CERTIFYING THE NEAR-OPTIMALITY OF BYTE-PAIR ENCODING ON NATURAL LANGUAGE CORPORA

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models operate on tokens, contiguous strings of bytes drawn from a finite token alphabet. The token alphabet is typically chosen so that a training corpus is represented with as few tokens as possible. Byte-pair encoding (BPE) is the most widely used procedure for inducing a token alphabet from data. It greedily merges tokens until a pre-determined vocabulary size is reached. The goal of this paper is to construct a scalable optimization algorithm that certifies how close to optimal a token alphabet, such as the one BPE learns, is on training corpora of realistic size. Our algorithm is based on dual decomposition, and it bounds the gap between standard tokenizers and the optimal token alphabet. To do so, we cast three previously proposed, declaratively specified tokenization objectives—direct vocabulary selection, optimal pair encoding, and optimal merge sequences—as integer linear programs (ILPs). For the first two formulations, dual decomposition allows us to exploit the fact that subproblems of the ILPs can be solved with dynamic programming. On ClimbMix—a common LM pretraining dataset—BPE's token count on held-out data is at most 3.45% above the DVS optimum, with the gap shrinking to 1.80% as vocabulary size increases, and BPE remains $2.5\times$ to $5.3\times$ faster. These results show that, under the evaluated constraints, BPE achieves near-optimal compression at substantially lower computational cost.

1 INTRODUCTION

Language models do not generate text byte by byte. Instead, a tokenizer first maps a string of bytes to a sequence of tokens, contiguous strings of bytes, and the model is trained and evaluated on these strings of tokens. The most widely applied principle for choosing the token alphabet is compression: among token alphabets of a given size, prefer the one under which a training corpus is represented with the fewest tokens (Zouhar et al., 2023). Fewer tokens mean more text per context window and less computation per byte, and better compression correlates with better downstream performance (Gallé, 2019; Goldman et al., 2024). The most widely used heuristic for inducing a token alphabet is byte-pair encoding (BPE; Gage, 1994; Sennrich et al., 2016), which starts from single bytes and repeatedly merges the most frequent adjacent pair of tokens in the training corpus into a new token. The algorithm halts when the vocabulary reaches a target size. Each merge is a locally greedy step toward representing the corpus with fewer tokens, but greedy steps need not add up to the best vocabulary. This paper asks how much BPE's greediness costs, i.e., how many more tokens BPE uses than the best vocabulary of the same size.

Answering this question requires more machinery than simply finding a better vocabulary. First, it is not *a priori* clear what optimization problem BPE solves, since it is *defined* as a greedy procedure rather than by a declarative objective. BPE differs from an unrestricted vocabulary in three ways: it chooses merges greedily, it applies each merge to every occurrence in the corpus, and it builds each new token by concatenating two existing tokens. Relaxing these restrictions one at a time yields three objectives that have been proposed in the literature (Figure 1): (i) **optimal merge sequences** (OMS; Zouhar et al., 2023), which drop only the greedy choice; (ii) **optimal pair encoding** (OPE; Kozma & Voderholzer, 2026), which also allows a merge to apply at only some occurrences; and (iii) **direct vocabulary selection** (DVS; Whittington et al., 2025), which drops the pair-construction requirement entirely and allows any substrings as tokens, up to the vocabulary budget. Each problem's feasible set contains the previous one's, so a lower bound for DVS bounds the gain available to *any* of these methods, including ones that abandon BPE's construction rules. For OPE, Kozma

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

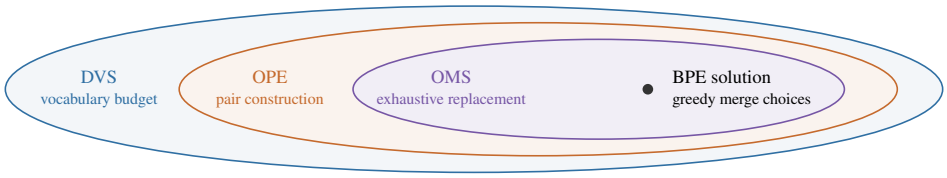


Figure 1: OPE adds pair construction to DVS; OMS additionally requires exhaustive replacement. The point is one OMS solution returned by greedy BPE. Problem parameters are matched as specified in § B.5.

& Voderholzer (2026) prove that BPE is at least $1/3$-optimal, i.e., on every input text, the number of symbols BPE's merges save is at least a third of the number saved by the optimal pair encoding. However, this is a *worst-case* guarantee specifically for OPE, so the question remains whether BPE compresses even better in practice on natural language corpora and whether other objectives, e.g., DVS and OMS, admit similar bounds.

Second, a better vocabulary alone cannot tell us how far BPE is from optimal. Search procedures that improve on BPE's greedy one (Tempus et al., 2026) find vocabularies that use fewer tokens, which shows that *at least* some improvement is possible. But any vocabulary we find gives only an *upper* bound on the minimum token count; it cannot tell us how much further improvement remains. For that we need a *lower* bound on the minimum token count, which in turn caps the improvement any method can achieve over BPE. Like prior work (Lim et al., 2025; Tempus et al., 2026; Singh et al., 2026), we cast tokenization as an integer linear program (ILP), and we likewise focus on its linear programming (LP) relaxation, whose optimal value is exactly such a lower bound. However, unlike previous work, we derive a custom solver that is memory-efficient enough to scale to large corpora. Prior work hands the LP to an off-the-shelf solver, which limits the size of the corpus it can handle: Singh et al. (2026) restrict their LP to the 100k–200k most frequent pretokens,[1] and their largest solve still takes about 94 hours; Tempus et al. (2026) solve a single LP over a sample of roughly 600k ClimbMix documents.[2]

Our scalable solver is based on dual decomposition (Dantzig & Wolfe, 1960; Geoffrion, 1974; Rush & Collins, 2012). The key observation is that, for OPE and DVS, the ILP couples strings only through the shared vocabulary. In other words, once the vocabulary is fixed, it decomposes into independent subproblems, one per string, and finding each string's minimum-token segmentation is a shortest-path problem that an exact dynamic program solves efficiently.[3] We relax the constraints that tie each string's segmentation to the shared vocabulary, moving them into the objective with one Lagrange multiplier per token occurrence. Given the multipliers, every string can be segmented independently, so the millions of distinct strings in a corpus are processed in large GPU batches, while the vocabulary is chosen separately by keeping the tokens with the largest summed multipliers. The multipliers are then updated by subgradient ascent to penalize disagreement between the segmentations and the vocabulary, and the process repeats. Every iteration yields both a dual and a primal solution. The dual solution, i.e., the current multipliers, gives a lower bound on the minimum token count; the primal solution, i.e., the selected vocabulary, gives an upper bound, since its token count can be computed exactly with the dynamic program. When the two bounds are close, the vocabulary is certified to be near-optimal without the true optimum ever being computed. To further reduce memory, we introduce an active set, i.e., multipliers are stored only for a working set of token occurrences, and all others are treated as zero. Iterations restricted to the working set guide the optimization, while the lower bounds we report come only from periodic sweeps over the full problem, so they remain valid.

We experiment on ClimbMix, a language-model pretraining corpus (Diao et al., 2025) that consists of 28.2 GB of text. Because adding a token to a vocabulary never increases the minimum token

[1] Before learning a token vocabulary, the text is **pretokenized**. In practice, this means that a regular expression splits it into coarse chunks, called **pretokens**, e.g., words, numbers, punctuation, and runs of whitespace. Tokens may not cross pretoken boundaries, so each distinct pretoken can be segmented independently.

[2] Their implementation (`github.com/JanTempus/tokenisation_lp`), by default, further restricts the maximum token length to at most five bytes. Consistent with this, most tokens in their released ClimbMix vocabularies are longer than five bytes, e.g., 10080 of 16384.

[3] The same does not hold for OMS, whose exhaustive merges make each step depend on the corpus state left by the previous merges (§4); thus, while we give an ILP for OMS, we cannot give a bespoke scheme based on dual decomposition as we do for OPE and DVS.

count—even on held-out data—token counts are only comparable between vocabularies of the same size. We therefore compare methods for a fixed token alphabet size K. In particular, we consider $K \in \{4096, 8192, 16384, 32768, 65536\}$. For $K = 4096$, we find that BPE uses at most 3.45% more tokens than the optimal DVS vocabulary of the same size at. However, at $K = 65536$, this bound decreases to 1.80%. Indeed, we observe a general monotonic decrease in upper bound as a function of K, suggesting that BPE becomes more optimal for larger token alhabets. Importantly, our dual decomposition scheme only takes 2.5–5.3× more wall-clock time to obtain. Our results lead us the titular conclusion that—on large, natural language corpora—BPE is nearly optimal.

