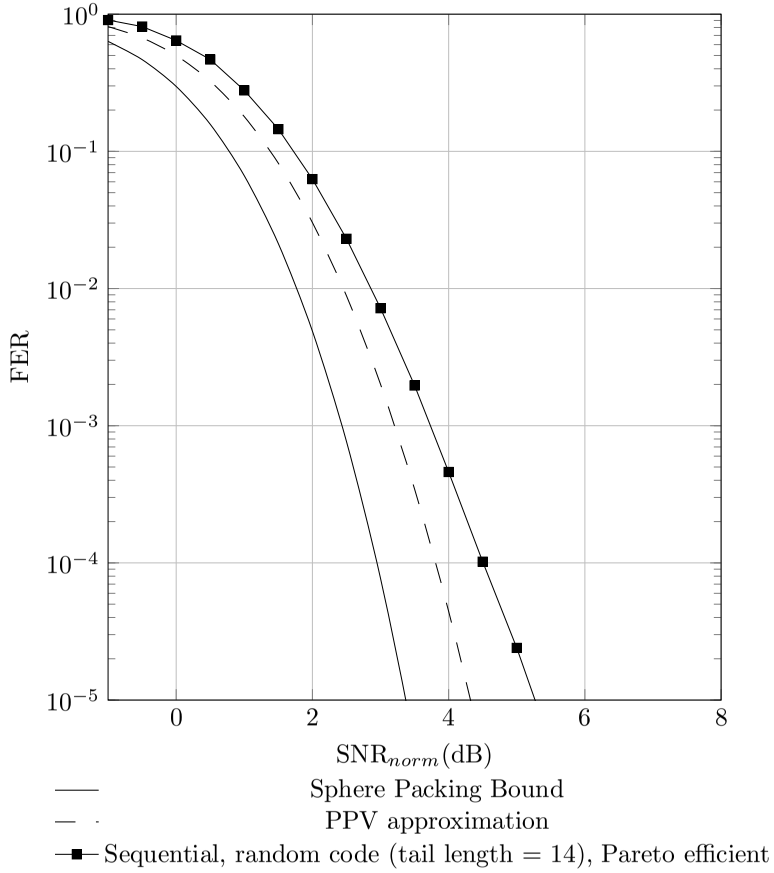

Figure C.205: Frame error rate versus normalized SNR for random code with sequential decoder, $R = 3/4$, $N = 64$

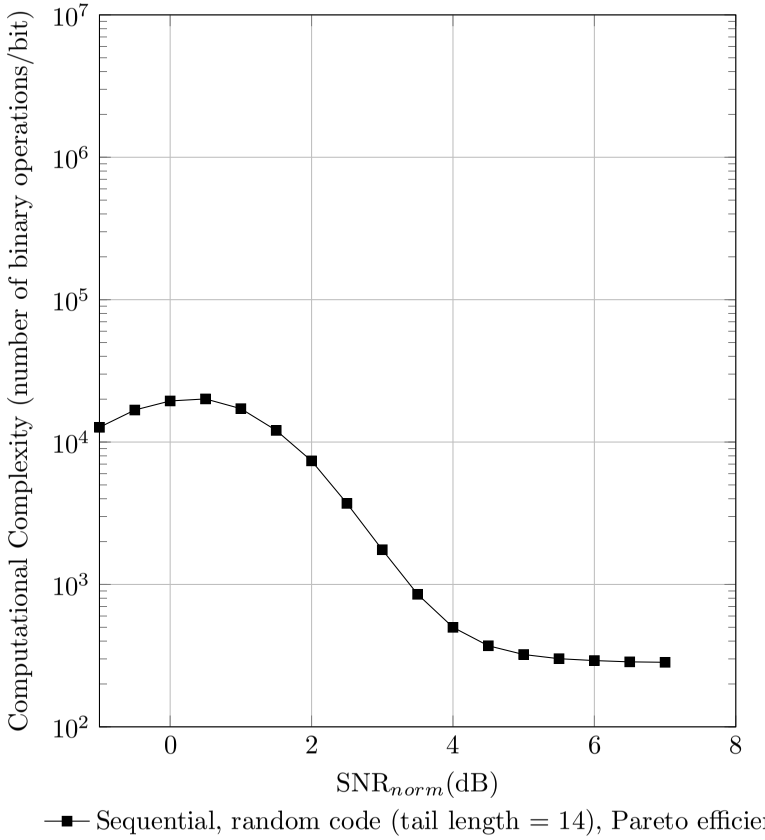

Figure C.206: Computational complexity versus normalized SNR for random code with sequential decoder, $R = 3/4$, $N = 64$

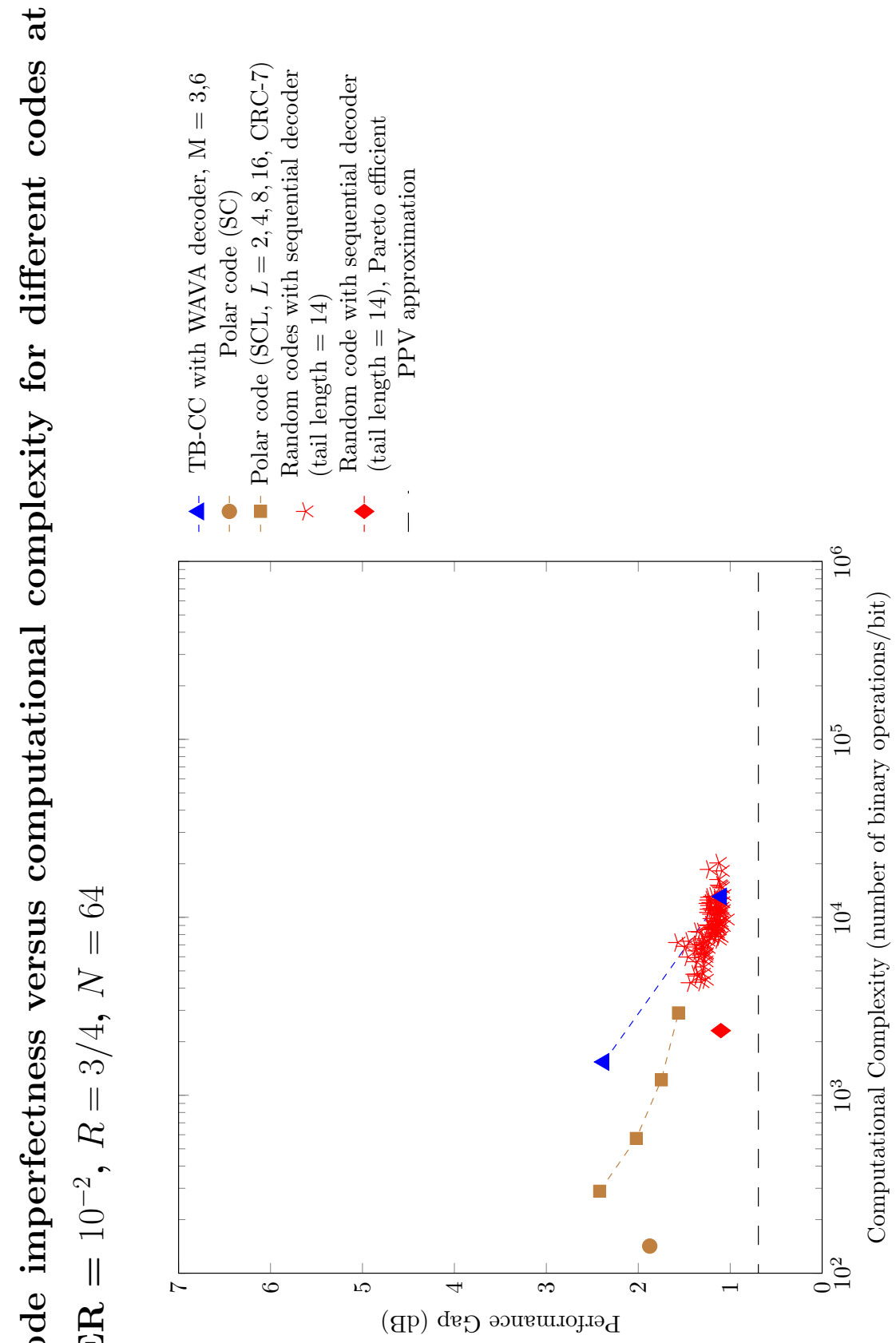

Figure C.207: Code imperfectness versus computational complexity at FER = 10^{-2} for different codes with $R = 3/4$, $N = 64$

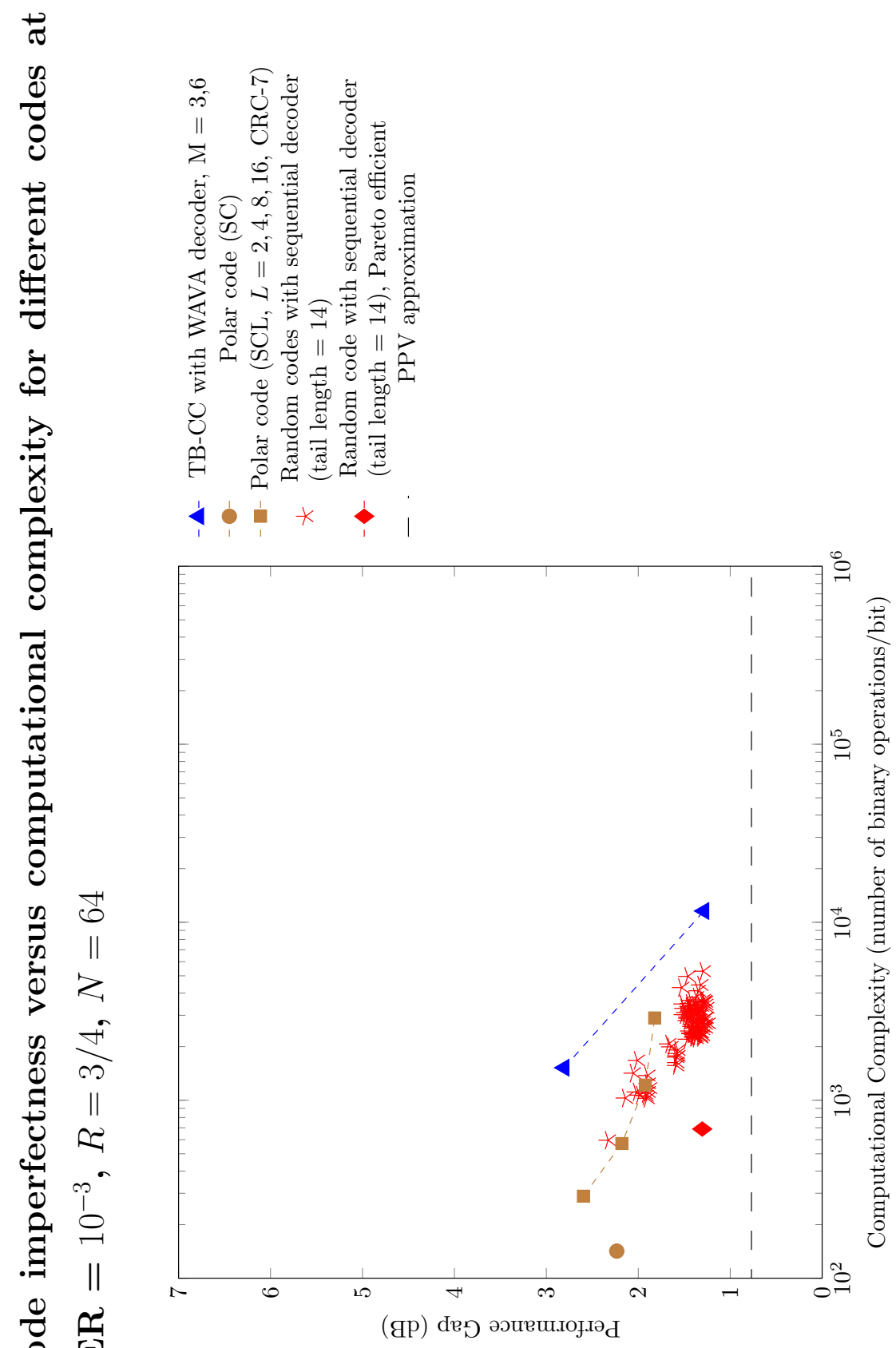

Figure C.208: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 3/4$, $N = 64$

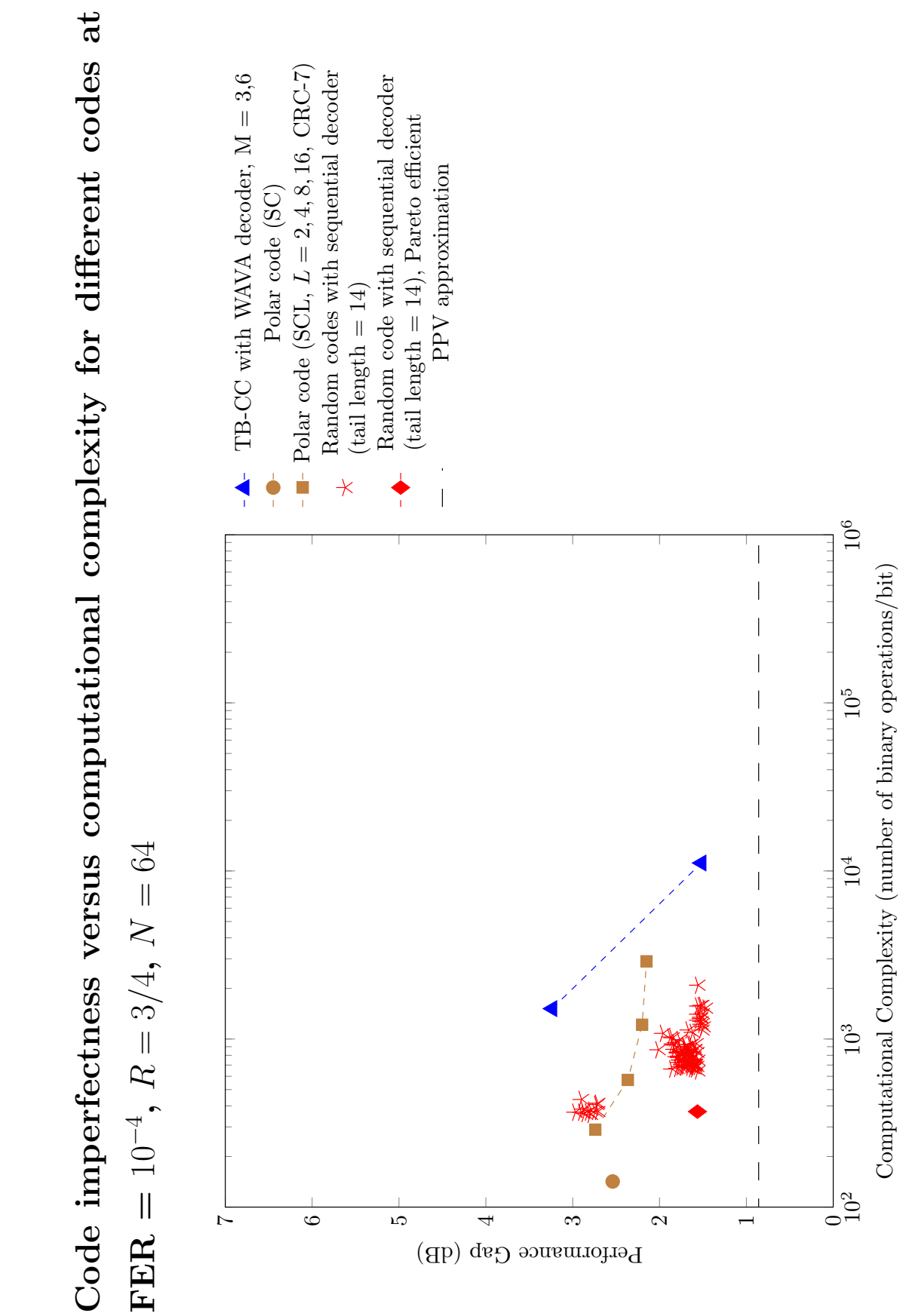

Figure C.209: Code imperfectness versus computational complexity at FER = 10^{-4} for different codes with $R = 3/4$, $N = 64$

