# Memory-Efficient Pipeline for Large-Scale Linear Code Classification

GPT-5.2, Gemini 3 Pro, Claude Sonnet 4.5, Liner AI, NotebookLM, ChatEXAONE


| **Sohyun Lee** | **Chaerin Yu** | **Woohyuk Jung** | **Junwon Lee** |
|---|---|---|---|
| Myong-Ji.univ | Myong-Ji.univ | Dan-kook.univ | Chung-ang.univ |
| Yongin, Korea | Yongin, Korea | Yongin, Korea | Seoul, Korea |
| leesh03x@mju.ac.kr | chaelin1097@mju.ac.kr | joh1791@dankook.ac.kr | ljw11332@cau.ac.kr |



## Absract

This study addresses the linear code classification problem from a projective geometric perspective, reformulating it as an integer linear programming (ILP) problem based on lattice point enumeration. To overcome the computational complexity and memory bottlenecks inherent in large-scale parameter exploration, we propose a hierarchical verification framework named MES (Memory Efficiency Stability). Conventional ILP-based approaches often encounter significant limitations where the explosion of search nodes during infeasibility proofs leads to excessive memory consumption. To mitigate this, the MES framework adopts a "proof-preserving" structure designed to postpone high-cost ILP calls. Specifically, it introduces a pre-search phase incorporating theoretical pruning based on the Griesmer bound and Pless power moments (Phase 0.5) and linear relaxation (Phase 0). During the search phase, an orbital branching technique (Phase 1.5) leveraging projective geometric symmetry is applied to structurally suppress redundant isomorphic branches. Experimental results demonstrate that the proposed MES algorithm reduces memory usage by up to 27.9% compared to the baseline, significantly enhancing operational stability in high-dimensional problem sets. This research provides a practical methodology for large-scale linear code classification and existence determination, even within resource-constrained computing environments.


## 1. Introduction

From a projective geometric perspective, a linear code corresponds to a multiset of points in the projective space PG(k−1,q), and this representation provides a basis for geometrically analyzing the code's structure and weight distribution. Leveraging this, the linear code classification problem can be reformulated as a lattice point enumeration problem by introducing integer variables $x_i \in \mathbb{Z}_{\geq 0}$ that represent the selection count for each point $P_i$ in the point set P. In this context, the length and weight constraints are expressed as follows: $\sum P x_i = n$, $\forall H \in H$, $\sum_{P_i \in H} x_i \in \{n - w \mid w \in W\}$.

While such a formulation transforms the code classification problem into an integer solution search problem, it entails computational difficulties as the search space expands rapidly with increasing

parameters. In particular, the problem of extending an existing [n,k]q code into an [n+r,k+1]q code and classifying the extended candidates up to isomorphism is known to be a theoretically significant yet computationally formidable task. During the extension process, the number of candidate solutions increases exponentially, including numerous invalid solutions in the search process. This creates a structural paradox where the cost of proving that an extension is impossible exceeds the cost of finding actual valid solutions.

To alleviate the computational costs associated with infeasibility proofs and the memory bottleneck issues occurring in conventional linear code classification, this study proposes a hierarchical verification and search framework named Memory Efficiency Stability (MES). While maintaining linear code classification as an integer solution search problem based on lattice point enumeration, MES adopts a proof-preserving structure that postpones high-cost ILP calls as much as possible. Instead, it sequentially applies theoretical necessary conditions, linear relaxation, and symmetry analysis in the preceding stages. Through this approach, the framework aims to eliminate explicitly infeasible cases before the search phase and structurally suppress redundant branching caused by isomorphism during the search, thereby simultaneously ensuring computational efficiency and memory stability.

The structure of this paper is as follows. In Section 2, we strictly define the linear code classification problem addressed in this study and summarize the projective geometric representation, isomorphism, search space, and underlying assumptions. Section 3 mathematically justifies the pre-search infeasibility determination conditions, such as the Griesmer bound, Pless power moments, and orbit decomposition based on group actions, as well as the principles of symmetry-based search reduction. In Section 4, the entire procedure and core design principles of the MES algorithm are presented based on these theoretical foundations. Section 5 describes the experimental setup and benchmark parameters, and Section 6 reports the results of the performance and memory usage comparison with the baseline technique.

