entropy

MDPI

*Article*

# Restart Mechanisms for the Successive-Cancellation List-Flip Decoding of Polar Codes

**Charles Pillet [1,*,†], Ilshat Sagitov [1,†], Alexios Balatsoukas-Stimming [2] and Pascal Giard [1,*]**

[1] LaCIME, Department of Electrical Engineering, École de technologie supérieure (ÉTS), 1100 Notre-Dame St. West, Montréal, QC H3C 1K3, Canada; ilshat.sagitov.1@ens.etsmtl.ca
[2] Department of Electrical Engineering, Eindhoven University of Technology, Groene Loper 19, 5600 MB Eindhoven, The Netherlands; a.k.balatsoukas.stimming@tue.nl
* Correspondence: charles.pillet.1@ens.etsmtl.ca (C.P.); pascal.giard@etsmtl.ca (P.G.)
† These authors contributed equally to this work.

**Abstract:** Polar codes concatenated with a cyclic redundancy check (CRC) code have been selected in the 5G standard with the successive-cancellation list (SCL) of list size L = 8 as the baseline algorithm. Despite providing great error-correction performance, a large list size increases the hardware complexity of the SCL decoder. Alternatively, flip decoding algorithms were proposed to improve the error-correction performance with a low-complexity hardware implementation. The combination of list and flip algorithms, the successive-cancellation list flip (SCLF) and dynamic SCLF (DSCLF) algorithms, provides error-correction performance close to SCL-32 with a list size L = 2 and $T_{\max}$ = 300 maximum additional trials. However, these decoders have a variable execution time, a characteristic that poses a challenge to some practical applications. In this work, we propose a restart mechanism for list–flip algorithms that allows us to skip parts of the decoding computations without affecting the error-correction performance. We show that the restart location cannot realistically be allowed to occur at any location in a codeword as it would lead to an unreasonable memory overhead under DSCLF. Hence, we propose a mechanism where the possible restart locations are limited to a set and propose various construction methods for that set. The construction methods are compared, and the tradeoffs are discussed. For a polar code of length N = 1024 and rate 1/4, under DSCLF decoding with a list size L = 2 and a maximum number of trials $T_{\max}$ = 300, our proposed approach is shown to reduce the average execution time by 41.7% with four restart locations at the cost of approximately 1.5% in memory overhead.

**Keywords:** polar codes; decoding; execution time; complexity; energy efficiency



Academic Editors: Yongpeng Wu and Peihong Yuan

Received: 11 February 2025
Revised: 10 March 2025
Accepted: 11 March 2025
Published: 14 March 2025

**Citation:** Pillet, C.; Sagitov, I.; Balatsoukas-Stimming, A.; Giard, P. Restart Mechanisms for the Successive-Cancellation List-Flip Decoding of Polar Codes. *Entropy* **2025**, *27*, 309. https://doi.org/10.3390/e27030309



## 1. Introduction

Polar codes [1] are a class of linear block codes that were shown to asymptotically achieve the channel capacity under low-complexity successive-cancellation (SC) decoding as the code length tends to infinity [1]. However, the SC decoding algorithm does not provide sufficient error-correction performance at short-to-moderate code lengths. The concatenation of a cyclic redundancy check (CRC) code to the polar code results in the CRC-aided (CA)–polar code scheme; this scheme was selected to protect the control channel of the enhanced mobile broadband (eMBB) service in 5G [2]. SC does not take advantage of the CRC code.

Successive-cancellation list (SCL) decoding [3] was proposed to improve the error-correction performance of SC decoding. SCL decoding retains $L$ decoding paths, providing

$L$ different candidates of a codeword. To achieve this, the number of paths is doubled, and when it reaches $2L$, the best $L$ paths according to path metrics (PMs) are selected to continue the decoding. The true potential of the SCL decoding algorithm is highlighted with the CA–polar code scheme. The candidate with the smallest PM that verifies the CRC code is elected as the codeword estimate.

Another decoding algorithm for the CA–polar codes is the flip algorithm [4,5]. Unlike SCL decoding, successive-cancellation flip (SCF) and dynamic SCF (DSCF) sequentially attempt the decoding with a single SC instance, providing up to $T_{\max}$ candidates of a codeword. At each additional trial, one of the decision bits, based on a metric, is flipped during the course of SC decoding before normal decoding is resumed. DSCF better approximates the reliability of a decision, providing a more accurate list of bit-flipping candidates and improving the error-correction performance. Moreover, DSCF may also handle multiple bit flips per decoding trial. The latter requires a list of candidates that is dynamically updated along the decoding trials.

Successive-cancellation list flip (SCLF) decoding [6] combines the list and flip algorithms. Additional trials of SCLF correspond to the SCL algorithm having a reverse path selection on the path-flipping locations [7]. These locations are based on a flip metric computed during the initial SCL trial [8]. In [9], dynamic SCLF (DSCLF) decoding is proposed, which adapts the DSCF methodology to SCLF. That is, the metric that selects path-flipping locations is further improved, and the multi-path-flipping methodology of DSCF is adapted.

Since it combines list and flip algorithms, the algorithm is computationally complex and works according to the methodology outlined by [10–15], who investigated ways to reduce the complexity of the list–flip algorithm. In [10,11], a simplified metric and an adaptive list size $L$ are proposed, allowing us to reduce the complexity of $L \neq 2$. A low-complexity flip metric allowing fast decoding of some special nodes is proposed in [12]. The fast implementation of the SCLF decoder is provided in [13], reducing up to 73.4% of the average decoding latency for $N = 512$ and $K = 256$ codes. In [14], the number of additional trials is restricted to one, but this specific case can be improved with a more accurate method for locating the first error position. In [15], the authors created a scheme with early termination, average execution time reduction, and enhanced performance by protecting the codeword with multiple CRC codes. However, the total number of CRC bits is high and specific patterns are required.

Previously, we proposed restart mechanisms for SCF-based decoders [16,17]. The simplified restart mechanism (SRM) conditionally restarts SC decoding from the second half of the codeword if that is where the current bit-flipping candidate is located. The generalized restart mechanism (GRM) restarts SC decoding from any location of the bit-flipping candidate, which is achieved by storing the decoded codeword in its memory at the end of the initial unsuccessful decoding trial. It was shown that GRM is applicable to various types of SCF-based decoders. This work represents the continuation of [16,17] but for the list–flip algorithm. The restart mechanism for the list–flip decoder shown in this paper reduces the complexity less in comparison to the GRM, and we will show that embedding the GRM is impracticable for this decoder. Nevertheless, the complexity reduction is better with respect to the SRM, and the error-correction performance is not reduced.

*Contributions*

This work proposes the limited-locations restart mechanism (LLRM) for SCLF-based decoders for polar codes with the central idea to partially skip decoding computations that are identical between the initial trial and any additional trial. For any additional trial, the decoding restart is performed from one of the restart locations that is the closest to the path-flipping location from its right-hand side (RHS) in a codeword. The restart

locations are determined offline, i.e., by conducting simulations at the target frame-error rate (FER). An LLRM does not alter the error-correction performance of the original decoder. During a restart, the decoding tree is calculated from its root and partial-sum (PS) bits are recalculated. Relevant path information required to retrieve the decoding tree is stored to memory. When applying the LLRM with four restart locations to the DSCLF decoder that can flip up to three paths per decoding trial, the average flip time reduction is 51% for $N = 512$ and $K = 128$ while requiring approximately 1.7% of additional memory compared to the standard DSCLF decoder.

*Outline*

The remainder of this paper is organized as follows: Section 2 begins by introducing polar codes with SC decoding. Then, SCL, SCF, and SCLF decoders are described. Section 4 begins with a simulation-based statistical analysis that provides bit-flip location distributions. In Section 5, simulation results for SCLF and DSCLF decoders with LLRM, and previously proposed GRMs, for the standard scheme are provided. Comparisons are made in terms of error-correction performance, memory estimates, and average execution time.

## 2. Background

### 2.1. Construction of Polar Codes

An $(N, K)$ polar code [1] is a linear block code of length $N = 2^n$ and code dimension $K$, defining the code rate $R_{code} = K/N$. Polar codes encode an input vector $\boldsymbol{u} = [u_0, \ldots, u_{N-1}]$ to the codeword $\boldsymbol{x} = [x_0, \ldots, x_{N-1}]$ as $\boldsymbol{x} = \boldsymbol{u}\boldsymbol{T}_2^{\otimes n}$ where $\boldsymbol{T}_2^{\otimes n}$ denotes the Kronecker power of the binary kernel $\boldsymbol{T}_2 = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$. The matrix $\boldsymbol{T}_N = \boldsymbol{T}_2^{\otimes n}$ induces channel polarization [1], i.e., each of the $N$ bits $\boldsymbol{u}$ has its own bit channel defined by its own reliability. Classifying the $N$ bit channels according to their reliabilities is not an easy task; several methods exist to achieve this in this paper, which can also be found in [18]. The polar code construction consists of splitting the $N$ positions in $\boldsymbol{u}$ into two sets, $\boldsymbol{\mathcal{A}}$ and $\boldsymbol{\mathcal{A}}^c$ (with $|\boldsymbol{\mathcal{A}}| = K$), being the information set and the frozen set. The information set corresponds to the indices of the $K$ most reliable positions, while the remaining $(N - K)$ bits, called frozen bits, are set to predefined values that are known by the decoder, which are typically zeros. The vector $\boldsymbol{u}$ contains the message $\boldsymbol{m}$ of $K$ bits in the positions stated in $\boldsymbol{\mathcal{A}}$, i.e., $\hat{\boldsymbol{u}}_{\boldsymbol{\mathcal{A}}} = \boldsymbol{m}$ and contains $N - K$ frozen bits, i.e., $\hat{\boldsymbol{u}}_{\boldsymbol{\mathcal{A}}^c} = \boldsymbol{0}$. In this work, the binary phase-shift keying (BPSK) modulation over an additive white Gaussian noise (AWGN) channel is used, as well as the polar code construction of [18].

CA–polar codes include the concatenation of a CRC code during the polar encoding. The $r$ CRC bits are generated on the basis of the message $\boldsymbol{m}$ and are placed in the set of information bits $\boldsymbol{\mathcal{A}}$, increasing the number of information bits to $K_{\text{tot}} = K + r$. Next, information-bit indices of CA–polar codes are noted $a_1 < \cdots < a_{K+r}$. The $(N, K + r)$ notation is used throughout this work to indicate the code parameters of CA–polar codes.

### 2.2. SC Decoding

Polar codes have been proposed with the low-complexity SC algorithm in [1] to retrieve an estimate $\hat{\boldsymbol{u}}$ of the input vector $\boldsymbol{u}$. The description of SC decoding as a binary tree traversal was proposed in [19], where the tree is traversed depth-first, starting with the left branch. The decoding tree of an $(8, 4)$ polar code is shown in Figure 1, where the stages are $s \in \{n, \ldots, 0\}$ with the root node at $s = n$. The received vector of channel log-likelihood ratios (LLRs), denoted by $\boldsymbol{\alpha}_{\text{ch}} = [\alpha_{\text{ch}}(0), \ldots, \alpha_{\text{ch}}(N-1)]$, is at the tree root. For any node, denoted by $v$ and located at stage $s$, the input LLR vector is $\boldsymbol{\alpha}_v = [\alpha_v(0), \ldots, \alpha_v(2^s - 1)]$ from the parent node, and the two input partial-sum (PS) vectors are $\boldsymbol{\beta}^l = \left[\beta^l(0), \ldots, \beta^l\left(2^{s-1} - 1\right)\right]$ and $\boldsymbol{\beta}^r = \left[\beta^r(0), \ldots, \beta^r\left(2^{s-1} - 1\right)\right]$, from its left and right child nodes, respectively. The left and right child nodes of the node $v$ have the vectors $\boldsymbol{\alpha}_v^l = \left[\alpha_v^l(0), \ldots, \alpha_v^l\left(2^{s-1} - 1\right)\right]$

and $\boldsymbol{\alpha}_v^r = [\alpha_v^r(0), \ldots, \alpha_v^r(2^{s-1}-1)]$, respectively, where each LLR element is calculated as follows:

$$\alpha_v^l(j) = f\left(\alpha_v(j), \alpha_v\left(j+2^{s-1}\right)\right), \tag{1}$$

$$\alpha_v^r(j) = g\left(\alpha_v(j), \alpha_v\left(j+2^{s-1}\right), \beta^l(j)\right), \tag{2}$$

where $j \in \{0, \ \ldots, \ 2^{s-1}-1\}$. Going left, the $f$ function is calculated as [20]

$$f(\alpha_1, \alpha_2) = \text{sgn}(\alpha_1)\text{sgn}(\alpha_2)\min(|\alpha_1|, |\alpha_2|). \tag{3}$$

Going right, the $g$ function is calculated as

$$g(\alpha_1, \alpha_2, \beta_1) = (1-2\beta_1)\alpha_1 + \alpha_2. \tag{4}$$

Bit estimates, $\hat{\boldsymbol{u}} = [\hat{u}_0, \ldots, \hat{u}_{N-1}]$, are obtained either by taking a hard decision on the decision LLRs, $\boldsymbol{\alpha}_{\text{dec}} = [\alpha_{\text{dec}}(0), \ldots, \alpha_{\text{dec}}(N-1)]$, that reach the leaf nodes or using the frozen bit values. Namely, we have

$$\hat{u}_i = \begin{cases} \text{HD}(\alpha_{\text{dec}}(i)), & \text{if } i \in \boldsymbol{\mathcal{A}} \\ 0, & \text{if } i \in \boldsymbol{\mathcal{A}}^c. \end{cases} \tag{5}$$

where $\text{HD}(\cdot)$ represents the hard decision function. In Figure 1, the nodes represented in black and white correspond to information and frozen bits. The partial-sum vector, denoted by $\boldsymbol{\beta}$, is calculated from the bit estimates and is propagated up from children to parent nodes. At any node $v$, each bit of $\boldsymbol{\beta}$ is calculated as follows:

$$\beta_v(j) = \begin{cases} \beta^l(j) \oplus \beta^r(j), & \text{if } \ j < 2^{s-1}, \\ \beta^r(j), & \text{otherwise,} \end{cases} \tag{6}$$

where $j \in \{0, \ldots, 2^s-1\}$ and $\oplus$ is a bitwise XOR.