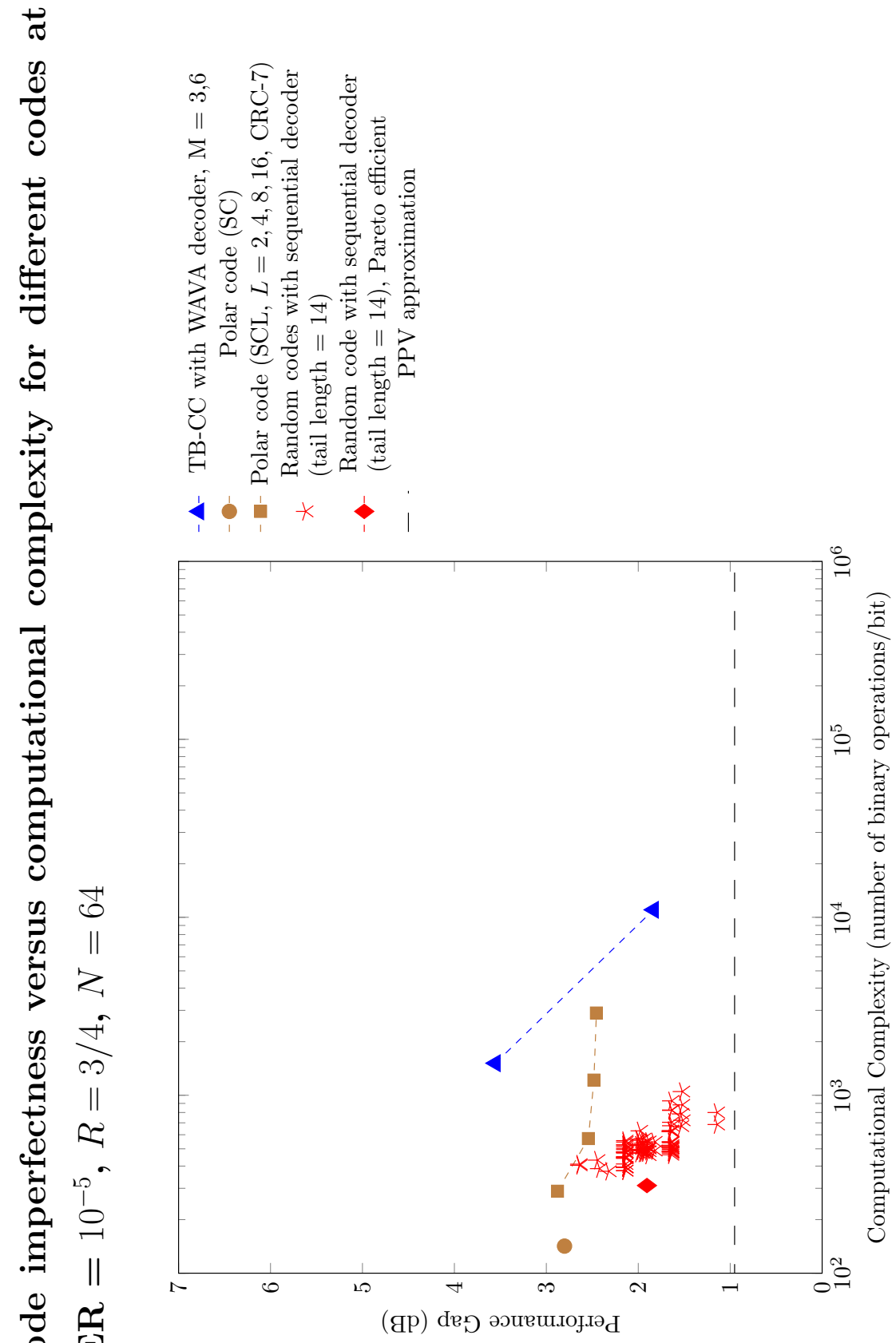

Figure C.210: Code imperfectness versus computational complexity at FER = 10^{-5} for different codes with $R = 3/4$, $N = 64$

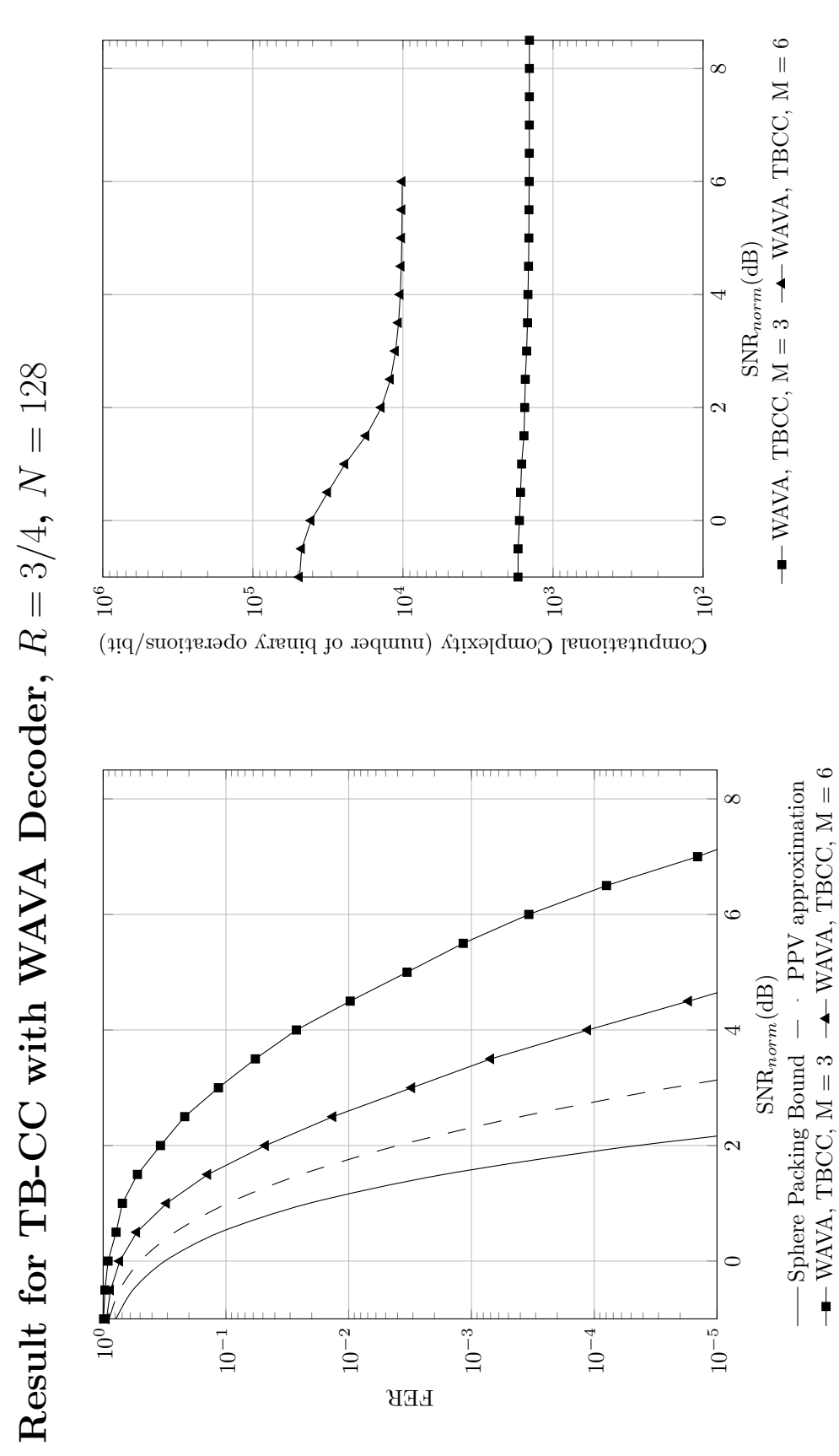

Figure C.211: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 3/4$, $N = 128$, maximum number of iterations = 4.

Figure C.212: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 3/4$, $N = 128$, maximum number of iterations = 4

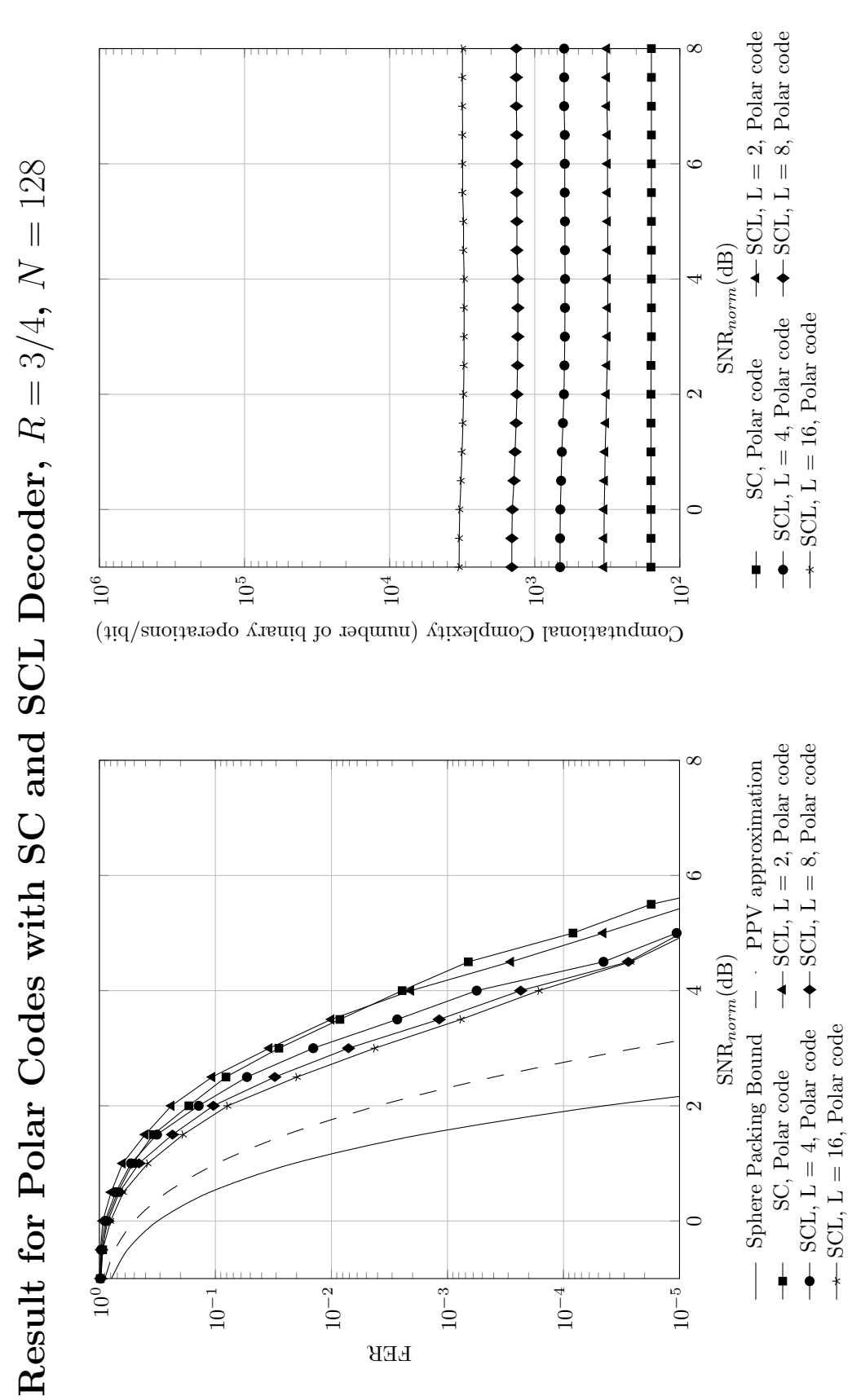

Figure C.213: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 3/4$, $N = 128$

Figure C.214: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 3/4$, $N = 128$

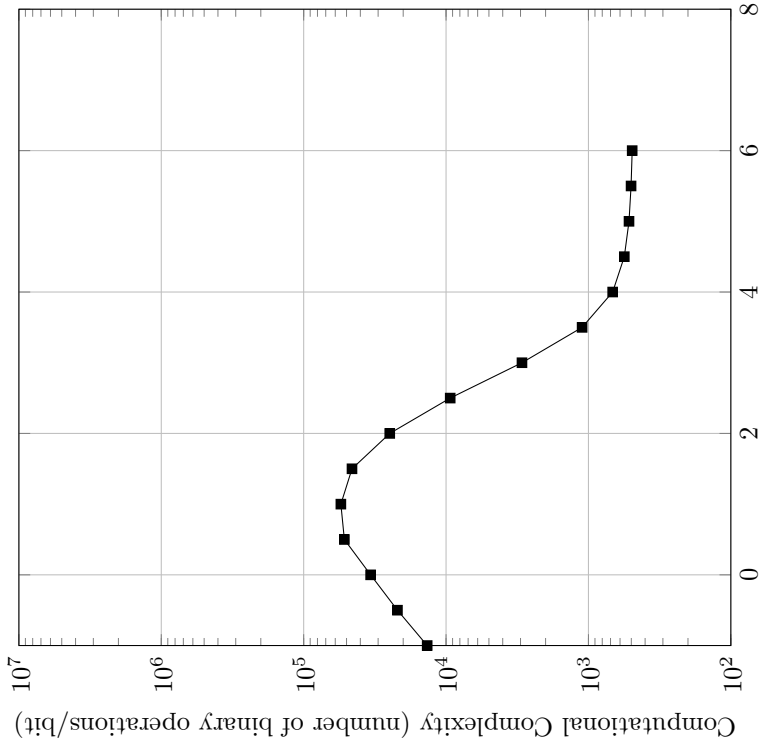

Figure C.216: Computational complexity versus normalized SNR for random code with sequential decoder, $R = 3/4$, $N = 128$

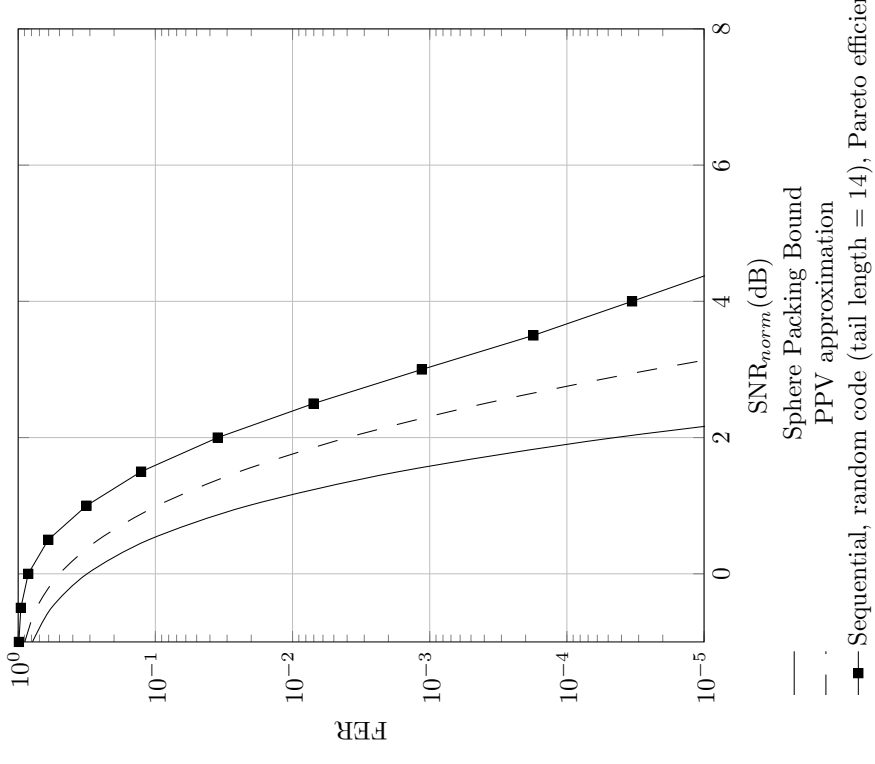

Figure C.215: Frame error rate versus normalized SNR for random code with sequential decoder, $R = 3/4$, $N = 128$