2 Tokenization as an Optimization Problem

2.1 Strings and Tokenizers

Let Σ be a finite, non-empty **base alphabet**, e.g., the set of Unicode characters or the set of bytes $\{0, \ldots, 255\}$. Write Σ^* for all finite strings over Σ, including the empty string ε, and $\Sigma^+ \stackrel{\text{def}}{=} \Sigma^* \setminus \{\varepsilon\}$ for the non-empty strings. A string $\boldsymbol{\sigma} = \sigma_1 \cdots \sigma_N$ has base symbols $\sigma_i \in \Sigma$ and length $|\boldsymbol{\sigma}| = N$.

A **token alphabet**, or vocabulary, is a finite set $\Sigma \subseteq \Delta \subset \Sigma^+$ of **tokens** δ, identifying base symbols with one-symbol strings. Including the alphabet ensures that every input can be segmented. Write Δ^* for all finite token sequences, including the empty sequence. For $\boldsymbol{\delta} = \delta_1 \cdots \delta_J \in \Delta^*$, $|\boldsymbol{\delta}| = J$ counts tokens, whereas $|\delta_j|$ counts base symbols.

Definition 1. *The **decoder** $\kappa\colon \Delta^* \to \Sigma^*$ concatenates the base strings in a token sequence: $\kappa(\boldsymbol{\delta}) = \delta_1 \cdot \delta_2 \cdots \delta_J$, with the empty sequence decoding to ε. A **tokenizer** is determined by its **encoder** $\tau\colon \Sigma^* \to \Delta^*$ (Gastaldi et al., 2025). It is **exact** if decoding reverses encoding:*

$$\kappa(\tau(\boldsymbol{\sigma})) = \boldsymbol{\sigma} \qquad \text{for every } \boldsymbol{\sigma} \in \Sigma^*. \tag{1}$$

Exactness permits multiple segmentations: BPE's merge-order encoder may differ from minimum-token segmentation over the same vocabulary. § B.4 extends the notation to distinct OMS token identities sharing a decoded string; decoding concatenates their decoded strings.

2.2 Corpus and Problem Parameters

Before optimization, we fix a partition of the corpus into strings, or pretokens. Tokens cannot cross these boundaries, but each pretoken may contain several learned tokens. All pretokens share one vocabulary. Repeated strings may be stored once, with their corpus frequencies recorded as multiplicities. Let $D = (\boldsymbol{\sigma}^{(1)}, \ldots, \boldsymbol{\sigma}^{(M)})$ contain $M \geq 1$ non-empty corpus strings, with lengths $N^{(m)} = |\boldsymbol{\sigma}^{(m)}|$ and multiplicities $c^{(m)} \in \mathbb{Z}_{>0}$ (all one when occurrences are listed separately). Empty strings contribute no tokens and are omitted. Number boundaries from 0 to $N^{(m)}$ and write $\boldsymbol{\sigma}^{(m)}_{i:j} \stackrel{\text{def}}{=} \sigma^{(m)}_{i+1} \cdots \sigma^{(m)}_{j}$ for the substring between boundaries $0 \leq i < j \leq N^{(m)}$. The vocabulary budget is a nonnegative integer K bounding optional tokens: $|\Delta \setminus \Sigma| \leq K$. Mandatory alphabet symbols do not count toward K; each token created by a merge does, even if unused in the final segmentation. For a nonnegative integer n, write $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$ and $[n]_0 \stackrel{\text{def}}{=} \{0, \ldots, n\}$, with $[0] = \varnothing$.

The three optimization problems we study, DVS, OPE, and OMS (formalized in §3), all minimize the same empirical token count,

$$\sum_{m=1}^{M} c^{(m)} \left| \tau(\boldsymbol{\sigma}^{(m)}) \right|, \tag{2}$$

by choosing a shared vocabulary and exact encoder subject to the budget and construction rules. DVS and OPE permit minimum-token segmentation; OMS follows a shared sequence of exhaustive merges. Our main empirical comparison scores every vocabulary, including BPE's, by minimum-token segmentation, isolating vocabulary quality under a common encoding rule.

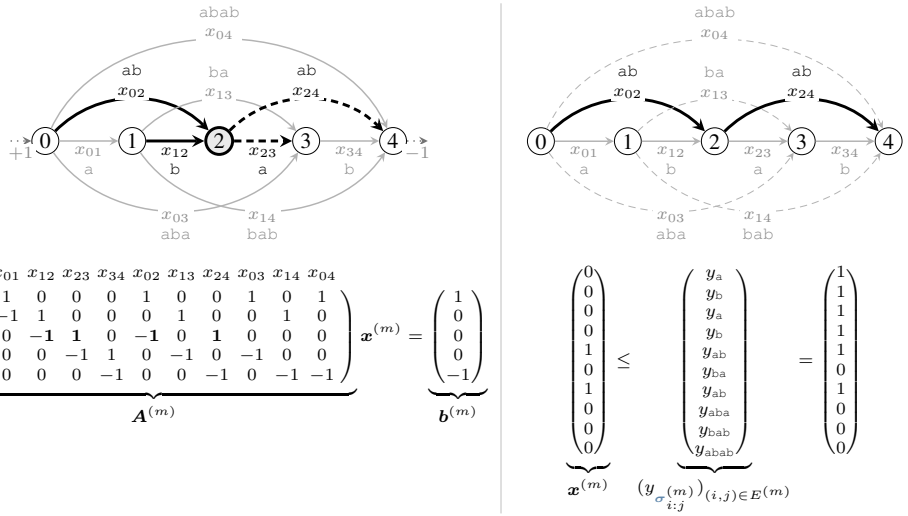


Figure 2: Lattice constraints for `abab` with all substrings as candidates. Left: boundary 2 and its flow-conservation row, with entering arcs contributing -1 and leaving arcs $+1$. Right: selecting `ab` enables both occurrences through the shared variable $y_{\texttt{ab}}$; the bold path gives the shortest segmentation. Thin dashed arcs correspond to unselected tokens.

3 The Optimization Landscape

The objective in (2) couples vocabulary selection with corpus segmentation. We first formulate DVS as an ILP, then add pair-construction and exhaustive-replacement constraints to obtain OPE and OMS. The resulting formulations make the hierarchy of feasible solutions explicit and reveal the structure used by the decompositions in §4.

3.1 Direct Vocabulary Selection

DVS jointly chooses a vocabulary and corpus segmentations to minimize token count. Fix a finite set of candidate strings $\Sigma \subseteq \Omega \subset \Sigma^+$, with optional tokens $\Omega^- \stackrel{\text{def}}{=} \Omega \setminus \Sigma$. A binary variable y_δ selects each candidate token (4g); all alphabet symbols are mandatory (4e), and at most K optional tokens may be selected (4d). This vocabulary is shared across the corpus.

For each corpus string, represent a candidate-token occurrence by an arc between its starting and ending boundaries:

$$E^{(m)} \stackrel{\text{def}}{=} \{(i,j) : 0 \le i < j \le N^{(m)},\ \boldsymbol{\sigma}_{i:j}^{(m)} \in \Omega\}. \tag{3}$$

The binary variable $x_{ij}^{(m)}$ selects that occurrence (4f). A complete segmentation is a path from boundary 0 to boundary $N^{(m)}$. In Figure 2, selecting `ab` uses one vocabulary slot and enables both occurrences in `abab`, giving the two-token path `ab` | `ab`.

The node–arc incidence matrix $\boldsymbol{A}^{(m)}$ has $+1$ where an arc starts and -1 where it ends. The vector $\boldsymbol{b}^{(m)}$ injects one unit of flow at boundary 0 and removes it at $N^{(m)}$, so the flow-conservation constraint $\boldsymbol{A}^{(m)}\boldsymbol{x}^{(m)} = \boldsymbol{b}^{(m)}$ in (4b) enforces a complete path. The linking constraint $x_{ij}^{(m)} \le y_{\boldsymbol{\sigma}_{i:j}^{(m)}}$ in (4c) permits an occurrence only if its token type is selected. Thus both occurrences of `ab`, and all its occurrences elsewhere in the corpus, depend on the same $y_{\texttt{ab}}$.

For token count, each selected arc costs the string's multiplicity, $w_{ij}^{(m)} = c^{(m)}$. Stacking these costs as $\boldsymbol{w}^{(m)}$ gives the objective of the complete ILP in Figure 3.

Write $X^{(m)}$ for the binary path set defined by (4b). Fixing $\boldsymbol{y}$ leaves an independent shortest-path problem for each string, solvable by Viterbi. The difficulty is choosing the shared vocabulary: the linking constraints (4c) couple all these paths to the same budgeted selection. § 4.1 exploits precisely this structure. The following proposition states that Program (4) is exact.

