# ADDITIVE COMPLEXITY OF SMALL LINEAR CODES: A CORRECTED CELL, TWO EXACT VALUES IN DIMENSION FIVE, AND THE EXTENDED TERNARY GOLAY CODE

KOREA SUPERINTELLIGENCE LABS

Abstract. For a prime power $q$, write $\mathrm{SSLP}_q(r,k,d)$ for the least number of additions and scalar multiplications needed to compute $Lv$, where $L^{\mathsf{T}}$ generates an $[r,k,d]_q$ code: the *additive* complexity of the best code with the given parameters. The quantity was introduced by Dumas, Lia and Sheekey, who tabulate eleven of its values and ask for precise results at more small parameters. We settle one of their cells against the published value, improve four others, and add a cell their table does not have. For $q \in \{2,3\}$ we prove $\mathrm{SSLP}_q(10,4,4) = 3$, not the 4 that appears in their table: three additions produce a $[10,4,4]_q$ code, and no two do. The witness computes two of the four standard basis vectors as intermediates and never outputs them, which is what the published lower bound overlooks; putting the generator matrix in systematic form is not available here, because a change of basis preserves the code but not the length of a program computing it. Over $\mathbb{F}_3$ in dimension five we prove $\mathrm{SSLP}_3(13,5,5) \le 7$, $\mathrm{SSLP}_3(12,5,5) \le 8$, $\mathrm{SSLP}_3(11,5,5) \le 9$ and $\mathrm{SSLP}_3(10,5,5) \le 10$, against the printed $\le 11$, $\le 12$, $\le 12$ and $\in [5,12]$; for the first of these we also record that $\le 8$ already follows from a binary program in the same paper, so the improvement there is from 8 to 7. In dimension six we prove $\mathrm{SSLP}_3(12,6,6) \le 14$, for the extended ternary Golay code — a cell the published table does not contain, and one that paper needs. Exhaustive searches reported separately, outside the machine-checked development, close two of these cells: $\mathrm{SSLP}_3(13,5,5) = 7$, and $\mathrm{SSLP}_3(10,5,5) = 10$ against a published bracket of $[5,12]$; they also bracket the new cell as $\mathrm{SSLP}_3(12,6,6) \in [10,14]$. Substituted into a lemma of Dumas, Lia and Sheekey, the values bound from below the cost of multiplication in algebras of order $3^5$ and $3^6$, and cap what that lemma can ever give. Every statement below that is labelled as a theorem, proposition or corollary is machine-checked in Lean 4, with no `sorry` and in a development that imports no mathematical library, with one exception that is marked as such in its own statement and proved on paper; the exhaustive searches that are not checked in the kernel are clearly marked and are stated as remarks, not results.

## 1. Introduction

**The quantity.** Let $q$ be a prime power and consider the multiplication of a finite field or finite semifield of order $q^n$, regarded as a bilinear map on $\mathbb{F}_q^n$. A rank-$r$ decomposition of the associated tensor presents that multiplication as

$$x \star y \;=\; P\big(Lx \odot Ry\big), \qquad L, R \in \mathbb{F}_q^{r\times n}, \quad P \in \mathbb{F}_q^{n\times r},$$

with $\odot$ the entrywise product. The number $r$ of base-field multiplications is the tensor rank, or *multiplicative* complexity; the additions and scalar multiplications spent on the three matrix–vector products $Lx$, $Ry$ and $P(\,\cdot\,)$ are the *additive* complexity. For small $n$ the two are of comparable weight, and the second is much the less studied of them.

Dumas, Lia and Sheekey [1] observe that the matrices occurring in such a decomposition are not arbitrary: $L^{\mathsf{T}}$, $R^{\mathsf{T}}$ and $P$ are generator matrices of linear codes whose length, dimension and minimum distance are forced by $r$ and $n$. So the additive cost of the best decomposition is bounded below by the additive cost of the best code, and they introduce the corresponding quantity. Their definition (Definition 10 of v1) reads, verbatim:

> "We denote by $\mathrm{SSLP}_q(r,k,d)$ (shortest straight-line program) the minimum number of additions and scalar multiplications required to compute $Lv$, where $L^{\mathsf{T}}$ generates an $[r,k,d]_q$ Hamming metric code."

It is followed immediately by the remark

> "Note that for $\mathbb{F}_2$ and $\mathbb{F}_3$ all operations can be achieved by addition, regarding negations as trivial",

which is why the two fields $q = 2$ and $q = 3$ are the ones for which the quantity has been tabulated, and the only ones considered here. The minimum in the definition is taken over *all* matrices $L$ whose transpose generates a code with the stated parameters, so $\mathrm{SSLP}_q(r, k, d)$ is a property of the parameter triple, not of a given matrix. That distinction is what separates it from the well-studied Shortest Linear Straight-Line Program problem of Boyar, Matthews and Peralta [3], and from the "XOR count" line for lightweight linear layers [4], both of which minimise gates for a *fixed* matrix over a field of characteristic two. It is also, and for the same reason, a different object from the tensor rank: everything below is about additive, not multiplicative or bilinear, complexity, and no bound here says anything about the rank of any tensor.

**What is known.** Table 1 reproduces the eleven entries of [1, Table 3], captioned "Table of some shortest straight-line programs for linear codes". Every reference in this paper is to arXiv:2602.09577v1, of 10 February 2026. That was still the only version of [1] when the computations reported here were run, on 2 and 3 September 2026, and when this text was finalised, on 3 September 2026; the arXiv listing was consulted on the last of those dates. Every definition, lemma and table of [1] is cited below by the number it carries in the compiled v1 e-print, with two exceptions that are cited by their content and flagged in the bibliography; those numbers could shift in a later version.

TABLE 1. The published values [1].

| $[r, k, d]_q$ | SSLP | $[r, k, d]_q$ | SSLP |
|---|---|---|---|
| $[8, 4, 4]_2$ | 6 | $[8, 4, 4]_3$ | 6 |
| $[9, 4, 4]_2$ | 5 | $[9, 4, 4]_3$ | 5 |
| $[10, 4, 4]_2$ | 4 | $[10, 4, 4]_3$ | 4 |
| $[13, 5, 5]_2$ | 8 | $[10, 5, 5]_3$ | $\in [5, 12]$ |
| | | $[11, 5, 5]_3$ | $\leq 12$ |
| | | $[12, 5, 5]_3$ | $\leq 12$ |
| | | $[13, 5, 5]_3$ | $\leq 11$ |

The six entries in dimension four rest on proofs (Theorem 18 and Lemmas 22 and 23 of v1), and so does the value 8 at $[13, 5, 5]_2$ (Lemma 21). The four upper bounds at $q = 3$, $k = 5$ are sourced in v1 to its Table 7, a table of multiplication algorithms in the same paper: they come from the addition counts of $L$-matrices produced by a randomised search over algorithms rather than from any search over codes. The lower bound 5 for $[10, 5, 5]_3$ comes from Lemma 16 of [1], which says that the generator matrix of a $[2k, k, k]_q$ code with $q \in \{2, 3\}$ and $k \geq 4$ has no repeated columns, so its ten distinct columns need at least $10 - 5$ additions. The first of the open problems with which [1] closes reads, verbatim:

> "Provide general lower bounds for $\mathrm{SSLP}_q(r, k, q)$, and establish precise results for more small parameters."

**Results.** We settle one cell of Table 1 against its published value, improve four others, and add a cell that table does not contain. Table 2 summarises.

The headline is the cell $[10, 4, 4]_q$. Three additions suffice (Theorem 3.1), two do not (Theorem 3.3), so $\mathrm{SSLP}_q(10, 4, 4) = 3$ for $q \in \{2, 3\}$ (Corollary 3.4) against the 4 of Table 1. The witness is short enough to display:

$$t_1 := \mathbf{e}_1 + \mathbf{e}_2, \qquad t_2 := \mathbf{e}_1 + \mathbf{e}_3, \qquad t_3 := \mathbf{e}_2 + \mathbf{e}_4,$$

TABLE 2. The cells of Table 1, and one cell beyond it, after this paper. Bold marks a value that changes. A dagger marks an entry that rests in part on an exhaustive computation outside the machine-checked development (Section 7).

| cell | published | here | where |
|---|---|---|---|
| $[8,4,4]_q$, $[9,4,4]_q$, $q \in \{2,3\}$ | 6, 5 | 6, 5 | Prop. 4.2, Rmk. 4.3 |
| $[10,4,4]_q$, $q \in \{2,3\}$ | 4 | **3** | Thm. 3.1, Thm. 3.3, Cor. 3.4 |
| $[13,5,5]_2$ | 8 | $8^\dagger$ | Prop. 5.4, Rmk. 7.1 |
| $[10,5,5]_3$ | $\in [5,12]$ | $\mathbf{10}^\dagger$ | Thm. 5.6, Rmk. 7.5 |
| $[11,5,5]_3$ | $\leq 12$ | $\in [\mathbf{7},\mathbf{9}]^\dagger$ | Thm. 5.3, Rmk. 7.3 |
| $[12,5,5]_3$ | $\leq 12$ | $\in [\mathbf{7},\mathbf{8}]^\dagger$ | Thm. 5.2, Rmk. 7.3 |
| $[13,5,5]_3$ | $\leq 11$ | $\mathbf{7}^\dagger$ | Thm. 5.1, Rmk. 7.4 |
| $[12,6,6]_3$ | — | $\in [\mathbf{10},\mathbf{14}]^\dagger$ | Thm. 6.1, Rmk. 7.6 |

with the ten output columns taking each of $\mathbf{e}_3, \mathbf{e}_4, t_1, t_2, t_3$ twice. Two of the four inputs, $\mathbf{e}_1$ and $\mathbf{e}_2$, are used only as intermediates and never appear among the outputs. That is exactly the possibility the published lower bound does not consider, and Section 3.3 locates the step, quotes it, and says why the systematic-form reduction that the step performs is unavailable for this quantity. The correction is confined to that one cell: the neighbouring values 6 and 5 at $[8,4,4]_q$ and $[9,4,4]_q$ are unaffected, and the four-addition program that [1] exhibits for $[10,4,4]$ is itself correct (Proposition 4.2) — it is the matching lower bound, not the construction, that fails.