Result for TB-CC with Sequential Decoder, $R = 3/4$, $N = 128$

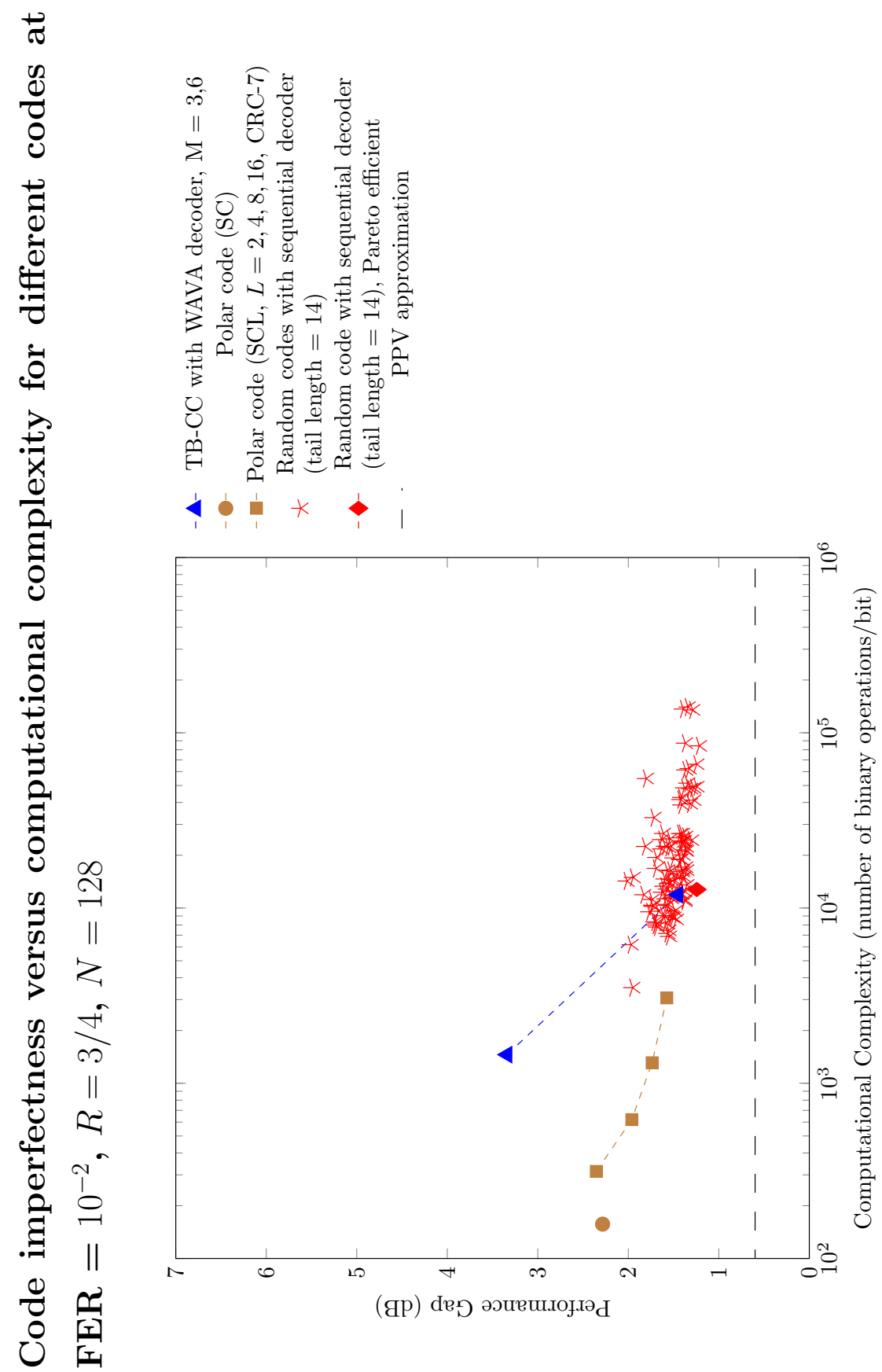

Figure C.217: Code imperfectness versus computational complexity at FER = 10^{-2} for different codes with $R = 3/4$, $N = 128$

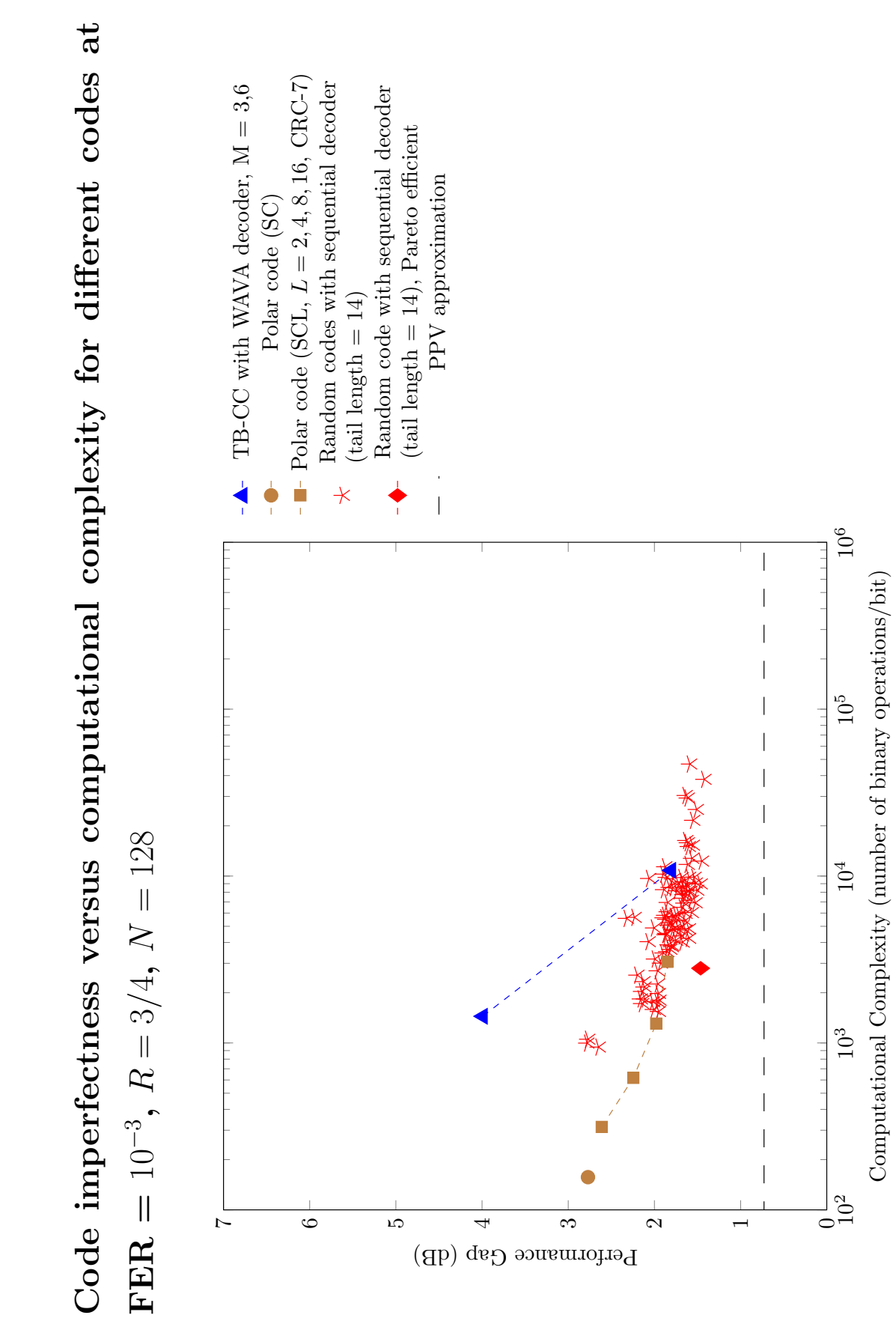

Figure C.218: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 3/4$, $N = 128$

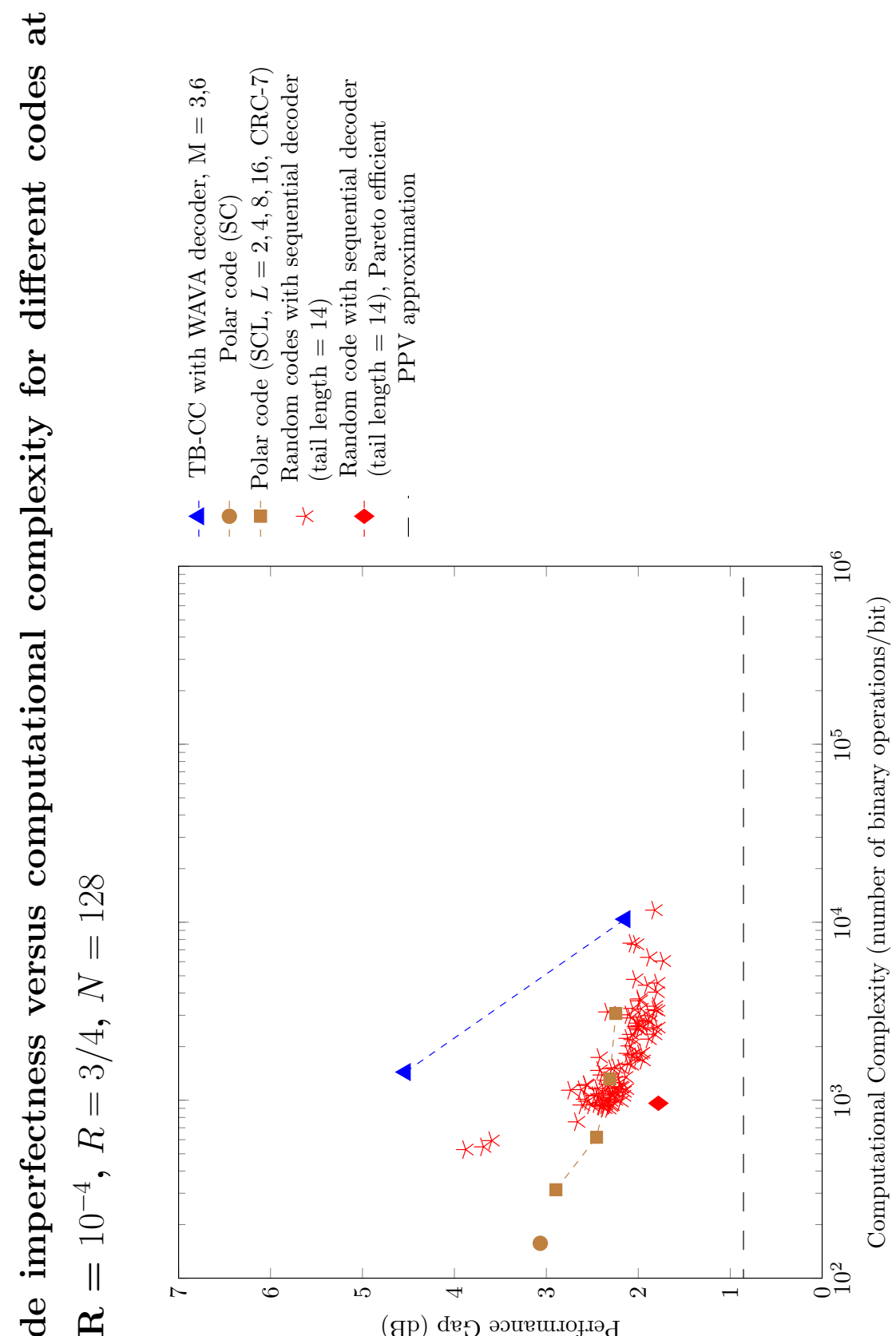

Figure C.219: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 3/4$, $N = 128$

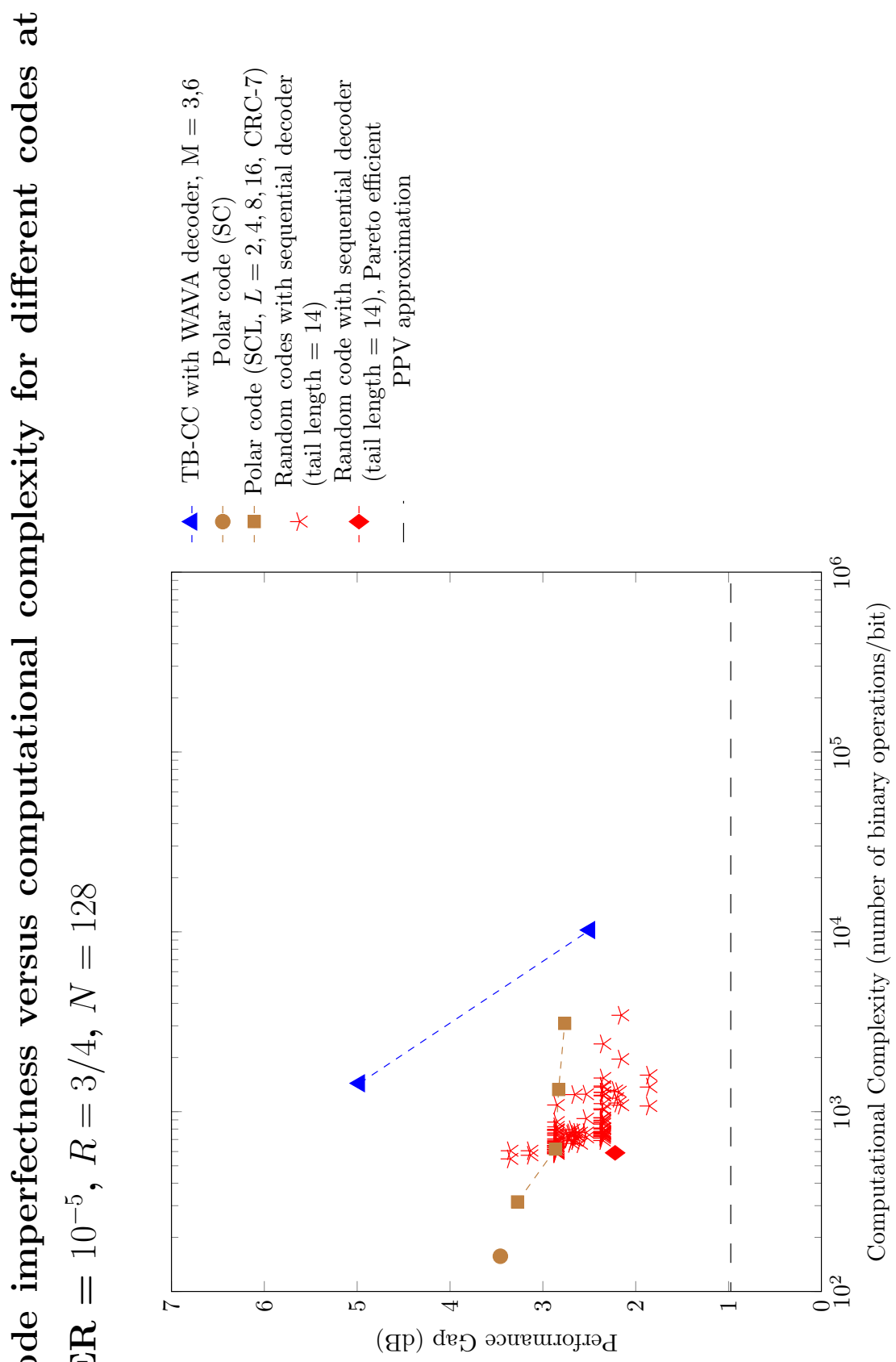

Figure C.220: Code imperfectness versus computational complexity at FER = 10^{-5} for different codes with $R = 3/4$, $N = 128$