DVS

$$\min_{\boldsymbol{x},\boldsymbol{y}} \quad \sum_{m=1}^{M} \boldsymbol{w}^{(m)\top}\boldsymbol{x}^{(m)} \qquad \text{(minimize total token cost)} \qquad (4a)$$

$$\text{s.t.} \quad \boldsymbol{A}^{(m)}\boldsymbol{x}^{(m)} = \boldsymbol{b}^{(m)} \qquad \forall m \in [M] \qquad \text{(flow conservation)} \qquad (4b)$$

$$x_{ij}^{(m)} \le y_{\boldsymbol{\sigma}_{i:j}^{(m)}} \qquad \forall m,\ (i,j) \in E^{(m)} \qquad \text{(arcs need selected tokens)} \qquad (4c)$$

$$\sum_{\delta \in \Omega^-} y_\delta \le K \qquad \text{(vocabulary budget)} \qquad (4d)$$

$$y_\sigma = 1 \qquad \forall \sigma \in \Sigma \qquad \text{(alphabet always selected)} \qquad (4e)$$

$$\boldsymbol{x}^{(m)} \in \{0,1\}^{E^{(m)}} \qquad \forall m \in [M] \qquad \text{(binary arcs)} \qquad (4f)$$

$$\boldsymbol{y} \in \{0,1\}^{\Omega} \qquad \text{(binary tokens)} \qquad (4g)$$

Figure 3: DVS ILP. Blue marks the backbone inherited by the later programs.

Proposition 1 (DVS ILP). *The integer feasible points of Program* (4) *correspond to the pairs of a vocabulary* Δ *with* $\Sigma \subseteq \Delta \subseteq \Omega$ *and* $|\Delta \setminus \Sigma| \le K$ *and a segmentation of every corpus string into tokens of* Δ*, and corresponding points have equal objective value.*

We prove Proposition 1 in § B.2, which also extends the objective to general additive token costs.

3.2 Optimal Pair Encoding

Optimal pair encoding (OPE) chooses at most K merges, each creating a token from two previously available tokens. A merge may replace any non-overlapping subset of matching occurrences (Kozma & Voderholzer, 2026). DVS can add `abc` for one vocabulary slot; OPE must also construct `ab` or `bc` as an intermediate token. That intermediate consumes another slot even if it is unused in the final segmentation.

Reuse the candidate strings and occurrence arcs from DVS. Let $\mathcal{R}$ be the set of allowed merges $(a, b \to g)$, where $a, b \in \Omega$, $g \in \Omega^-$, and $g = ab$. Write $L(r)$, $R(r)$, and $O(r)$ for merge r's left input, right input, and output. A binary variable u_r chooses a merge as a token's **creator**. OPE adds just these construction constraints to the direct ILP:

$$\sum_{r \in \mathcal{R}: O(r) = g} u_r = y_g \qquad \forall g \in \Omega^-, \qquad (5a)$$

$$u_r \le y_{L(r)}, \qquad u_r \le y_{R(r)} \qquad \forall r \in \mathcal{R}. \qquad (5b)$$

The first row chooses one creator for each selected optional token; the second requires both inputs to be selected. Such a vocabulary is **merge-closed**: every optional token can be built recursively from selected shorter tokens, down to the alphabet. One creator per optional token also makes the vocabulary budget a merge budget.

An OPE sequence may create two distinct tokens that decode to the same string, e.g., by building `abc` both as `ab` + `c` and as `a` + `bc`. Such duplicates never help.

Lemma 1 (One token per string). *Every at-most-K OPE solution can be rewritten so that distinct merges create tokens with distinct decoded strings. The rewriting uses no more merges and preserves every final token boundary, hence the total token count.*

Consequently, for the stated candidate strings and allowed splits, this representation exactly characterizes OPE, with one token per decoded string and the same final-path variables as DVS.

Proposition 2 (OPE ILP). *Fix candidate strings* Ω *and allowed string merges* $\mathcal{R}$*. Every at-most-K OPE solution whose created tokens decode to strings in* Ω *and whose splits belong to* $\mathcal{R}$ *maps to an integer feasible point of Program* (18) *with the same total token count, and every integer feasible point of Program* (18) *maps to such an OPE solution with the same final segmentation.*

Corollary 1 (Completeness of the substring formulation). *Let* Ω *contain the alphabet and every distinct nonempty corpus substring, with one candidate per string, and let* $\mathcal{R}$ *contain every split*

$(a, b \to g)$ *with* $a, b, g \in \Omega$ *and* $g = ab$. *Program* (18) *then attains the unrestricted at-most-*K *OPE optimum.*

We prove Lemma 1, Proposition 2, and Corollary 1 in § B.3. Partial replacement can realize any final segmentation over the selected merge-closed vocabulary. Construction therefore changes the global selection problem while leaving fixed-vocabulary segmentation as a shortest-path problem.

3.3 Optimal Merge Sequences

Optimal merge sequences (OMS) optimize the sequence of at most K merge operations that BPE chooses greedily (Zouhar et al., 2023; Whittington et al., 2025). Each chosen merge must replace all currently matching pairs, scanning left to right to resolve overlaps. Earlier choices therefore determine which later merges can act. For example,

$$\texttt{a} \mid \texttt{b} \mid \texttt{c} \mid \texttt{d} \xrightarrow{\texttt{b+c}\to\texttt{bc}} \texttt{a} \mid \texttt{bc} \mid \texttt{d}. \tag{6}$$

After this step, neither `ab` nor `cd` can be formed in this string. Starting with those two merges instead produces `ab` | `cd`. OPE may skip the `bc` occurrence; OMS must apply it if that merge is scheduled first.

Our OMS ILP, Program (24), consequently tracks a shared merge schedule and each string's segmentation after every stage.

Proposition 3 (OMS ILP)**.** *The integer feasible points of Program* (24) *correspond to the sequences of at most* K *allowed merges, padded with no-ops to length* K*, in which each merge is applied exhaustively with the left-to-right overlap convention; corresponding points have objective value equal to the final multiplicity-weighted token count.*

We prove Proposition 3 in § B.4. A final vocabulary and path alone do not specify a valid execution. These intermediate states prevent the same decomposition into Viterbi and vocabulary selection from carrying over (§4).

When the corpus boundaries, candidates, permitted merges, and budget are matched, the three formulations satisfy

$$\mathrm{OPT}_{\mathrm{direct}} \;\le\; \mathrm{OPT}_{\mathrm{OPE}} \;\le\; \mathrm{OPT}_{\mathrm{OMS}} \;\le\; L_{\mathrm{BPE}}, \tag{7}$$

where L_{BPE} counts the tokens produced by BPE's merge sequence. § B.5 gives the precise compatibility conditions. A lower bound for DVS therefore also bounds the potential improvement over BPE in the more restrictive construction models.

4 Dual Decomposition for DVS and OPE

Once a vocabulary is fixed, each corpus string can be segmented independently. The difficulty is choosing the shared vocabulary: tokens useful in different parts of the corpus compete for the same limited budget. Dual decomposition separates local segmentation from global vocabulary selection and uses Lagrange multipliers to coordinate their decisions (Geoffrion, 1974; Rush & Collins, 2012). Minimizing the separated problems gives a lower bound on the optimal token count; recovering and evaluating a feasible vocabulary gives an upper bound. Together, these bounds quantify how much improvement can remain even when the optimal tokenizer is unknown.

4.1 Direct Vocabulary Selection

Fixed-vocabulary Viterbi. For any allowed arc set $F^{(m)} \subseteq E^{(m)}$ and arc costs q_{ij}, minimum-cost segmentation is a shortest path through the string boundaries. The Viterbi algorithm computes it by

$$\alpha_0 = 0, \qquad \alpha_j = \min_{i:(i,j)\in F^{(m)}} \alpha_i + q_{ij}, \quad j = 1, \ldots, N^{(m)}, \tag{8}$$

with backpointers. A vocabulary Δ restricts the arcs to $E^{(m)}(\Delta) = \{(i,j) \in E^{(m)} : \boldsymbol{\sigma}_{i:j}^{(m)} \in \Delta\}$ with costs $w_{ij}^{(m)}$, yielding M independent paths.

Once arcs are enumerated, each Viterbi pass takes $O(N^{(m)} + |F^{(m)}|)$ time. The fixed-vocabulary flow polytope is integral because its node–arc incidence matrix is totally unimodular (Schrijver, 1986, Ch. 19), so relaxing its binary variables does not change the optimum. This property makes segmentation tractable; it does not make joint vocabulary selection integral or submodular.