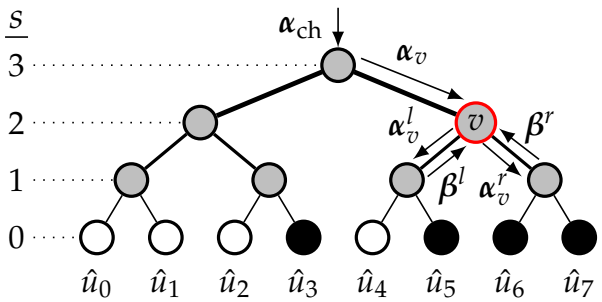



**Figure 1.** SC decoding tree of an $(8, 4)$ polar code.

### *2.3. SCL Decoding*

SCL decoding was proposed in [3] and uses $L$ parallel instances of the SC decoder, where $L$ is a power of 2. During the decoding, the $L$ best decoding paths are kept through the computation of PMs. At the end of the decoding, a set of $L$ bit-estimate candidates, denoted as $\hat{\boldsymbol{\mathcal{U}}}_{N-1} = \{\hat{\boldsymbol{u}}(1), \ldots, \hat{\boldsymbol{u}}(L)\}$, is provided. Next, the SCL decoding is described in more detail.

First, for all paths, the bit-estimate vectors are initialized to 0, i.e., $\forall l \in [1, L], \hat{\boldsymbol{u}}(l) = \boldsymbol{0}$, with $l$ denoting the path index. Moreover, all PMs are also initialized to 0, i.e., $\forall l \in [1, L], \text{PM}(l) = 0$. The decoding starts with a single path, and intermediate LLRs are computed according to the SC schedule and the update rules (1) and (2). When the decoder

reaches the first information bit $a_1$, the single path is duplicated, i.e., all intermediate partial sums and all intermediate LLRs including the decision LLR $\alpha_{\text{dec}}(a_1)$ are duplicated to another path structure. Contrary to SC decoding where the bit-estimate $\hat{u}_{a_1}$ is taken according to the hard decision $\text{HD}(\alpha_{\text{dec}}(a_1))$ (5), the path duplication in SCL decoding allows us to consider both options $\{0,1\}$ regardless of the sign of the LLR $\alpha_{\text{dec}}(a_1)$. However, the PMs are updated based on the chosen bit decision and the sign of the decision LLR $\alpha_{\text{dec}}$ on their path. Namely, the PM is penalized if the bit decision does not follow the sign of the decision LLR, i.e., for the $l$-th path, the path metric $\text{PM}(l)$ is updated as [21]

$$\text{PM}(l) = \begin{cases} \text{PM}(l) + |\alpha_{\text{dec}}(l,a)|, & \text{if } \text{HD}(\alpha_{\text{dec}}(l,a)) \neq \hat{u}_a(l) \\ \text{PM}(l), & \text{otherwise,} \end{cases} \tag{7}$$

where $\alpha_{\text{dec}}(l,a)$ is the decision LLR for the $l$-th path at index $a \in \boldsymbol{\mathcal{A}}$, while $\hat{u}_a(l)$ is the bit decision.

For the first information indices $\boldsymbol{\mathcal{A}}_{\text{dup}} \triangleq \{a_1, \ldots, a_{log_2(L)}\}$, the duplication of the paths does not result in more than $L$ paths; hence, no sorting and selection of the best $L$ paths are required. However, for all information bits $a \in \boldsymbol{\mathcal{A}}_{\text{sort}} \triangleq \{a_{log_2(L)+1}, \ldots, a_{K+r}\}$, $2L$ paths are obtained after duplication. Hence, a selection of the best $L$ paths is required and is performed through the sorting operation of the $2L$ PMs. The $L$ paths having the smallest PM remain.

After duplication, the set of $2L$ PMs is noted $\mathbf{PM}_{\text{dup}_a} = \{\text{PM}(1), \ldots, \text{PM}(2L)\}$ with $a \in \boldsymbol{\mathcal{A}}_{\text{sort}}$, $\forall \text{PM}(l) \in \mathbf{PM}_{\text{dup}_a}$, and $\text{PM}(l)$ is updated as (7). For $\forall a \in \boldsymbol{\mathcal{A}}_{\text{sort}}$, the set $\hat{\boldsymbol{\mathcal{U}}}_{\text{dup}_a} = \{\hat{\boldsymbol{u}}_0^a(1), \ldots, \hat{\boldsymbol{u}}_0^a(2L)\}$ gathers the $2L$ partial candidates after duplication at index $a$. After the sorting of $\mathbf{PM}_{\text{dup}_a}$, the following notations are used: $\mathbf{PM}_{\text{sort}_a} = \{\mathbf{PM}_a, \mathbf{PM}_{\text{worst}_a}\}$ and $\hat{\boldsymbol{\mathcal{U}}}_{\text{sort}_a} = \{\hat{\boldsymbol{\mathcal{U}}}_a, \hat{\boldsymbol{\mathcal{U}}}_{\text{worst}_a}\}$. Each subset contains the information of $L$ paths. The SCL decoding continues with the best $L$ paths composed of the partial bit estimates $\hat{\boldsymbol{\mathcal{U}}}_a = \{\hat{\boldsymbol{u}}_0^a(1), \ldots, \hat{\boldsymbol{u}}_0^a(L)\}$ and the corresponding $L$ PMs $\mathbf{PM}_a = \{\text{PM}(1), \ldots, \text{PM}(L)\}$. For $i \in \boldsymbol{\mathcal{A}}^c$, the $L$ paths takes the frozen decision (5) and the update of the $L$ PMs is performed according to (7).

SCL decoding shows its true potential for CA–polar codes. At the end of the decoding, a CRC check is performed on the $L$ candidates stored in $\hat{\boldsymbol{\mathcal{U}}}_{N-1}$. The final decoded codeword is the candidate that satisfies the CRC while minimizing the path metric. If none of the bit estimates satisfy the CRC, a decoding failure is declared.

### *2.4. SCF Decoding*

The SCF algorithm was proposed in [4] for CA–polar codes. This algorithm reuses a single SC instance to perform the decoding. If the CRC check fails at the end of the initial SC decoding trial, up to $T_{\text{max}}$ additional trials are performed, each of them involving a bit flip in the bit-estimate $\hat{\boldsymbol{u}}$. The decoding stops when one additional trial returns a candidate $\hat{\boldsymbol{u}}$ checking the CRC code or after $T_{\text{max}}$ additional trials resulting in a CRC check failure. The DSCF decoding was proposed to enhance the performance of SCF [5]; DSCF is defined by the decoding order $\omega$, stating the maximum number of bit flips that can be performed in the additional trial. If $\omega = 1$, the list of bit-flip candidates is known at the end of the initial trial; if $\omega = \{2,3\}$, the list of bit-flip candidates is adjusted in all additional trials, i.e., the list is dynamic. A greater decoding order $\omega$ results in greater error-correction gain but also requires more $T_{\text{max}}$ trials as well as more memory because of the storage of the flip metrics and of the bit-flip locations. For the $t$-th additional trial, $1 \leq t \leq T_{\text{max}}$, the set $\varepsilon_t$ stores the bit-flip locations with $|\varepsilon_t| \leq \omega$. The list of all bit-flipping candidates is denoted as $\boldsymbol{\mathcal{B}}_{\text{flip}} = \{\varepsilon_1, \ldots, \varepsilon_t, \ldots \varepsilon_{T_{\text{max}}}\}$.

For $\omega \geq 2$, the latency of DSCF is its main drawback due to the sequential trials and the greater $T_{\max}$ to approach the full potential of the decoding algorithm. Options exist to improve the decoding latency of DSCF. The baseline algorithm can be the fast-SSC algorithm [22], having special nodes in order to not traverse the full decoding tree. However, for $\omega \geq 2$, the special nodes need to be adapted and reduced in size, leading to a more complex implementation [23] with respect to $\omega = 1$. Another option is the restart mechanism [16,17].

*An optimal restart mechanism for flip decoders: the generalized restart mechanism (GRM):*

Authors in [17] proposed the GRM to restart the decoding elsewhere other than the first leaf for the additional trials. At additional trial $1 \leq t \leq T_{\max}$, the restart location $\psi_t$ depends on the information set $\boldsymbol{\mathcal{A}}$ and the first bit-flip location $\min(\varepsilon_t)$. This mechanism requires only the storage of $\hat{\boldsymbol{u}}$, leading to a small memory overhead that allows the retrieval of the intermediate partial sum $\boldsymbol{\beta}_{\text{int}}$. A module to perform the correct $f$ and $g$ sequence is required to retrieve the intermediate LLRs $\boldsymbol{\alpha}_{\text{int}}$. This sequence is retrieved with the binary representation of $\psi_t$ [17]. Figure 2 depicts the additional trial for a flip decoder embedding the GRM. The first bit-flip location is $\min(\varepsilon_t) = 9$, leading to the restart location $\psi_t = 11$. The skipped nodes are shown in blue, while the restart path is depicted in red. The GRM can be embedded into any baseline algorithm, i.e., standard SC [1] or the fast-SSC [22] decoding algorithm. The average reduction brought by the GRM depends on the code rate and the baseline algorithm. For $N = 1024$ and code rate $R_{code} = 1/8$, the reduction is up to 56.9% with SC as the baseline algorithm and 20.9% with fast-SSC as the baseline algorithm.

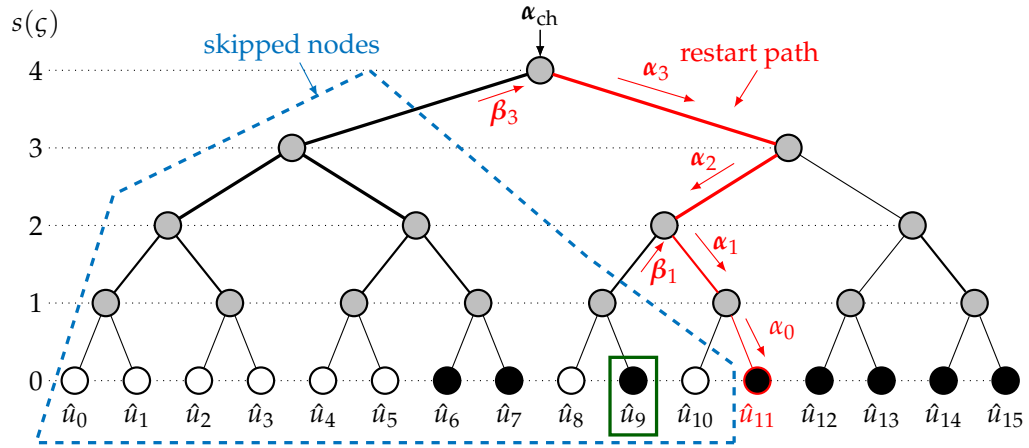



**Figure 2.** Additional trial for flip decoders embedding the GRM; the first bit flip is $\min(\varepsilon_t) = 9$ and the restart location is $\psi_t = 11$.