In dimension five over $\mathbb{F}_3$ we exhibit programs of lengths 7, 8 and 9 for the cells $r = 13, 12, 11$ (Theorems 5.1, 5.2 and 5.3). One of the three comparisons in Table 2 needs a caveat that we prefer to state in the introduction rather than bury: the binary program of [1] for $[13,5,5]_2$, read over $\mathbb{F}_3$, again gives a $[13,5,5]_3$ code (Proposition 5.4), so $\mathrm{SSLP}_3(13,5,5) \leq 8$ was already implied by that paper's own data, four better than the $\leq 11$ its table prints. The improvement we claim at that cell is therefore from 8 to 7, and the step from 11 to 8 is an observation about [1] rather than a new computation. No such caveat applies at $r = 11$ or $r = 12$: $\mathrm{SSLP}_q(r,k,d)$ is non-increasing in $r$ (Lemma 2.4), so a program of length 13 says nothing about shorter lengths, and [1] exhibits no binary program at those lengths.

The remaining cell of that row, $[10,5,5]_3$, is the one where the *upper* bound is hard, and for a structural reason. The length 10 is Griesmer-optimal in dimension five at distance five, $5+2+1+1+1=10$, so no $[9,5,5]_3$ code exists at all, and by Lemma 16 of [1] the ten output columns of a $[10,5,5]_3$ generator matrix are ten *distinct* points of the projective space: nothing can be saved by repeating a column, and a program has to produce all ten values. We exhibit a ten-addition program (Theorem 5.6), two better than the published $\leq 12$, and a second one computing a code from the other of the two orbits into which the $[10,5,5]_3$ codes containing a standard frame fall (Remark 7.2).

Beyond the published table we add the cell $[12,6,6]_3$. Table 1 has rows for $k = 4$ and $k = 5$ only, while the first open problem of [1] asks for "precise results for more small parameters"; $[12,6,6]_3$ is the next Griesmer-optimal cell of the family $[2k,k,k]_3$, and it is one that paper itself needs. Its Lemma 9, bounding the partially symmetric tensor rank of $\mathbb{F}_{3^6}$ below by 13, requires generator matrices of $[12,6,6]_3$ codes, records that the extended ternary Golay code is the unique such code up to equivalence [6], and prints a generator matrix for it. Fourteen additions suffice for that code (Theorem 6.1); we know of no published bound for this cell in either direction. That last sentence is the outcome of a search, not of a published negative, and the search is worth naming: the phrases "shortest straight-line program" and "additive complexity" beside "linear code", and "ternary Golay" beside any of *addition*, *straight-line*, *XOR*, *circuit* and *complexity*, over an 86-gigabyte local full-text corpus of arXiv and over the live arXiv search interface; and the citation graph of [1] in OpenAlex, which was empty. The corpus lines that survive those filters are about *decoding* complexity, LP decoding, coset partitions, code transforms and Golay *sequences*, and none

of them costs the additions needed to compute a generator matrix of this code. All of it was re-run on 3 September 2026.

Four computations, all exhaustive, all carried out in C outside the formal development, are reported in Section 7: an independent reproduction of the exhaustive search that [1] asserts without describing for the cell $[13,5,5]_2$; a search over addition chains in dimension five over $\mathbb{F}_3$ which raises every lower bound in that row to 7 and so, with Theorem 5.1, gives $\mathrm{SSLP}_3(13,5,5)=7$ exactly; the corresponding search for a fixed target code, which finishes the cell $[10,5,5]_3$ from below and, with Theorem 5.6, gives $\mathrm{SSLP}_3(10,5,5)=10$ exactly, closing the published bracket $[5,12]$ to a single value; and the same search for the extended ternary Golay code, giving $\mathrm{SSLP}_3(12,6,6)\geq 10$. All four are stated as remarks, and nothing stated as a theorem depends on them.

The values are not decoration in [1]. Its Lemma 11 turns $\mathrm{SSLP}_q(r,n,n)$ into a lower bound on the number of operations that multiplication in a field or semifield of degree $n$ over $\mathbb{F}_q$ needs when it is computed with $r$ multiplications, and the last of the three open problems that paper closes with asks precisely about $r=10$ and $r=11$ at order $3^5$. Substituting the values obtained here gives $10M+25A$ and $11M+15A$ there, where that paper's own table supported $10M+10A$ and, at $r=11$, nothing at all; and it gives $12M+24A$ for every rank-12 algorithm at order $3^6$, a degree for which the table has no row. The same substitution prices the method as well as improving it: since $\mathrm{SSLP}_3(10,5,5)=10$ is exact, Lemma 11 can never yield more than $10M+25A$ at $r=10$, so the distance from there to the $10M+43A$ algorithm that paper exhibits cannot be closed by this route. That is Proposition 8.1, the one numbered statement below that is proved on paper rather than in the kernel: it is arithmetic on top of a lemma of [1] which we quote and use but do not verify, and its inputs are the computations of Section 7.

## 2. Preliminaries

Throughout, $q\in\{2,3\}$, and $\mathbf{e}_1,\dots,\mathbf{e}_k$ is the standard basis of $\mathbb{F}_q^k$. We unwind the definition of [1] quoted in Section 1 into the model that its own search recipe describes. That recipe reads, verbatim:

> "Begin with a matrix consisting of the standard basis vectors in $\mathbb{F}_q^k$. Add a row as a sum or difference of two previous rows, or as a repeat of a previous row. Repeat until we obtain a code with minimum distance at least $n$, or until we reach a chosen maximum number $r_{max}\geq r$ of rows. Compute the minimum distance of the code obtained by selecting every multiset of $r$ rows. If this is at least $d$, then $\mathrm{SSLP}_q(r,k,d)\leq r_{max}-k-\#\text{repeated rows}$."

**Definition 2.1.** A *straight-line program of length $L$ on $k$ inputs* is a sequence of steps $\sigma=(\sigma_1,\dots,\sigma_L)$, $\sigma_t=(a_t,b_t,\varepsilon_t)$ with $a_t,b_t<k+t-1$ and $\varepsilon_t\in\{+,-\}$. Its *values* $v_1,\dots,v_{k+L}\in\mathbb{F}_q^k$ are

$$v_i=\mathbf{e}_i\quad(1\leq i\leq k),\qquad v_{k+t}=v_{a_t+1}\;\varepsilon_t\;v_{b_t+1}\quad(1\leq t\leq L).$$

An *output selection* of size $r$ is a vector $m\in\mathbb{Z}_{\geq 0}^{k+L}$ with $\sum_i m_i=r$; it presents the $k\times r$ matrix $G(\sigma,m)$ whose columns are the values $v_i$, the value $v_i$ repeated $m_i$ times. We say the pair $(\sigma,m)$ *computes an $[r,k,\geq d]_q$ code* when

$$\mathrm{wt}(xG(\sigma,m))\;=\;\sum_{i\,:\,\langle x,v_i\rangle\neq 0} m_i\;\geq\;d\qquad\text{for every } x\in\mathbb{F}_q^k\setminus\{0\},$$

and set

$$\mathrm{SSLP}_q(r,k,d)\;=\;\min\bigl\{\,L\;:\;\text{some }(\sigma,m)\text{ of length }L\text{ and size }r\text{ computes an }[r,k,\geq d]_q\text{ code}\,\bigr\}.$$

Three points of the model carry the results below and are worth isolating.

*Remark* 2.2. The $k$ inputs are free and the outputs are *selected* from the pool of computed values. Nothing requires an input, or any particular intermediate, to be among the outputs, and nothing requires the outputs to be distinct: repetition costs nothing, which is precisely what "or as a repeat

of a previous row" and the term $-\#$repeated rows in the recipe of [1] say. This is the point at which the published lower bound for $[10, 4, 4]$ parts company with the definition; see Section 3.3.

*Remark* 2.3. We impose minimum distance *at least* $d$, following "if this is at least $d$" in the recipe above. The convention makes upper bounds no easier and lower bounds strictly harder, so every statement below is the conservative reading. Where the distinction could matter we control it: each witness exhibited here is checked to have minimum distance exactly $d$, so the codes produced really are $[r, k, d]_q$ codes and not codes of larger distance.

**Lemma 2.4.** *For $d \geq 1$ the map $x \mapsto xG(\sigma, m)$ is injective, so a code computed in the sense of Definition 2.1 has dimension exactly $k$; and $\mathrm{SSLP}_q(r, k, d)$ is non-increasing in $r$.*

*Proof.* If $xG = 0$ with $x \neq 0$ then $\mathrm{wt}(xG) = 0 < d$. For the second claim, note that every weight $\mathrm{wt}(xG(\sigma, m))$ is a sum of coordinates of $m$ over a subset of indices depending only on $\sigma$ and $x$; so increasing any coordinate of $m$ by one decreases no weight, and turns a selection of size $r$ into one of size $r + 1$ for the same program $\sigma$. □

We write $\mathbf{e}_i^*$ for the dual basis, so that the codeword indexed by $x = \mathbf{e}_i^*$ is the $i$-th row of $G$.