Relaxing the shared vocabulary. The licensing rows $x_{ij}^{(m)} \le y_{\sigma_{i:j}^{(m)}}$ are the only constraints coupling a corpus path to the shared vocabulary. We relax these rows, attaching a Lagrange multiplier to each candidate-token occurrence in a stored string. Segmentation pays the multiplier of each occurrence it uses, while vocabulary selection earns their sum for each selected token. Attach a multiplier $\lambda_{ij}^{(m)} \ge 0$ to each row and let the aggregated multiplier of token δ be

$$\Lambda_\delta \stackrel{\text{def}}{=} \sum_{m=1}^{M} \sum_{\substack{(i,j)\in E^{(m)},\\ \sigma_{i:j}^{(m)}=\delta}} \lambda_{ij}^{(m)} \ge 0. \tag{9}$$

Here, each multiplier relaxes a licensing constraint for a stored string, aggregating across its repeated copies. For token count, the adjusted arc cost is $c^{(m)} + \lambda_{ij}^{(m)}$, equivalently $c^{(m)}(1 + p_{ij}^{(m)})$ for a per-copy multiplier $p_{ij}^{(m)} = \lambda_{ij}^{(m)}/c^{(m)}$. Moving the relaxed rows into the objective yields

$$g(\boldsymbol{\lambda}) = \sum_{m=1}^{M} \min_{\boldsymbol{x}^{(m)}} \sum_{(i,j)\in E^{(m)}} (w_{ij}^{(m)} + \lambda_{ij}^{(m)}) x_{ij}^{(m)} + \min_{\boldsymbol{y}} \left(- \sum_{\delta\in\Omega} \Lambda_\delta y_\delta \right), \tag{10}$$

where $\boldsymbol{x}^{(m)} \in X^{(m)}$ and $\boldsymbol{y}$ satisfies the budget, alphabet, and binary constraints. The first term finds the cheapest segmentation of each string using the adjusted costs. The second selects a shared vocabulary using the aggregated multipliers.

The vocabulary subproblem keeps the alphabet and the at most K optional tokens with the largest positive Λ_δ, which takes $O(|\Omega|)$ expected time with linear-time selection.

By weak duality, every nonnegative multiplier vector gives a valid lower bound. Maximizing this bound defines the dual optimum $g^\star \stackrel{\text{def}}{=} \max_{\boldsymbol{\lambda}\ge\mathbf{0}} g(\boldsymbol{\lambda})$. Because each segmentation block is integral and the vocabulary block is a cardinality polytope, the integrality property of Lagrangian relaxation (Geoffrion, 1974) implies that $g^\star$ equals the optimal value of the LP relaxation of Program (4), so the decomposition loses nothing relative to the monolithic LP.

The equality concerns the dual optimum: a finite run may return a weaker bound, and the LP relaxation itself may lie below the integer optimum. Because DVS is NP-complete (Whittington et al., 2025) and the LP is solvable in polynomial time, the LP value cannot equal the integer optimum on every instance, and the certificate below is informative only when its observed gap is small.

Optimization and certification. At iteration t, Viterbi and top-K return $\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{y}}_t$ and projected subgradient ascent updates

$$\lambda_{t+1,ij}^{(m)} = \left[\lambda_{t,ij}^{(m)} + \eta_t \big(\hat{x}_{t,ij}^{(m)} - \hat{y}_{t,\sigma_{i:j}^{(m)}} \big) \right]_+. \tag{11}$$

The multiplier rises when a path uses a token rejected by the vocabulary oracle, making that occurrence less attractive to segmentation and its token more attractive to vocabulary selection. The two oracles do not need to agree for their dual value to remain a valid lower bound. Let $D_t \stackrel{\text{def}}{=} \max_{s\le t} g(\boldsymbol{\lambda}_s)$. Re-running Viterbi without multipliers under each selected vocabulary and retaining the best value gives a feasible upper bound P_t. At every iteration,

$$D_t \le \mathrm{OPT}_\Omega \le P_t, \qquad \mathrm{gap}_t \stackrel{\text{def}}{=} \frac{P_t - D_t}{P_t}. \tag{12}$$

The interval $[D_t, P_t]$ contains the unknown optimum of Program (4). Its width bounds the improvement still possible over the returned vocabulary; comparing BPE's token count with D_t similarly bounds the improvement possible beyond BPE. § C.2 gives pseudocode and the standard $O(1/\sqrt{T})$ convergence statement (Polyak, 1969; Nemirovsky & Yudin, 1983).

4.2 Optimal Pair Encoding

The OPE decomposition lets us measure how retaining pair-construction constraints affects compression and its optimality certificate. Pair construction introduces dependencies between vocabulary entries: selecting `abc` requires a way to build it from shorter tokens (§ 3.2). Relaxing these dependencies preserves the separation between final segmentation and global selection.

Proposition 4 (Exact OPE subproblem oracles)**.** *For fixed multipliers, the OPE dual is computed by (i) Viterbi with arc cost $c^{(m)} + \lambda_{ij}^{(m)}$, (ii) one merge scan for each output token's cheapest creator, and (iii) top-K selection by net token profit.*

As for DVS, every nonnegative multiplier vector gives a lower bound for Program (18). After relaxation, the creator block is the independent-set polytope of a truncated partition matroid and hence integral, so by the same integrality property (Geoffrion, 1974) the strongest such bound equals the LP optimum. Feasible vocabulary recovery supplies the corresponding upper bounds; § C.3 gives the multiplier definitions and recovery procedure.

Exhaustive merge sequences. Exhaustive merge sequences additionally require intermediate token states: an earlier merge can destroy an adjacency needed later. Relaxing the schedule links therefore leaves trajectory subproblems that cannot be solved by the final-segmentation Viterbi oracle used above.

5 Experiments

Matched instance. All methods use the same preprocessed corpus and candidate-token restrictions. We pretokenize ClimbMix shards 0–6 (Diao et al., 2025) using the nanochat split pattern, then apply GPT-2 ByteLevel preprocessing. Each pretoken is segmented independently, so tokens cannot cross pretoken boundaries. We store each distinct pretoken once, together with its frequency. These frequencies weight both optimization and evaluation, so deduplication preserves the full-corpus token-count objective. The resulting table contains 1996113 distinct pretokens representing 1.766B bytes of corpus text. Every vocabulary includes all 256 single-byte tokens. Optional tokens are selected from observed substrings within pretokens that are two to five bytes long and occur at least twice. The vocabulary budget K counts only optional token types. We sweep $K \in \{4096, 8192, 16384, 32768, 65536\}$.

Common score. Let $T(\Delta)$ be the multiplicity-weighted token count from exact Viterbi segmentation over Δ. Every returned vocabulary, including BPE's, uses this common score, isolating vocabulary choice from merge-order encoding. We report the feasible improvement over the Viterbi-scored BPE vocabulary as

$$G_q(K) \stackrel{\text{def}}{=} 100\,\frac{T(\Delta_{\mathrm{BPE},K}) - T(\Delta_{q,K})}{T(\Delta_{\mathrm{BPE},K})}. \tag{13}$$

For a formulation q with optimum $T_q^\star$ and DD lower bound L_q, we also report the certified BPE excess

$$100\left(\frac{T(\Delta_{\mathrm{BPE},K})}{T_q^\star} - 1\right) \le C_q(K) \stackrel{\text{def}}{=} 100\left(\frac{T(\Delta_{\mathrm{BPE},K})}{L_q} - 1\right). \tag{14}$$

These percentages answer different questions: G_q is an achieved reduction relative to BPE's token count, while C_q bounds BPE's excess relative to the unknown optimum using a certified lower bound. A 3.45% excess certificate implies that even an optimal method can reduce BPE's token count by at most $1 - 1/1.0345 \approx 3.335\%$ under the same constraints; the observed gains below 2% concern the feasible DD vocabularies.