(i) Proposed Memory Stable Layer Discovery Framework for Linear Code Classification
(ii) Introducing a new pruning strategy to structurally reduce the cost of proving infeasibility
(iii) It is about providing a methodology that can practically solve the problem of large-scale code classification while maintaining the Proof-preserving property.

## 2. Prior research

The problem of linear code classification has been a major research topic in computational coding theory for a long time, and in recent years, an approach combining lattice point enumeration and integer linear programming has been actively studied. This study takes the classification framework proposed in Kurz (2019) and Kurz (2024) as an important prior study, in particular.

### 2.1 Lattice-point enumeration-based code extension

Kurz (2019)[1] proposed a lattice-point enumeration-based expansion and classification algorithm for linear codes on finite fields. This study presented the existing [n,k]q in a reduced form with the [n+r,k+1]q symbol, and introduced a filtering strategy to remove isomorphic symbols that arise during the expansion process. In addition, the connection between the code structure and geometric representation was clarified by geometrically analyzing the weights of the codes by utilizing the point distribution and hyperplane structure in projected space. This study experimentally proved that lattice-point enumeration-based classification can also be applied to large-scale code problems.

[1] S. Kurz. LINCODE — COMPUTER CLASSIFICATION OF LINEAR CODES. arXiv preprint arXiv:1912.09357, 2019.

### 2.2 An ILP-based approach to feasibility analysis

Kurz (2024)[2] proposed a pipeline with a Phase 0–1–2 structure to determine early on whether linear codes admit feasible extensions under given weight constraints. This approach adopts a strategy of performing ILP-based feasibility checks at the pre-grid point enumeration stage to eliminate non- extendable candidates early in the exploration. In particular, by modeling the discontinuous weight spectrum in a linear constraint formula using binary and slack variables, it is designed to eliminate unacceptable weight intervals early. However, in some instances, it has been reported that these ILP steps cause rapid expansion of the branch-and-bound navigation tree and increased memory usage, which can act as a major bottleneck in the overall computational time.

### 2.3 Previous Research Considerations

Existing studies have greatly expanded the computational possibilities of linear code classification by combining lattice point enumeration with ILP, but it still has the following limitations. First, the ILP feasibility verification to eliminate infeasible instances early can be the biggest computational bottleneck. Second, symmetry and group action are not fully utilized at the beginning of exploration, so the effect of reducing search space is limited. Third, as the frequency of ILP calls increases in large instances, memory usage and computational stability decreases. This limitation poses a structural problem: "The cost of proving impossible can be greater than the actual cost of searching," raising the need for a more efficient and reliable search framework.

### 2.4 Research methods

To solve the above problems, this study proposes a hierarchical approach that minimizes high-cost ILP calls and reduces the search space in advance while keeping the linear code classification problem as a lattice-point enumeration-based integer as a search problem. The proposed method aims to simultaneously improve navigation efficiency and computational stability by step-by-step utilizing theoretical requirements, linear relaxation, and symmetry analysis. Specific problem formulation, algorithm structure, and mathematical modeling are described in detail in the following sections (problem definition and assumption, algorithm design, and mathematical formulation).

## 3. Problem Description & Assumptions

### 3.1 Problem Setting and Objective

This study addresses the problem of determining the exitence [n,k]q linear codes over a finite field Fq satisfy a weight constraint W, classifying such codes up to isomorphism. Kurz (2024)'s ILP-based approach faces a significant bottleneck in infeasibility proofs, where the explosion of search nodes leads to excessive computational costs. To address this, we propose a proof-preserving verification framework that introduces Phase 0.5 (theoretical pruning) and Phase 1.5 (symmetry analysis and branching). The goal of this paper is to optimize the existing Phase 0–1–2 pipeline, currently dominated by high-cost ILP, by presenting an algorithm that ensures memory efficiency and operational stability in large-scale execution environments. We refer to this hierarchical framework as the MES (Memory Efficiency Stability) pipeline.