### *2.5. SCLF Decoding*

SCLF decoding is a combination of list decoding [3] and flip decoding [4,5]. It was first proposed in [6], but the baseline algorithm described in [8] is used in this paper. The least reliable bit positions, called *path-flipping locations*, are identified thanks to the flip metrics computed during the initial SCL trial and are stored in $\boldsymbol{\mathcal{B}}_{\text{flip}}$. For $a \in \boldsymbol{\mathcal{A}}_{\text{sort}}$, the flip metric $\text{FM}_a$ in [8] is computed using the $2L$ PMs after the sorting operation $\mathbf{PM}_{\text{sort}_a}$ as

$$\text{FM}_a = \ln\left(\frac{\sum_{l=0}^{L-1}\exp(-\text{PM}(l))}{\left(\sum_{l=0}^{L-1}\exp(-\text{PM}(l+L))\right)^p}\right), \tag{8}$$

where $\ln(\cdot)$ indicates the natural logarithm. The constant value $p \approx 1.0$ is found via simulations. A simplified metric of (8) has been proposed in [12] and is computed as follows:

$$\mathrm{FM}_a = -\mathrm{PM}(1) + p \cdot \mathrm{PM}(L+1), \tag{9}$$

where $\mathrm{PM}(1)$ represents the best metric while $\mathrm{PM}(L+1)$ is the best metric among the paths that will be discarded at index $a \in \boldsymbol{\mathcal{A}}_{\text{sort}}$. The use of (9), which has also been applied to partitioned SCLF (PSCLF) decoding [15], leads to a negligible error-correction performance degradation. Thus, the metric (9) is used in this work for SCLF decoding. During the initial trial, $|\boldsymbol{\mathcal{A}}_{\text{sort}}|$ flip metrics were computed. The metrics are sorted when computed while keeping track of the corresponding index $a \in \boldsymbol{\mathcal{A}}_{\text{sort}}$. The $T_{\max}$ smallest flip metrics are stored in the set $\boldsymbol{\mathcal{M}}_{\text{flip}}$. The $T_{\max}$ corresponding indices are stored in $\boldsymbol{\mathcal{B}}_{\text{flip}} = \{\varepsilon_1, \ldots, \varepsilon_{T_{\max}}\}$ where $\varepsilon_t$ corresponds to the path-flipping location in the $t$-th additional trial of SCLF.

In SCLF, $|\varepsilon_t| = 1$, i.e., the path-flipping only occurs once per additional trial. During the additional trial $t$, the standard SCL trial is performed until the path-flipping location $\varepsilon_t$ is reached. At this position, the $L$ worst paths are selected instead of the $L$ best ones, i.e., *path flipping* is performed [7]. Following the path flipping, the standard SCL decoding is resumed for the remaining part of the codeword.

### *2.6. Dynamic SCLF Decoding*

In [9], the DSCF decoding strategy is adapted to SCLF decoding, and DSCLF-$\omega$ decoding is proposed where several path-flipping locations potentially occur in an additional trial. The metric computations of the DSCLF and DSCF decoders are very similar. $\boldsymbol{\mathcal{B}}_{\text{flip}}$ is dynamically updated for each unsuccessful additional trial and is then composed of a set of path-flipping candidates. For the $t$-th additional trial, the path-flipping candidate $\varepsilon_t$ then becomes a set with $|\varepsilon_t| \leq \omega$. During the $t$-th additional trial, if $|\varepsilon_t| < \omega$, for all information indices $a > \max(\varepsilon_t)$, a flip metric is computed for the extended set $\varepsilon_t \cup a$ and the metric is calculated as follows [9]:

$$\mathrm{FM}_{\varepsilon_t \cup a} = \sum_{\epsilon \in \boldsymbol{\varepsilon}_t \cup a} \mathrm{FM}_\epsilon + \sum_{\substack{j \leq a \\ j \in \boldsymbol{\mathcal{A}}}} \mathcal{J}\left(\mathrm{FM}_j\right), \tag{10}$$

where $\mathrm{FM}_j$ corresponds to the simplified metric (9) and $\mathcal{J}(x)$ is calculated as follows [5,9]:

$$\mathcal{J}(x) = \frac{1}{z} \ln(1 + \exp(-z \cdot x)), \tag{11}$$

where $z$ is a constant value, at $0.0 < z \leq 1.0$, and is found via simulations. In [10], the piece-wise linear approximation function of (10) is proposed. In [5], a step-approximation function of (10) is derived for the DSCF decoder, and it is calculated as follows:

$$\mathcal{J}_{\text{step}}(x) = \begin{cases} 1.5\,, & \text{if } \ 0 \leq x \leq 5.0, \\ 0\,, & \text{otherwise.} \end{cases} \tag{12}$$

For DSCLF-$\omega$, $\omega \geq 2$, during an additional trial, several flip metrics (10) are potentially computed; as soon as one flip metric is computed, sorting is performed to update $\boldsymbol{\mathcal{B}}_{\text{flip}}$ and $\boldsymbol{\mathcal{M}}_{\text{flip}}$ of size $T_{\max}$. If the metric $\mathrm{FM}_{\varepsilon_t \cup a}$ exceeds the largest metric in $\boldsymbol{\mathcal{M}}_{\text{flip}}$, no sorting is required, and the set $\varepsilon_t \cup a$ is discarded. If not, the set $\varepsilon_t \cup a$ is inserted to $\boldsymbol{\mathcal{B}}_{\text{flip}}$ and $\mathrm{FM}_{\varepsilon_t \cup a}$ is inserted in $\boldsymbol{\mathcal{M}}_{\text{flip}}$ while maintaining the ascending order.

To the best of our knowledge, the use of (9) in the dynamic metric of DSCLF (10) using the approximation (12) has never been investigated in previous works. This combination makes a low complexity flip metric for the DSCLF decoding algorithm. However, the main

focus of this paper is on a restart mechanism to reduce the average execution time; hence, no more comments on this reduced complexity flip metric will be made later on.

## 3. Restart Mechanism for the List-Flip Decoder

In [17], the GRM showed great potential in reducing the complexity and the average execution time of flip-based decoders, i.e., SCF and DSCF-$\omega$, regardless of the baseline algorithm (with or without fast decoding of special nodes). The GRM only requires the storage of the candidate $\hat{\boldsymbol{u}}$ of the initial SC trial to enable the restart of an additional trial in any location $a \in \boldsymbol{\mathcal{A}}$. Next, the feasibility of embedding the GRM to the SCLF algorithm is discussed based on the memory overhead required. Then, a restart mechanism tailored for the list–flip algorithm is proposed and described.

### *3.1. Memory Requirements of List-Flip Algorithm*

First, the memory requirements of SCLF are studied. The SCL decoder uses a structure of $N$ bits to store $\boldsymbol{\mathcal{A}}$. The SCLF (DSCLF-$\omega$) decoder corresponds to an SCL decoder, used up to $T_{\max}$ times, an additional module to compute the flip metric (9) and (10), and a module to construct $\boldsymbol{\mathcal{B}}_{\text{flip}}$. The SCL decoder is considered to be parallel and is viewed as $L$ SC instances in terms of memory. The memory requirements, expressed in bits, for the LLRs and the partial sums of an SC instance is described in [17] and is

$$\Lambda_{\text{SC}} = \underbrace{Q_{\text{ch}} \cdot N}_{\boldsymbol{\alpha}_{\text{ch}}} + \underbrace{Q_{\text{int}} \cdot (N-1)}_{\boldsymbol{\alpha}_{\text{int}}} + \underbrace{2N-1}_{\hat{\boldsymbol{u}}+\boldsymbol{\beta}_{\text{int}}}, \tag{13}$$

where $Q_{\text{ch}}$ and $Q_{\text{int}}$ represent the quantization in bits for the channel LLRs $\boldsymbol{\alpha}_{\text{ch}}$ and the intermediate LLRs $\boldsymbol{\alpha}_{\text{int}}$. $N$ bits are used to store the current candidate $\hat{\boldsymbol{u}}$ and $N-1$ bits are needed for the intermediate partial sums $\boldsymbol{\beta}_{\text{int}}$. The SCL algorithm requires the storage of the structure $\mathbf{PM}_{\text{dup}}$, i.e.,

$$\Lambda_{\text{SCL}} = L \cdot \Lambda_{\text{SC}} + \underbrace{N}_{\boldsymbol{\mathcal{A}}} + \underbrace{2L \cdot Q_{\text{PM}}}_{\mathbf{PM}_{\text{dup}}}, \tag{14}$$

where $Q_{\text{PM}}$ is the quantization in bits used for the PMs.

Regarding the flip algorithm, DSCLF-$\omega$ requires the storage of $\boldsymbol{\mathcal{B}}_{\text{flip}}$ and $\boldsymbol{\mathcal{M}}_{\text{flip}}$ leading to

$$\Lambda_{\text{flip}} = \underbrace{T_{\max} \cdot Q_{\text{flip}}}_{\boldsymbol{\mathcal{M}}_{\text{flip}}} + \underbrace{\omega \cdot T_{\max}}_{\boldsymbol{\mathcal{B}}_{\text{flip}}}, \tag{15}$$

where $Q_{\text{flip}}$ is the quantization in bits used for the flip metric in (9) and (10). In bits, the memory of the list–flip decoder is

$$\Lambda_{\text{list-flip}} = \Lambda_{\text{SCL}} + \Lambda_{\text{flip}}. \tag{16}$$

### *3.2. Generalized Restart Mechanism*

The GRM is a restart mechanism that reduces the complexity of flip-based decoders while not affecting the error-correction performance [17]. The small memory overhead used to embed the GRM is one of the main advantages of this mechanism, allowing a reduction of 56.9% for the average execution time at the cost of 3.4% memory overhead for DSCF$-3$ decoding of a $(1024, 128+11)$ polar code. Another key advantage is that the restart location for an additional trial is always optimal, as the set of restart locations in

the GRM is defined as $\mathcal{R}_{\text{GRM}} = \mathcal{A}_{\text{sort}}$. Next, the estimation of the memory overhead is performed to embed the GRM to SCLF.

At the additional trial $1 \leq t \leq T_{\max}$, the GRM embedded into the flip decoder allows us to restart the decoding at the restart location $\psi_t \in \mathcal{A}_{\text{sort}}$, which is the next information-bit location after $\min(\varepsilon_t) \in \mathcal{A}_{\text{sort}}$. The restart path retrieves the intermediate partial sum $\boldsymbol{\beta}_{\text{int}}$ based on the $\hat{\boldsymbol{n}}$ of the initial SC trial. The intermediate LLRs, $\boldsymbol{\alpha}_{\text{int}}$, are retrieved with the channel LLRs, $\boldsymbol{\alpha}_{\text{ch}}$, and the intermediate partial sum $\boldsymbol{\beta}_{\text{int}}$. The sequence of $f$ and $g$ functions to perform during the restart path depends on the binary representation of $\psi_t$ as described in [17]. Hence, the restart path allows us to restore the status of the tree at position $\psi_t$ without performing the usual SC schedule. Next, we describe why the SCL cannot be restarted with only the storage of the final candidate $\hat{\boldsymbol{u}}_{N-1}$.

At a path-flipping location, the $L$ worst paths instead of the $L$ best continue the decoding. The $L$ worst path information, i.e., the sets $\mathbf{PM}_{\text{worst}_a}$ and $\hat{\boldsymbol{u}}_{\text{worst}_a}$, changes at each information bit $a \in \mathcal{A}_{\text{sort}}$ and becomes overwritten during the initial SCL trial. At a certain position, if the best $L$ paths out of the $2L$ are generated through less than $L$ parents, the information carried of some discarded paths is lost and cannot be retrieved with $\hat{\boldsymbol{u}}_{N-1}$.

As an example, SCLF with $L = 2$ decoding an $(8, 3+1)$ code with $\mathcal{A} = \{3,5,6,7\}$ ($\mathcal{A}_{\text{sort}} = \{5,6,7\}$) is shown. In this example, only the bit values at position $\mathcal{A}$ are shown. At position 3, SCL considers $L = 2$ paths, i.e., $\hat{\boldsymbol{u}}_3 = \{[0],[1]\}$. After duplication at position 5, we have $\hat{\boldsymbol{u}}_{\text{dup}_5} = \{[0,0],[0,1],[1,0],[1,1]\}$, which, after sorting, will be divided as $\hat{\boldsymbol{u}}_5 = \{[0,0],[0,1]\}$ and

$$\hat{\boldsymbol{u}}_{\text{worst}_5} = \{[1,0],[1,1]\}. \tag{17}$$

If we forward to the end with $\hat{\boldsymbol{u}}_7 = \{[0,0,1,0],[0,0,0,1]\}$, this is not able to pass the CRC. Only storing $\hat{\boldsymbol{u}}_7$ will not be enough to retrieve the worst paths at position 5 for example. Indeed, from $\hat{\boldsymbol{u}}_7$, we can state that $\{[0,0]\} \subset \hat{\boldsymbol{u}}_5$, meaning that $\hat{\boldsymbol{u}}_{\text{worst}_5} \subset \{[0,1],[1,0],[1,1]\}$, without knowing $\hat{\boldsymbol{u}}_{\text{worst}_5}$ (17), which is not enough information to perform the path flipping through the GRM.