Methods. BPE is trained once to the largest budget and each prefix vocabulary is rescored by Viterbi. DVS and OPE DD run for 300 rounds, returning feasible vocabularies and lower bounds; § E.4 examines longer runs. The monolithic DVS LP (Tempus et al., 2026) uses top-K recovery. § E.2 compares all LPs and exact OMS on small controlled instances. An active set reduces memory by retaining occurrence-specific multipliers only for a working subset of licensing constraints, with unstored multipliers treated as zero. Restricted iterations guide optimization, while full-oracle

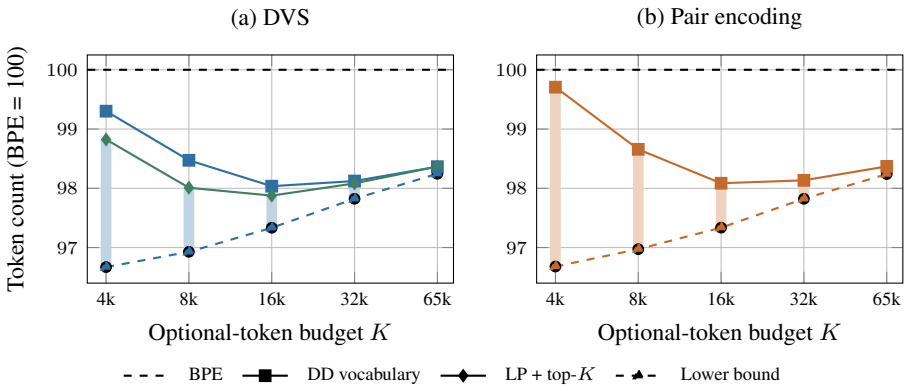


Figure 4: Optimal token-count intervals on 1.766 GB of ClimbMix, normalized to BPE's Viterbi score (100). Each thick segment joins a certified lower bound (triangle) to the best feasible score shown (square or diamond), using the better of DD and LP recovery in panel (a). Lower is better. Certificates apply at the tested budgets; connecting lines guide the eye.

sweeps provide the reported lower bounds. Both DD systems evaluate length-sorted pretokens in GPU batches using a vectorized trie and on-device Viterbi traceback. Streamed execution keeps persistent multipliers in host memory and rebuilds occurrence tensors one batch at a time. § C.2 details the active set representation and certification.

Can Certification Scale? Both DD systems scale to 28.232B bytes of text with roughly 34 GB of peak host memory, whereas the monolithic DVS LP exceeds 476 GB while still building its program on 7.064B bytes, and BPE is 2.5× and 5.3× faster than the two decompositions; § E.1 gives the runtime and memory details.

How Close Is BPE to Each Optimum? Figure 4 brackets each unknown optimum between a lower bound and a feasible token count, with BPE normalized to 100. DVS and OPE DD use 0.29%–1.96% fewer Viterbi tokens than BPE and differ by at most 0.41%. The bounds certify BPE within 3.45% of both matched optima throughout the sweep, tightening to 1.80% at $K = 65536$. At this budget, DVS and OPE primal gaps are 0.12% and 0.13%, supporting the small observed gains with tight bounds on remaining improvement. Fewer tokens need not mean fewer encoded bits: DVS DD improves held-out Huffman compression at $K \geq 16384$, but is worse at smaller budgets despite using fewer tokens (§ E.3). §D explores choosing vocabulary size while accounting for vocabulary storage cost.

6 Conclusion

We developed a shared optimization framework that separates vocabulary selection, pair construction, and exhaustive merging, and identifies which restrictions each optimality certificate addresses. For DVS and OPE, dual decomposition coordinates tractable segmentation and vocabulary-selection problems while preserving the corresponding LP bounds at the dual optimum. This makes certification possible beyond the memory limits of our monolithic implementation; exhaustive OMS retains intermediate segmentation states that prevent the same decomposition. We discuss how this framework relates to prior work on tokenization objectives, vocabulary construction, and optimality certificates in §A.

AI use statement

In this work, we used generative AI tools to assist with implementing portions of the experiments and to review the presentation and internal consistency of the methodology, experiments, and mathematical arguments. We additionally used generative AI tools to edit the manuscript for clarity and concision and to revise LaTeX formatting, figures, captions, cross-references, and bibliography entries. We did not use generative AI to generate experimental data or conduct experiments. All

AI-assisted code and manuscript revisions were reviewed by the authors; the code was tested, experiments were run by the authors, and reported values were checked against the underlying results. We take responsibility for the final content of this work, including all AI-assisted text, claims, code, and artifacts.

REFERENCES

George B. Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960. doi: 10.1287/opre.8.1.101. URL https://doi.org/10.1287/opre.8.1.101.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Yingyan Celine Lin, Jan Kautz, and Pavlo Molchanov. Nemotron-CLIMB: CLustering-based iterative data mixture bootstrapping for language model pre-training, 2025. URL https://arxiv.org/abs/2504.13161.

Philip Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38, 1994. URL https://jacobfilipp.com/DrDobbs/articles/CUJ/1994/9402/gage/gage.htm.

Matthias Gallé. Investigating the effectiveness of BPE: The power of shorter sequences. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China, 2019. Association for Computational Linguistics. URL https://aclanthology.org/D19-1141.

Juan Luis Gastaldi, John Terilla, Luca Malagutti, Brian DuSell, Tim Vieira, and Ryan Cotterell. The foundations of tokenization: Statistical and computational concerns. In *Proceedings of the 13th International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2407.11606.

Arthur M. Geoffrion. Lagrangean relaxation for integer programming. *Mathematical Programming Study*, 2:82–114, 1974.

Omer Goldman, Avi Caciularu, Matan Eyal, Kris Cao, Idan Szpektor, and Reut Tsarfaty. Unpacking tokenization: Evaluating text compression and its correlation with model performance. In *Findings of the Association for Computational Linguistics: ACL 2024*, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-acl.134.

David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. doi: 10.1109/JRPROC.1952.273898.

László Kozma and Johannes Voderholzer. Theoretical analysis of byte-pair encoding. In *34th Annual European Symposium on Algorithms (ESA 2026)*, volume 388 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 80:1–80:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi: 10.4230/LIPIcs.ESA.2026.80. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ESA.2026.80.

Jia Peng Lim, Shawn Tan, Davin Choo, and Hady W. Lauw. A partition cover approach to tokenization. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2501.06246.

Arkadi S. Nemirovsky and David B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. John Wiley & Sons, Chichester, 1983.

Boris T. Polyak. Minimization of unsmooth functionals. *USSR Computational Mathematics and Mathematical Physics*, 9(3):14–29, 1969.

Alexander M. Rush and Michael Collins. A tutorial on dual decomposition and Lagrangian relaxation for inference in natural language processing. *Journal of Artificial Intelligence Research*, 45: 305–362, 2012.

Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, Cambridge, MA, 2010. Association for Computational Linguistics. URL https://aclanthology.org/D10-1001/.

Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, 1986.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany, 2016. Association for Computational Linguistics. URL https://aclanthology.org/P16-1162.

Adhiraj Singh, Deepanshu Mody, Ghina Al Shdaifat, Hamza Alshamy, Adam Wiemerslage, Varshini Reddy, and Craig W. Schmidt. Joint optimization for greedy longest-match tokenization, 2026. URL https://arxiv.org/abs/2607.23362.

Jan Tempus, Philip Whittington, Craig W. Schmidt, Dennis Komm, and Tiago Pimentel. Tokenisation via convex relaxations, 2026. URL https://arxiv.org/abs/2605.22821.

Philip Whittington, Gregor Bachmann, and Tiago Pimentel. Tokenisation is NP-complete. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vienna, Austria, 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.acl-long.1365/.

Vilém Zouhar, Clara Meister, Juan Luis Gastaldi, Li Du, Tim Vieira, Mrinmaya Sachan, and Ryan Cotterell. A formal perspective on byte-pair encoding. In *Findings of the Association for Computational Linguistics: ACL 2023*, Toronto, Canada, 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.findings-acl.38.

A Related Work

Our framework brings together research on compression objectives, vocabulary construction, and optimization guarantees. Empirical links between compression and downstream performance (Gallé, 2019; Goldman et al., 2024) motivate the token-count objective. Zouhar et al. (2023) analyze greedy BPE relative to optimal merge sequences; Kozma & Voderholzer (2026) study approximation and hardness for pair encoding; and Whittington et al. (2025) establish NP-completeness for direct and bottom-up tokenization. We express DVS, OPE, and OMS through shared segmentation variables and explicit construction constraints, making these families comparable and specifying which notion of optimality each certificate addresses.

ConvexTok (Tempus et al., 2026) relates to the DVS instantiation of this framework: it already uses an LP relaxation to learn vocabularies and certify their distance from optimality. We place this approach alongside the OPE and OMS formulations, so that certification can be studied across their different feasible sets. For the DVS instance, our scaling study also exposes a computational bottleneck: the monolithic LP is terminated during construction after using hundreds of gigabytes of memory (§ E.1).

To compute useful bounds at corpus scale, we draw on Lagrangian relaxation (Geoffrion, 1974) and dual decomposition for NLP (Rush et al., 2010; Rush & Collins, 2012), which coordinate tractable component solvers. We derive segmentation and selection oracles for DVS and OPE, avoiding the full programs while preserving their LP relaxation strength at convergence. The framework also explains why these oracles do not carry over to exhaustive OMS: its constraints retain intermediate sequences.