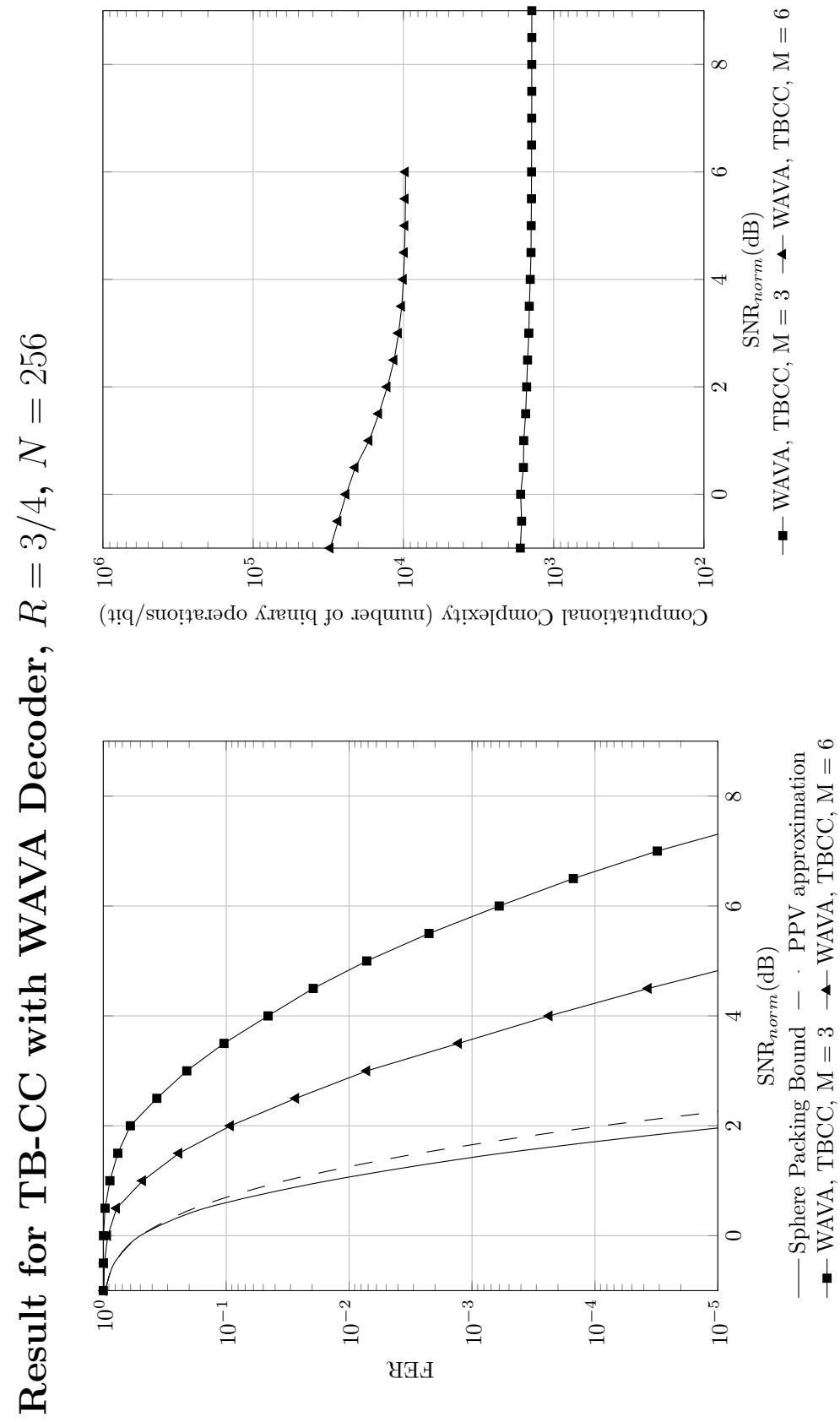

Figure C.221: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 3/4$, $N = 256$, maximum number of iterations = 4.

Figure C.222: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 3/4$, $N = 256$, maximum number of iterations = 4

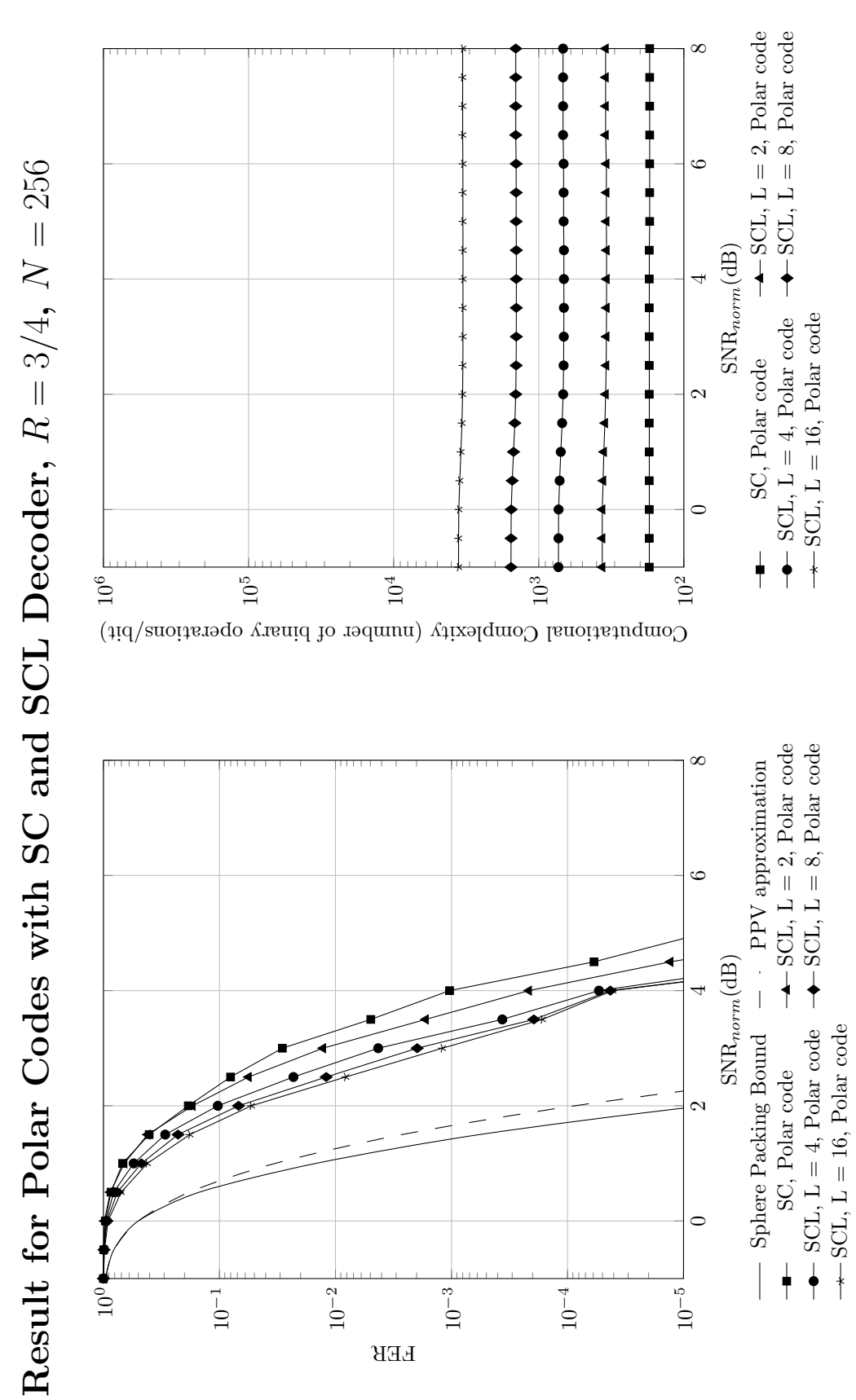

Figure C.223: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 3/4$, $N = 256$

Figure C.224: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 3/4$, $N = 256$

Result for TB-CC with Sequential Decoder, $R = 3/4$, $N = 256$

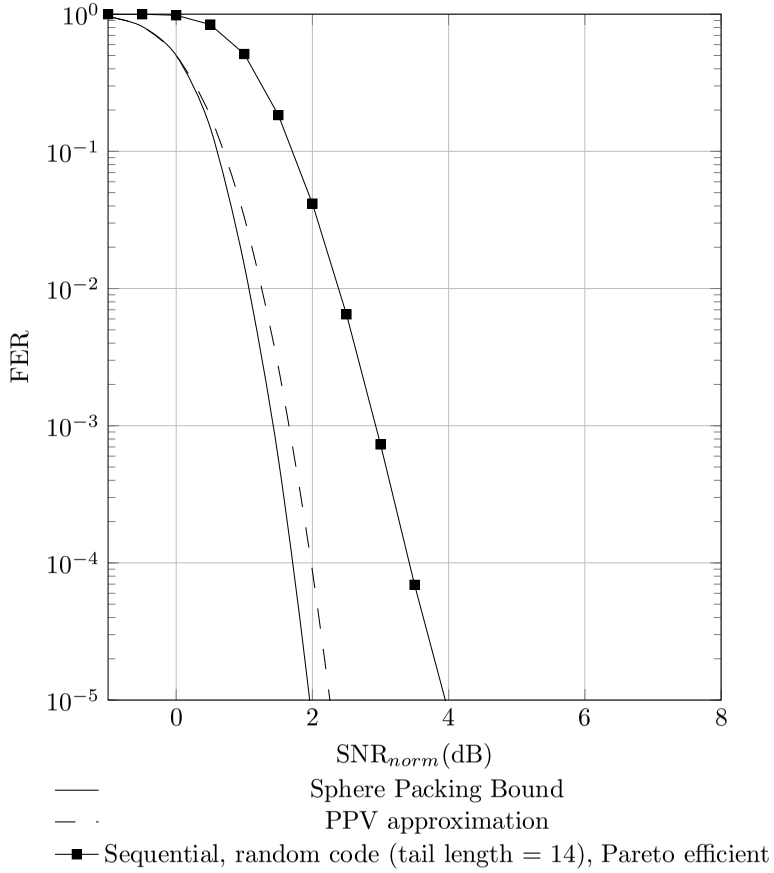

Figure C.225: Frame error rate versus normalized SNR for random code with sequential decoder, $R = 3/4$, $N = 256$

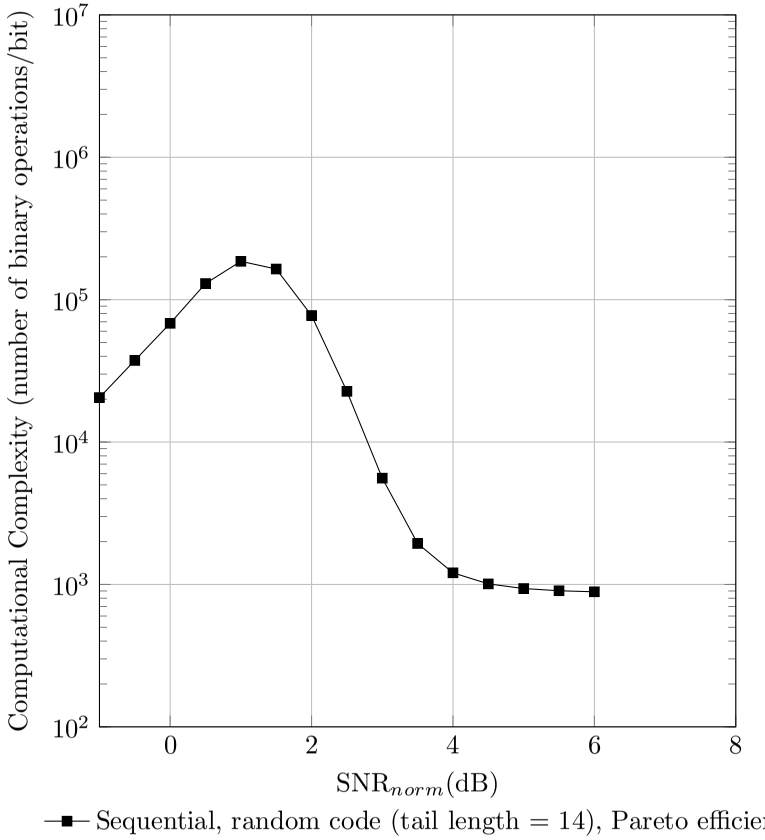

Figure C.226: Computational complexity versus normalized SNR for random code with sequential decoder, $R = 3/4$, $N = 256$

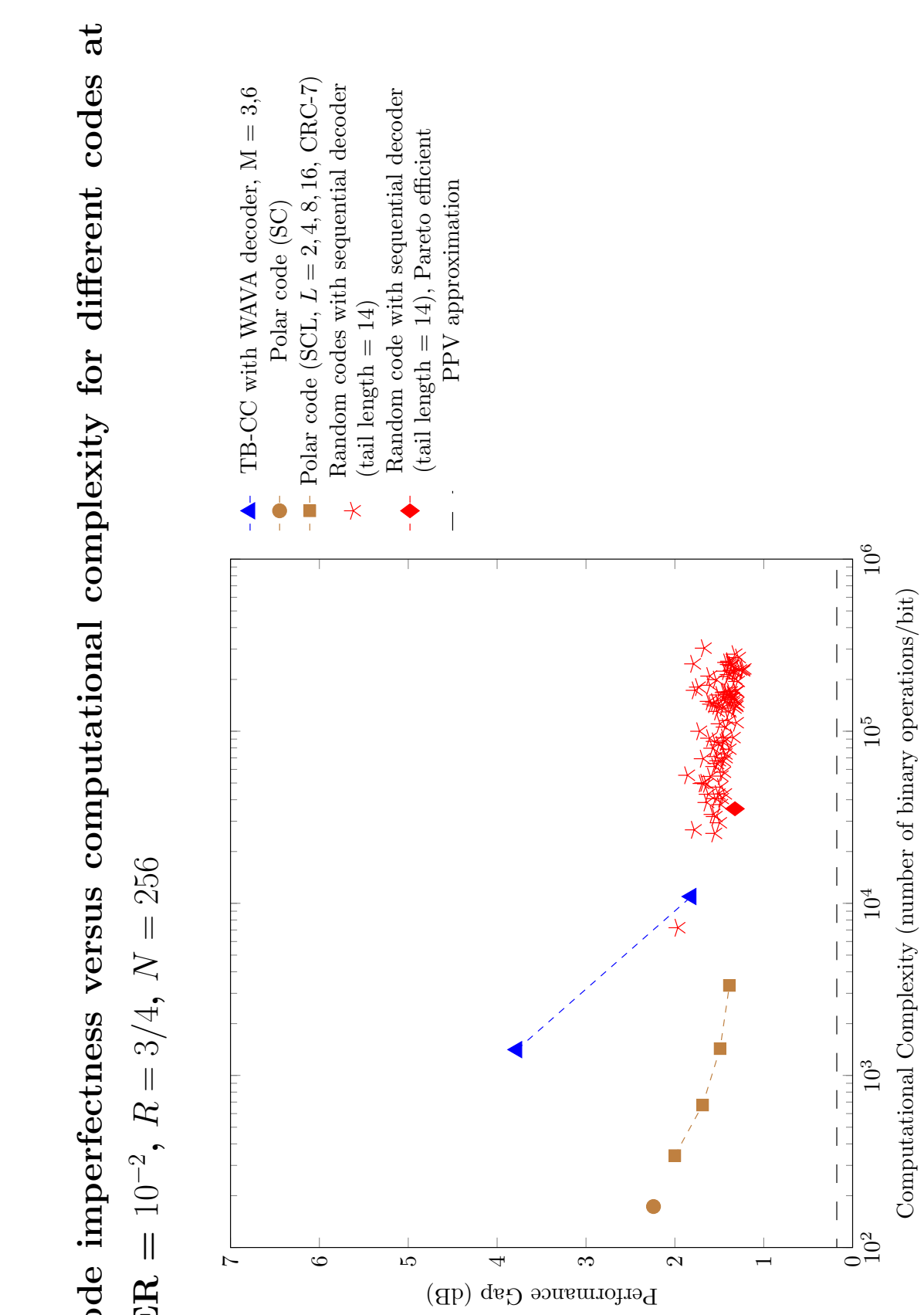

Figure C.227: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 3/4$, $N = 256$