Hence, in order to resume the decoding in any path-flipping location $\min(\varepsilon_t)$, the $L$ worst path information, i.e., $\hat{\boldsymbol{u}}_{\text{worst}_a}$ and $\mathbf{PM}_{\text{worst}_a}$, need to be stored for all $a \in \mathcal{A}_{\text{sort}}$. In order to store the path metric information, $L \times Q_{\text{PM}} \times |\mathcal{A}_{\text{sort}}|$ additional bits are required. In order to save memory for the storage of the $L$ worst partial candidate $\hat{\boldsymbol{u}}_{\text{worst}_a}$, only the message bits can be stored, i.e., for $a_i \in \mathcal{A}_{\text{sort}}$, $L \cdot i$ bits are stored instead of $L \cdot a_i$. Hence, the total memory requirement $\Lambda_{\text{GRM}}$ for embedding the GRM to the list–flip algorithms is

$$\Lambda_{\text{GRM}} = \sum_{i=\log_2(L)+1}^{K+r} (|\mathbf{PM}_{\text{worst}}| + L \cdot i), \tag{18}$$

$$\Lambda_{\text{GRM}} = L \times \left( Q_{\text{PM}} \cdot |\mathcal{A}_{\text{sort}}| + \sum_{i=\log_2(L)+1}^{K+r} i \right). \tag{19}$$

The memory requirement (19) grows with the code rate since $|\mathcal{A}_{\text{sort}}| = K + r - \log 2(L)$ will grow as well. Moreover, the memory will grow if the list size grows (19). Figure 3 depicts the memory sketch of the DSCLF-$\omega$ algorithm embedding the GRM.

The storage of the $L$ worst partial candidates leads to impractically large memory requirements. As an example, for the list–flip decoder of the $(1024,\ 512+16)$ polar code with $L = 2$, $\omega = 2$, and $T_{\max} = 50$ and the quantization scheme $Q_{\text{ch}} = 6$, $Q_{\text{int}} = 7$,

$Q_{\text{PM}} = 8$, and $Q_{\text{flip}} = 9$, derived from [16,23], the memory requirement of the decoder $\Lambda_{\text{list-flip}}$ and embedding the GRM $\Lambda_{\text{GRM}}$ are estimated to be

$$\Lambda_{\text{list-flip}} = 31760\,\text{bits}, \tag{20}$$

$$\Lambda_{\text{GRM}} = 287758\,\text{bits}. \tag{21}$$

Hence, the GRM induces a memory overhead of $\Delta_{\text{mem}}^{\text{GRM}} = 906\%$, while it was 6.1% for DSCF-2 with the same $T_{\max}$ [17].

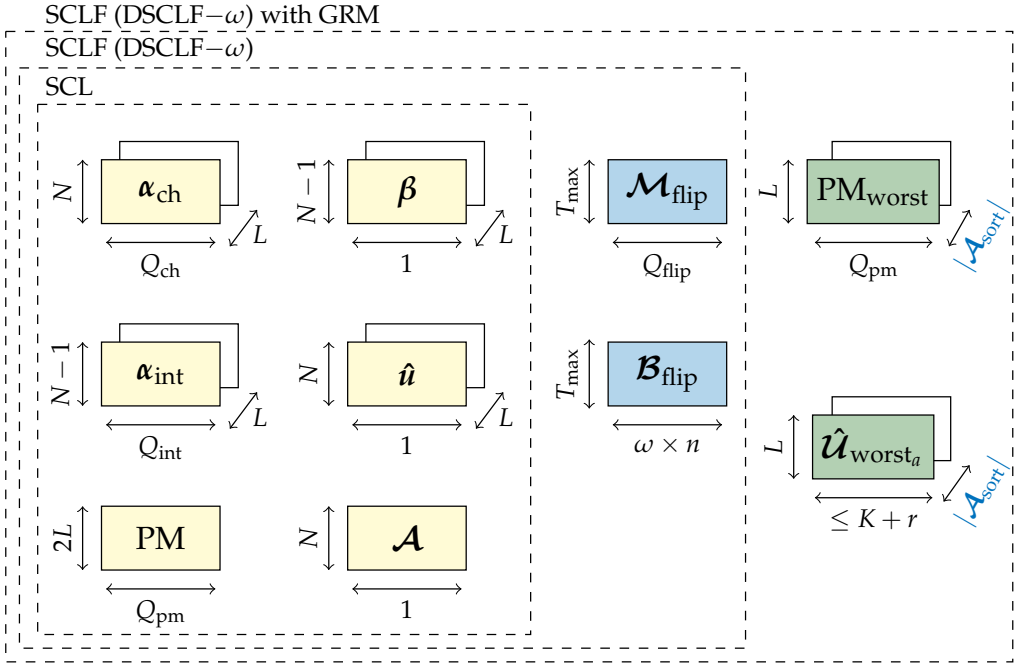



**Figure 3.** Memory sketch of SCLF with GRM.

The memory requirements to embed the GRM to list–flip algorithms, i.e., having $\mathcal{R}_{\text{GRM}} = \mathcal{A}_{\text{sort}}$, has been estimated with the conclusion that the GRM is unfeasible for the list–flip decoder. Next, the LLRM is proposed to tackle this issue.

### *3.3. Limited Location Restart Mechanism*

The LLRM is derived from the GRM and proposed as a way to skip part of the tree traversal of the SCL decoding. However, the set of restart locations $\mathcal{R}$ is predefined and its size $|\mathcal{R}|$ is very limited compared to $|\mathcal{R}_{\text{GRM}}|$. The effectiveness of the proposed approach is discussed in Section 5. First, the proposed LLRM is defined.

For an additional trial $t$, the list–flip decoder embedding the proposed LLRM skips non-negligible parts of the decoding tree by recovering the path information at the location $\psi_t \in \mathcal{R}$, where $\mathcal{R} = \{\mathcal{R}(1), \ldots, \mathcal{R}(R)\} \subset \mathcal{A}_{\text{sort}}$ represents the set of restart locations and $R = |\mathcal{R}|$ is the number of restart locations. It should be noted that $a_{\log_2(L)+1} \in \mathcal{A}_{\text{sort}}$ will always be included in $\mathcal{R}$. A notation variant of $\mathcal{R}$ is the subset $\mathcal{R}_{\mathcal{A}} = \{\mathcal{R}_{\mathcal{A}}(1), \ldots, \mathcal{R}_{\mathcal{A}}(R)\} \subset [\log_2(L) + 1, K + r]$, storing the index of each restart location in the information set $\mathcal{A}$, i.e., $\mathcal{R} = \left\{a_{\mathcal{R}_{\mathcal{A}}(1)}, \ldots, a_{\mathcal{R}_{\mathcal{A}}(R)}\right\}$.

By embedding the proposed LLRM, the SCL additional trial is modified by not restarting at position 0. The corresponding modified SCL trial is denoted by $\text{SCL}(\psi_t, \boldsymbol{\varepsilon}_t)$, indicating the restart location $\psi_t \in \mathcal{R}$ and the bit-flipping set $\boldsymbol{\varepsilon}_t$. The savings in terms of computations and decoding time come at the cost of storing all path information in the positions stated in $\mathcal{R}$ during the initial SCL trial. As described next, restart mechanisms exist [16,24] for SCF (DSCF-$\omega$) but on positions permitting an easy restart procedure.

In [16], the simplified restart mechanism (SRM) was proposed and can be seen as an instance of LLRM with one of the most simple sets of restart locations $\boldsymbol{\mathcal{R}}$. In the SRM, $R = 2$ restart locations are possible and include $\boldsymbol{\mathcal{R}}_{\text{SRM}} = \left\{0, \frac{N}{2}\right\}$ which allows a very simple restart path, where the restart always begins from the upper stage involving the channel LLRs $\boldsymbol{\alpha}_{\text{ch}}$. Only $\frac{N}{2}$ bits are required to apply this mechanism. However, the restart location $R_{\text{SRM}}(1) = 0$ does not allow a reduction in the complexity. Hence, only the restart location $R_{\text{SRM}}(2) = \frac{N}{2}$ reduces the decoding complexity. The authors in [24] derive the SRM to propose the set of restart locations $\boldsymbol{\mathcal{R}}_{\text{FRM}} = \left\{0, \frac{N}{2}, \frac{3N}{4}, \ldots, \frac{(N-1)N}{N} = N-1\right\}$. Despite being a more advanced set of restart locations, it comes at the cost of storing LLRs. As for the SRM, the restart locations are only on the right-hand side of the tree, limiting the application of the mechanism as discussed in [16].

The optimal restart mechanism is the GRM [17] as it combines the ability of restarting at any location in $\boldsymbol{\mathcal{A}}$, i.e., $\boldsymbol{\mathcal{R}}_{\text{GRM}} = \boldsymbol{\mathcal{A}}$, while not having to store the intermediate LLRs. This is possible thanks to the *generalized restart path*, which describes the algorithm that restores all intermediate LLRs and all partial sums on the basis of the binary representation of the restart location $\psi_t \in \boldsymbol{\mathcal{A}}$ and on the SC candidate $\hat{\boldsymbol{u}}$ of the initial trial.

By reusing the generalized restart path, the restart locations in the proposed LLRM will not be selected according to the ease of performing the restart path as in [16,24] but on picking the restart locations to improve, as much as possible, the average execution time reduction. The first consequence is that the first restart location for the proposed LLRM is not 0 but will be $\mathcal{R}(1) = a_{\log_2(L)+1} \in \boldsymbol{\mathcal{A}}_{\text{sort}}$ ($\mathcal{R}_{\mathcal{A}}(1) = \log_2(L) + 1$), representing the first information-bit location involving duplication in SCL. This will allow the avoidance of computations on the left-hand side of the decoding tree mostly composed of frozen bits. However, before finding the other restart locations, the proposed LLRM is described in more detail.

The memory overhead of the LLRM is first discussed. It is decomposed into two sets: the set $\boldsymbol{\mathcal{P}} = \{\boldsymbol{\mathcal{P}}(1), \ldots, \boldsymbol{\mathcal{P}}(R)\}$ that will store the relevant PMs and the set $\hat{\boldsymbol{\mathcal{M}}} = \{\hat{\boldsymbol{\mathcal{M}}}(1), \ldots, \hat{\boldsymbol{\mathcal{M}}}(R)\}$ that will store the relevant messages. During the initial SCL trial, when the decoding reaches the first restart location $\mathcal{R}(1) \in \boldsymbol{\mathcal{R}}$, the $2L$-sorted PMs are stored in $\boldsymbol{\mathcal{P}}(1)$, while the partial message candidates are stored in $\hat{\boldsymbol{\mathcal{M}}}(1)$. For the other restart locations, path information is also stored in the corresponding index of $\boldsymbol{\mathcal{P}}$ and $\hat{\boldsymbol{\mathcal{M}}}$. At the end of the initial SCL trial, the $R$ elements of $\boldsymbol{\mathcal{P}}$ and $\hat{\boldsymbol{\mathcal{M}}}$ gather the full path information at the restart locations stored in the $\boldsymbol{\mathcal{R}}$ of the $L$ best and $L$ worst paths.

The LLRM is activated whenever an additional trial is performed, i.e., if the CRC check fails for all $L$ candidates in $\hat{\boldsymbol{\mathcal{U}}}_{N-1}$. For $1 \leq t \leq T_{\text{max}}$, the $t$-th additional trial is defined by the path-flipping locations stored in $\boldsymbol{\mathcal{B}}_{\text{flip}}(t) = \varepsilon_t$. The first path-flipping location is defined by $\min(\varepsilon_t)$. The restart location $\psi_t \in \boldsymbol{\mathcal{R}}$ for the $t$-th additional trial corresponds to the closest element in $\boldsymbol{\mathcal{R}}$, verifying that

$$\psi_t \leq \min(\varepsilon_t). \tag{22}$$

The additional SCL trial restarting at $\psi_t$ and flipping at locations stored in $\varepsilon_t$ is noted for $\text{SCL}(\psi_t, \boldsymbol{\varepsilon}_t)$. Next, $\rho$ denotes the index in $\boldsymbol{\mathcal{R}}$, defining $\psi_t$, i.e., $\psi_t = \mathcal{R}(\rho)$.

If $\psi_t = \min(\varepsilon_t)$, the restart location is also a path-flipping location; hence, the worst $L$ paths are chosen to continue the decoding. Thus, the path metric structure $\mathbf{PM}_{\psi_t}$ of $\text{SCL}(\psi_t, \boldsymbol{\varepsilon}_t)$ is restored with the $L$ worst PMs stored in $\boldsymbol{\mathcal{P}}(\rho)$. Similarly, the partial candidate $\hat{\boldsymbol{\mathcal{U}}}_{\psi_t}$ is restored on the basis of the $L$ worst partial candidates stored in $\boldsymbol{\mathcal{P}}(\rho)$. If $\min(\varepsilon_t) > \psi_t$, the restart location precedes the path-flipping location; hence, the best $L$ paths are chosen to continue the decoding. Thus, the path metric structure $\mathbf{PM}_{\psi_t}$ of $\text{SCL}(\psi_t, \boldsymbol{\varepsilon}_t)$ is restored

with the best path metric stored in $\mathcal{P}(\rho)$. Similarly, the partial candidate $\hat{\mathcal{U}}_{\psi_t}$ is restored on the basis of the $L$ best partial candidates stored in $\mathcal{P}(\rho)$.