## 3. Three additions suffice, and two do not

### 3.1. The witness.

**Theorem 3.1.** *Let $q \in \{2, 3\}$. The straight-line program of length three*

$$t_1 := \mathbf{e}_1 + \mathbf{e}_2, \qquad t_2 := \mathbf{e}_1 + \mathbf{e}_3, \qquad t_3 := \mathbf{e}_2 + \mathbf{e}_4$$

*on four inputs, together with the output selection giving multiplicity 2 to each of $\mathbf{e}_3, \mathbf{e}_4, t_1, t_2, t_3$ and multiplicity 0 to $\mathbf{e}_1$ and $\mathbf{e}_2$, computes a $[10, 4, 4]_q$ code. Hence $\mathrm{SSLP}_q(10, 4, 4) \leq 3$.*

The generator matrix is

$$G \;=\; \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix},$$

its ten columns being $\mathbf{e}_3, \mathbf{e}_3, \mathbf{e}_4, \mathbf{e}_4, t_1, t_1, t_2, t_2, t_3, t_3$.

*Proof.* Each of the five values $\mathbf{e}_3, \mathbf{e}_4, t_1, t_2, t_3$ occurs exactly twice among the columns, so for $x = (x_1, x_2, x_3, x_4) \neq 0$,

$$\mathrm{wt}(xG) \;=\; 2 \cdot \#\{\, i : f_i(x) \neq 0 \,\}, \qquad (f_1, \ldots, f_5) = (x_3,\ x_4,\ x_1 + x_2,\ x_1 + x_3,\ x_2 + x_4).$$

It therefore suffices to show that at least two of the five forms are nonzero at every $x \neq 0$. If all five vanish then $f_1$ and $f_4$ give $x_3 = x_1 = 0$, then $f_3$ gives $x_2 = 0$ and $f_5$ gives $x_4 = 0$, so $x = 0$. Suppose exactly one is nonzero. If $f_1(x) \neq 0$, the vanishing of $f_2$ and $f_5$ gives $x_4 = x_2 = 0$, then $f_3$ gives $x_1 = 0$ and $f_4$ gives $x_3 = 0$, contradicting $f_1(x) = x_3 \neq 0$. If $f_2(x) \neq 0$, the vanishing of $f_1$ and $f_4$ gives $x_1 = 0$, then $f_3$ gives $x_2 = 0$ and $f_5$ gives $x_4 = 0$, contradicting $f_2(x) = x_4 \neq 0$. In each of the three remaining cases the two forms $f_1, f_2$ vanish, whence $x_3 = x_4 = 0$, and then any two of $f_3, f_4, f_5$ force $x_1 = x_2 = 0$, so the third vanishes as well. Thus $\mathrm{wt}(xG) \geq 4$ for all $x \neq 0$, and the bound is attained: at $x = \mathbf{e}_1^*$ only $f_3$ and $f_4$ are nonzero, so that codeword has weight exactly 4 and $G$ generates a $[10, 4, 4]_q$ code, not one of larger distance. □

*Remark* 3.2. The argument uses no property of the coefficient field, so the displayed $G$ generates a $[10, 4, 4]$ code over every field. Only $q \in \{2, 3\}$ is checked in the formal development, those being the fields for which the definition of [1] counts every operation as an addition.

3.2. **The matching lower bound.**

**Theorem 3.3.** *Let $q \in \{2,3\}$. No straight-line program of length at most 2, with any output selection of size 10, computes a $[10,4,4]_q$ code. Hence $\mathrm{SSLP}_q(10,4,4) \geq 3$.*

*Proof by exhaustive search.* The check is finite by construction, with no reduction of the search space and no appeal to symmetry. At step $t$ of a program on $k=4$ inputs there are $2(k+t-1)^2$ admissible steps, so there are 1, 32 and 1600 well-formed programs of lengths 0, 1 and 2 respectively — 1633 in all. An output selection of size 10 on $k+L$ values is a composition of 10 into $k+L$ non-negative parts, of which there are $\binom{13}{3} = 286$, $\binom{14}{4} = 1001$ and $\binom{15}{5} = 3003$ for $L = 0,1,2$. For each field, every one of the

$$1 \cdot 286 \;+\; 32 \cdot 1001 \;+\; 1600 \cdot 3003 \;=\; 4\,837\,118$$

resulting pairs $(\sigma, m)$ is tested against the condition of Definition 2.1 on all $|\mathbb{F}_q^4| - 1$ nonzero dual vectors, and none passes.

The completeness of the two enumerations — that every well-formed program of length $L$ is in the list of programs, and every composition of 10 into $k+L$ parts is in the list of selections — is proved, not assumed; the two lemmas are part of the formal development described in Section 9. The sweep is also checked not to be vacuously true: the identical enumeration over the identical 1600 programs of length 2, with output selections of size 12 instead of 10, returns *false*, and the two-addition witness it finds for a $[12,4,4]_3$ code is exhibited (it is the program $t_1 := \mathbf{e}_1 + \mathbf{e}_2$, $t_2 := \mathbf{e}_1 + \mathbf{e}_3$ with columns $\mathbf{e}_2, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_3, \mathbf{e}_4^{\times 4}, t_1, t_1, t_2, t_2$). □

**Corollary 3.4.** $\mathrm{SSLP}_q(10,4,4) = 3$ *for $q=2$ and for $q=3$.*

*Proof.* Theorems 3.1 and 3.3. □

3.3. **Where the published argument breaks.** The value 4 in Table 1 is the conclusion of Lemma 23 of [1], stating $\mathrm{SSLP}(10,4,4) = 4$ for $q \in \{2,3\}$. Its upper bound is correct and we re-check it below (Proposition 4.2). Its lower bound begins:

> "If the generator matrix of a $[10,4,4]$ code has a zero column then it requires at least 5 operations by [Lemma 22, the $[9,4,4]$ lemma]. To require fewer than 4 operations, it thus must have at least four standard basis and two repeated columns."

and the rest of the proof is a case analysis on matrices of the shape $G = [\, I_4 \mid M \,]$.

The second sentence is the step that does not follow. Counting "four standard basis columns" presumes that the $k$ free inputs of the program appear among the $r$ output columns. In the witness of Theorem 3.1 they do not: $\mathbf{e}_1$ and $\mathbf{e}_2$ are computed for free, used twice each as operands, and never output, so only two standard basis vectors occur among the ten columns and the case analysis never reaches the matrix $G$ displayed above.

The reduction to systematic form is unavailable here for a structural reason worth stating, since it is the reason the oversight is easy to make. Replacing $L$ by $LX$ for invertible $X$ replaces a generator matrix of a code by another generator matrix of the *same* code, so for questions about the code — its length, dimension and minimum distance — one may assume $G = [\, I_k \mid M \,]$. But the quantity being minimised is the length of a straight-line program computing $Lv$ from the standard inputs $v_1, \ldots, v_k$, and that length is not invariant under $L \mapsto LX$: the change of basis is itself something a program would have to compute. So "without loss of generality $G$ is systematic" is not a permissible reduction for $\mathrm{SSLP}_q(r,k,d)$, even though it is permissible for every parameter appearing in its statement.

Two remarks, in fairness to [1]. First, the failure is a lower-bound failure only, and it is confined to this cell: the neighbouring values $\mathrm{SSLP}_q(8,4,4) = 6$ and $\mathrm{SSLP}_q(9,4,4) = 5$ survive the independent search reported in Remark 4.3, and every straight-line program printed in [1] that we have checked — those for $[8,4,4]$, $[9,4,4]$ and $[10,4,4]$, and the one for $[13,5,5]_2$ — is correct (Propositions 4.2 and 5.4). Of the two neighbouring lower bounds, only one is affected in form. Lemma 22 of v1, at $[9,4,4]$, takes the same step, in the words "w.l.o.g. we can suppose that four of them are the standard basis vector the rest are repeated columns"; a $[9,4,4]_q$ code may repeat columns, so fewer than four

standard basis vectors can occur among them, and the case analysis that follows is incomplete in the same way. Theorem 17 of v1, at $[8,4,4]$, is not affected: there Lemma 16 makes the eight columns pairwise distinct while five additions on four inputs make only nine values available, so either all four inputs are output — the systematic case, disposed of just before — or exactly three are, which is the case the proof goes on to analyse. Neither conclusion is re-proved here: both are supported only by the search of Remark 4.3, which is outside the formal development. Second, nothing in the correction disturbs the use [1] makes of the table: the value enters its complexity bounds through Lemma 11 of v1, "the minimum number of operations needed to compute multiplication in a field or semifield of degree $n$ over $\mathbb{F}_q$ using $r$ multiplications is at least $rM + (3\,\mathrm{SSLP}_q(r,n,n) + n - r)A$", where a *smaller* SSLP weakens the lower bound on the cost of multiplication rather than invalidating any algorithm.

## 4. The rest of the row in dimension four

The cell $[10,4,4]$ sits inside a row that Table 1 tabulates only for $r = 8, 9, 10$. For completeness, and because a corrected cell is more credible when the row around it is complete and its ends are controlled, we record the remaining values. They are routine: with enough output columns the problem collapses, and the point of the row is where it stops.