### 3.2 Formal Definitions and Classification Problem

A linear code $C \leq \mathrm{F}_q^n$ is regarded as a multiset defined by a multiplicity function $m: P \rightarrow Z \geq 0$ over the point set $P$ of the projective geometry $PG(k-1, q)$. The column vectors of the generator matrix $G$ correspond to

[2] S. Kurz. Computer classification of linear codes based on lattice point enumeration and integer linear programming. arXiv preprint arXiv:2403.17509, 2024.

points in the projective space, and the weight of a codecword *w(H*) is expressed by the following linear relationship:

$$w(H) = n - \Sigma P \in H, m(P) \in W \quad (1)$$

The input for the classification problem consists of parameters n, k, q and the weight constraint *W*, while the output is the set of unique non-equivalent codes *Cunique* that satisfy these conditions.

### 3.3 Search Space, Equivalence, and Symmetry Reduction

Phase 1.5 (Symmetry Analysis) is introduced to mitigate the search space expansion caused by monomial equivalence. This phase serves as the foundation for the orbital branching technique, which utilizes the automorphism group to partition points into orbits and selects only representative elements for branching. This symmetry-based branching is specifically designed to eliminate redundant equivalent branches without excluding any isomorphism class from the search. Consequently, by pruning unnecessary duplicate paths through symmetry, the framework strictly preserves the completeness of the search, ensuring that the final existence of determination and classification results remain entirely unaffected.

### 3.4 Assumptions

This study follows the standard setting of projective code assumptions and weight constraints of integer intervals. In Phase 0.5 (Theoretical Pruning), only necessary conditions, such as the Griesmer bound and Pless power moments, are inspected to comply with the proof-preserving principle, ensuring that no valid solution is omitted. Unlike certain empirical heuristic approaches, the filtering steps in this study are based on mathematically rigorous requirements. This design aims to determine the fundamental limits of the search space in advance, grounded in mathematical justification rather than simple computational acceleration.

## 4. Mathematical Modeling

This section defines the hierarchical validation pipeline used in this study as a pure set of mathematical constraints. The linear code expansion ILP model itself presented in previous studies is used as it is, but this study systematically organizes the dictionary impossibility determination conditions that can be applied in the previous stage by separating them from the formula level.

This study refers to the mathematical modeling of the code expansion process presented by Kurz (2024). This study describes the isomorphic overlap and symmetrical structure that occurs in linear code expansion from the perspective of group action. Based on these formulas, this study mathematically d the applicable impossibility discriminant condition and the search space reduction constraint in the pre-exploration phase.

All conditions set forth in this section are necessary conditions for the possibility of the existence of a definitive extension and in no event exclude possible harm. In other words, the purpose of this study is not to replace search, but to build a hierarchical mathematical model that eliminates unnecessary search in advance.

### 4.1 Length constraints by Griesmer boundaries

In order for a linear code [n,k]q on a finite body $\mathrm{F_q}$ to have a minimum distance, it must satisfy the Griesmer boundary.

$$n \geq \sum_{i=0}^{k-1} \left\lceil \frac{d}{q^i} \right\rceil \quad (2)$$

In this study, given a set of weights, we use a minimum non-zero weight, defined as. If the above inequality is not true, there can be no code for the parameter, which can be determined mathematically immediately before any integer scheme model or search.

$$W \in N\, d := \min\, W[n, k]_q \tag{3}$$

**4.2 Weight Distribution Constraints According to Pless Power Moments**

Let's say {Aw}nw=0 the weight distribution of [n,k]q linear codes. Pless power moments provide a series of linear relationships that this distribution must satisfy, and, the first and second moments are given as follows:

$$\sum_{w=0}^{n} A_w = q^k, \sum_{w=0}^{n} w\, A_w = q^{k-1}(q-1)n \tag{4}$$

In this study, since the weight set is restricted to W, the additional condition W Aw =0 for all w∉W∪{0} is imposed. Consequently, the above moment equations reduce to a system of linear equations defined over the restricted variable set {Aw :w∈W}. If this system admits no nonnegative integer solution, then no linear code satisfying the given weight set can exist. This condition therefore serves as a criterion for detecting infeasibility at the level of weight enumeration, without explicitly constructing the underlying code structure.