Once the structures $\mathbf{PM}_{\psi_t}$ and $\hat{\mathcal{U}}_{\psi_t}$ are correctly restored, for all $L$ paths, the intermediate values in the decoding tree, i.e., the restart path, are used to attain the next leaf in the decoding trees of the $L$ paths, i.e., $\psi_t + 1$, as shown in Figure 2. The binary representation of $\psi_t + 1$ describes the series of functions to be performed in the restart path. If $g$-functions are required, the partial sum $\boldsymbol{\beta}_{\text{int}}$ is restored through the $L$-independent restart paths. These functions allow us to accurately retrieve the intermediate LLRs $\boldsymbol{\alpha}_{\text{int}}$ for all $L$ paths. When the $L$ restart paths attain the leaf, if $\psi_t + 1 \in \mathcal{A}^c$, for all $L$ paths, $\hat{u}_{\psi_t+1} = 0$ and the $L$ path metrics are updated using (7). If $\psi_t + 1 \in \mathcal{A}$, $2L$ paths are generated, as well as $2L$ path metrics, and the decision on the $2L$ paths are taken based on the path metrics and the nature of position $\psi_t + 1$.

### *3.4. Example of the LLRM*

The modified SCL trial in SCLF embedding the LLRM, $\text{SCL}(\psi_t, \varepsilon_t)$, is explained with an example for the $(16, 10+1)$ code defined by $\mathcal{A} = \{2, 3, 5, 6, 7, 9, 11, 12, 13, 14, 15\}$ with $T_{\max} = 1$ and $L = 2$. The set of restart locations is set prior to decoding to $\mathcal{R} = \{3, 6, 9\}$. This example is shown in Figure 4. The initial trial starts with $\mathcal{P} = \{\emptyset, \emptyset, \emptyset\}$ and $\hat{\mathcal{M}} = \{\emptyset, \emptyset, \emptyset\}$. When reaching the first information bit $a_1 = 2$, the decoding considers both $\hat{u}_{a_1} = \{0, 1\}$, leading to the structure $\hat{\mathcal{U}}_{a_1} = \{[0, 0, 0], [0, 0, 1]\}$. The SCL does not need sorting since $L = 2$. The path metric structure is also updated as $\mathbf{PM}_{a_1} = \{\text{PM}(1), \text{PM}(2)\}$. The next bit $a_2 = 3 \in \mathcal{A}$ is also $\mathcal{R}(1)$. As explained in Section 3.3, $\hat{\mathcal{M}}(1)$ will store the $2L$ partial candidates, i.e., $\hat{\mathcal{M}}(1) = \{[0, 0], [0, 1], [1, 0], [1, 1]\}$, before the selection. Similarly, the $2L$ PMs in $\mathbf{PM}_{\text{sort}_{a1}}$ are stored in $\mathcal{P}(1)$. The same storage process happens when the decoding reaches $\mathcal{R}(2) = 6$ and $\mathcal{R}(3) = 9$.

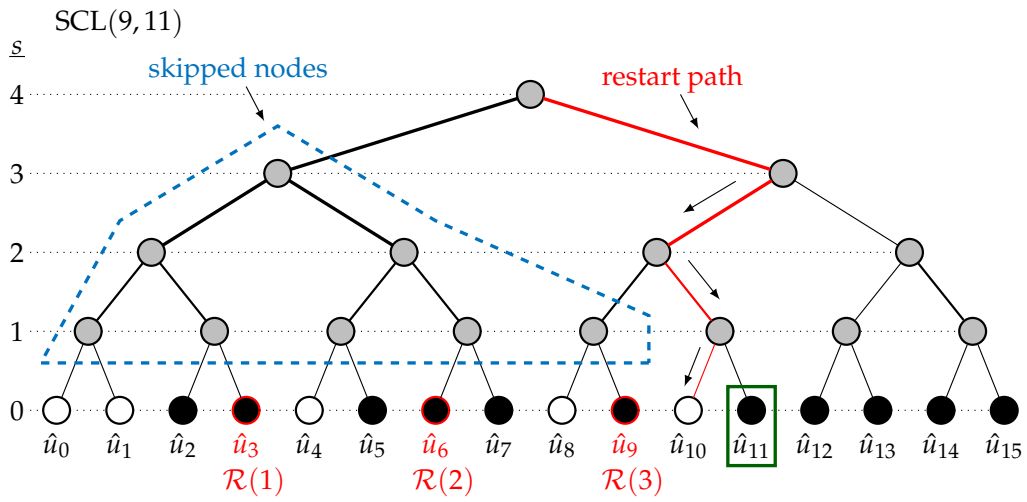



**Figure 4.** The modified trial SCL $(9, 11)$ with $\mathcal{R} = \{3, 6, 9\}$.

After the failure of the initial trial, the path-flipping location is considered to be $\varepsilon_1 = \{11\}$ in this example. The restart location $\psi_1$ is selected from $\mathcal{R}$ as being the closest to $\varepsilon_1$, verifying $\psi_1 \leq \varepsilon_1$. Hence, the restart location is $\psi_1 = \mathcal{R}(3) = 9$. The $\text{SCL}(\psi_1, \varepsilon_1)$ begins by restoring $\hat{\mathcal{U}}_9$ and $\mathbf{PM}_9$ with the $L$ best partial candidates in $\hat{\mathcal{M}}(3)$ and the $L$ best PMs in $\mathcal{P}(3)$ since $\psi_1 \neq \varepsilon_1$. For all $L$ paths, the restart path connecting the root of the tree at stage $s = n = 4$ with the leaf $\hat{u}_{10}$ at stage $s = 0$ is traversed, which allows us to retrieve the intermediate partial sum $\boldsymbol{\beta}_{\text{int}}$ and the intermediate LLR $\boldsymbol{\alpha}_{\text{int}}$. After traversing the restart path, all intermediate information is retrieved, allowing us to resume the standard course of SCL; since $10 \in \mathcal{A}^c$, only the $L$ path metrics are updated based on the decision LLR

computed at the end of the restart path for the $L$ paths. When $\varepsilon_1 = 11$ is reached, path flipping is performed, i.e., the $L$ worst paths are picked. When the last index is reached, the CRC is verified on all decoding paths. It either returns a decoding candidate verifying the CRC or returns a decoding failure since the decoding has reached its maximum number of additional trials $T_{\max} = 1$.

### *3.5. Memory Model*

In order to resume the decoding in one of the $R$ restart locations, the LLRM requires the storage of

1. the set of restart locations $\boldsymbol{\mathcal{R}}$;
2. the $2L$ path metric information $\boldsymbol{\mathcal{P}}$ on each restart location $\mathcal{R}(\rho) \in \boldsymbol{\mathcal{R}}$;
3. the $2L$ partial message candidate $\hat{\boldsymbol{\mathcal{M}}}$ on each restart location $\mathcal{R}(\rho) \in \boldsymbol{\mathcal{R}}$.

As a consequence, the memory overhead depends on $R$ but also on the positions of the restart locations, since the partial message is stored. For the $\rho$-th restart location, $2 \times L \times \mathcal{R}_{\mathcal{A}}(\rho)$ bits are required to store the partial messages. Moreover, the information on the $2L$ path is stored, as discussed in Section 3.3. The total memory overhead of the LLRM is denoted as $\Lambda_{\text{LLRM}}$ and is

$$\Lambda_{\text{LLRM}} = \underbrace{n \cdot R}_{\boldsymbol{\mathcal{R}}} + \underbrace{R \cdot 2L \cdot Q_{\text{PM}}}_{\boldsymbol{\mathcal{P}}} + \underbrace{\sum_{\rho=1}^{R} 2L\mathcal{R}_{\mathcal{A}}(\rho)}_{\hat{\boldsymbol{\mathcal{M}}}}. \tag{23}$$

The memory sketch of the SCLF decoder with the LLRM is provided in Figure 5. The green blocks correspond to the memory overhead (23) when embedding the proposed LLRM.

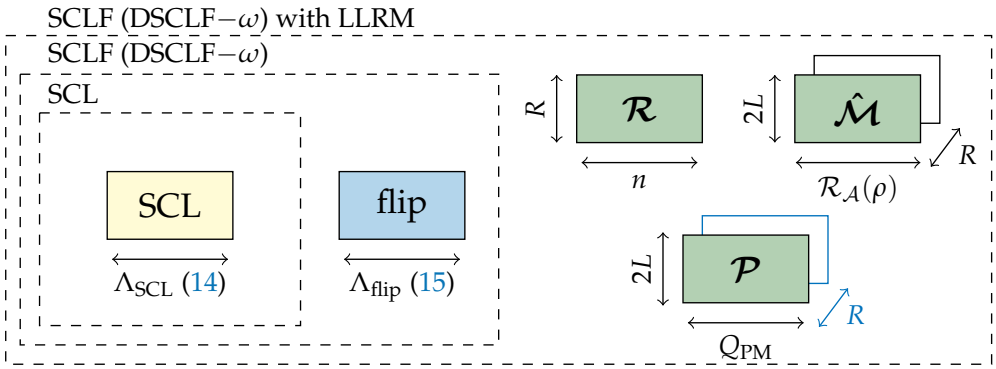



**Figure 5.** Memory sketch of SCLF with the LLRM.

## 4. Obtaining the Restart Locations

In this section, three designs for the set of restart locations are proposed. The choice of restart locations aims at improving the execution time reduction provided by the LLRM. Two are based on the structure of the polar code, while one is proposed based on offline simulations. For the latter, similarly to [17], offline simulations are used to retrieve the probability-mass function (PMF) of the first path-flipping candidate $\min(\varepsilon_t)$ for list–flip decoders. These statistics are then used to design the set $\boldsymbol{\mathcal{R}}$.

### *4.1. Structural Design*

Before describing the simulation-based design, two simple sets of restart locations are described. Both sets use the structure of the polar code. The first design of restart locations, noted as $\boldsymbol{\mathcal{R}}_{\text{divN}}$, equally divides the codeword into $R$ segments. Since the codeword is of length $N$, each segment is of size $\frac{N}{R}$. This particular design does not take into account

the information set and is constructed as $\mathcal{R}_{\text{divN}} = \left\{0, N/R, \ldots, \frac{N(R-1)}{R}\right\}$. For $R = 2$, the restart locations $\mathcal{R}_{\text{divN}} = \left\{0, \frac{N}{2}\right\}$ correspond to the restart locations of the SRM [16]. Restarting at 0 does not allow us to save any computations, which limits the effect of this mechanism for high-rate codes, as discussed in [16].

The second design, generating $\mathcal{R}_{\text{divK}}$, defines the restart locations on the basis of the information set $\mathcal{A}$. With the exception of the first restart location, any two successive restart locations are separated by $\left\lceil \frac{K+r}{R} \right\rceil$ information bits. The first restart location is $\mathcal{R}_{\text{divK}}(1) = \mathcal{A}_{\log_2(L)+1}$, since no restart is possible if $a \notin \mathcal{A}_{\text{sort}}$. All restart locations correspond to $\mathcal{R}_{\text{divK}} = \left\{a_{\log_2(L)+1}, a_{\lceil (K+r)/R \rceil}, \ldots, a_{\left\lceil \frac{(K+r)(R-1)}{R} \right\rceil}\right\}$ or $\mathcal{R}_{\mathcal{A}} = \left\{\log_2(L)+1, \left\lceil \frac{K+r}{R} \right\rceil, \ldots, \left\lceil \frac{(K+r)(R-1)}{R} \right\rceil\right\}$. For any additional trials, part of the computations will be avoided since $\mathcal{R}_{\text{divK}}(1) \neq 0$. This design pushes the restart locations towards the end of the codeword, allowing us to avoid many computations. However, these restart locations may not often used since $\min(\varepsilon_t)$ is expected to be close to the first information-bit indices.

### *4.2. Design Based on the First Path-Flipping Location*

In the following, the value of the first path-flipping location is denoted as $i_1 = \min(\varepsilon_t)$. The simulation-based design requires us to know the probability that $i_1$ is the first path-flipping location. Next, the probability that $a \in \mathcal{A}_{\text{sort}}$ is the first path-flipping location occurring during an additional trial of list–flip decoders is denoted by $\mathbb{P}(i_1 = a)$. The algorithm to obtain the PMF by simulation is described in Algorithm 1. A design signal-to-noise ratio (SNR) is chosen to match a desired FER. Moreover, the list size $L$ and the number of additional trials $T_{\max}$ are chosen to define the list–flip decoder. If the decoder performs an additional trial, the value of $i_1 = \min(\varepsilon_t) \in \mathcal{A}_{\text{sort}}$ is stored in a structure denoted as Occ. After transmitting $C$ codewords, with $C$ being large enough, a reliable probability distribution describing $\mathbb{P}(i_1 = a)$ for all $a \in \mathcal{A}_{\text{sort}}$ calculations is returned. This distribution is then used to choose the restart location $\mathcal{R}$.