**Proposition 4.1.** *For $q \in \{2,3\}$,*

$$\mathrm{SSLP}_q(11,4,4) = 3, \qquad \mathrm{SSLP}_q(12,4,4) = \mathrm{SSLP}_q(13,4,4) = 2, \qquad \mathrm{SSLP}_q(14,4,4) = 1.$$

*Proof by witness and exhaustive search.* Each upper bound is one explicit program: the program of Theorem 3.1 with an eleventh column on $\mathbf{e}_3$ for $r = 11$; the two-addition program $t_1 := \mathbf{e}_1 + \mathbf{e}_2$, $t_2 := \mathbf{e}_1 + \mathbf{e}_3$ with columns $\mathbf{e}_2^{\times 2}, \mathbf{e}_3^{\times 2}, \mathbf{e}_4^{\times 4}, t_1^{\times 2}, t_2^{\times 2}$ for $r = 12$ and the same with a third copy of $\mathbf{e}_2$ for $r = 13$; and the one-addition program $t_1 := \mathbf{e}_1 + \mathbf{e}_2$ with columns $\mathbf{e}_1^{\times 2}, \mathbf{e}_2^{\times 2}, \mathbf{e}_3^{\times 4}, \mathbf{e}_4^{\times 4}, t_1^{\times 2}$ for $r = 14$. Each matching lower bound is the sweep of Theorem 3.3 run at the relevant $r$ over every program of every smaller length and every output selection of that size.

Both ends of the row are controlled. With no additions at all the codeword indexed by $\mathbf{e}_i^*$ has weight equal to the multiplicity of $\mathbf{e}_i$, so $4 \cdot 4 = 16$ columns are needed; the sweep confirms that no zero-addition program works at $r = 15$, and the selection giving each basis vector multiplicity 4 works at $r = 16$, which also shows the $r = 15$ sweep is not vacuous. □

**Proposition 4.2.** *The straight-line programs printed in* [1] *for* $[8,4,4]$ *and* $[9,4,4]$ *compute codes with the stated parameters over $\mathbb{F}_2$ and over $\mathbb{F}_3$, and the one printed for* $[10,4,4]$ *does so over $\mathbb{F}_3$. In particular* $\mathrm{SSLP}_q(8,4,4) \le 6$ *and* $\mathrm{SSLP}_q(9,4,4) \le 5$ *for $q \in \{2,3\}$, and* $\mathrm{SSLP}_3(10,4,4) \le 4$.

*Proof.* The three programs are transcribed from the listings of [1] and evaluated against Definition 2.1. In the notation of that definition they are, with the output selection written as a multiplicity per computed value:

$$\begin{aligned}
[8,4,4]: &\quad t_1 := \mathbf{e}_1 + \mathbf{e}_3,\ t_2 := \mathbf{e}_2 + \mathbf{e}_4,\ t_3 := \mathbf{e}_2 + t_1,\ t_4 := \mathbf{e}_1 + t_2,\ t_5 := \mathbf{e}_4 + t_1,\ t_6 := \mathbf{e}_3 + t_2,\\
&\quad \text{columns } \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4, t_3, t_4, t_5, t_6;\\
[9,4,4]: &\quad t_1 := \mathbf{e}_1 + \mathbf{e}_4,\ t_2 := \mathbf{e}_2 + \mathbf{e}_3,\ t_3 := \mathbf{e}_4 + t_2,\ t_4 := \mathbf{e}_3 + t_1,\ t_5 := \mathbf{e}_2 + t_1,\\
&\quad \text{columns } \mathbf{e}_1^{\times 2}, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4, t_2, t_3, t_4, t_5;\\
[10,4,4]: &\quad t_1 := \mathbf{e}_4 + \mathbf{e}_3,\ t_2 := t_1 + \mathbf{e}_2,\ t_3 := \mathbf{e}_2 + \mathbf{e}_1,\ t_4 := t_3 + \mathbf{e}_3,\\
&\quad \text{columns } \mathbf{e}_1^{\times 2}, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4^{\times 2}, t_1, t_2, t_3, t_4.
\end{aligned}$$

The first two are checked over both fields and the third over $\mathbb{F}_3$; the $\mathbb{F}_2$ case at $r = 10$ is not checked here and is not needed, since Theorem 3.1 already gives $\mathrm{SSLP}_2(10,4,4) \le 3$. The purpose of the proposition is to separate the correction of Section 3.3 from any disagreement about the constructions of [1]: the constructions are sound, and only the lower bound at $r = 10$ fails. □

*Remark* 4.3 (a computation outside the formal development). An independently written exhaustive search in C, run before any of the above was formalised, computes for each program length $L$ the least $r$ for which some value set reachable in $L$ additions supports an $[r, 4, \geq 4]_q$ code. The search enumerates reachable *sets* of projective points rather than chains — the test depends only on the set of computed values — fixes the first addition to $\mathbf{e}_1 + \mathbf{e}_2$ under the group of signed coordinate permutations, and discards steps that produce zero or a value already present, both of which can be deleted from a shortest program. It returns $14, 12, 10, 10, 9, 8$ for $L = 1, \ldots, 6$ over both fields, whence $\mathrm{SSLP}_q(r, 4, 4) = 6, 5, 3, 3, 2, 2, 1$ at $r = 8, \ldots, 14$. This reproduces the published 6 and 5 at $r = 8, 9$, agrees with Corollary 3.4 and Proposition 4.1, and contradicts only the cell $[10, 4, 4]$. The lower bounds at $r = 8$ and $r = 9$ rest on this computation alone and are not part of the machine-checked development; the values there are those of [1], and we claim no independent proof of them.

## 5. Dimension five over $\mathbb{F}_3$

Table 1 leaves the four cells $[r, 5, 5]_3$, $r = 10, \ldots, 13$, bounded only from above, and only from a table of multiplication algorithms. Searching over codes instead does substantially better at the three cells $r = 11, 12, 13$. Each witness below was found by a hill-climbing search over addition chains, re-derived from Definition 2.1 by an independently written implementation — which also has to recover the signs, the chain search working with projective representatives — and only then checked in the kernel. No claim of exhaustiveness attaches to any of them: they are upper bounds.

**Theorem 5.1.** $\mathrm{SSLP}_3(13, 5, 5) \leq 7$. *Explicitly, the seven-addition program on five inputs*

$$u_1 := \mathbf{e}_3 - \mathbf{e}_2, \quad u_2 := u_1 - \mathbf{e}_5, \quad u_3 := \mathbf{e}_4 + \mathbf{e}_5, \quad u_4 := u_3 + \mathbf{e}_2,$$
$$u_5 := \mathbf{e}_5 + \mathbf{e}_1, \quad u_6 := u_4 + u_1, \quad u_7 := \mathbf{e}_1 + u_4,$$

*with output multiplicities* $1, 1, 2, 1, 0$ *on* $\mathbf{e}_1, \ldots, \mathbf{e}_5$ *and* $0, 2, 1, 0, 2, 1, 2$ *on* $u_1, \ldots, u_7$*, computes a* $[13, 5, 5]_3$ *code.*

**Theorem 5.2.** $\mathrm{SSLP}_3(12, 5, 5) \leq 8$. *Explicitly, the eight-addition program on five inputs*

$$u_1 := \mathbf{e}_5 + \mathbf{e}_3, \quad u_2 := \mathbf{e}_1 + \mathbf{e}_4, \quad u_3 := \mathbf{e}_4 + \mathbf{e}_2, \quad u_4 := \mathbf{e}_1 + \mathbf{e}_3,$$
$$u_5 := u_1 - \mathbf{e}_2, \quad u_6 := \mathbf{e}_5 + u_2, \quad u_7 := u_3 - \mathbf{e}_3, \quad u_8 := \mathbf{e}_2 + u_4,$$

*with output multiplicities* $0, 0, 1, 1, 1$ *on* $\mathbf{e}_1, \ldots, \mathbf{e}_5$ *and* $1, 0, 1, 1, 1, 2, 1, 2$ *on* $u_1, \ldots, u_8$*, computes a* $[12, 5, 5]_3$ *code.*

**Theorem 5.3.** $\mathrm{SSLP}_3(11, 5, 5) \leq 9$. *Explicitly, the nine-addition program on five inputs*

$$u_1 := \mathbf{e}_2 + \mathbf{e}_5, \quad u_2 := \mathbf{e}_1 - \mathbf{e}_2, \quad u_3 := \mathbf{e}_3 + u_2, \quad u_4 := \mathbf{e}_1 + u_1, \quad u_5 := \mathbf{e}_2 + u_3,$$
$$u_6 := \mathbf{e}_4 - u_4, \quad u_7 := u_3 + \mathbf{e}_4, \quad u_8 := u_1 + \mathbf{e}_3, \quad u_9 := u_1 + u_7,$$

*with output multiplicities* $0, 1, 1, 1, 1$ *on* $\mathbf{e}_1, \ldots, \mathbf{e}_5$ *and* $0, 1, 0, 0, 1, 2, 1, 1, 1$ *on* $u_1, \ldots, u_9$*, computes an* $[11, 5, 5]_3$ *code.*

*Proof of Theorems 5.1, 5.2 and 5.3.* In each case the values of the program are evaluated in $\mathbb{F}_3^5$ and the condition of Definition 2.1 is checked at all $3^5 - 1 = 242$ nonzero dual vectors. Each of the three codes is confirmed to have minimum distance exactly 5, so the parameters are as stated. Two further checks accompany the first witness: read over $\mathbb{F}_2$ — where its subtractions are a different operation — the same program and the same output selection give a code of minimum distance 3, so the evaluation is genuinely field-sensitive; and dropping its last addition and moving the freed columns onto $\mathbf{e}_1$ gives minimum distance 3, so the length of the witness is not decorative. □