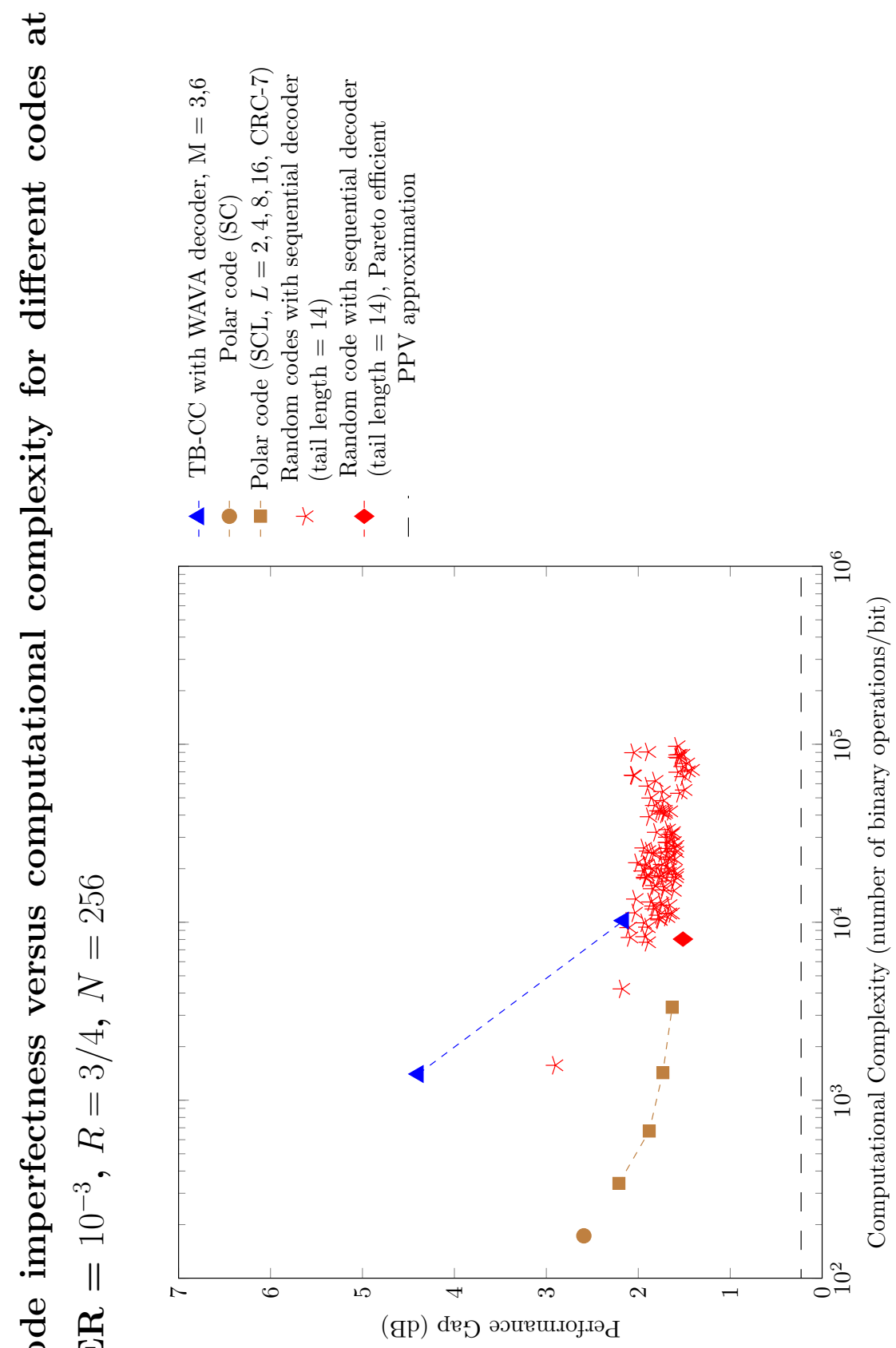

Figure C.228: Code imperfectness versus computational complexity at FER = 10^{-3} for different codes with $R = 3/4$, $N = 256$

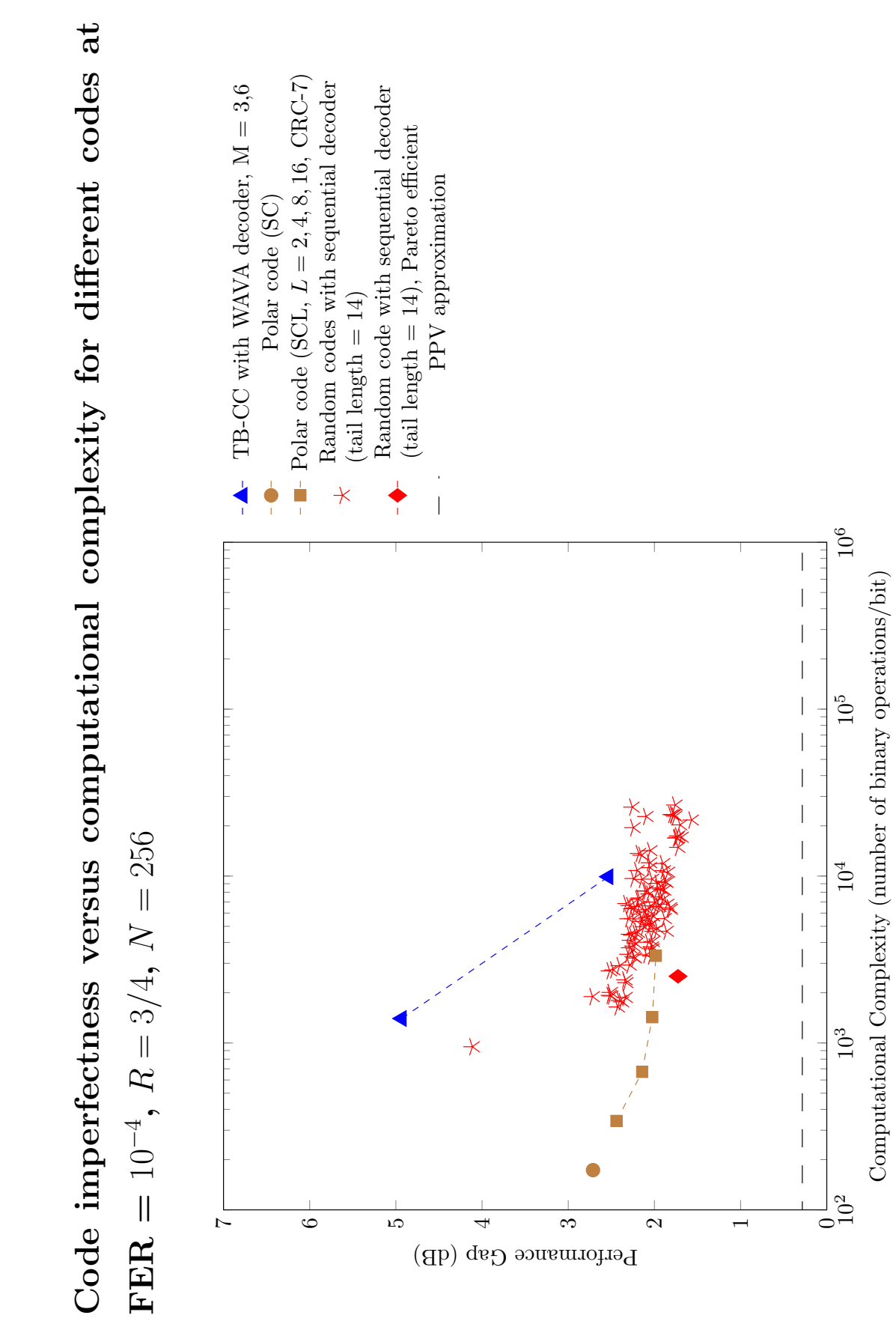

Figure C.229: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 3/4$, $N = 256$

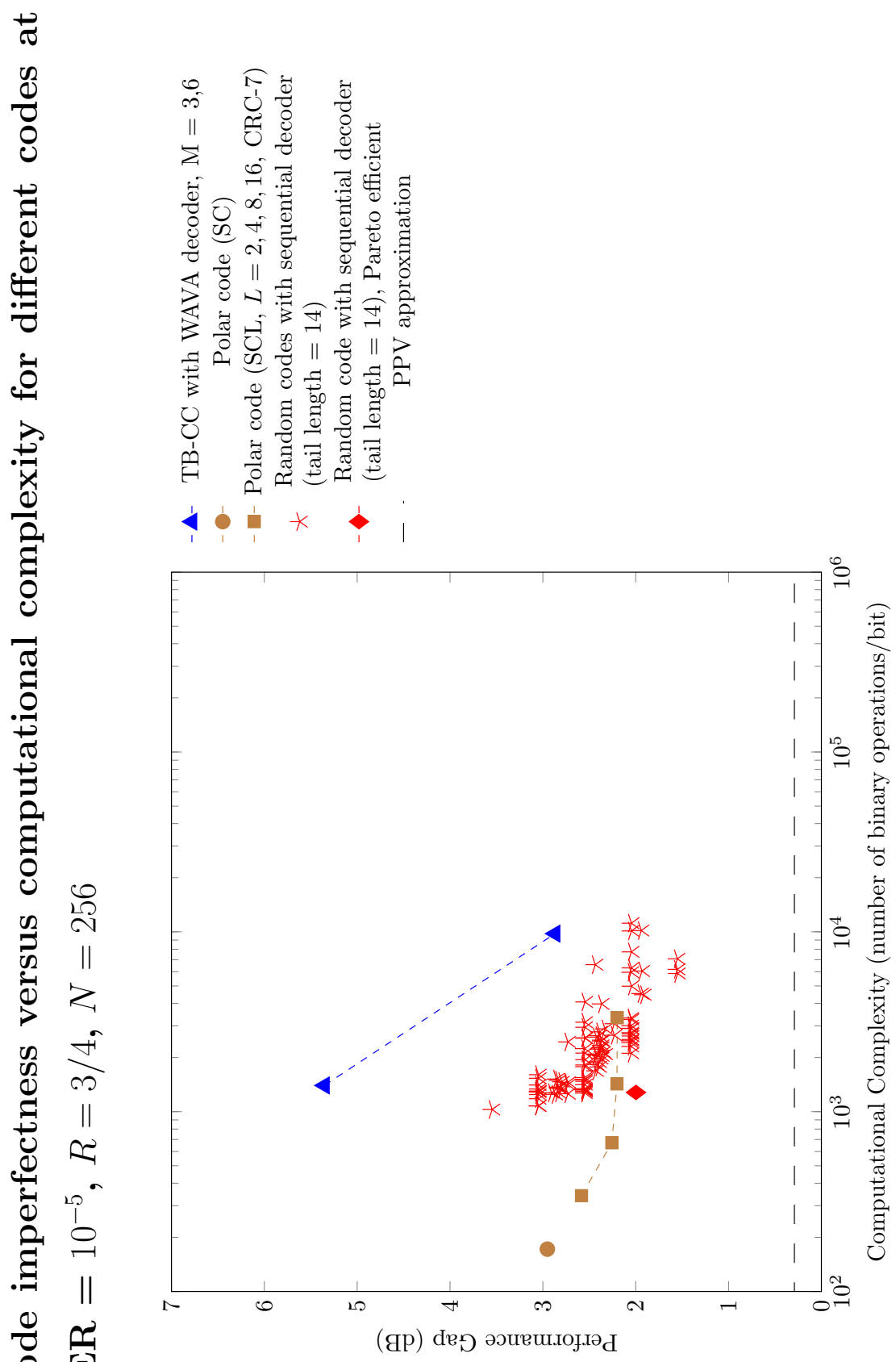

Figure C.230: Code imperfectness versus computational complexity at FER = 10^{-5} for different codes with $R = 3/4$, $N = 256$

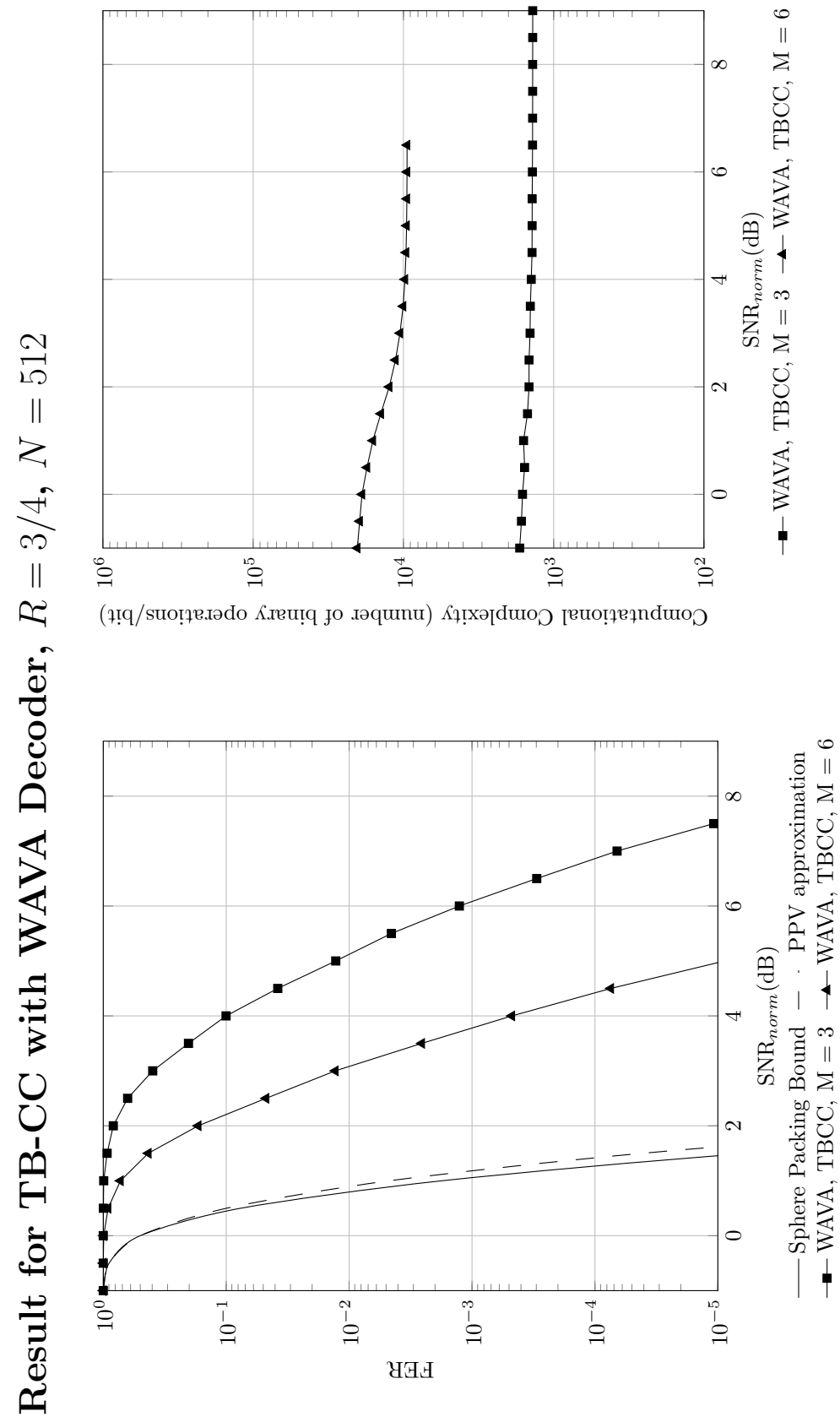

Figure C.231: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 3/4$, $N = 512$, maximum number of iterations = 4.