**Algorithm 1** Obtaining the distribution of the first path-flipping candidates occurring in SCLF by simulation

**procedure** DISTR_BIT_FLIP_SCLF($C$, $\mathcal{A}$, $T_{\max}$, $L$, SNR)
  **for** $j = \log_2(L) + 1 : |\mathcal{A}|$ **do**
    $\mathbb{P}(i_1 = a_j) \leftarrow 0$ ▷ Initialize $\mathbb{P}, \forall a \in \mathcal{A}_{\text{sort}}$
    $\text{Occ}(j) \leftarrow 0$ ▷ Initialize counter $\forall a \in \mathcal{A}_{\text{sort}}$
  **end for**
  $\mathcal{T}_{\text{sim}} \leftarrow 0$ ▷ Total number of additional trials performed
  **for** $c = 0 : C - 1$ **do**
    $\boldsymbol{x} \leftarrow$ POLAR_ENCODING($\boldsymbol{u}$)
    $\boldsymbol{\alpha}_{\text{ch}} \leftarrow \text{AWGN}(\text{SNR}, \boldsymbol{x})$ ▷ Channel LLRs for the decoding
    $(\mathcal{B}_{\text{flip}}, t) \leftarrow \text{SCLF}(\boldsymbol{\alpha}_{\text{ch}}, \mathcal{A}, T_{\max})$ ▷ $\mathcal{B}_{\text{flip}}$: Set of flipping locations in SCLF, $t$: Number of additional trials performed
    **if** $t > 0$ **then** ▷ Initial trial has failed
      $\mathcal{T}_{\text{sim}} \leftarrow \mathcal{T}_{\text{sim}} + t$ ▷ Update the total number of additional trials
      **for** $\tau = 1 : t$ **do**
        $\varepsilon_\tau \leftarrow \mathcal{B}_{\text{flip}}(\tau)$
        $a_k \leftarrow \min(\varepsilon_\tau)$ ▷ Extract first bit-flipping $a_k \in \mathcal{A}_{\text{sort}}$
        $\text{Occ}(k) \leftarrow \text{Occ}(k) + 1$ ▷ Increase by 1 the occurence $i_1 = a_k$
      **end for**
    **end if**
  **end for**
  **for** $j = \log_2(L) + 1 : |\mathcal{A}|$ **do**
    $\mathbb{P}(i_1 = a_j) \leftarrow \text{Occ}(j)/\mathcal{T}_{\text{sim}}$ ▷ Estimate the probability-mass function $\mathbb{P}(i_1 = a_j)$
  **end for**
  **return** the distribution $\mathbb{P}(i_1 = a), \forall a \in \mathcal{A}_{\text{sort}}$
**end procedure**

Ultimately, the design consists of choosing the restart locations by equally dividing the distribution in segments sharing the same probability of having $\min(\varepsilon_t)$. The algorithm to obtain the set of restart locations $\boldsymbol{\mathcal{R}}_{\text{prob}}$ according to the PMF is described in Algorithm 2. The number of restart locations $R = |\boldsymbol{\mathcal{R}}|$ is selected in advance. The restart location $\mathcal{R}(\rho)$, $1 \leq \rho \leq R$, is the first location verifying $\mathbb{P}(i_1 = \mathcal{R}(\rho)) > \frac{(\rho-1)}{R}$. We note that $\mathcal{R}(1)$ is set to the smallest $a \in \boldsymbol{\mathcal{A}}_{\text{sort}}$, verifying $\mathbb{P}(i_1 = a) > 0$, and it is usually $a_{\log_2(L)+1}$.

**Algorithm 2** Design of $\boldsymbol{\mathcal{R}}$ with the distribution of the first path-flipping location

**procedure** DESIGN_RESTART_WITH_PMF($\mathbb{P}$, $R$, $\boldsymbol{\mathcal{A}}$)
  $sum\mathbb{P} \leftarrow 0$
  $\rho \leftarrow 1$
  **for** $j = \log_2(L) + 1 : K + r$ **do**
    $sum\mathbb{P} \leftarrow sum\mathbb{P} + \mathbb{P}\left(i_1 = a_j\right)$
    **if** $sum\mathbb{P} > \frac{\rho-1}{R}$ **then** ▷ Divide equally with resp. to $\mathbb{P}$
      $\mathcal{R}(\rho) \leftarrow a_j$
      $\rho \leftarrow \rho + 1$ ▷ Next restart location
    **end if**
    **if** $\rho > R$ **then**
      **return** $\boldsymbol{\mathcal{R}}$ ▷ Already $R$ restart locations in $\boldsymbol{\mathcal{R}}$
    **end if**
  **end for**
**end procedure**

### *4.3. Simulation Setup and Results*

For this analysis, a polar code of length $N = 1024$ with rate $1/2$ and a CRC of $r = 16$ bits is simulated over the AWGN channel with the BPSK modulation. Simulations are for a minimum of $C = 10^5$ codewords and are run until at least $10^3$ errors are observed. The target FER is $10^{-2}$ as in [17], which is obtained at $E_b/N_0 = 1.625$ dB. The DSCLF-$\omega$ decoder with $\omega = 3$ and $T_{\max} = 300$ is simulated. The list size is set to $L = 2$. The value of $T_{\max}$ was selected to achieve the optimal error-correction performance at the target FER.

The PMF is depicted in Figure 6. In the considered example, the set of restart locations is $R = 4$. Locations found by Algorithm 2 are denoted by $\boldsymbol{\mathcal{R}}_{\text{prob}}$ and equally divide the distributions according to their probabilities. For this example, the restart locations are $\boldsymbol{\mathcal{R}}_{\text{prob}} = \{191, 248, 370, 451\}$. Regarding the structural designs, the restart locations are $\boldsymbol{\mathcal{R}}_{\text{divN}} = \{0, 256, 512, 756\}$ and $\boldsymbol{\mathcal{R}}_{\text{divK}} = \{191, 499, 741, 890\}$.

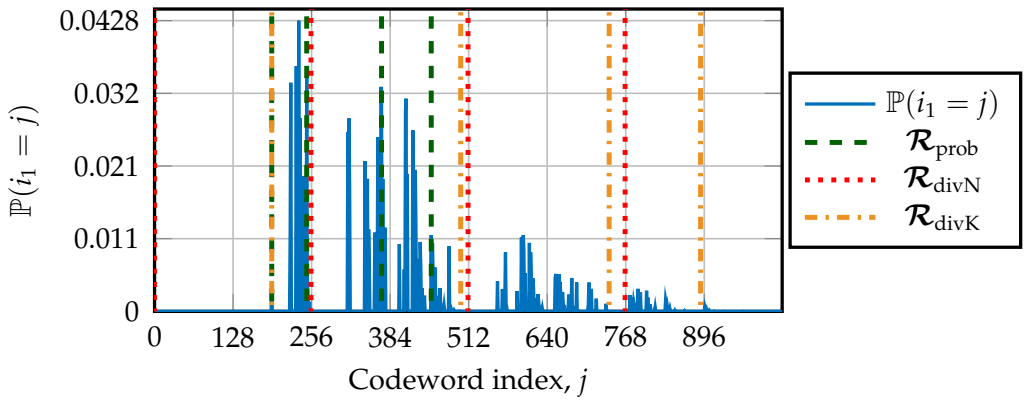



**Figure 6.** Distribution describing $\mathbb{P}(i_1 = j)$ under DSCLF-3 decoding for $(1024, 512 + 16)$ code. Vertical lines indicate restart locations $\boldsymbol{\mathcal{R}}$.

## 5. Simulation Results

Next, simulation results are provided. This section comprises error-correction performance, memory estimations, and average execution time reductions for list–flip decoders. The polar codes are constructed with a design SNR $E_b/N_0 = \{1.5, 2.0, 3.4\}$ dB for rates $R_{code} = \{1/4, 1/2, 3/4\}$, respectively. For all simulations, the number of restart locations is set to $R = 4$. The list size is selected as $L = 2$ to maintain a low-complexity list decoder. The maximum number of trials is selected as $T_{\max} = \{30, 50, 300\}$ for SCLF and DSCLF-$\omega$ decoders with $\omega = \{2, 3\}$, respectively. The BPSK modulation is used over an AWGN channel.

### *5.1. Error-Correction Performance*

The FER for SCLF and DSCLF-$\omega$ with various $\omega$ are shown in Figure 7 for polar codes of $N = 1024$ with rate $R_{code} = 1/2$ and in Figure 8 for polar codes with rate $R_{code} = 1/4$. For reference, standard SCL decoders with $L = 8$ and $L = 32$ are provided. The LLRM is derived from the GRM, a mechanism that does not affect error-correction performance [17]. Both figures show that the proposed LLRM does not affect the error-correction performance of the original SCLF and DSCLF decoders either. Moreover, higher order DSCLF decoders greatly increase the error-correction performance compared to SCLF. The performance of DSCLF-3 with $T_{\max} = 300$ is close to the performance of the SCL decoder with $L = 32$ for both rates and even matches for $R_{code} = 1/4$ and FER $= 10^{-3}$.

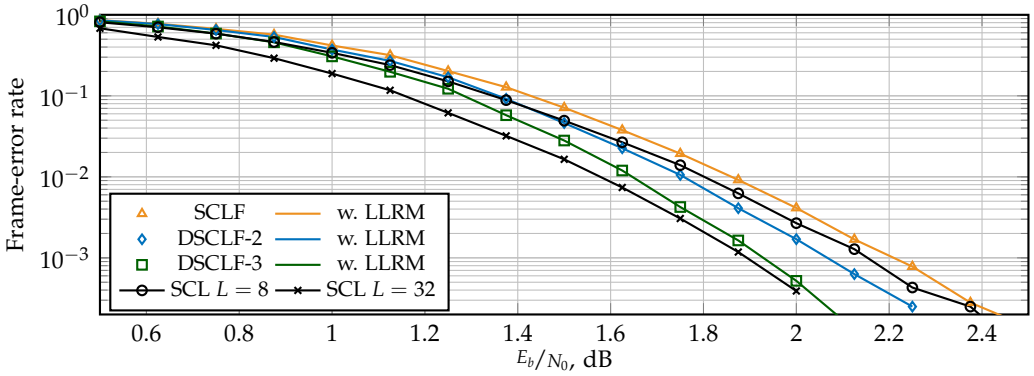



**Figure 7.** Error-correction performance of SCLF and DSCLF decoders with $L = 2$ for $R = 1/2$ codes and $N = 1024$.

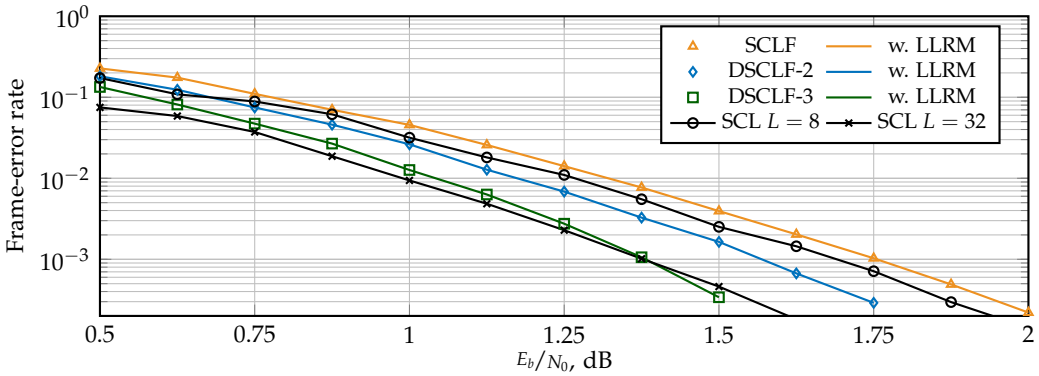



**Figure 8.** Error-correction performance of SCLF and DSCLF decoders with $L = 2$ for $R = 1/4$ codes and $N = 1024$.

### *5.2. Memory Estimations*

Memory requirements are estimated with (14), (15), (19), and (23), where the same quantization scheme as that of [16,23] is used. Hence, the blocks based on LLR values are

quantized by $Q_{\text{ch}} = 6$, $Q_{\text{int}} = 7$, $Q_{\text{flip}} = 7$ bits, and $Q_{\text{pm}} = 8$ bits, respectively. The sizes of these blocks also depend on the values of $N$, $L$, and $T_{\max}$. The number of restart locations $R = 4$ is selected for all estimations, which was similarly carried out in the example in Section 4. All sets of restart locations are considered. Table 1 provides the memory overhead, expressed in percent, induced by embedding the restart mechanism to the polar code decoder for code lengths $N = \{512, 1024, 2048\}$ and rates $R_{code} = \{1/4, 1/2, 3/4\}$. For $N = 1024$, the memory overhead is given for SCLF, DSCLF-2, and DSCLF-3, while for $N = \{512, 2048\}$, the memory overhead for DSCLF-3 is provided. By analyzing Table 1, the memory overhead to embed the LLRM depends on the set of restart locations $\boldsymbol{\mathcal{R}}$, since the location affects the size of $\hat{\boldsymbol{\mathcal{M}}}$ in (23). Moreover, for $\boldsymbol{\mathcal{R}}_{\text{divK}}$, the increase in the code rate increases the memory overhead since the size of $\hat{\boldsymbol{\mathcal{M}}}$ becomes larger. For a code rate of $R_{code} = 1/4$, the memory overhead is around 1.5% for $\boldsymbol{\mathcal{R}}_{\text{prob}}$ while it is 5% for $\boldsymbol{\mathcal{R}}_{\text{divK}}$. Meanwhile, for the GRM with $\boldsymbol{\mathcal{R}}_{\text{GRM}} = \boldsymbol{\mathcal{A}}_{\text{sort}}$ ($|\boldsymbol{\mathcal{A}}_{\text{sort}}| = K + r - \log_2(L)$), the memory overhead depends on the code rate and the code length but not the decoder. By doubling the code length $N$, the overhead approximately doubles as well. For $N = 512$ and rate $R_{code} = 3/4$, the overhead is 623.8% and 3204.5% for $N = 2048$. Hence, Table 1 shows that the overhead induced by the GRM makes it unfeasible. However, the overhead induced by the LLRM remains feasible for an implementation.