**Proposition 5.4.** *The eight-addition program printed in* [1, Lemma 21] *for* $[13, 5, 5]_2$*,*

$$u_1 := \mathbf{e}_5 + \mathbf{e}_4, \quad u_2 := \mathbf{e}_3 + \mathbf{e}_4, \quad u_3 := \mathbf{e}_5 + \mathbf{e}_1, \quad u_4 := \mathbf{e}_5 + \mathbf{e}_2,$$
$$u_5 := u_2 + \mathbf{e}_2, \quad u_6 := u_3 + \mathbf{e}_3, \quad u_7 := u_4 + u_6, \quad u_8 := u_7 + u_2,$$

*with all thirteen computed values output once, computes a* $[13,5,5]_2$ *code, so* $\mathrm{SSLP}_2(13,5,5) \le 8$. *Read over* $\mathbb{F}_3$ *— every step of it being an addition — the same program with the same output selection computes a* $[13,5,5]_3$ *code, so* $\mathrm{SSLP}_3(13,5,5) \le 8$ *as well.*

*Proof.* Both evaluations are direct. What is read over $\mathbb{F}_3$ is the *program*, not the 0/1 matrix that [1] displays beside it: over $\mathbb{F}_3$ the last two values of the program are $\mathbf{e}_1+\mathbf{e}_2+\mathbf{e}_3+2\mathbf{e}_5$ and $\mathbf{e}_1+\mathbf{e}_2+2\mathbf{e}_3+\mathbf{e}_4+2\mathbf{e}_5$, so the generator matrix it produces there is a different matrix. What the bound needs is the number of additions, and that is the same eight over both fields. □

*Remark* 5.5. Proposition 5.4 is the honest baseline for Theorem 5.1. The bound $\mathrm{SSLP}_3(13,5,5) \le 8$ is a one-line consequence of a lemma already in [1], four better than the $\le 11$ that its Table 1 entry records for that cell, and it appears nowhere in that paper. The new content of Theorem 5.1 is therefore the step from 8 to 7. The other two cells admit no such transfer: by Lemma 2.4 a program of length 13 bounds nothing at $r = 11$ or $r = 12$, and [1] exhibits no binary program at those lengths. The only other route to such a transfer would be a binary $[n,5,5]$ code of length below 13, and there is none: Grassl's tables [7] give 4 as both the lower and the upper bound for the largest minimum distance of a binary $[12,5]$ code, and that largest distance is non-decreasing in $n$, so no binary $[n,5,5]$ code with $n \le 12$ exists, while $[13,5,5]_2$ does. None of the statements above depends on this.

5.1. **The cell** $[10,5,5]_3$**.** The remaining cell of the row is the one at which the upper bound is hard. The length is Griesmer-optimal, $\sum_{i<5}\lceil 5/3^i\rceil = 5+2+1+1+1 = 10$, so no $[9,5,5]_3$ code exists at all, and by Lemma 16 of [1] the ten columns of a $[10,5,5]_3$ generator matrix are ten *distinct* points of $\mathrm{PG}(4,3)$: nothing is saved by repeating a column, and a program has to produce all ten of them. The published bracket for the cell is $[5,12]$, sourced in [1] to its Lemma 16 and its Table 7 of multiplication algorithms; the upper end is the addition count of the $L$-matrix of a rank-10 algorithm that paper reports for a semifield of order 243, not the result of any search over codes. Ten additions suffice.

**Theorem 5.6.** $\mathrm{SSLP}_3(10,5,5) \le 10$. *Explicitly, the ten-addition program on five inputs*

$$u_1 := \mathbf{e}_2 - \mathbf{e}_4, \quad u_2 := \mathbf{e}_1 + \mathbf{e}_5, \quad u_3 := \mathbf{e}_4 - u_2, \quad u_4 := \mathbf{e}_1 + \mathbf{e}_3, \quad u_5 := \mathbf{e}_4 + u_4,$$
$$u_6 := \mathbf{e}_2 + u_2, \quad u_7 := u_5 + u_6, \quad u_8 := \mathbf{e}_3 + u_7, \quad u_9 := \mathbf{e}_4 - u_8, \quad u_{10} := u_2 + u_8,$$

*with output multiplicities* $1,1,1,0,1$ *on* $\mathbf{e}_1,\dots,\mathbf{e}_5$ *and* $1,0,1,0,1,0,1,0,1,1$ *on* $u_1,\dots,u_{10}$, *computes a* $[10,5,5]_3$ *code; and so does the ten-addition program*

$$u_1 := \mathbf{e}_1 + \mathbf{e}_3, \quad u_2 := \mathbf{e}_4 + u_1, \quad u_3 := \mathbf{e}_2 + u_2, \quad u_4 := \mathbf{e}_1 - \mathbf{e}_2, \quad u_5 := u_1 + u_4,$$
$$u_6 := \mathbf{e}_5 + u_4, \quad u_7 := \mathbf{e}_3 - u_6, \quad u_8 := u_3 - u_7, \quad u_9 := u_1 - u_8, \quad u_{10} := u_4 - u_8,$$

*with output multiplicities* $1,0,0,1,1$ *on* $\mathbf{e}_1,\dots,\mathbf{e}_5$ *and* $0,1,1,0,1,1,1,0,1,1$ *on* $u_1,\dots,u_{10}$.

*Proof.* In each case the fifteen values are evaluated in $\mathbb{F}_3^5$ and the condition of Definition 2.1 is checked at all $3^5-1 = 242$ nonzero dual vectors. Each program outputs ten of its fifteen values once each and none of them twice, as it must. Both codes are confirmed to have minimum distance exactly 5, so the parameters are as stated, and both have the same weight enumerator,

$$1 + 72z^5 + 60z^6 + 90z^8 + 20z^9,$$

in which $72+60+90+20 = 242$ accounts for every nonzero dual vector, so no other weight occurs in either code. (That is the weight enumerator of the $[10,5,5]_3$ code, which [1] records as unique up to equivalence [5]; the count is made here so that a transcription slip in either program would have to reproduce it in order to go unnoticed, and no bound in this paper depends on the uniqueness.) Three controls accompany the first witness: read over $\mathbb{F}_2$ the same program and output selection give minimum distance 3, so the evaluation is field-sensitive; dropping its last addition, or its last two, and moving the freed columns onto $\mathbf{e}_1$ gives minimum distance 4 and then 3; and *nine* output columns on the same fifteen values never reach minimum distance 5, whichever nine are chosen —

a sweep over all $\binom{23}{14} = 817\,190$ multiplicity vectors of length 15 summing to 9. The last of these is the Griesmer bound $\sum_{i<5}\lceil 5/3^i\rceil = 10$ confirmed at this instance; it is a control on the witness, not a lower bound for the cell. □

*Remark* 5.7. One witness already proves the bound, $\mathrm{SSLP}_3(10,5,5)$ being a minimum over *all* $[10,5,5]_3$ codes. The second is here because the two searches ran on the two orbit representatives of Remark 7.2 — the two frame orbits into which the $[10,5,5]_3$ codes containing a standard frame fall — and the exhaustive lower bound of Remark 7.5 has to run on both of them; a ten-addition program for only one representative would leave the two halves of the cell talking about different objects. Both programs were found by a purpose-written randomised greedy chain builder with restarts and a one-step lookahead, then lifted to the standard basis and re-derived from Definition 2.1 by an independently written implementation before being checked in the kernel. The lift is not a formality: the search works with projective points and starts from an arbitrary independent five-set — which is exactly the freedom a program has — while the model starts from $\mathbf{e}_1,\dots,\mathbf{e}_5$, so the change of basis and the sign of every step have to be recovered. Three million restarts per representative with the length capped at 9 found nothing shorter; that is evidence, and the proof that nothing shorter exists is the exhaustive computation of Remark 7.5.

## 6. A cell beyond the table: the extended ternary Golay code

Table 1 has rows for $k = 4$ and $k = 5$ only, and [1] asks for precise results at more small parameters. The next Griesmer-optimal cell of the family $[2k,k,k]_3$ is $[12,6,6]_3$, with $\sum_{i<6}\lceil 6/3^i\rceil = 6+2+1+1+1+1 = 12$, and it is a cell that paper itself needs. Lemma 9 of v1, proving that the partially symmetric tensor rank of $\mathbb{F}_{3^6}$, and of any semifield of order $3^6$, is at least 13, opens:

> "We require that $L^{\mathsf{T}} = R^{\mathsf{T}}$ and $P$ are generator matrices for $[12,6,6]_3$ codes. There is a unique $[12,6,6]_3$ code up to equivalence, namely the extended ternary Golay code [6]. Thus we may assume that $L^{\mathsf{T}}$ is any generator matrix for this code; for example, …"

and then prints $L^{\mathsf{T}} = [\,I_6 \mid B\,]$ with

$$B = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 & 2 & 1 \\ 1 & 1 & 0 & 1 & 2 & 2 \\ 1 & 2 & 1 & 0 & 1 & 2 \\ 1 & 2 & 2 & 1 & 0 & 1 \\ 1 & 1 & 2 & 2 & 1 & 0 \end{pmatrix}.$$

That is the code used below. Since $\mathrm{SSLP}_3(12,6,6)$ is a minimum over all $[12,6,6]_3$ codes, an upper bound for one of them is an upper bound for the cell, and the uniqueness citation is not needed for Theorem 6.1; it is needed for the matching lower bound, and Remark 7.6 says so.