Figure C.232: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 3/4$, $N = 512$, maximum number of iterations = 4

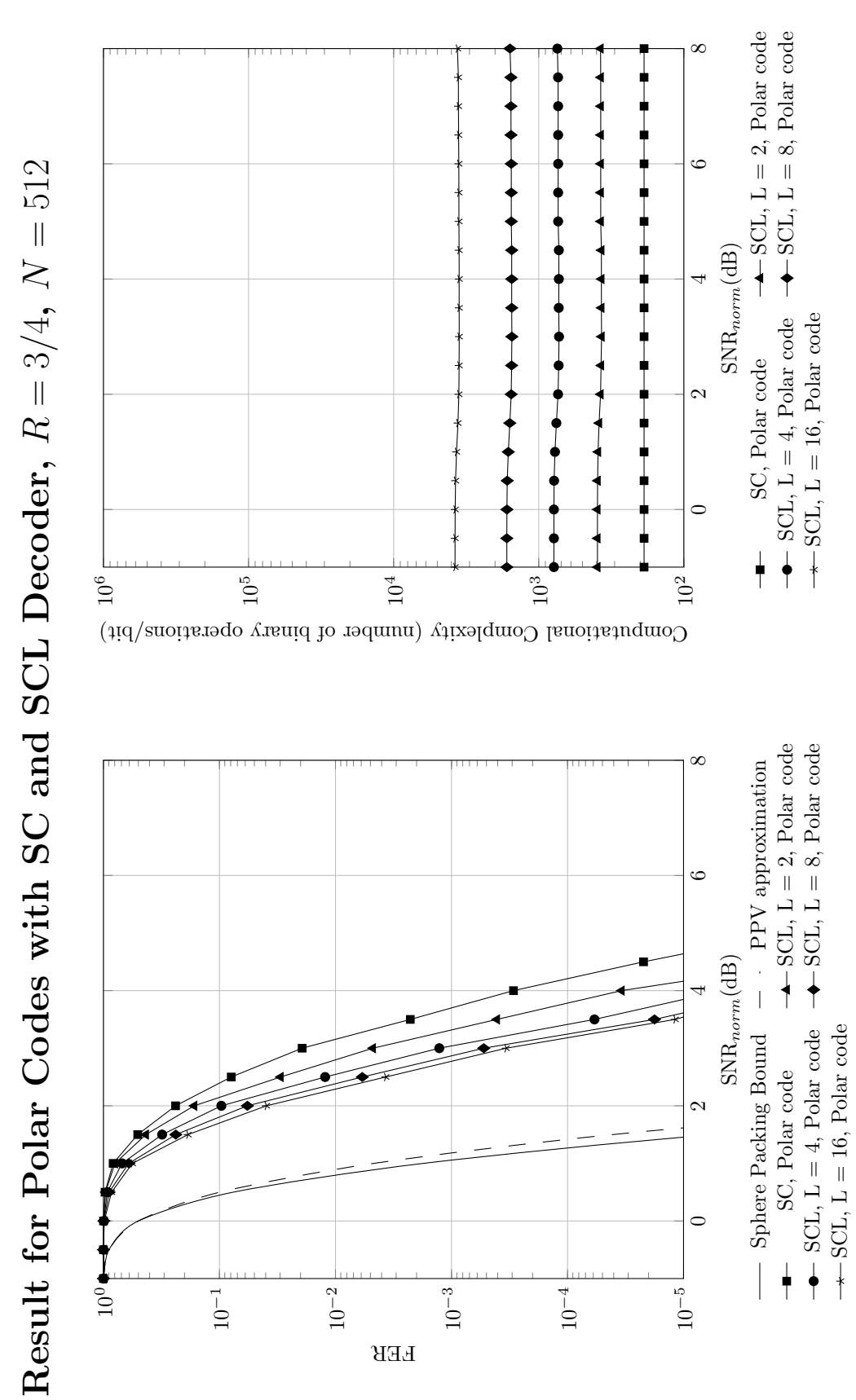

Figure C.233: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 3/4$, $N = 512$

Figure C.234: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 3/4$, $N = 512$

Result for TB-CC with Sequential Decoder, $R = 3/4$, $N = 512$

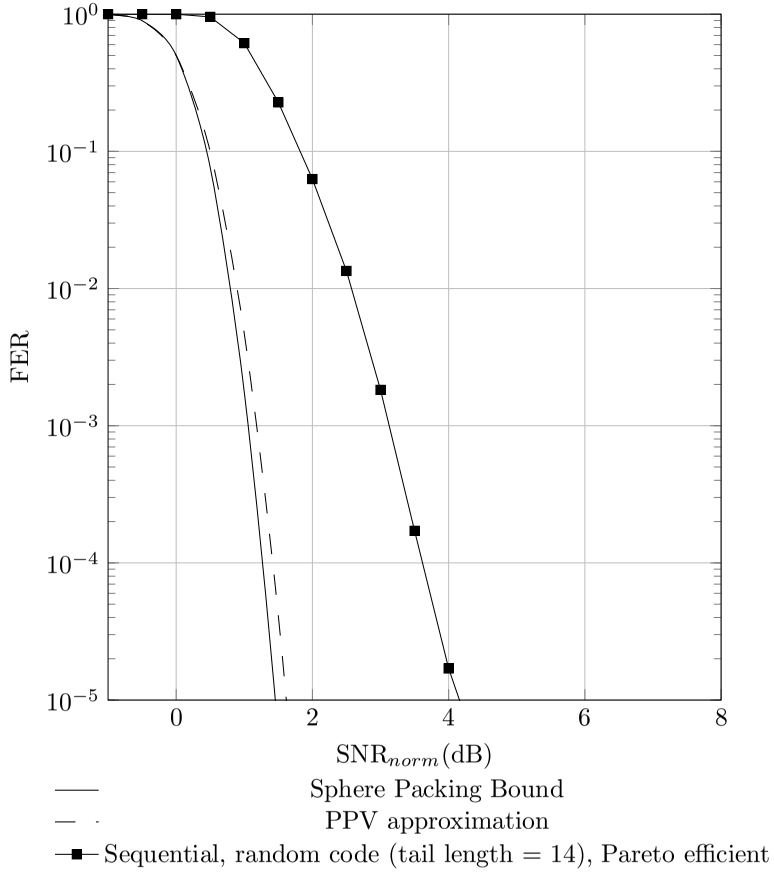

Figure C.235: Frame error rate versus normalized SNR for random code with sequential decoder, $R = 3/4$, $N = 512$

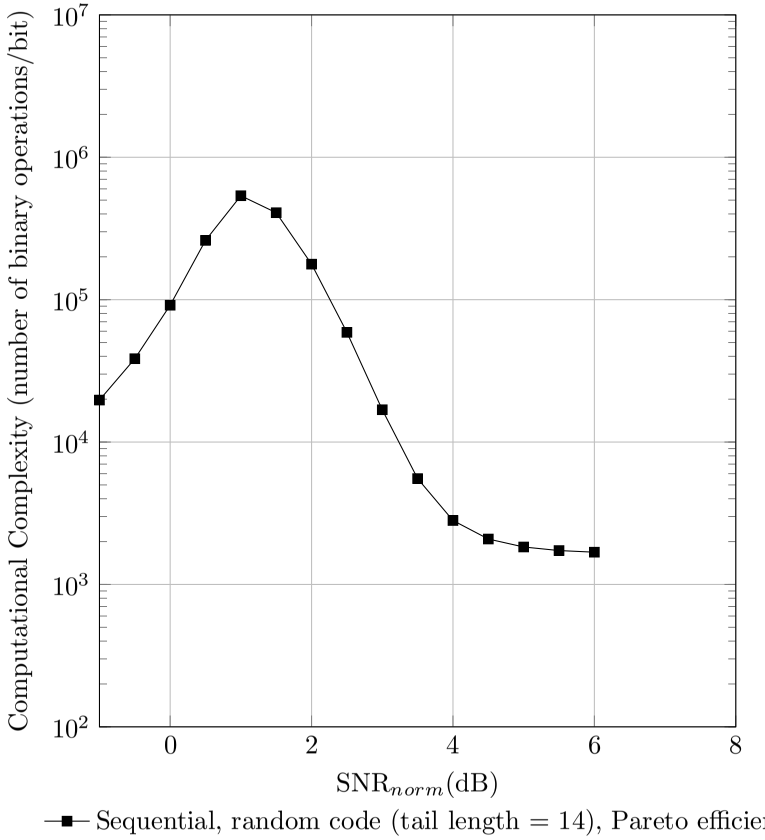

Figure C.236: Computational complexity versus normalized SNR for random code with sequential decoder, $R = 3/4$, $N = 512$

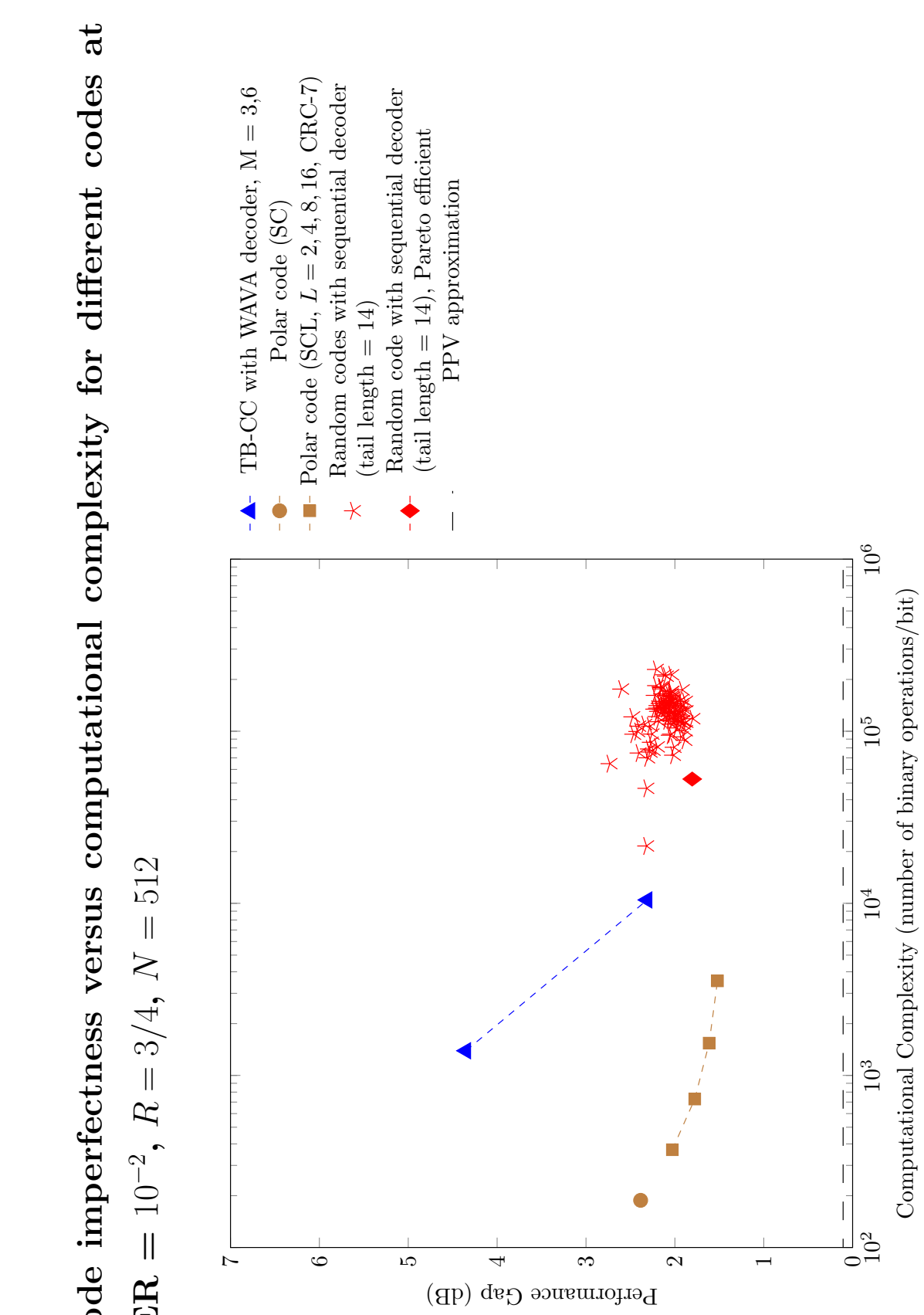

Figure C.237: Code imperfectness versus computational complexity at FER = 10^{-2} for different codes with $R = 3/4$, $N = 512$

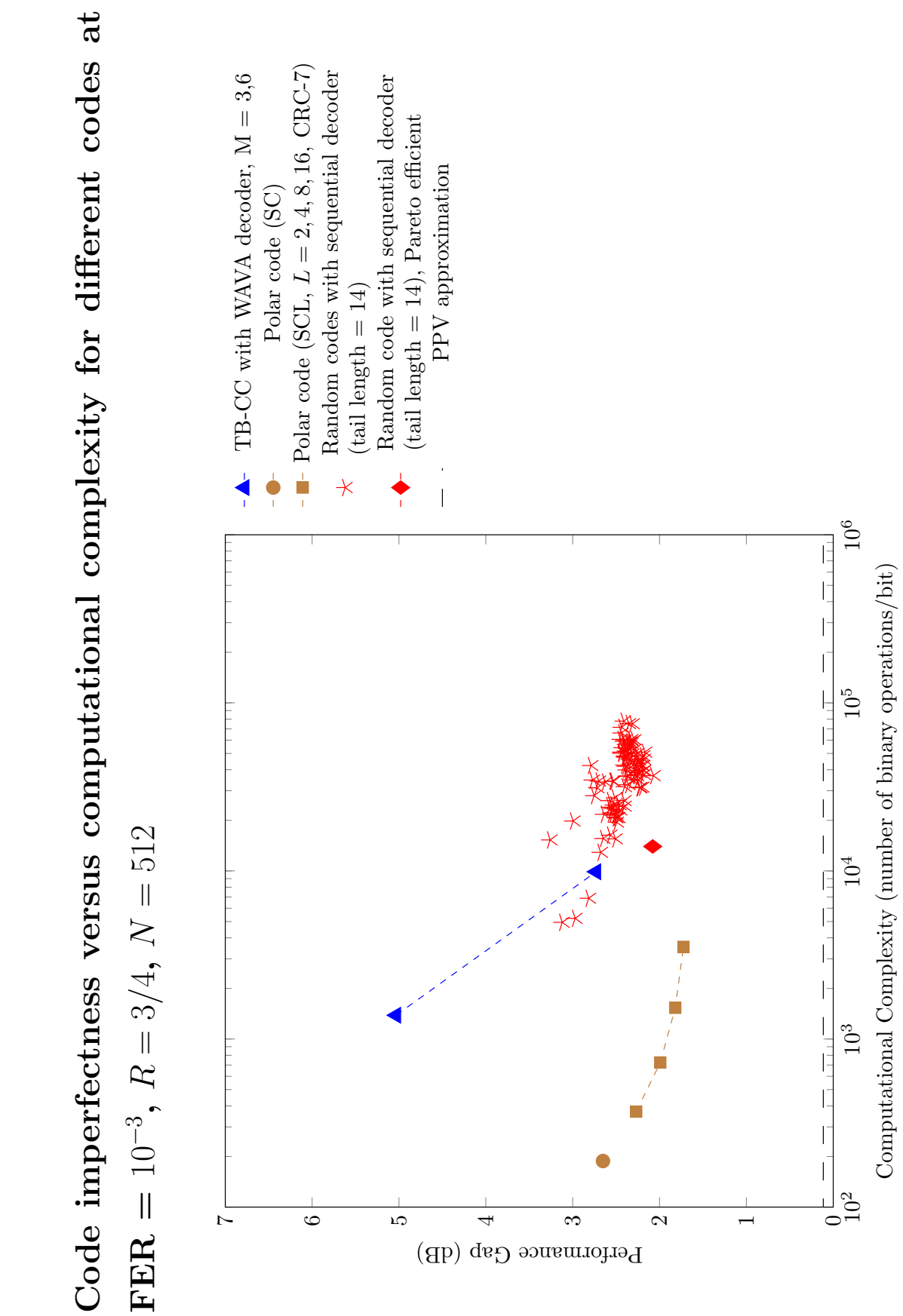

Figure C.238: Code imperfectness versus computational complexity at FER = 10^{-3} for different codes with $R = 3/4$, $N = 512$