**Table 1.** Memory estimates and overhead for SCLF-based decoders with GRMs, LLRMs, and the original decoders.

| $N$ | $\omega$ | $T_{\max}$ | $\Lambda_{\text{list-flip}}$ | $R_{code}$ | $\Delta_{\text{mem}}^{\text{LLRM}}$ | | | $\Delta_{\text{mem}}^{\text{GRM}}$ |
|---|---|---|---|---|---|---|---|---|
| | | | | | $\boldsymbol{\mathcal{R}}_{\text{prob}}$ | $\boldsymbol{\mathcal{R}}_{\text{divK}}$ | $\boldsymbol{\mathcal{R}}_{\text{divN}}$ | |
| | | | **bits** | | % | % | % | % |
| 1024 | 1 | 30 | 32,270 | 1/4 | 1.8 | 5.5 | 2.1 | 235.0 |
| | | | | 1/2 | 7.7 | 10.3 | 6.1 | 875.1 |
| | | | | 3/4 | 4.6 | 15.0 | 12.0 | 1921.5 |
| 1024 | 2 | 50 | 33,110 | 1/4 | 2.2 | 5.5 | 2.1 | 229.0 |
| | | | | 1/2 | 2.2 | 10.0 | 6.0 | 853.0 |
| | | | | 3/4 | 2.7 | 14.7 | 11.7 | 1872.7 |
| 1024 | 3 | 300 | 42,860 | 1/4 | 1.5 | 4.2 | 1.6 | 176.9 |
| | | | | 1/2 | 1.4 | 7.7 | 4.6 | 658.9 |
| | | | | 3/4 | 1.8 | 11.3 | 11.7 | 1446.7 |
| 512 | 3 | 300 | 21,682 | 1/4 | 1.7 | 3.9 | 1.7 | 83.1 |
| | | | | 1/2 | 2.4 | 6.8 | 4.2 | 290.7 |
| | | | | 3/4 | 1.8 | 9.8 | 7.6 | 623.8 |
| 2048 | 3 | 300 | 75,504 | 1/4 | 1.5 | 4.4 | 1.6 | 374.0 |
| | | | | 1/2 | 1.6 | 8.5 | 4.9 | 1442.1 |
| | | | | 3/4 | 1.1 | 12.5 | 13.0 | 3204.5 |

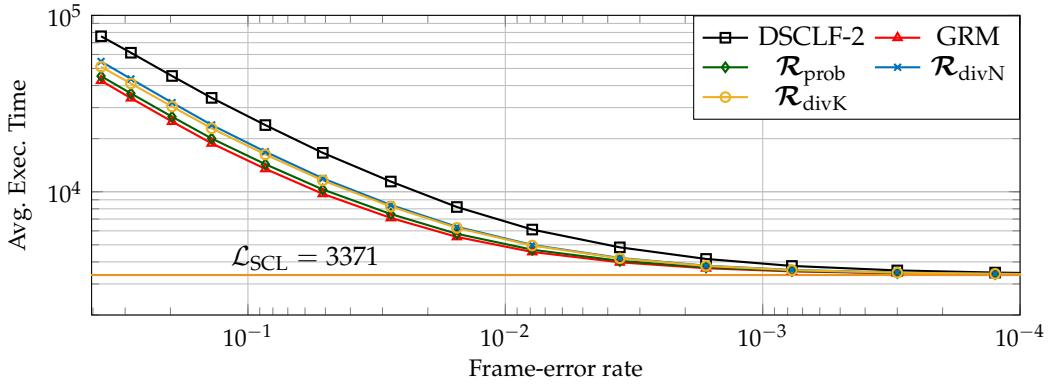
### *5.3. Average Execution Time Reduction Induced by the LLRM*

Next, with respect to the standard list–flip decoder (—□—), the reduction brought by the LLRM is estimated for various code lengths, code rates, and various restart locations designed according to simulations $\boldsymbol{\mathcal{R}}_{\text{prob}}$ (—◇—) or designed according to code properties such as $\boldsymbol{\mathcal{R}}_{\text{divN}}$ (—×—) and $\boldsymbol{\mathcal{R}}_{\text{divK}}$ (—○—). In order to compute it, the chosen architectural execution-time model is from [17]. Moreover, the number of processing elements, having an impact on the average execution time of decoders, is $P = 64$ as in [23,25]. Moreover,

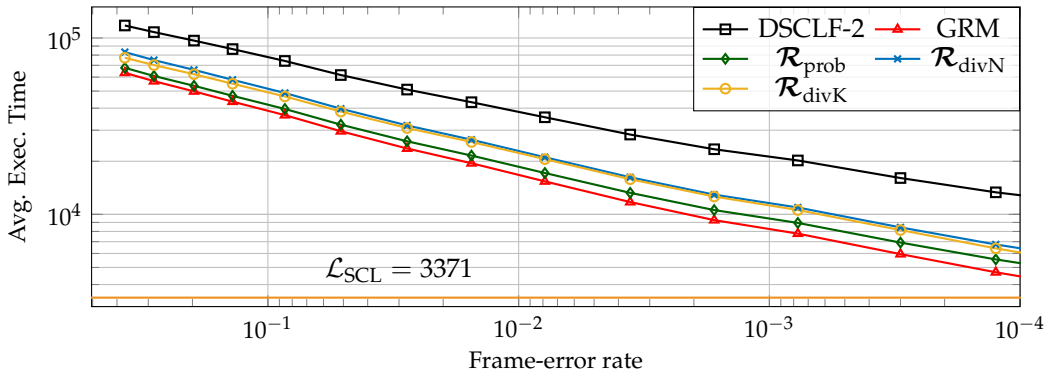
the latency of SCL is estimated as in [15], i.e., one clock cycle is added whenever the SCL encounters an information bit.

Two types of reduction will be discussed next, the average execution time reduction ($\Delta_{\mathbf{GRM}}$ and $\Delta_{\mathbf{LLRM}}$) and the average flip time reduction ($\Delta^{\mathbf{flip}}_{\mathbf{GRM}}$ and $\Delta^{\mathbf{flip}}_{\mathbf{LLRM}}$). The first consists of the reduction over all the simulated frames. The second consists of the reduction over the time spent during the flip part of the list–flip algorithm; since the flipping part is not required in all frames, this reduction is expected to be greater, i.e., $\Delta^{\mathbf{flip}}_{\mathbf{LLRM}} > \Delta_{\mathbf{LLRM}}$ ($\Delta^{\mathbf{flip}}_{\mathbf{GRM}} > \Delta_{\mathbf{GRM}}$). The reduction brought by the unfeasible GRM serves as a bound, i.e., $\Delta_{\mathbf{GRM}} > \Delta_{\mathbf{LLRM}}$ and $\Delta^{\mathbf{flip}}_{\mathbf{GRM}} > \Delta^{\mathbf{flip}}_{\mathbf{LLRM}}$, and the GRM reduction is shown in the figures as (—△—).

Figures 9 and 10 depict the average execution time over all frames (a) and over the flipping part (b) for the DSCLF-2 and DSCLF-3 decoders for the $(1024, 256 + 16)$ polar code. For both decoders, the reduction is clearly visible for all sets of restart locations. However, the set of restart locations $\boldsymbol{\mathcal{R}}_{\text{prob}}$ allows for a greater reduction and is closer to the optimal reduction retrieved with the GRM. The reduction over all frames reduces with the FER since the flipping part is required less often, and the mechanism permits a reduction only on the flipping part. In Figure 10a, the reduction with $\boldsymbol{\mathcal{R}}_{\text{prob}}$ at FER $= 0.1$ is $\Delta_{\mathbf{LLRM}} = 41.8\%$ while the reduction is $\Delta_{\mathbf{LLRM}} = 4.7\%$ at FER $= 10^{-4}$. If the focus is made on the flipping part solely, the reduction in the average flip time increases when the FER diminishes. In Figure 10b, the reduction with $\boldsymbol{\mathcal{R}}_{\text{prob}}$ at FER $= 0.1$ is $\Delta^{\mathbf{flip}}_{\mathbf{LLRM}} = 43.0\%$, while the reduction is $\Delta^{\mathbf{flip}}_{\mathbf{LLRM}} = 52.9\%$ at FER $= 10^{-4}$.

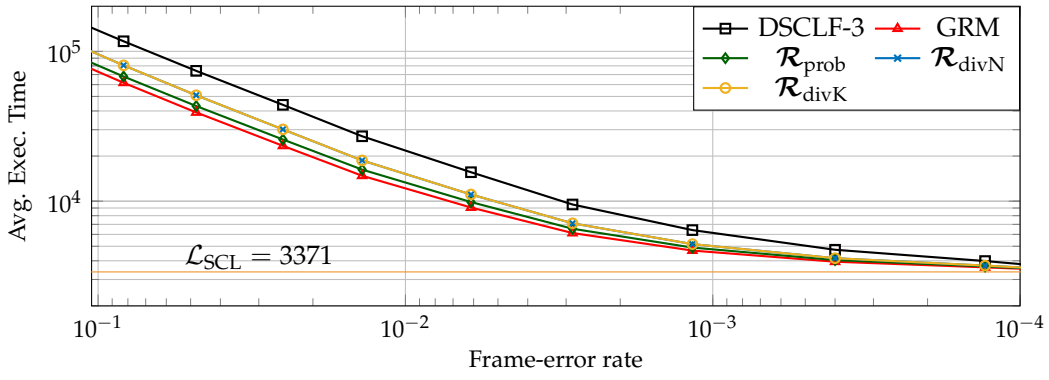



(**a**)

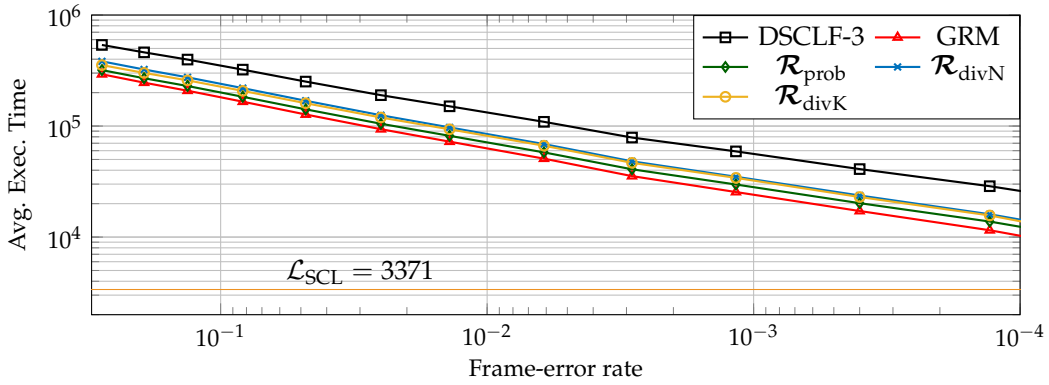



(**b**)

**Figure 9.** Execution time (**a**) and execution flip time (**b**) of DSCLF-2 decoder of polar codes with rate $1/4$ and $N = 1024$.




(**a**)




(**b**)

**Figure 10.** Execution time (**a**) and execution flip time (**b**) of DSCLF-3 decoders of polar codes with rate $1/4$ and $N = 1024$.

Table 2 recapitulates all aforementioned reductions with respect to the standard decoding for $N = \{512, 1024, 2048\}$ and code rates $K/N = \{1/4, 1/2, 3/4\}$. For $N = 1024$, the execution time reduction by embedding one of the restart mechanisms on SCLF and DSCLF-2 is also given. The reduction is computed at FER $= 10^{-2}$ as in [17]. Regardless of the decoders, code lengths, or code rates, the proposed design $\mathcal{R}_{\text{prob}}$ provides the best reduction. The reduction induced by the LLRM with $\mathcal{R}_{\text{prob}}$ is highlighted in boldface. As the code rate increases, the reduction reduces since the restart locations tend towards the beginning of the decoding tree, as explained in [16,17]. A reduction of 51.7% with respect to the DSCLF-2 algorithm is estimated for the code rate $1/4$ and $N = 1024$. For these parameters, the other designs provide a reduction of 40.7 and 42.1%. The smallest reduction observed is for DSCLF-2 with rate $3/4$ and $N = 1024$. The reduction is 7.0%, while the optimal reduction is 9.8%. On average, 3 to 7% is lost with respect to the reduction provided by the GRM.