**Theorem 6.1.** $\mathrm{SSLP}_3(12,6,6) \le 14$. *Explicitly, the fourteen-addition program on six inputs*

$$u_1 := \mathbf{e}_1 - \mathbf{e}_6,\quad u_2 := \mathbf{e}_2 + u_1,\quad u_3 := \mathbf{e}_3 + \mathbf{e}_4,\quad u_4 := \mathbf{e}_2 - u_3,\quad u_5 := \mathbf{e}_4 - \mathbf{e}_5,$$
$$u_6 := u_3 + u_5,\quad u_7 := \mathbf{e}_1 - u_5,\quad u_8 := \mathbf{e}_2 + u_7,\quad u_9 := \mathbf{e}_5 - \mathbf{e}_6,\quad u_{10} := u_4 - u_9,$$
$$u_{11} := u_7 - u_{10},\quad u_{12} := u_1 + u_9,\quad u_{13} := \mathbf{e}_2 - u_{12},\quad u_{14} := u_6 - u_{12},$$

*with output multiplicities* $1,0,1,1,0,1$ *on* $\mathbf{e}_1,\dots,\mathbf{e}_6$ *and* $0,1,0,1,0,1,0,1,0,1,1,0,1,1$ *on* $u_1,\dots,u_{14}$, *computes a* $[12,6,6]_3$ *code.*

*Proof.* The twenty values are evaluated in $\mathbb{F}_3^6$ and the condition of Definition 2.1 is checked at all $3^6 - 1 = 728$ nonzero dual vectors; twelve of the twenty values are output once each. The minimum distance is exactly 6, and the weight enumerator is

$$1 + 264z^6 + 440z^9 + 24z^{12},$$

with $264 + 440 + 24 = 728$, so every nonzero codeword has one of the three weights $6, 9, 12$. That is the weight enumerator of the extended ternary Golay code, which is what this code must be, being a $[12, 6, 6]_3$ code and that code being unique up to equivalence [6]; the computation is redundant given the uniqueness, and is there so that a transcription slip in the program would have to reproduce the Golay weight distribution in order to go unnoticed. Three controls accompany the witness: read over $\mathbb{F}_2$ the same program and columns give minimum distance 4; dropping the last addition and moving the freed column onto $\mathbf{e}_1$ gives 5; and stacking all twelve columns on three of the values gives 0. □

*Remark* 6.2. The program was found by a randomised greedy search over addition chains for a fixed target code, then lifted and re-derived from Definition 2.1 exactly as in Remark 5.7. Fourteen is a plateau for two unrelated heuristics rather than a first try: about 1.2 million greedy restarts over four runs and four seeds never beat it, and 420 seconds of local search conducted directly in the space in which the exhaustive search of Remark 7.6 lives, aimed at length 13, never closed more than 17 of the 19 points it needed. For comparison, the obvious hand construction costs 21: the all-ones vector $j$ in five additions, the six vectors $j - \mathbf{e}_i$ in six more — the first column of $B$ is $j - \mathbf{e}_1$ — and each of the five remaining columns of $B$ as some $j - \mathbf{e}_i$ plus two standard basis vectors, at two additions each. No exhaustive search reaches length 14 at any budget we can price, so 14 is an upper bound and not a claim of optimality; the best lower bound we have for the cell is the 10 of Remark 7.6.

## 7. Exhaustive searches outside the formal development

The computations in this section are exhaustive searches carried out in C, independently written from Definition 2.1, and *not* checked in the kernel. They are stated as remarks for that reason, and nothing in Sections 3–6 depends on them. They come in two shapes. The searches over addition *chains* (Remarks 7.1 and 7.3) enumerate the value sets reachable in $L$ additions and ask how few output columns any of them supports; they use the same three reductions as Remark 4.3: deduplication of chains by their value set, deletion of steps producing zero or a repeated value, and a canonical first addition under the group of signed coordinate permutations. Only the last is a symmetry argument; the other two are minimality arguments valid cell by cell. The searches for a *fixed* target code (Remarks 7.2, 7.5 and 7.6) go the other way round, enumerating the sets of extra values a program of length $L$ could hold besides the code's own columns; the same minimality argument makes exhausting those sizes upward exhaustive over program lengths, since at a shortest program all $k + L$ values are distinct and nonzero.

*Remark* 7.1 (lower bounds in dimension five). Running the search of Remark 4.3 at $k = d = 5$ with the output size capped at 13 shows that no value set reachable in $L$ additions supports as many as 13 output columns, for every $L \le 5$ over $\mathbb{F}_3$ (the top level holding 131 433 deduplicated value sets and $1.81 \times 10^9$ leaf nodes of the search) and for every $L \le 7$ over $\mathbb{F}_2$ (97 635 sets, $1.06 \times 10^{10}$ leaf nodes). Hence $\mathrm{SSLP}_3(r, 5, 5) \ge 6$ and $\mathrm{SSLP}_2(r, 5, 5) \ge 8$ for every $r \le 13$. The $\mathbb{F}_2$ half is a reproduction rather than a new bound: the proof of Lemma 21 of [1], the lemma for that cell, ends "Fewer than 8 additions would require at least one repeated column. An exhaustive search shows that this is not possible", with no method, size or running time given, and the computation just described is that search, independently written, agreeing. Combined with Proposition 5.4 it gives $\mathrm{SSLP}_2(13, 5, 5) = 8$. Combined with Theorems 5.1, 5.2 and 5.3 it brackets the three $\mathbb{F}_3$ cells as $\mathrm{SSLP}_3(11, 5, 5) \in [6, 9]$, $\mathrm{SSLP}_3(12, 5, 5) \in [6, 8]$ and $\mathrm{SSLP}_3(13, 5, 5) \in [6, 7]$.

*Remark* 7.2 (the cell $[10, 5, 5]_3$). For a *fixed* target code the problem collapses, and the cell $[10, 5, 5]_3$ is small enough to be settled from below that way. A program of length $L$ computes a code with column set $P$ exactly when its value set $S \supseteq P$ has $|S| = 5 + L$ and $S$ is generated from some independent 5-subset of itself by the operations $(x, y) \mapsto \pm x \pm y$ performed *inside* $S$; the starting basis being arbitrary is exactly the freedom to change coordinates, so working in the target's own

coordinates loses nothing, and since the closure operation is monotone a greedy closure decides the question. Every $[10,5,5]_3$ code spans $\mathbb{F}_3^5$, so five of its ten columns form a basis and may be carried to the five standard basis points. Enumerating the ten-point sets of $\mathrm{PG}(4,3)$ that contain the five standard basis points and meet every hyperplane in at most five points gives 576 sets, falling into two orbits, of sizes 96 and 480, under the group of signed coordinate permutations; so every $[10,5,5]_3$ code is equivalent to one of two explicit representatives. (Two orbits is not a disagreement with the uniqueness of the $[10,5,5]_3$ code that [1] takes from [5]. The group acting here is only the stabiliser of the five standard basis points, of order $5!\,2^5 = 3840$, and not the full equivalence group, which may also change basis; and the two representatives are in fact carried onto one another by an element of $\mathrm{GL}_5(3)$ — an exhaustive search finds 1440 of them — so they do generate equivalent codes. The enumeration is written this way so that the lower bound below rests on nothing but itself.) Both representatives are infeasible at $L = 5, 6, 7$ and at $L = 8$, whence

$$\mathrm{SSLP}_3(10,5,5) \;\geq\; 9.$$

The published upper bound $\leq 12$ is untouched, so the published interval $[5,12]$ becomes $[9,12]$. The decisive cost saving is the observation that an element of $S$ that is not $\pm a \pm b$ for any $a, b \in S$ can never be produced by a step, so it must be one of the five free inputs: if more than five elements of $S$ are inexpressible then $S$ is unreachable, and otherwise every candidate free basis must contain the inexpressible ones. The level $L = 9$ has since been run, on both representatives, and Remark 7.5 reports it; $L = 10$ is out of reach at the budget used here.

*Remark* 7.3 (the dimension-five row from below). The chain search of Remark 7.1 was carried one level further over $\mathbb{F}_3$. At $L = 6$ the level holds 2 262 515 deduplicated value sets and $8.63 \times 10^{10}$ leaf nodes, and again no reachable value set supports as many as thirteen output columns at minimum distance 5. The least number of columns a reachable set supports is non-increasing in $L$ — extend the set by one more value — so every length $L \leq 6$ falls at once:

$$\mathrm{SSLP}_3(r,5,5) \;\geq\; 7 \qquad \text{for every } r \leq 13,$$

one better than Remark 7.1 across the whole row. The run cost 1598 seconds of CPU time and 428 megabytes, and before its new level was believed it reproduced the earlier run's level sizes $1, 25, 462, 7799, 131\,433$ at $L \leq 5$ and its leaf-node count $1\,810\,422\,341$ digit for digit. A plateau-walking hill climber, 170 006 chain evaluations at $L = 6$ in 240 seconds of CPU time, likewise found no six-addition chain supporting thirteen columns: heuristic agreement rather than a second proof, but the check that would have fired had the exhaustive search been mis-pruned.

*Remark* 7.4 (the cell $[13,5,5]_3$, closed). Remark 7.3 and Theorem 5.1 meet:

$$\mathrm{SSLP}_3(13,5,5) \;=\; 7.$$

Table 1 prints $\leq 11$ for this cell; as Remark 5.5 says, the honest baseline is the $\leq 8$ that the binary program of [1] already gives when it is read over $\mathbb{F}_3$; what closes the cell is therefore the step from 8 to 7 above (Theorem 5.1) together with the step from 6 to 7 below (Remark 7.3). The same lower bound with Theorems 5.2 and 5.3 brackets the neighbouring cells as

$$\mathrm{SSLP}_3(12,5,5) \in [7,8], \qquad \mathrm{SSLP}_3(11,5,5) \in [7,9],$$

both of which Table 1 prints as $\leq 12$. Deciding the first of them is exactly the level $L = 7$ priced in Remark 7.8.