**4.3 Group action according to projective geometric symmetry**

In the projective space PG(k-1, q), the projective general linear group PGL(k, q) acts naturally on the point set P. With respect to this group action, two points P and Q in P are said to belong to the same orbit if there exists an element g in PGL(k, q) such that g(P) = Q. Under this action, the point set P is decomposed into a disjoint union of orbits.

$$P = \bigsqcup_{i=1}^{r} O_i \tag{5}$$

This orbital decomposition means that point selections that can be converted to each other by projective isomorphism produce essentially identical codes. In other words, the selection between points belonging to the same orbit corresponds to overlap in terms of the isomorphism of the code, and does not actually provide a new structure. Therefore, only when representative points are selected in different orbits are cases that are essentially different. From this point of view, orbital branching constraints can be understood as not just computational heuristics, but as a structural and theoretical device to eliminate homomorphic duplication caused by group action in advance. This group action explains the cause of homomorphic redundancy that occurs in the process of code expansion, and subsequently serves as the basis for mathematically defining why the orbital branching constraint introduced in this study is valid.

**4.4 Exploration Space Reduction Constraints According to Orbital Branching**

For each orbit Oi, a representative point pi∈Oi is fixed. When a point selection variable is called xP∈Z≥0(P∈P), the search is branched by the following constraints:

$$x_{p_i} \geq 1,\ x_{p_j} = 0 \text{ for all } j < i. \tag{6}$$

This constraint divides the search space based on different orbital representative points, and each branch is responsible for the non-isomorphic sea space. This is a mathematical condition that prevents the repeated generation of solutions of the same structure without changing the existence of the solution.

## 5. MRE Suggested(Memory Resource Efficiency) Algorithm

**5.1 Problem Definition and Geometric Canonization**

The Memory Resource Efficiency (MRE) algorithm proposed in this study defines the classification

problem of [n,k]q linear codes as a lattice point enumeration problem within the projective geometry *PG(k−1, q)*. For $P_i$in the projective space, let denote the selection count for each point Pi. Under this formulation, the code classification problem is defined as the process of identifying an integer vector X that satisfies the following constraints.

1. Length Constraints: $\Sigma Pi \in P\ xi = n$ (7)
2. Weights: $\forall H \in H, \sum Pi \in H\ xi \in \{n - w \mid w \in W\}$ (8)

The existing standard ILP approach (hereinafter referred to as Baseline) directly calls the integer programming solver based on defined constraints to explore the solution. However, the explosion of search nodes that occurs during the proof of infeasibility process acts as a decisive bottleneck that causes the threshold of computational resources to be exceeded. To overcome these limitations, the MES algorithm adopts a hierarchical filtering structure that minimizes high-cost ILP calls, and strictly maintains the principle of proof-preserving that does not lose solution space at all stages. Each of the constraints that appear in this formulation corresponds to the mathematical requirements justified in Section 3.

**5.2 MES Algorithm Overview and Pseudocode**

The detailed operation procedure of the MES algorithm is as follows in Algorithm 1. For technical consistency, theoretical pruning is defined as Phase 0.5, linear easing as Phase 0, symmetry analysis as Phase 1, and precision navigation as Phase 1.5.

**Table1**

Procedure of MRE algorithm

| **MRE (Memory Efficiency Stability) algorithm** |
|---|
| Input: Length n, Dimension k, Field size q, Weight set W |
| Output: Unique linear codes C_unique or Proof of Infeasibility |
| 1: initialize geometry: generate $\mathcal{P}$ and $\mathcal{G}$ for PG(k-1, q) |
| 2: compute incidence matrix M ∈ {0, 1}^{\|$\mathcal{G}$\| × \|$\mathcal{P}$\|} |
| 3: compute orbits $\mathcal{O}$ = {$\mathcal{O}$₁, …, $\mathcal{O}$_r} under PGL(k, q) action |
| 4: set $\boldsymbol{\mathcal{R}}$ ← {p₁, …, p_r} as representatives where p_i ∈ $\mathcal{O}$_i |
| 5: **if** GriesmerBound(n, k, q, W) is violated **then**<br>**return** Infeasible |
| 6: **if** PlessPowerMoments(n, k, q, W) is unsatisfiable **then**<br>**return** Infeasible |
| 7: **if** Relaxed_LP_Model(M, W) is infeasible **then**<br>**return** Infeasible |
| 8: set $\mathcal{S}$_cand ← ∅ |
| 9: **for** i = 1 to r do |
| 10: construct optimized ILP model with<br>x_{p_i} ≥ 1<br>and ∑{j < i} ∑{P ∈ $\mathcal{O}$_j} x_P = 0 |
| 11: solve ILP via exact solver (Gurobi)<br>to find all feasible solutions |
| 12: update candidate set:<br>$\mathcal{S}$_cand ← $\mathcal{S}$_cand ∪ {Found solutions} |
| 13: **end for** |
| 14: filter $\mathcal{S}$_cand using weight distribution invariants |

and isomorphism checks

15: **return** $\mathcal{C}$_unique as non-equivalent codes

The efficiency of each step of this algorithm will be quantitatively evaluated in terms of {ILP} call reduction, memory usage, and validation stability compared to baseline in Section 5.