**Table 2.** Execution time reduction by embedding the LLRM and the GRM to list–flip decoders of various polar codes at the FER of $10^{-2}$.

| $N$ | $\omega$ | $T_{\max}$ | $R_{code}$ | $E_b/N_0$ | $\Delta_{\text{GRM}}$ | $\Delta_{\text{LLRM}}$ (%) | | | $\Delta^{\text{flip}}_{\text{GRM}}$ | $\Delta^{\text{flip}}_{\text{LLRM}}$ (%) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | dB | (%) | $\mathcal{R}_{\text{prob}}$ | $\mathcal{R}_{\text{divN}}$ | $\mathcal{R}_{\text{divK}}$ | (%) | $\mathcal{R}_{\text{prob}}$ | $\mathcal{R}_{\text{divN}}$ | $\mathcal{R}_{\text{divK}}$ |
| 1024 | 1 | 30 | $1/4$ | 1.34 | 18.5 | 16.9 | 12.3 | 14.3 | 51.9 | 47.4 | 34.6 | 40.1 |
| | | | $1/2$ | 1.87 | 12.8 | 9.9 | 7.9 | 8.1 | 35.8 | 27.7 | 22.1 | 22.7 |
| | | | $3/4$ | 3.03 | 10.4 | 7.8 | 7.1 | 5.9 | 28.3 | 21.0 | 19.1 | 16.1 |

**Table 2.** *Cont.*

| $N$ | $\omega$ | $T_{\max}$ | $R_{code}$ | $E_b/N_0$ | $\Delta_{\mathrm{GRM}}$ | $\Delta_{\mathrm{LLRM}}$ (%) | | | $\Delta^{\mathrm{flip}}_{\mathrm{GRM}}$ | $\Delta^{\mathrm{flip}}_{\mathrm{LLRM}}$ (%) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | dB | (%) | $\mathcal{R}_{\mathrm{prob}}$ | $\mathcal{R}_{\mathrm{divN}}$ | $\mathcal{R}_{\mathrm{divK}}$ | (%) | $\mathcal{R}_{\mathrm{prob}}$ | $\mathcal{R}_{\mathrm{divN}}$ | $\mathcal{R}_{\mathrm{divK}}$ |
| 1024 | 2 | 50 | 1/4 | 1.21 | 25.4 | 23.1 | 18.2 | 18.9 | 56.7 | 51.7 | 40.7 | 42.1 |
| | | | 1/2 | 1.78 | 19.1 | 14.8 | 12.5 | 12.2 | 36.0 | 27.9 | 23.5 | 22.9 |
| | | | 3/4 | 3.00 | 9.8 | 7.0 | 5.8 | 5.1 | 23.1 | 16.5 | 13.7 | 12.0 |
| 1024 | 3 | 300 | 1/4 | 1.06 | 45.5 | 41.7 | 31.0 | 33.2 | 52.0 | 47.6 | 35.4 | 37.9 |
| | | | 1/2 | 1.66 | 27.8 | 22.2 | 16.7 | 17.2 | 32.2 | 25.7 | 19.3 | 19.9 |
| | | | 3/4 | 2.86 | 15.3 | 11.7 | 7.1 | 7.5 | 18.7 | 14.3 | 8.7 | 9.2 |
| 512 | 3 | 300 | 1/4 | 1.39 | 47.3 | 43.3 | 34.4 | 35.5 | 55.7 | 51.0 | 40.5 | 41.9 |
| | | | 1/2 | 1.92 | 30.8 | 28.1 | 20.0 | 20.3 | 35.5 | 32.4 | 23.1 | 23.4 |
| | | | 3/4 | 3.08 | 14.5 | 11.5 | 6.6 | 6.8 | 18.2 | 14.4 | 8.2 | 8.5 |
| 2048 | 3 | 300 | 1/4 | 0.84 | 44.1 | 38.0 | 31.8 | 26.7 | 55.2 | 47.6 | 39.8 | 33.4 |
| | | | 1/2 | 1.49 | 27.8 | 21.8 | 17.2 | 17.7 | 33.7 | 26.5 | 20.8 | 21.5 |
| | | | 3/4 | 2.73 | 12.5 | 8.9 | 4.7 | 4.8 | 15.5 | 11.0 | 5.8 | 6.0 |

## 6. Conclusions

In this paper, a restart mechanism is proposed for list–flip decoders of polar codes. We first show that an optimal mechanism is unfeasible due to a large memory overhead, which can increase up to 3755%. Thus, an limited-locations restart mechanism (LLRM) is proposed, allowing us to restart in predefined locations of the tree if an additional trial is performed. This mechanism requires the storage of path information at these locations. The choice of the restart locations influences the effectiveness of the LLRM. Three designs of restart locations are proposed and compared to each other. A thorough analysis is performed for various list–flip decoders, as well as various code lengths and code rates. The design requiring simulations achieves a reduction of 41.7% with respect to the DSCLF-3 decoding of the $(1024, 256 + 16)$ code at the cost of 1.5% memory overhead. This reduction is only 4% smaller than the optimal mechanism which comes at the cost of 177% memory overhead.

**Author Contributions:** Conceptualization, C.P., I.S., A.B.-S., and P.G.; methodology, C.P. and I.S.; software, C.P. and I.S; validation, C.P. and I.S.; writing—original draft preparation, C.P. and I.S.; writing—review and editing, C.P., A.B.-S., and P.G.; visualization, C.P. and I.S.; supervision, A.B.-S. and P.G.; project administration, P.G.; funding acquisition, P.G. All authors have read and agreed to the published version of the manuscript.

**Funding:** We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), RGPIN-2018-04284.

**Data Availability Statement:** The original contributions presented in the study are included in the article; further inquiries can be directed to the corresponding authors.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| AWGN | additive white Gaussian noise |
| BPS | binary phase-shift keying |
| CA | CRC-aided |
| CRC | cyclic redundancy check |
| DSCF | dynamic successive-cancellation flip |

| | |
|---|---|
| DSCLF | dynamic successive-cancellation list flip |
| eMBB | enhanced mobile broadband |
| FER | frame-error rate |
| GRM | generalized restart mechanism |
| LLR | log-likelihood ratio |
| LLRM | limited location restart mechanism |
| PM | path metric |
| PMF | probability-mass function |
| PS | partial sum |
| PSCLF | successive-cancellation list flip |
| RHS | right-hand side |
| SC | successive cancellation |
| SCL | successive-cancellation list |
| SCLF | successive-cancellation list flip |
| SCF | successive-cancellation flip |
| SCF | simplified restart mechanism |
| SNR | signal-to-noise ratio |

## References


1. Arıkan, E. Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels. *IEEE Trans. Inf. Theory* **2009**, *55*, 3051–3073. [CrossRef]
2. 3GPP. *Multiplexing and Channel Coding*; Technical Report TS 38.212; Release 16.5; 2018. Available online: https://www.etsi.org/deliver/etsi_ts/138200_138299/138212/16.05.00_60/ts_138212v160500p.pdf (accessed on 9 March 2025).
3. Tal, I.; Vardy, A. List decoding of polar codes. *IEEE Trans. Inf. Theory* **2015**, *61*, 2213–2226. [CrossRef]
4. Afisiadis, O.; Balatsoukas-Stimming, A.; Burg, A. A low-complexity improved successive cancellation decoder for polar codes. In Proceedings of the 2014 48th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 2–5 November 2014; pp. 2116–2120. [CrossRef]
5. Chandesris, L.; Savin, V.; Declercq, D. Dynamic-SCFlip Decoding of Polar Codes. *IEEE Trans. Commun.* **2018**, *66*, 2333–2345. [CrossRef]
6. Yu, Y.; Pan, Z.; Liu, N.; You, X. Successive Cancellation List Bit-flip Decoder for Polar Codes. In Proceedings of the 2018 10th International Conference on Wireless Communications and Signal Processing (WCSP), Hangzhou, China, 18–20 October 2018; pp. 1–6. [CrossRef]
7. Cheng, F.; Liu, A.; Zhang, Y.; Ren, J. Bit-Flip Algorithm for Successive Cancellation List Decoder of Polar Codes. *IEEE Access* **2019**, *7*, 58346–58352. [CrossRef]
8. Pan, Y.H.; Wang, C.H.; Ueng, Y.L. Generalized SCL-Flip Decoding of Polar Codes. In Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM), Taipei, Taiwan, 7–11 December 2020; pp. 1–6. [CrossRef]
9. Shen, Y.; Balatsoukas-Stimming, A.; You, X.; Zhang, C.; Burg, A.P. Dynamic SCL Decoder With Path-Flipping for 5G Polar Codes. *IEEE Wireless Commun. Lett.* **2022**, *11*, 391–395. [CrossRef]
10. Lv, H.; Yin, H.; Yang, Z.; Wang, Y.; Dai, J. Adaptive List Flip Decoder for Polar Codes with High-Order Error Correction Capability and a Simplified Flip Metric. *Entropy* **2022**, *24*, 1806. [CrossRef] [PubMed]
11. Li, J.; Zhou, L.; Li, Z.; Gao, W.; Ji, R.; Zhu, J.; Liu, Z. Deep Learning-Assisted Adaptive Dynamic-SCLF Decoding of Polar Codes. *IEEE Trans. Cogn. Commun. Netw.* **2024**, *10*, 836–851. [CrossRef]
12. Ivanov, F.; Morishnik, V.; Krouk, E. Improved generalized successive cancellation list flip decoder of polar codes with fast decoding of special nodes. *J. Commun. Netw.* **2021**, *23*, 417–432. [CrossRef]
13. Doan, N.; Hashemi, S.; Gross, W. Fast Successive-Cancellation List Flip Decoding of Polar Codes. *IEEE Access* **2022**, *10*, 5568–5584. [CrossRef]
14. Wang, Y.; Qiu, S.; Chen, L.; Wang, Q.; Zhang, Y.; Liu, C.; Xing, Z. A Low-Latency Successive Cancellation Hybrid Decoder for Convolutional Polar Codes. In Proceedings of the 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 5105–5109. [CrossRef]
15. Pillet, C.; Sagitov, I.; Domer, G.; Giard, P. Partitioned Successive-Cancellation List Flip Decoding of Polar Codes. In Proceedings of the 2024 IEEE Workshop on Signal Processing Systems (SiPS), Cambridge, MA, USA, 4–6 November 2024; pp. 19–24. [CrossRef]
16. Sagitov, I.; Pillet, C.; Balatsoukas-Stimming, A.; Giard, P. Successive-Cancellation Flip Decoding of Polar Code with a Simplified Restart Mechanism. In Proceedings of the 2023 IEEE Wireless Communications and Networking Conference (WCNC), Glasgow, Scotland, 26–29 March 2023; pp. 1–6. [CrossRef]

17. Sagitov, I.; Pillet, C.; Balatsoukas-Stimming, A.; Giard, P. Generalized Restart Mechanism for Successive-Cancellation Flip Decoding of Polar Codes. *J. Signal Process. Syst.* **2025** . [CrossRef]
18. Tal, I.; Vardy, A. How to Construct Polar Codes. *IEEE Trans. Inf. Theory* **2013**, *59*, 6562–6582. [CrossRef]
19. Alamdar-Yazdi, A.; Kschischang, F.R. A Simplified Successive-Cancellation Decoder for Polar Codes. *IEEE Commun. Lett.* **2011**, *15*, 1378–1380. [CrossRef]
20. Leroux, C.; Tal, I.; Vardy, A.; Gross, W.J. Hardware architectures for successive cancellation decoding of polar codes. In Proceedings of the 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Prague, Czech Republic, 22–27 May 2011; pp. 1665–1668. [CrossRef]
21. Balatsoukas-Stimming, A.; Parizi, M.; Burg, A. LLR-Based Successive Cancellation List Decoding of Polar Codes. *IEEE Trans. Signal Process.* **2015**, *63*, 5165–5179. [CrossRef]
22. Sarkis, G.; Giard, P.; Vardy, A.; Thibeault, C.; Gross, W.J. Fast Polar Decoders: Algorithm and Implementation. *IEEE J. Sel. Areas Commun.* **2014**, *32*, 946–957. [CrossRef]
23. Ercan, F.; Tonnellier, T.; Doan, N.; Gross, W.J. Practical Dynamic SC-Flip Polar Decoders: Algorithm and Implementation. *IEEE Trans. Signal Process.* **2020**, *68*, 5441–5456. [CrossRef]
24. Xiyue, X.; Meilin, H.; Rui, G. Flexible Restart Mechanism for Successive Cancellation Flip Decoding of Polar Codes. *IEEE Commun. Lett.* **2024**, *28*, 2459–2463. [CrossRef]
25. Giard, P.; Balatsoukas-Stimming, A.; Müller, T.C.; Bonetti, A.; Thibeault, C.; Gross, W.J.; Flatresse, P.; Burg, A. PolarBear: A 28-nm FD-SOI ASIC for Decoding of Polar Codes. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2017**, *7*, 616–629. [CrossRef]