*Remark* 7.5 (the cell $[10,5,5]_3$, closed). The level $L = 9$ left unrun in Remark 7.2 has since been run on both representatives. A program of length 9 holds $5 + 9 = 14$ values, ten of which are the code's columns, so the search is over the $\binom{111}{4} = 5\,989\,005$ four-element sets of extra points of $\mathrm{PG}(4,3)$. It was run in two chunks per representative, split on the index of the first extra point, holding 2 524 165 and 3 464 840 candidate sets; all four chunks are infeasible, and $2\,524\,165 + 3\,464\,840 = 5\,989\,005$

exactly, which is the check that the split dropped nothing. With the levels $L \le 8$ of Remark 7.2 this gives $\mathrm{SSLP}_3(10,5,5) \ge 10$, and with Theorem 5.6

$$\mathrm{SSLP}_3(10,5,5) \;=\; 10\,:$$

the published bracket $[5,12]$ closes to a single value. The level cost 36.0 minutes of CPU time per representative, 1.20 CPU-hours in all, at 0.354 milliseconds per candidate set. That is the half-hour per representative that version 1 of this paper estimated for the level (see the note on version 1 at the end), but only after the rewrite described in Remark 7.7: with the inner loop as it stood when that estimate was made, the same level cost 1.12 milliseconds per set, about 1.9 CPU-hours per representative.

*Remark* 7.6 (the cell $[12,6,6]_3$ from below). The same fixed-code search, run on the extended ternary Golay code as Section 6 presents it. A program of length $L$ holds $6+L$ values, twelve of which are the code's columns, so the search is over the sets of $|E| = L-6$ extra points of $\mathrm{PG}(5,3)$, which has 364 points, 352 of them outside the code. At $|E| = 0,1,2,3$ — that is, at lengths $L = 6,7,8,9$ — there are 1, 352, 61 776 and $\binom{352}{3} = 7\,207\,200$ candidate sets, and every one of them is infeasible. Hence

$$\mathrm{SSLP}_3(12,6,6) \;\ge\; 10, \qquad \text{so} \qquad \mathrm{SSLP}_3(12,6,6) \in [10,14]$$

with Theorem 6.1, for a cell where we know of no previous bound in either direction. The level $L = 9$ carries essentially all of the cost, at 124.5 seconds of CPU time with the search as it now stands.

What the lower bound rests on should be said plainly, because it differs from the $[10,5,5]_3$ case. The search takes the code as fixed, so what it proves is a statement about the extended ternary Golay code; it becomes a statement about the cell $\mathrm{SSLP}_3(12,6,6)$ — a minimum over all $[12,6,6]_3$ codes — through the uniqueness of that code up to equivalence, which is Pless [6] as cited by [1] in the lemma quoted in Section 6, and which is not re-derived here. In dimension five no such citation is needed, because the enumeration described in Remark 7.2 produces the codes itself; the analogous enumeration in dimension six is over the $\binom{358}{6} = 2.8 \times 10^{12}$ twelve-point subsets of $\mathrm{PG}(5,3)$ containing the standard frame before pruning, and since the dimension-five case pruned its $\binom{116}{5} = 1.6 \times 10^8$ candidates down to a fifth of a second of work, the pruned cost is not predictable from the raw count and was not measured. The upper bound of Theorem 6.1 needs none of this.

*Remark* 7.7 (on the instrument). Midway through these computations the reachability core of the fixed-code search was rewritten — productions kept as bit masks, independence of the candidate basis maintained incrementally — for a measured speed-up of 3.2. Since that core produces every "infeasible" verdict in Remarks 7.5 and 7.6, the old version was not discarded but run against the new one: on every level up to $L = 8$ for both $[10,5,5]_3$ representatives, and on one chunk of the $L = 9$ level, the two return identical verdicts and identical candidate counts (1, 111, 6105, 221 815, and 108 019 for the chunk); on a positive control — a ten-point set that *is* reachable in five additions — both print the same chain, so the search can still say yes; and the $[12,6,6]_3$ lower bound was run to completion with each of them, at 396.6 and 124.5 seconds, so that verdict has two independent implementations behind it. The candidate count is printed beside every verdict for a reason: a search whose answer is always "no" reads exactly the same when it has silently tested nothing, and one chunked run in this work did test nothing before the counter was added.

*Remark* 7.8 (what would settle the rest, and what it would cost). Three further levels would move the bounds, and all three were priced on the machine used here rather than guessed; none was run. The chain search at $L = 7$ over $\mathbb{F}_3$ in dimension five would decide whether $\mathrm{SSLP}_3(12,5,5)$ is 7 or 8 and would settle $\mathrm{SSLP}_3(11,5,5)$ to within one: level sizes grow by a factor of about 17 ($131\,433 \to 2\,262\,515$), so that level holds roughly $3.8 \times 10^7$ value sets, and at the measured 0.69 milliseconds per set scaled by the measured per-set growth of 2.8 it costs about 21 CPU-hours. The fixed-code search for $[12,6,6]_3$ at $L = 10$ is $\binom{352}{4} = 628\,828\,200$ candidate sets, 87.25 times the $L = 9$ count, about 3.9 CPU-hours after the rewrite; it would give $\mathrm{SSLP}_3(12,6,6) \ge 11$. The same search

for $[10,5,5]_3$ at $L=10$ is $\binom{111}{5} = 128\,164\,707$ sets per representative, about 16 CPU-hours, and is no longer needed.

## 8. What the values are worth for multiplication

The cells of Table 1 are tabulated in [1] because they bound the cost of multiplication in a small algebra. Lemma 11 of that paper reads, verbatim:

> "The minimum number of operations needed to compute multiplication in a field or semifield of degree $n$ over $\mathbb{F}_q$ using $r$ multiplications is at least
>
> $$rM + \big(3\,\mathrm{SSLP}_q(r,n,n) + n - r\big)A.$$"

Here $M$ counts multiplications and $A$ additions in $\mathbb{F}_q$; the three copies of SSLP are the three matrices $L^{\mathsf{T}}$, $R^{\mathsf{T}}$ and $P$ of a rank-$r$ decomposition, and the term $n-r$ is the transposition principle applied to $P$. Substituting the values established above is two lines of arithmetic, and it is worth doing because the last of the three open problems with which [1] closes reads, verbatim:

> "Determine whether or not there exists an algorithm for a semifield of order $3^5$ requiring $10M$ and fewer than $43A$ operations, or $11M$ and fewer than $44A$."

**Proposition 8.1** (proved on paper; not part of the formal development)**.** *Grant Lemma 11 of* [1] *and the values of Remarks 7.3, 7.5 and 7.6. Then the minimum number of operations needed to compute multiplication in a field or semifield of order* $3^5$ *using* $r$ *multiplications is at least*

$$10M+25A \quad (r=10), \qquad 11M+15A \quad (r=11),$$

$$12M+14A \quad (r=12), \qquad 13M+13A \quad (r=13);$$

*and in a field or semifield of order* $3^6$ *using* 12 *multiplications it is at least* $12M+24A$*. Moreover Lemma 11 cannot give more than* $10M+25A$ *at* $r=10$*,* $n=5$*, nor more than* $12M+36A$ *at* $r=12$*,* $n=6$*, whatever further values of* $\mathrm{SSLP}_3$ *become known.*

*Proof.* At $q=3$, $n=5$ the lemma reads $rM+(3\,\mathrm{SSLP}_3(r,5,5)+5-r)A$. With $\mathrm{SSLP}_3(10,5,5)=10$ (Remark 7.5) the coefficient at $r=10$ is $3\cdot 10+5-10=25$; with $\mathrm{SSLP}_3(r,5,5)\geq 7$ for $r\leq 13$ (Remark 7.3) it is at least $3\cdot 7+5-r$, that is 15, 14 and 13 at $r=11,12,13$. At $q=3$, $n=6$, $r=12$ the lemma reads $12M+(3\,\mathrm{SSLP}_3(12,6,6)-6)A$, and $\mathrm{SSLP}_3(12,6,6)\geq 10$ (Remark 7.6) makes the coefficient at least $3\cdot 10-6=24$. For the last sentence, $\mathrm{SSLP}_3(10,5,5)=10$ is exact, so $3\cdot 10+5-10=25$ is also the largest value the lemma can take at $r=10$, $n=5$; and $\mathrm{SSLP}_3(12,6,6)\leq 14$ (Theorem 6.1) caps the coefficient at $r=12$, $n=6$ by $3\cdot 14-6=36$. □

Three things about this proposition should be explicit. It is the only numbered statement in this paper that is not machine-checked: Lemma 11 is quoted from [1] and used, not verified here, and the values it is fed with are the computations of Section 7, which are outside the formal development as well. Its inputs at $r=11,12,13$ and at $n=6$ are lower bounds, so the conclusions there are lower bounds on a lower bound and would improve if those cells were settled. And it bounds algorithms from below only: nothing here says that an algorithm attaining any of these counts exists.