### 5.3 Step-by-step acceleration and optimization mechanism

#### 5.3.1 Determining the preemptiveness (Pre-Search Pruning / Phase 0.5 & 0)

In this step, before starting the search engine, the possibility of the existence of the code is verified through O(k) level arithmetic operations and linear relaxation (LP). Unlike Baseline, which immediately enters a high-cost exploration without such a pre-check, Phase 0.5 checks for combinatorial contradictions through Griesmer bound and Pless moments, and Phase 0 immediately terminates exploration if the LP model with the integer constraint removed is Infeasible. This is a key device that minimizes the frequency of ILP calls, which are the entire bottleneck and the dominant term of the verification.

#### 5.3.2 Exploring Symmetry-Based Optimization (Symmetry-Aware Search / Phase 1 & 1.5)

Phase 1 performs geometric symmetry analysis without invoking an optimization solver, while Phase 1.5 performs strict ILP search under orbit-based branching constraints. By applying the Orbital Branching technique, the width of the navigation tree is reduced, significantly reducing the number of navigation nodes compared to the baseline. Since each branch corresponds to a different orbit under the action of the group, no isomorphism class is excluded from the search, which ensures that the proof preservation characteristics of the MES are strictly maintained. In addition, depending on the size of the target weight set, equation constraints are applied to maximize the Presolve performance of the Gurobi solver.

#### 5.3.3 Post-validation and Isomorphic Filtering (Verification & Filtering / Phase 2)

For the explored solutions, the weight distribution is calculated as an invariant and grouped. Taking advantage of the property that codes with different distributions can never be Isomorphic, primary filtering is performed, and strict isomorphism tests are performed only within the same group to ensure computational efficiency and classification accuracy at the same time.

## 6. Results

This chapter quantitatively compares and analyzes the performance of the Baseline algorithm and the newly proposed Memory Resource Efficiency algorithm. It was conducted on a set of parameters of varying difficulty, and it is described with a particular focus on the trend and efficiency of memory usage, which is the most critical factor among system resources.

### 6.1 Configure experiment environment and dataset

Table 2: parameter set

| | n | K | Q | w |
|---|---|---|---|---|
| case1 | 18 | 6 | 7 | 3 |
| case2 | 35 | 4 | 8 | 28, 32 |
| case3 | 66 | 5 | 4 | 48, 56 |

The experiment was conducted by recording memory usage for a total of 60 minutes in 1-minute increments through a function that monitors memory usage in the same hardware environment. The dataset to be compared was configured in such a way that the size of the search space was different depending on

the dimension (k) of the linear code and the size of the body (q). In the case of case1, it was used as an instance where the existence of a simple bound method could not be immediately determined. For case2 and case3, parameters that were approached in the process of proving the existence of solutions in S. Kurz's previous research were used.

### 6.2 Analyze memory usage by algorithm

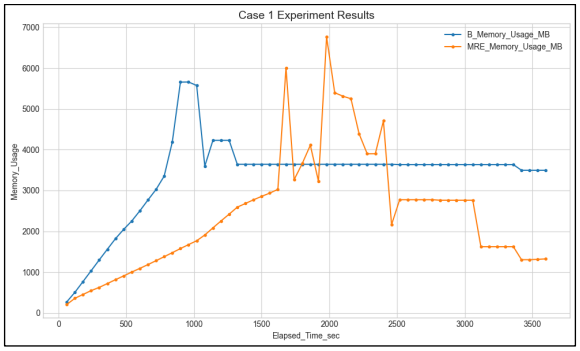


Figure 1: case1 experiment results

The Baseline algorithm shows a rapid increase in memory usage from the beginning of exploration, reaching a maximum level of 5.6 GB, and then maintains a high figure of around 3.6 GB for a long time. On the other hand, the MRE algorithm shows a relatively modest increase at the beginning of exploration, and then a temporary spike is observed at the time of the start (phase 1 entry section). However, after this initial spike, memory usage drops sharply, maintaining a stable downward trend in the long run.