B Token-Count Formulations and Correctness

This appendix proves the DVS formulation in § 3.1 and gives the complete OPE and OMS programs with their correctness proofs. Their shared color scheme records inherited structure: blue marks the DVS backbone, orange pair construction shared by OPE and OMS, and purple the ordered state transitions unique to OMS.

B.1 Shared Segmentation Representation

The bottom half of Figure 2 encodes segmentation paths algebraically. The node–arc incidence matrix $\boldsymbol{A}^{(m)} \in \{-1, 0, 1\}^{(N^{(m)}+1)\times|E^{(m)}|}$ has one row per boundary and one column per arc, with $+1$ where the arc starts and -1 where it ends. Row 2 of the example reads $x_{23}+x_{24}-x_{12}-x_{02} = 0$: the path may use one of the arcs leaving boundary 2 only if it also used one of the arcs entering it. Row k of $\boldsymbol{A}^{(m)}\boldsymbol{x} = \boldsymbol{b}^{(m)}$ is this flow-conservation condition at boundary k, and the right-hand side $\boldsymbol{b}^{(m)}$ supplies the one exception, injecting a unit of flow at boundary 0 and removing it at boundary $N^{(m)}$ (the dotted arcs). The binary solutions

$$X^{(m)} \stackrel{\text{def}}{=} \{\boldsymbol{x} \in \{0,1\}^{E^{(m)}} : \boldsymbol{A}^{(m)}\boldsymbol{x} = \boldsymbol{b}^{(m)}\} \tag{15}$$

are therefore exactly the paths from boundary 0 to boundary $N^{(m)}$, that is, the segmentations of string m.

Vocabulary selection is represented by binary variables y_δ, one for each candidate token. The constraint $x_{ij}^{(m)} \le y_{\sigma_{i:j}^{(m)}}$ permits an occurrence only when its token is selected. Because the corpus fixes $\sigma_{i:j}^{(m)}$, the subscript identifies a fixed vocabulary variable and the constraint is linear. In Figure 2, both x_{02} and x_{24} are bounded by $y_{\texttt{ab}}$; occurrences in other corpus strings use that same selection variable.

Remark 1 (Integrality does not imply submodularity). Integrality of the segmentation problem holds only for a fixed set of available tokens. It does not establish diminishing returns as the vocabulary changes. In fact, the token savings of DVS are not submodular. Let $T(S)$ be the minimum token count with vocabulary $\Sigma \cup S$, and let $F(S) \stackrel{\text{def}}{=} T(\varnothing) - T(S)$. On the single string `abcd`, take $A = \{\texttt{bc}\}$, $B = \{\texttt{bc}, \texttt{cd}\}$, and $e = \texttt{ab}$. Then

$$T(A) = T(A \cup \{e\}) = T(B) = 3, \qquad T(B \cup \{e\}) = 2. \tag{16}$$

Adding ab to A only exchanges one three-token segmentation for another; adding it to B enables ab · cd. Thus

$$F(A \cup \{e\}) - F(A) = 0 < 1 = F(B \cup \{e\}) - F(B), \tag{17}$$

violating diminishing returns although $A \subseteq B$. The cost T itself is not submodular either: adding ab to $\varnothing$ changes T by -1, whereas adding it to $\{\texttt{bc}\}$ changes T by 0. Integrality of each fixed-vocabulary path problem therefore does not make vocabulary selection a submodular optimization problem.

B.2 Direct Vocabulary Selection

Program (4) in § 3.1 combines segmentation paths with a shared vocabulary budget.

Additive token costs. The same program supports any additive cost $w\colon \Sigma^+ \to \mathbb{R}$ by setting $w_{ij}^{(m)} \stackrel{\text{def}}{=} c^{(m)} w(\boldsymbol{\sigma}_{i:j}^{(m)})$ and stacking these values as $\boldsymbol{w}^{(m)}$. The main text uses $w \equiv 1$, giving multiplicity-weighted token count. The characterization below and the DVS dual decomposition also hold for these general costs.

Rows (4b)–(4c) form complete, licensed paths; (4d)–(4e) enforce the budget and alphabet. We now prove that Program (4) exactly characterizes DVS.

Proposition 1 (DVS ILP)**.** *The integer feasible points of Program* (4) *correspond to the pairs of a vocabulary* Δ *with* $\Sigma \subseteq \Delta \subseteq \Omega$ *and* $|\Delta \setminus \Sigma| \leq K$ *and a segmentation of every corpus string into tokens of* Δ*, and corresponding points have equal objective value.*

Proof of Proposition 1. Let $(\boldsymbol{x}, \boldsymbol{y})$ be feasible for Program (4) and set $\Delta(\boldsymbol{y}) \stackrel{\text{def}}{=} \{\delta \in \Omega : y_\delta = 1\}$. Constraints (4d) and (4e) make this a feasible vocabulary. Because every occurrence arc moves from a smaller to a larger boundary, a binary unit flow satisfying (4b) selects a sequence of adjacent spans from 0 to $N^{(m)}$. Constraint (4c) ensures that the substring on every selected span belongs to $\Delta(\boldsymbol{y})$, so these substrings form a valid segmentation with objective (4a).

Conversely, any feasible vocabulary and any segmentation of each corpus string determine consecutive occurrence arcs. Setting their x variables to one and selecting the vocabulary's y variables satisfies (4b)–(4g) with the same cost. Both constructions preserve the objective. ■

B.3 Optimal Pair Encoding

For the candidate strings and allowed merges of § 3.2, the variables $\boldsymbol{u}$ select one creator per optional token and require both inputs to be selected. The complete program is:

Why one token per decoded string suffices. DVS can add abc for one unit of budget. Constructing it by pairs instead takes two merges, and there are two ways to do so:

$$\texttt{a} + \texttt{b} \to \texttt{ab}, \quad \texttt{ab} + \texttt{c} \to \texttt{abc} \qquad \text{or} \qquad \texttt{b} + \texttt{c} \to \texttt{bc}, \quad \texttt{a} + \texttt{bc} \to \texttt{abc}. \tag{19}$$

Either way the intermediate token, ab or bc, consumes budget even if only abc appears in the final segmentation. Thus choosing abc requires choosing at least one of ab and bc; the construction is itself a decision.

An OPE sequence can run both constructions above, spending four merges to create the intermediate tokens and two distinct tokens that both decode to abc. We distinguish the latter as abc and abc′: a later merge can specify either token as an input. The prime distinguishes the tokens and is not part of their decoded text. The next lemma says that such duplicates never pay off, so for the token-count objective a token can be identified with the string it decodes to.

Lemma 1 (One token per string)**.** *Every at-most-K OPE solution can be rewritten so that distinct merges create tokens with distinct decoded strings. The rewriting uses no more merges and preserves every final token boundary, hence the total token count.*

The result applies to unrestricted merges and to restrictions specifying allowed decoded token strings and binary splits. Its proof keeps one split per created string and rebuilds each final token at freely chosen replacement sites.

Optimal pair encoding

$$\min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{u}} \quad \sum_{m=1}^{M} c^{(m)} \mathbf{1}^\top \boldsymbol{x}^{(m)} \qquad \text{(minimize total token count)} \qquad (18a)$$

$$\text{s.t.} \quad \boldsymbol{A}^{(m)} \boldsymbol{x}^{(m)} = \boldsymbol{b}^{(m)} \qquad \forall m \in [M] \qquad \text{(flow conservation)} \qquad (18b)$$

$$x_{ij}^{(m)} \le y_{\boldsymbol{\sigma}_{i:j}^{(m)}} \qquad \forall m,\ (i,j) \in E^{(m)} \qquad \text{(arcs need selected tokens)} \qquad (18c)$$

$$\sum_{\delta \in \Omega^-} y_\delta \le K \qquad \text{(vocabulary budget)} \qquad (18d)$$

$$y_\sigma = 1 \qquad \forall \sigma \in \Sigma \qquad \text{(alphabet always selected)} \qquad (18e)$$

$$\sum_{r:O(r)=g} u_r = y_g \qquad \forall g \in \Omega^- \qquad \text{(one creator per token)} \qquad (18f)$$

$$u_r \le y_{L(r)}, \quad u_r \le y_{R(r)} \qquad \forall r \in \mathcal{R} \qquad \text{(both inputs selected)} \qquad (18g)$$

$$\boldsymbol{x}^{(m)} \in \{0,1\}^{E^{(m)}} \qquad \forall m \in [M] \qquad \text{(binary arcs)} \qquad (18h)$$

$$\boldsymbol{y} \in \{0,1\}^{\Omega} \qquad \text{(binary tokens)} \qquad (18i)$$

$$\boldsymbol{u} \in \{0,1\}^{\mathcal{R}} \qquad \text{(binary merges)} \qquad (18j)$$

Figure 5: Optimal pair-encoding ILP. Blue is inherited from DVS; orange is the creator and merge-closure structure added by OPE.