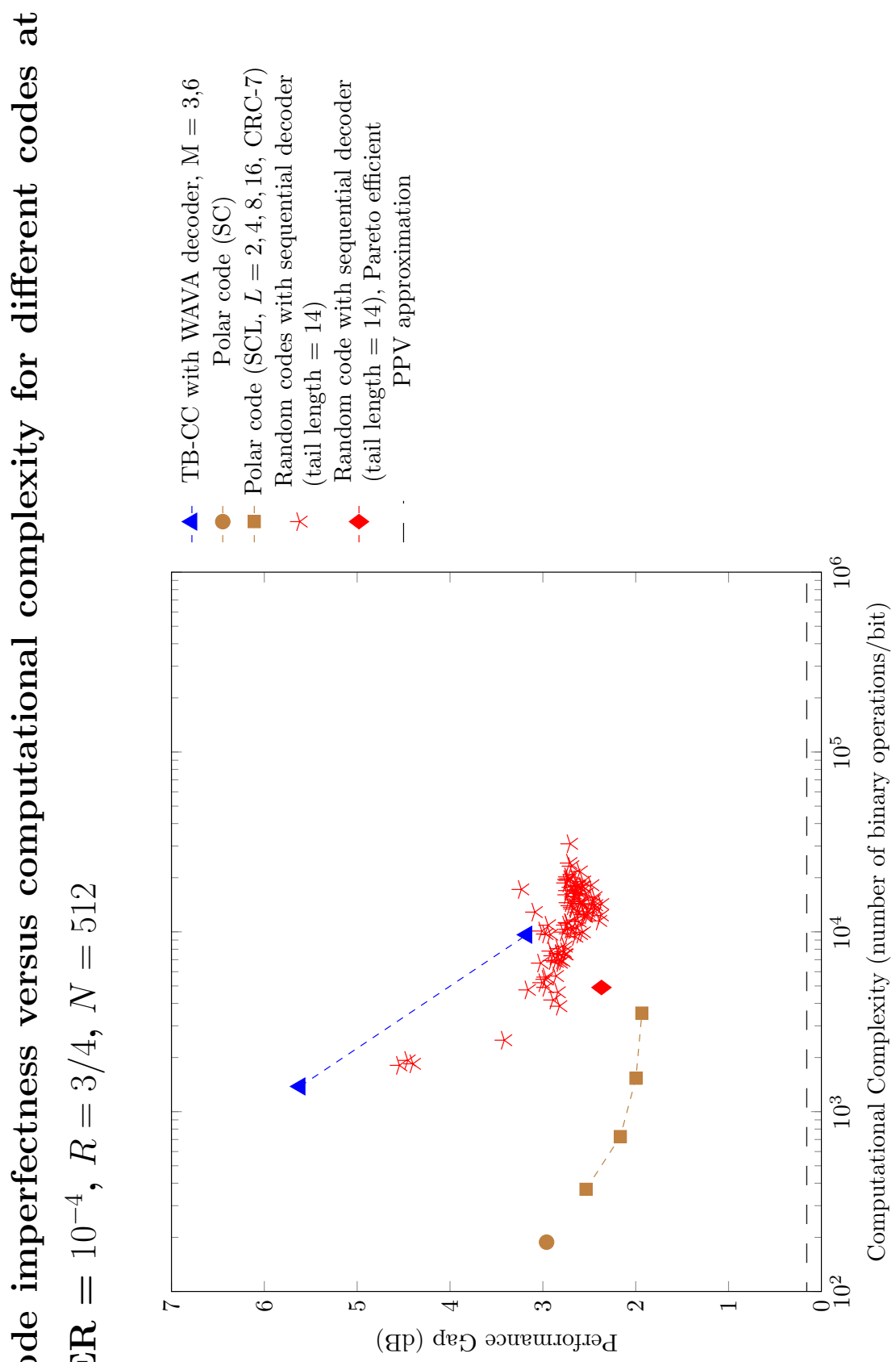

Figure C.239: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 3/4$, $N = 512$

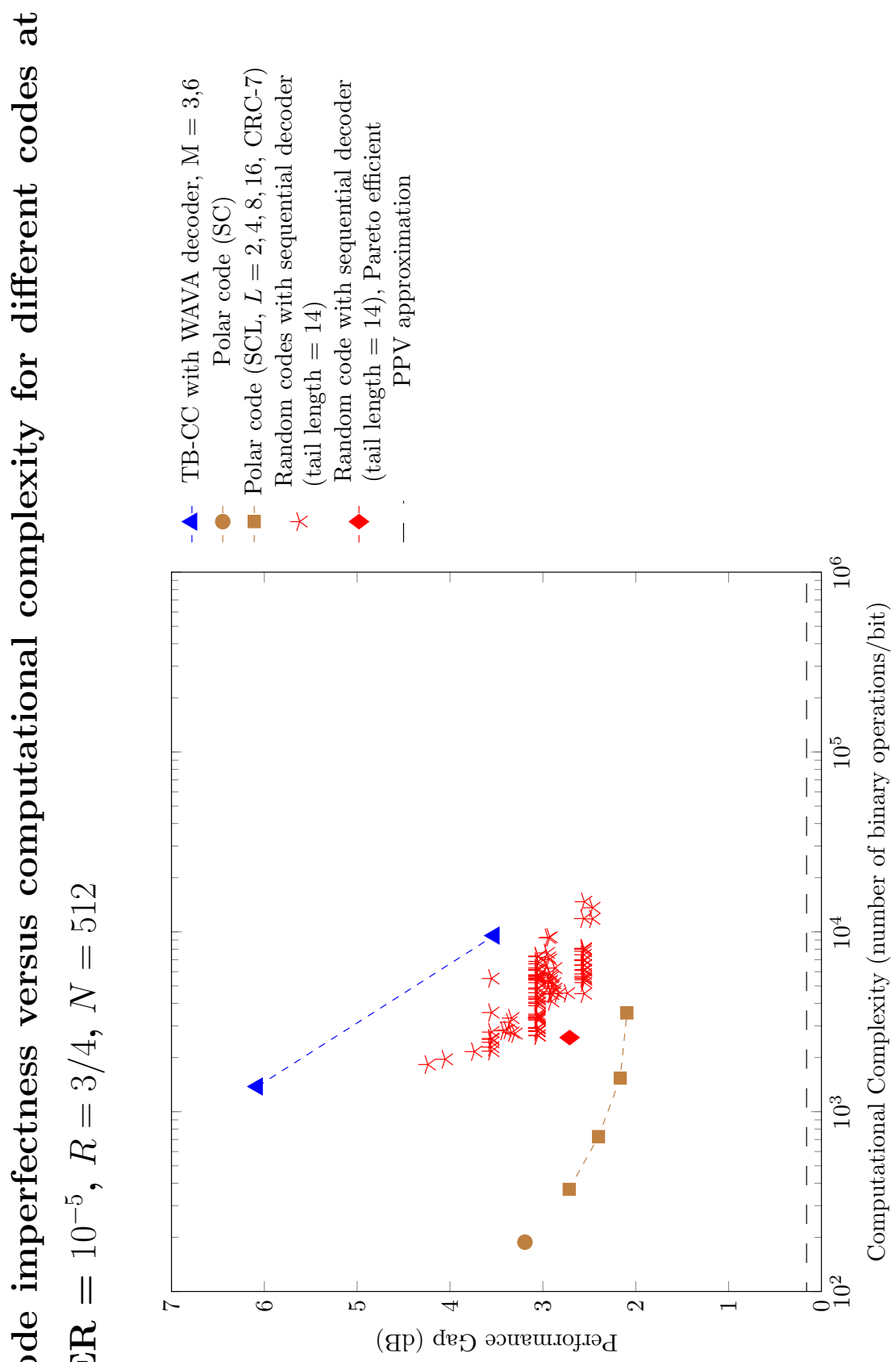

Figure C.240: Code imperfectness versus computational complexity at FER = 10^{-5} for different codes with $R = 3/4$, $N = 512$

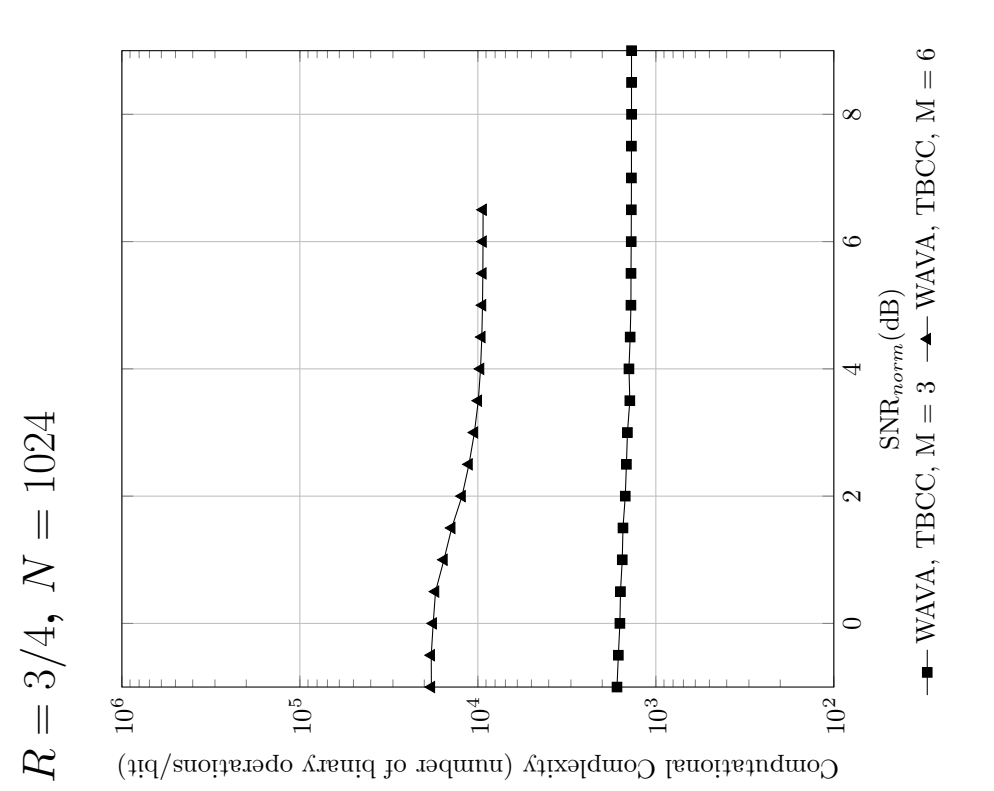

Figure C.241: Frame error rate versus normalized SNR for TB-CC with WAVA decoder, $R = 3/4$, $N = 1024$, maximum number of iterations = 4.

Figure C.242: Computational complexity versus normalized SNR for TB-CC with WAVA decoder, $R = 3/4$, $N = 1024$, maximum number of iterations = 4

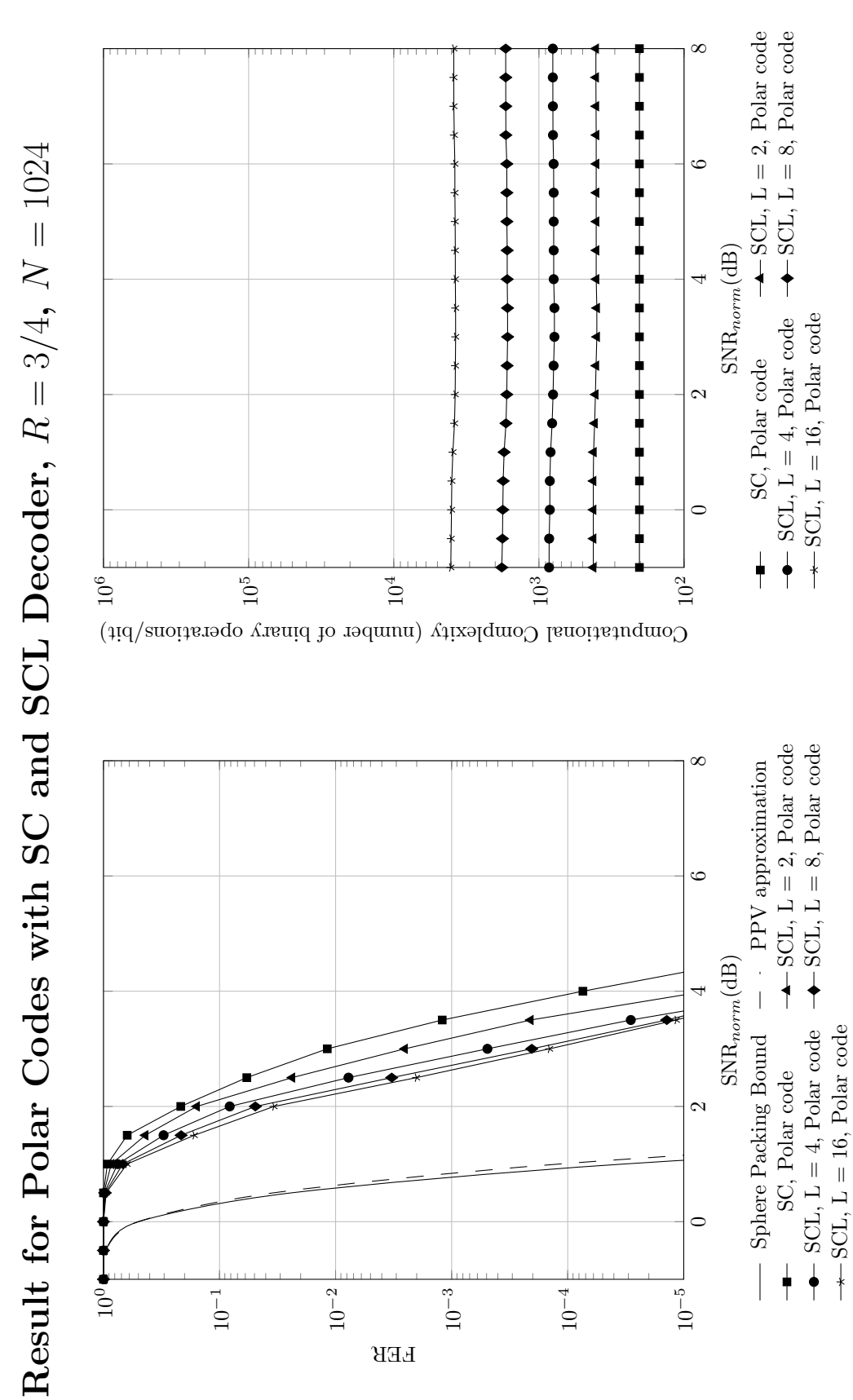

Figure C.243: Frame error rate versus normalized SNR for polar codes with SC and SCL decoder, $R = 3/4$, $N = 1024$

Figure C.244: Computational complexity versus normalized SNR for polar codes with SC and SCL decoder, $R = 3/4$, $N = 1024$

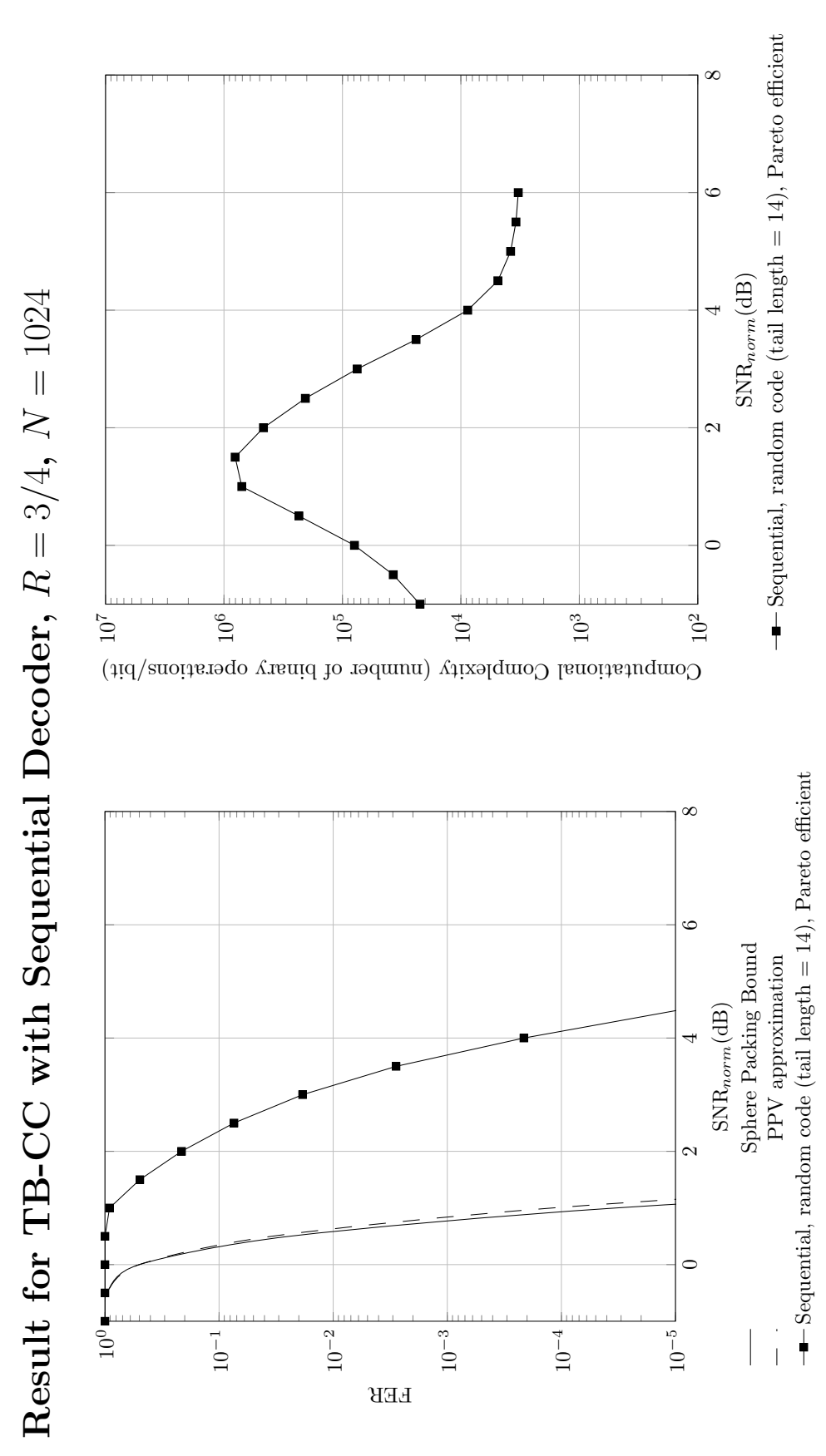

Figure C.245: Frame error rate versus normalized SNR for random code with sequential decoder, $R = 3/4$, $N = 1024$