With that said, the arithmetic answers a question [1] could not. For the two cases its third open problem names, the published data supported $10M+10A$ at $r=10$ — its own $\mathrm{SSLP}_3(10,5,5)\geq 5$ substituted into Lemma 11 — and nothing at all at $r=11$, since Table 1 carries no lower bound at $[11,5,5]_3$. The windows those questions live in therefore narrow from "at least $10A$, fewer than $43A$" to "at least $25A$, fewer than $43A$", and from "no bound, fewer than $44A$" to "at least $15A$, fewer than $44A$"; neither is settled. At order $3^6$ the paper has no dimension-six row to substitute at all, while its own table of known tensor ranks, [1, Table 1], gives the rank of $\mathbb{F}_{3^6}$ as lying in $[12,15]$, so $r=12$ is a live case and $12M+24A$ is a bound it could not state. Finally, the exact value at $r=10$ prices the method itself: the algorithm [1] exhibits for a semifield of order $3^5$ uses $10M+43A$, and Lemma 11 can never reach past $10M+25A$ there, so whatever closes that gap will not be this lemma.

### Changes from version 1 of this paper

Nothing stated in version 1 is withdrawn: every theorem, proposition and corollary of that version stands here with its statement unchanged. What is added is Theorem 5.6 with Section 5.1, Theorem 6.1 with Section 6, Remarks 7.3–7.8, and Proposition 8.1 with Section 8; the abstract, the introduction, Table 2 and Section 9 are rewritten to carry the two new cells, the two cells now known exactly, and the two further Lean modules.

Version 1's own text is changed rather than added to in three places, and we label all three. The level $L = 9$ that Remark 7.2 said was affordable and had not been run has since been run, so that remark's closing sentence now points at Remark 7.5 instead of leaving the level open; and its estimate of half an hour of CPU time per representative turns out to have been right only after the rewrite reported in Remark 7.7, the same level having cost about 1.9 CPU-hours per representative before it. The section of exhaustive computations, which held two of them in version 1, now holds four and is retitled accordingly. And two sentences describing how [1] arrives at the four upper bounds of its $q = 3$, $k = 5$ row — one in Section 1, one opening Section 5 — are weakened: version 1 said that all four numbers *are* addition counts of $L$-matrices printed in that paper's Table 7, whereas the entry at $r = 12$ follows from the count at $r = 11$ only through the monotonicity of Lemma 2.4, and the entry at $r = 10$ is the count for a rank-10 algorithm that [1] reports in its text rather than in that table.

Two version-1 statements are weaker than what is now known and are deliberately left standing as they were, being superseded rather than corrected: the bound $\mathrm{SSLP}_3(r, 5, 5) \geq 6$ of Remark 7.1, which Remark 7.3 raises to 7, and the bracket $\mathrm{SSLP}_3(10, 5, 5) \in [9, 12]$ of Remark 7.2, which Remark 7.5 closes to the single value 10. Both remain true as printed.

## 9. Verification

Every statement above labelled theorem, proposition or corollary is machine-checked in Lean 4, in six modules that import no mathematical library at all — not Mathlib, not anything but the repository's own tagging module — with the single exception of Proposition 8.1, which says in its own statement that it is proved on paper and which rests on a lemma of [1] that we do not verify. A vector of $\mathbb{F}_q^k$ is a list of $k$ natural numbers and all arithmetic is natural-number arithmetic modulo $q$, which is what keeps the exhaustive sweeps affordable: the whole development compiles in about five minutes of CPU time, and no module in it needs as much as three gigabytes of memory — the heaviest is the dimension-six check, whose evaluations walk the 728 nonzero dual vectors of $\mathbb{F}_3^6$.

Every witness — the three-addition program of Theorem 3.1, the three $\mathbb{F}_3$ programs of Section 5, the two ten-addition programs of Theorem 5.6, the fourteen-addition program of Theorem 6.1, the transcribed programs of Propositions 4.2 and 5.4, and every negative control — is verified by the kernel's own evaluator and depends on the single axiom `propext`. So do the identifications: the weight enumerators quoted in the proofs of Theorems 5.6 and 6.1, and the counts 242 and 728 of nonzero dual vectors that make them exhaustive, are kernel computations, the two cardinalities depending on no axiom at all. Three statements are run by Lean's compiled evaluator instead: Theorem 3.3, Proposition 4.1, and the nine-column control quoted in the proof of Theorem 5.6, whose sweep covers 817 190 multiplicity vectors. Each of the three carries one evaluation axiom per sweep — three per field for Theorem 3.3 — and the first two, together with Corollary 3.4, which rests on Theorem 3.3, depend additionally on `Classical.choice` and `Quot.sound`, which the nine-column control does not use. The completeness of the two enumerators used by the sweeps is proved in the kernel rather than assumed, so the passage from "the enumeration found nothing" to "no program of that length exists" is not part of the trusted base. There is no `sorry` anywhere in the development and it declares no axiom of its own.

The computations of Section 7 are outside this development, as their statements say. They were written independently in C from Definition 2.1, and each of them had to reproduce the previous run's numbers — level sizes, leaf-node counts, the 576 frame-containing $[10, 5, 5]_3$ codes in their two

orbits — before its new level was believed; the $[12, 6, 6]_3$ lower bound was run to completion by two implementations of the inner search (Remark 7.7). The repository is private: repository reference to be added at submission.

## References


[1] J.-G. Dumas, S. Lia and J. Sheekey, *Computational Explorations on Semifields*, arXiv:2602.09577v1 (cs.SC), submitted 10 February 2026. Title and author list are those of the e-print's own title block and of the arXiv metadata. All quotations, table values and lemma statements attributed to [1] above are from v1, which the arXiv listing gave as the only version on 3 September 2026; no journal version is recorded for it in the arXiv metadata. Numbered items are cited by the numbers they carry in the compiled v1 e-print: Table 1 (known tensor ranks), Lemma 9 (the partially symmetric tensor rank of $\mathbb{F}_{3^6}$, quoted in Section 6), Definition 10 (the definition of $\mathrm{SSLP}_q(r, k, d)$), Lemma 11 (the operation count for multiplication), Lemma 16 (no repeated columns in a $[2k, k, k]_q$ code), Theorem 17 and Theorem 18 ($[8, 4, 4]$), Lemma 21 ($[13, 5, 5]_2$), Lemma 22 ($[9, 4, 4]$), Lemma 23 ($[10, 4, 4]$), Table 3 (the table reproduced here as Table 1) and Table 7 (multiplication in extensions with 243 elements). Those numbers were read off a build of the e-print's own TEX source; the labelled ones agree with the cross-reference file it writes. Lemmas 22 and 23 are in Appendix A, in a source file of the e-print separate from the section that refers to them, and Lemma 9 carries no label of its own.

[2] J.-G. Dumas, C. Pernet and A. Sedoglavic, *Towards automated generation of fast and accurate algorithms for recursive matrix multiplication*, arXiv:2506.19405v1 (math.NA, cross-listed cs.SC), 24 June 2025. The library behind the searches of [1]. It uses the phrase "the shortest straight-line program for its computation" for a *given* linear operator, citing [3] for NP-hardness; it does not define or tabulate $\mathrm{SSLP}_q(r, k, d)$.

[3] J. Boyar, P. Matthews and R. Peralta, *Logic minimization techniques with applications to cryptology*, Journal of Cryptology **26** (2013), no. 2, 280–312; `doi:10.1007/s00145-012-9124-7`. The source of the Shortest Linear Straight-Line Program problem over GF(2) and of the heuristic and SAT-based literature that grew from it; the problem there is to minimise operations for a fixed matrix.

[4] The "XOR count" line for lightweight linear layers continues [3] in the same fixed-matrix, characteristic-two setting; see for instance B. Xu and X. Sun, *Utilizing Circulant Structure to Optimize the Implementations of Linear Layers*, arXiv:2511.18226v2 (cs.CR), 23 November 2025, revised 24 May 2026, and S. Mesnager, K. H. Kim, D. Jo, J. Choe, M. Han and D. N. Lee, *A Proof of the Beierle–Kranz–Leander Conjecture related to Lightweight Multiplication in* $\mathbb{F}_{2^n}$, arXiv:1812.09666v1 (cs.IT), 23 December 2018. Listed to mark the boundary of the literature search: none of this line concerns the minimum over all codes with given parameters, and none of it works over $\mathbb{F}_3$.

[5] M. van Eupen, *Four nonexistence results for ternary linear codes*, IEEE Transactions on Information Theory **41** (1995), no. 3, 800–805; `doi:10.1109/18.382031` (bibliographic data confirmed against Crossref and dblp). Cited by [1] for the uniqueness of the $[10, 5, 5]_3$ code; the result is *not* used in Remark 7.2.

[6] V. Pless, *On the uniqueness of the Golay codes*, Journal of Combinatorial Theory **5** (1968), no. 3, 215–228; `doi:10.1016/S0021-9800(68)80067-5` (bibliographic data confirmed against Crossref, and matching the entry in the bibliography of [1]). Cited by [1] for the uniqueness of the $[12, 6, 6]_3$ code up to equivalence; used here only where Remark 7.6 says so, and not at all for Theorem 6.1.

[7] M. Grassl, *Bounds on the minimum distance of linear codes and quantum codes*, `http://www.codetables.de`. The standard tables; the entries for binary $[12, 5]$ and $[13, 5]$ codes used in Remark 5.5 were consulted on 3 September 2026.

[8] L. de Moura and S. Ullrich, *The Lean 4 theorem prover and programming language*, in: A. Platzer and G. Sutcliffe (eds.), Automated Deduction — CADE 28, Lecture Notes in Computer Science **12699**, Springer, Cham, 2021, pp. 625–635; `doi:10.1007/978-3-030-79876-5_37`.