Proof of Lemma 1. Let an operational OPE solution use $k \le K$ merges. Collect in V the alphabet together with the distinct decoded strings of all its created tokens, including those used only as intermediate inputs. Each merge contributes one string, so $|V \setminus \Sigma| \le k$, strictly less exactly when two merges created the same string. For each $g \in V \setminus \Sigma$, choose one original merge whose output decodes to g and denote its decoded inputs by a_g, b_g. Then $g = a_g b_g$, both inputs belong to V, and both are strictly shorter than g.

Use one token named by each string in V and retain the single merge $(a_g, b_g \to g)$ for each $g \in V \setminus \Sigma$. The chosen merges are acyclic because length decreases along inputs. If allowed candidates and merges depend only on decoded strings, every retained merge remains allowed: its split was present in the original solution.

It remains to reproduce the original final encoding with the retained merges. Every token in that encoding occupies a contiguous interval of the input. Recursively expand that interval's decoded string using the chosen merges. This produces a binary parse tree with the same character leaves and root interval, although its internal boundaries may differ from the original derivation. The final token intervals partition each corpus string, so these trees form a disjoint parse forest.

Process the chosen merges by increasing output length. At the step for g, replace exactly the child pairs at forest nodes labeled g. Their descendants have already collapsed, making the required pairs adjacent. Nodes with the same label cannot be ancestors of one another, since a proper descendant has smaller length. Their intervals are therefore disjoint, so these replacements constitute a legal non-overlapping subset, independently in each corpus string. A merge with no forest occurrence may use the empty subset.

After all merges, each root is a single token on its original final interval. Thus all final boundaries and the token count are preserved, with $|V \setminus \Sigma| \le k$ merges and one token per string. This construction uses the freedom to choose replacement sites and does not preserve an exhaustive replacement trajectory. ■

Proposition 2 (OPE ILP)**.** *Fix candidate strings Ω and allowed string merges $\mathcal{R}$. Every at-most-K OPE solution whose created tokens decode to strings in Ω and whose splits belong to $\mathcal{R}$ maps to an integer feasible point of Program* (18) *with the same total token count, and every integer feasible point of Program* (18) *maps to such an OPE solution with the same final segmentation.*

Proof of Proposition 2. Fix the candidate strings Ω and allowed string merges $\mathcal{R}$.

Operational solution to ILP. Consider an at-most-K partial merge sequence whose output strings lie in Ω and whose decoded splits belong to $\mathcal{R}$. Apply Lemma 1. Set $y_\sigma = 1$ for the alphabet, $y_g = 1$ for each constructed optional token, and $u_r = 1$ for its chosen creator; set the other entries to zero. Each selected optional token has exactly one creator, both of whose inputs have selection value one. This gives (18e)–(18g), and $\sum_{\delta \in \Omega^-} y_\delta \le K$ enforces the vocabulary budget.

Every final token spans a contiguous substring. Set $x_{ij}^{(m)} = 1$ on its interval and zero elsewhere. The intervals partition the input, so these arcs form a unit path satisfying (18b). Each emitted token is available, giving (18c). Normalization preserves final token boundaries, so the ILP objective equals the operational token count.

ILP to operational solution. Let $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u})$ be integral and feasible. Each $\boldsymbol{x}^{(m)}$ is one source–sink path in the acyclic occurrence graph. For every selected nonalphabet token g, (18f) chooses a unique creator, and (18g) makes both of its inputs available. Recursively expanding these shorter inputs yields a finite parse tree for g with alphabet leaves. Use one token for each selected string.

Embed these trees in the intervals of the selected path arcs. Apply the increasing-length, forest-directed partial replacements from the preceding proof. They realize exactly the selected final token intervals using the chosen merges. Their number is $\sum_{\delta \in \Omega^-} y_\delta = \sum_{r \in \mathcal{R}} u_r \le K$, and the final multiplicity-weighted token count is (18a). Hence the program characterizes the feasible vocabularies and final segmentations, although it does not enumerate all operational histories. ■

Corollary 1 (Completeness of the substring formulation)**.** *Let Ω contain the alphabet and every distinct nonempty corpus substring, with one candidate per string, and let $\mathcal{R}$ contain every split $(a, b \to g)$ with $a, b, g \in \Omega$ and $g = ab$. Program* (18) *then attains the unrestricted at-most-K OPE optimum.*

Proof of Corollary 1. In any operational optimum, discard every merge that never creates an occurrence. A merge requiring such an output also cannot create an occurrence, so all remaining dependencies are retained. Every remaining output decodes to a contiguous corpus substring. Lemma 1 gives a solution with one token per such string and no more merges. Its chosen input strings and splits all appear in the formulation. Proposition 2 maps it to the ILP with the same objective; the reverse direction makes every ILP solution an unrestricted OPE solution. ■

The completeness result requires no copies indexed by the budget K. With maximum input length N, a single string has at most $N(N+1)/2$ distinct nonempty substrings, and a token g has at most $|g| - 1$ binary splits. Enumerating all occurrences and splits can nevertheless be expensive. Restricting candidate strings or allowed splits can change the optimum; the certificate for a restricted program bounds that program's optimum.

Why retain merge variables? For $g \in \Omega^-$, let $S(g) \stackrel{\text{def}}{=} \{(a, b) : (a, b \to g) \in \mathcal{R}\}$. For binary $\boldsymbol{y}$ with $y_\sigma = 1$ on the alphabet, the existence of a permitted construction is equivalently

$$y_g \le \sum_{(a,b) \in S(g)} y_a y_b. \tag{20}$$

A selected token needs at least one split whose two inputs are selected. Thus $\boldsymbol{u}$ is an auxiliary family for expressing a choice, rather than an additional restriction on integer vocabularies.

Eliminating $\boldsymbol{u}$ from the box-relaxed creator constraints gives the piecewise-linear condition

$$y_g \le \sum_{(a,b) \in S(g)} \min\{y_a, y_b\}. \tag{21}$$

Indeed, each creator has capacity $\min\{y_a, y_b\}$, and its nonnegative value can be chosen so that the creators sum to y_g exactly when their total capacity is at least y_g. This condition has a linear description using only $\boldsymbol{y}$: for every choice of one input from each allowed split, bound y_g by the sum of the chosen input variables. There can be exponentially many such choices. The explicit merge variables give a compact linear formulation and preserve the cheapest-creator and top-K oracles of § 4.2. Choosing one fixed split per token beforehand would instead restrict the integer feasible set.

Lemma 1 establishes equivalence of the integer optima under the stated string-based candidates. It does not assert equality between the LP relaxation of this program and an alternative formulation with distinct tokens with the same decoded string; The dual bound of § 4.2 concerns the relaxation of Program (18) itself.

B.4 OMS Formulation Details and Correctness

OMS applies each chosen merge exhaustively, scanning from left to right to resolve overlapping occurrences. Greedy BPE chooses one OMS-feasible merge sequence. Unlike OPE, its final segmentation is determined by the complete ordered sequence of merges.

On aaab, schedule $r_1 = (\mathtt{a}, \mathtt{a} \to \mathtt{aa})$ and then $r_2 = (\mathtt{a}, \mathtt{b} \to \mathtt{ab})$. The three snapshots are

$$\underbrace{\mathtt{a} \mid \mathtt{a} \mid \mathtt{a} \mid \mathtt{b}}_{t=0} \xrightarrow{r_1} \underbrace{\mathtt{aa} \mid \mathtt{a} \mid \mathtt{b}}_{t=1} \xrightarrow{r_2} \underbrace{\mathtt{aa} \mid \mathtt{ab}}_{t=2}. \tag{22}$$

At the first step, the two occurrences of aa overlap, so only the leftmost pair is merged. The final a and b remain available for the second merge. The program records this execution over K stages, with one shared merge or no-op at each stage and a segmentation snapshot for every string before and after each step. A no-op leaves all strings unchanged, allowing sequences shorter than K; the objective counts tokens in the final snapshots.

Let Ω_{OMS} contain the base alphabet and optional candidate tokens $\Omega^{-}_{\mathrm{OMS}} \stackrel{\text{def}}{=} \Omega_{\mathrm{OMS}} \setminus \Sigma$. Each candidate g has a decoded string $\mathrm{yield}(g)$, with $\mathrm{yield}(\sigma) = \sigma$ for alphabet tokens. We retain distinct candidates with the same decoded string, such as abc and abc′ from § B.3: the OPE normalization proof chooses replacement sites freely and does not preserve an exhaustive merge sequence. Primes distinguish tokens, not characters in the text. Allowed merges $r = (a, b \to g) \in \mathcal{R}_{\mathrm{OMS}}$ join candidates $a, b \in \Omega_{\mathrm{OMS}}$ into $g \in \Omega^{-}_{\mathrm{OMS}}$ with $\mathrm{yield}(g) = \mathrm{yield}(a)\,\mathrm{yield}(b)$. As in OPE, $L(r)$, $R(r)$, and $O(r)$ identify the input and output tokens.