Figure C.246: Computational complexity versus normalized SNR for random code with sequential decoder, $R = 3/4$, $N = 1024$

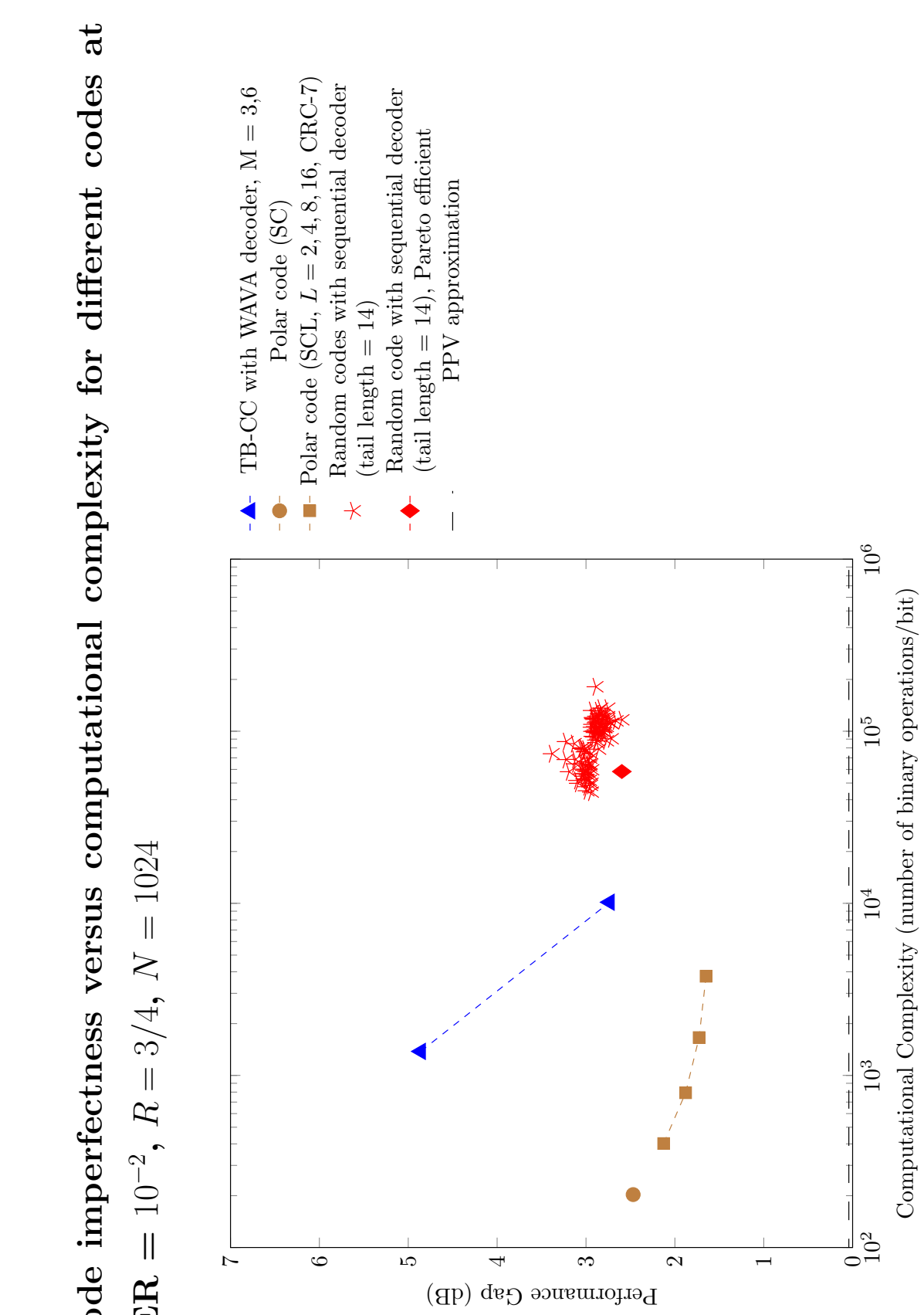

Figure C.247: Code imperfectness versus computational complexity at FER $= 10^{-2}$ for different codes with $R = 3/4$, $N = 1024$

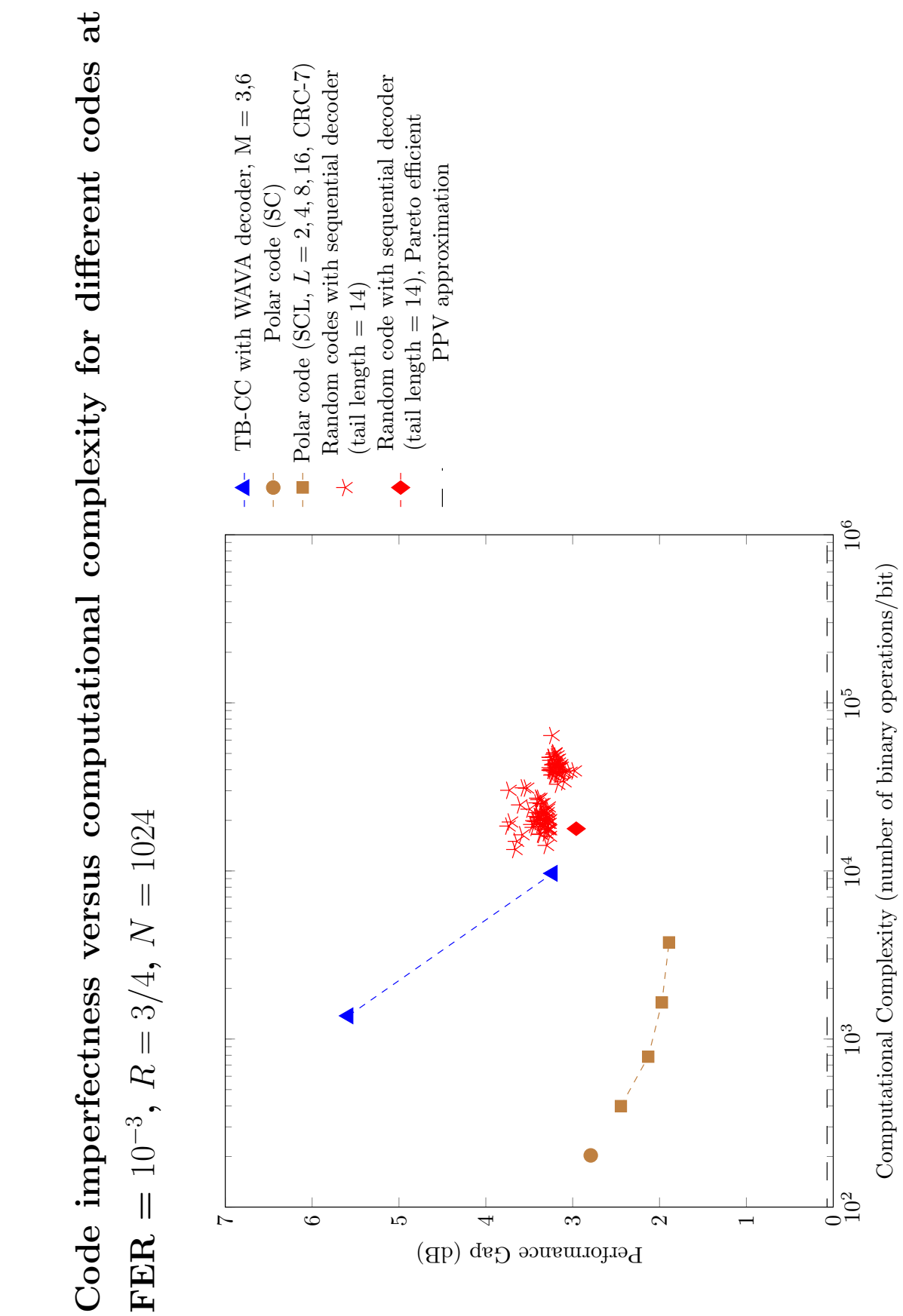

Figure C.248: Code imperfectness versus computational complexity at FER $= 10^{-3}$ for different codes with $R = 3/4$, $N = 1024$

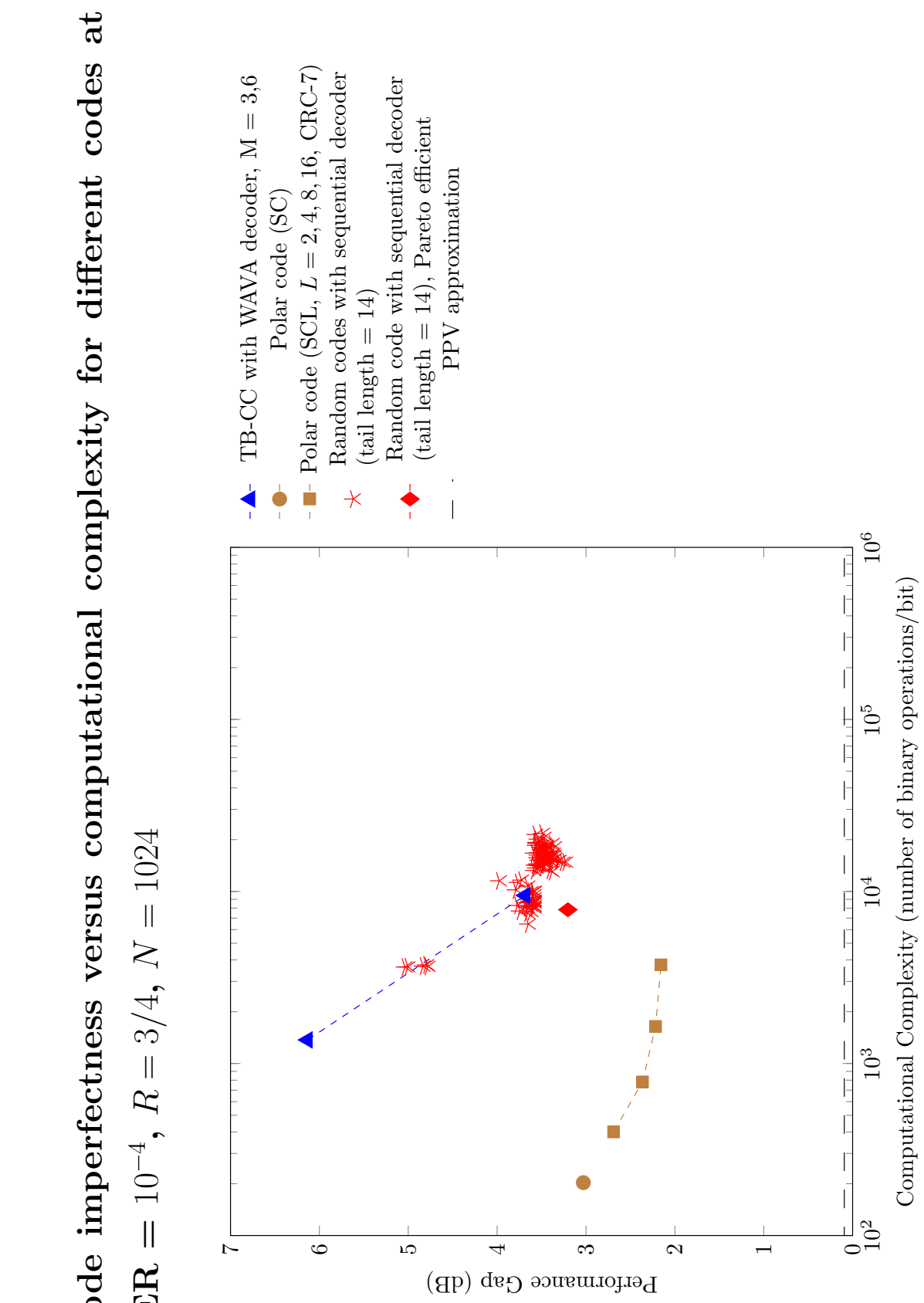

Figure C.249: Code imperfectness versus computational complexity at FER $= 10^{-4}$ for different codes with $R = 3/4$, $N = 1024$

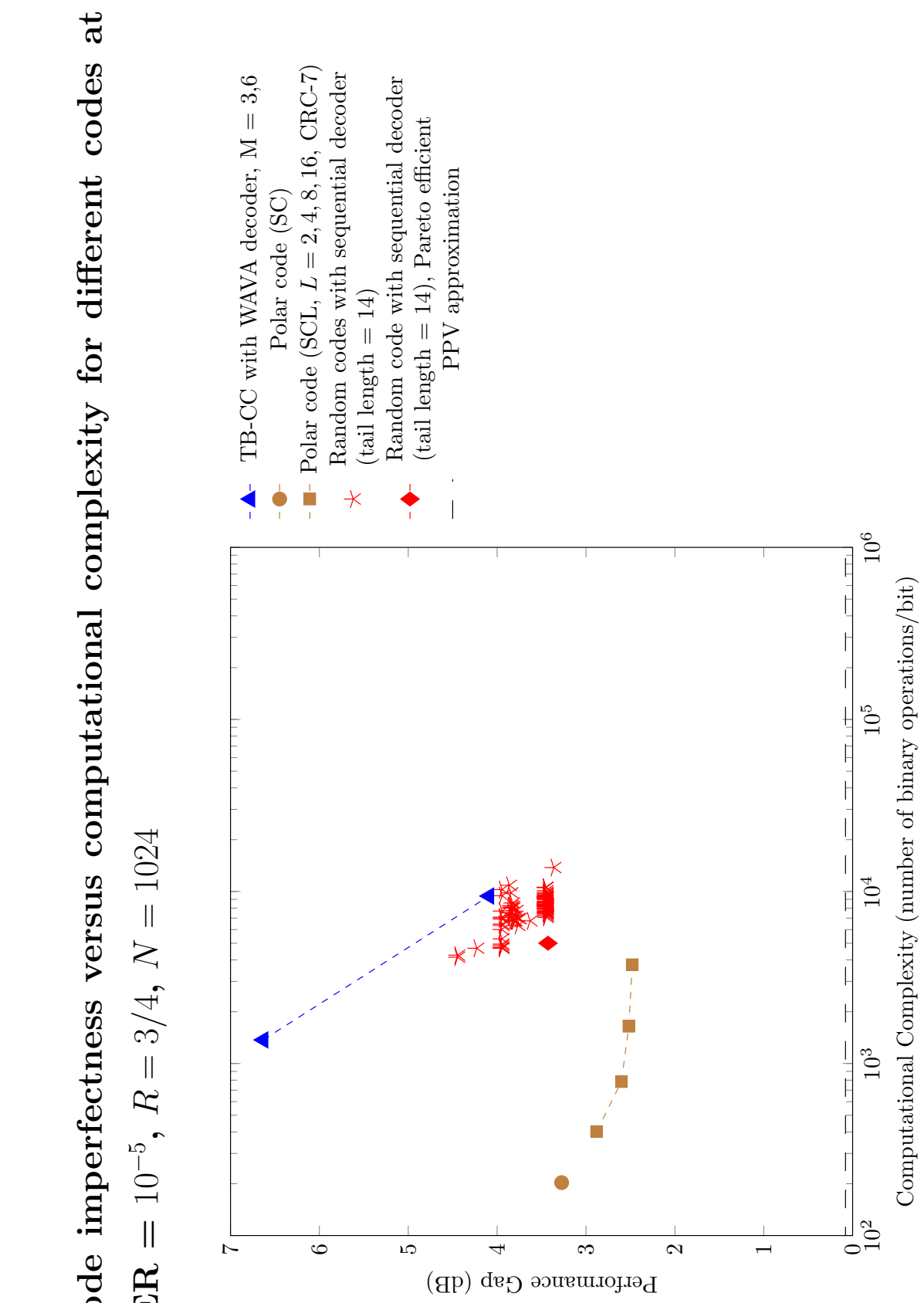

Figure C.250: Code imperfectness versus computational complexity at FER $= 10^{-5}$ for different codes with $R = 3/4$, $N = 1024$