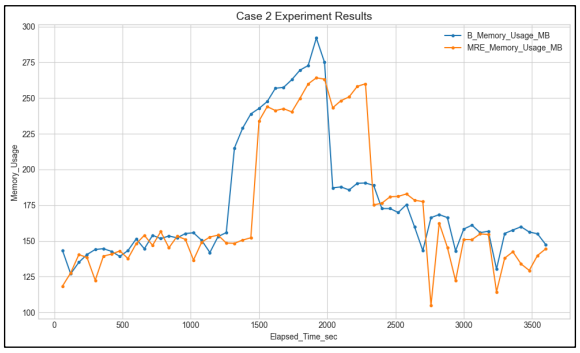


Figure 2: case2 experiment results

In scenarios with relatively small search spaces, both algorithms tend to keep memory usage within a few hundred MB. Both the baseline algorithm and the MRE algorithm record peaks at the level of 260 MB at similar point in time, but the overall fluctuation is smaller and less vibrating.

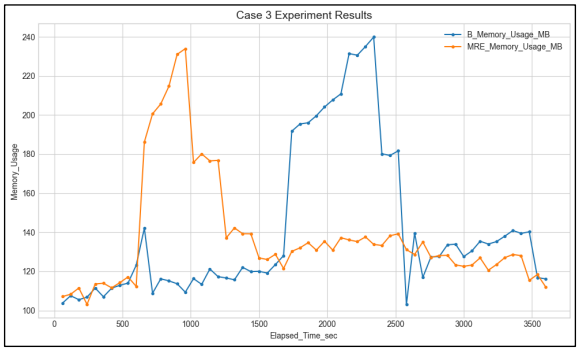

Figure 3: case3 experiment results

Case 3 is the experiment in which the structural benefits of the MRE algorithm are most clearly revealed. The memory usage of the Baseline algorithm gradually increases as the search progresses, while at certain intervals, it rises to the level of 240 MB and then decreases rapidly. On the other hand, in the case of MRE algorithms, a temporary memory increase is observed due to the symmetry preparation stage at the beginning of the search, but it remains stable in a narrow range as the search begins in earnest afterwards. Towards the end of the exploration, the MRE algorithm codeificantly reduces the volatility compared to the baseline algorithm, and there is almost no unnecessary memory reallocation.

This experiment confirmed that the proposed MRE algorithm reduces memory usage by up to approximately 27.9% compared to the basic algorithm. (Case 1: 27.9%, Case 2: 2.9%, Case 3: 2.1%) This memory efficiency has laid the foundation for solving high-dimensional problems within superior memory efficiency, resulting in a practical solution responsibility for MRE algorithms to solve complex linear code classification problems even in computing resource constrained environments. Therefore, in future studies and real-world symbol exploration projects, it makes sense to apply the symmetry-based multi-level pruning technique proposed in this study instead of a simple ILP approach in terms of resource efficiency.

**References**

[1] A. Basu, M. Conforti, M. Di Summa, and H. Jiang.: Complexity of branch-and-bound and cutting planes in mixed-integer optimization. Mathematical Programming, 183 (2020), 1–37.

[2] S. Kurz.: LinCode – computer classification of linear codes. Journal of Symbolic Computation, 95 (2019), 147–170.

[3] S. Kurz.: Computer classification of linear codes based on lattice point enumeration and integer linear programming. Decodes, Codes and Cryptography, 92 (2024), 1021–1056.

[4] T. Randolph and K. Węgrzycki.: Parameterized algorithms on integer sets with small doubling: Integer programming, subset sum and k-SUM. Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), 2025.

[5] D. Heinlein, T. Honold, M. Kiermaier, S. Kurz, and A. Wassermann.
Projective divisible binary codes.
Advances in Mathematics of Communications, 11 (2017), 1–35.

[6] R. Ganian and S. Ordyniak.: The complexity landscape of decompositional parameters for ILP. Artificial Intelligence, 257 (2018), 61–82.