For $t \in [K]$, binary variables ρ_{tr} and n_t select merge r or a no-op. Occurrence arcs retain token identities, and replacement sites join compatible adjacent arcs:

$$E^{(m)}_{\mathrm{OMS}} \stackrel{\text{def}}{=} \{(i, j, g) : 0 \le i < j \le N^{(m)},\ g \in \Omega_{\mathrm{OMS}},\ \mathrm{yield}(g) = \boldsymbol{\sigma}^{(m)}_{i:j}\},$$
$$Q^{(m)} \stackrel{\text{def}}{=} \{(i, h, j, r) : r \in \mathcal{R}_{\mathrm{OMS}},\ (i, h, L(r)), (h, j, R(r)), (i, j, O(r)) \in E^{(m)}_{\mathrm{OMS}}\}.$$

As before, $\boldsymbol{A}^{(m)}$ is the node–arc incidence matrix. For $t \in [K]_0$, $x^{(m,t)}_{ijg}$ indicates that token g occupies $[i, j)$ after stage t; for $t \in [K]$, $q^{(m,t)}_{ijg}$ indicates that it is copied unchanged from the previous snapshot. At site $s = (i, h, j, r)$, $e^{(m,t)}_s$ indicates eligibility: merge r is scheduled and both input occurrences are present. The variable $z^{(m,t)}_s$ indicates that the replacement is performed. Overlapping sites may both be eligible, but cannot both consume their shared token. All these variables are binary.

The fixed matrices in Program (24) express these relationships. Columns of $\boldsymbol{D}^{(m)}$ and $\boldsymbol{G}^{(m)}$ identify the input arcs consumed and the output arc produced at each site. Rows of $\boldsymbol{\Pi}^{(m)}$, $\boldsymbol{X}^{(m)}_{\mathrm{L}}$, and $\boldsymbol{X}^{(m)}_{\mathrm{R}}$ select its scheduled merge and two input occurrences. The product $\boldsymbol{p}^{(m,t)} = \boldsymbol{W}^{(m)}\boldsymbol{z}^{(m,t)}$ indicates whether each site's left input is consumed by the preceding overlapping replacement. Token availability before stage t is determined by prior creation:

$$\bar{a}_{t,\sigma} \stackrel{\text{def}}{=} 1 \quad (\sigma \in \Sigma), \qquad \bar{a}_{t,g} \stackrel{\text{def}}{=} \sum_{\tau < t} \sum_{r : O(r) = g} \rho_{\tau r} \quad (g \in \Omega^{-}_{\mathrm{OMS}}). \tag{23}$$

Each optional token is created at most once, so availability is binary. The matrix entries and a worked variable assignment for aaab are given below.

Program (24) maintains a complete segmentation at every stage and a valid shared merge schedule. Its balance equations carry each snapshot forward by copying or merging token occurrences, while the eligibility and overlap constraints enforce exhaustive left-to-right replacement. The earlier snapshots ensure that the final segmentation is produced by the chosen encoder. The remainder of this subsection explains the constraint blocks and proves the characterization.

Optimal merge sequences

Schedule and snapshots.

$$\min_{\boldsymbol{\rho},\boldsymbol{n},\boldsymbol{x},\boldsymbol{q},\boldsymbol{z},\boldsymbol{e}} \quad \sum_{m=1}^{M} c^{(m)} \mathbf{1}^\top \boldsymbol{x}^{(m,K)} \tag{24a}$$

$$\text{s.t.} \quad x_{ijg}^{(m,0)} = \mathbf{1}\{j = i+1,\ g = \boldsymbol{\sigma}_{i:j}^{(m)}\} \qquad \forall m \in [M],\ (i,j,g) \in E_{\mathrm{OMS}}^{(m)} \tag{24b}$$

$$\boldsymbol{A}^{(m)} \boldsymbol{x}^{(m,t)} = \boldsymbol{b}^{(m)} \qquad \forall m \in [M],\ t \in [K]_0 \tag{24c}$$

$$\mathbf{1}^\top \boldsymbol{\rho}_t + n_t = 1 \qquad \forall t \in [K] \tag{24d}$$

$$\textstyle\sum_{t=1}^{K} \sum_{r:O(r)=g} \rho_{tr} \le 1 \qquad \forall g \in \Omega_{\mathrm{OMS}}^{-} \tag{24e}$$

$$\rho_{tr} \le \bar{a}_{t,L(r)} \qquad \forall t \in [K],\ r \in \mathcal{R}_{\mathrm{OMS}} \tag{24f}$$

$$\rho_{tr} \le \bar{a}_{t,R(r)} \qquad \forall t \in [K],\ r \in \mathcal{R}_{\mathrm{OMS}} \tag{24g}$$

Copy or merge each token occurrence.

$$\boldsymbol{x}^{(m,t-1)} = \boldsymbol{q}^{(m,t)} + \boldsymbol{D}^{(m)} \boldsymbol{z}^{(m,t)} \qquad \forall m \in [M],\ t \in [K] \tag{24h}$$

$$\boldsymbol{x}^{(m,t)} = \boldsymbol{q}^{(m,t)} + \boldsymbol{G}^{(m)} \boldsymbol{z}^{(m,t)} \qquad \forall m \in [M],\ t \in [K] \tag{24i}$$

Force eligible replacements, leftmost first.

$$\boldsymbol{e}^{(m,t)} \le \boldsymbol{\Pi}^{(m)} \boldsymbol{\rho}_t, \quad \boldsymbol{e}^{(m,t)} \le \boldsymbol{X}_{\mathrm{L}}^{(m)} \boldsymbol{x}^{(m,t-1)}, \quad \boldsymbol{e}^{(m,t)} \le \boldsymbol{X}_{\mathrm{R}}^{(m)} \boldsymbol{x}^{(m,t-1)} \qquad \forall m \in [M],\ t \in [K] \tag{24j}$$

$$\boldsymbol{e}^{(m,t)} \ge \boldsymbol{\Pi}^{(m)} \boldsymbol{\rho}_t + \boldsymbol{X}_{\mathrm{L}}^{(m)} \boldsymbol{x}^{(m,t-1)} + \boldsymbol{X}_{\mathrm{R}}^{(m)} \boldsymbol{x}^{(m,t-1)} - 2 \cdot \mathbf{1} \qquad \forall m \in [M],\ t \in [K] \tag{24k}$$

$$\boldsymbol{p}^{(m,t)} \stackrel{\text{def}}{=} \boldsymbol{W}^{(m)} \boldsymbol{z}^{(m,t)} \qquad \forall m \in [M],\ t \in [K] \tag{24l}$$

$$\boldsymbol{e}^{(m,t)} - \boldsymbol{p}^{(m,t)} \le \boldsymbol{z}^{(m,t)} \le \boldsymbol{e}^{(m,t)} \qquad \forall m \in [M],\ t \in [K] \tag{24m}$$

$$\boldsymbol{\rho}, \boldsymbol{n}, \boldsymbol{x}, \boldsymbol{q}, \boldsymbol{z}, \boldsymbol{e} \in \{0,1\} \qquad \text{over the index sets above.} \tag{24n}$$

Figure 6: Optimal merge-sequence ILP. Blue retains the final token-count objective, orange retains pair-construction prerequisites, and purple marks the ordered, exhaustive state trajectory added by OMS.

Matrix construction. Fixed incidence matrices collect these decisions. Column s of $\boldsymbol{D}^{(m)}$ has a 1 on each input arc consumed at site s, and column s of $\boldsymbol{G}^{(m)}$ has a 1 on its output arc. Consequently

$$\underbrace{\boldsymbol{x}^{(m,t-1)}}_{\text{old tokens}} = \underbrace{\boldsymbol{q}^{(m,t)}}_{\text{copied tokens}} + \underbrace{\boldsymbol{D}^{(m)} \boldsymbol{z}^{(m,t)}}_{\text{consumed tokens}}, \qquad \underbrace{\boldsymbol{x}^{(m,t)}}_{\text{new tokens}} = \underbrace{\boldsymbol{q}^{(m,t)}}_{\text{copied tokens}} + \underbrace{\boldsymbol{G}^{(m)} \boldsymbol{z}^{(m,t)}}_{\text{merge outputs}}. \tag{25}$$

The first equality makes every old occurrence copied or consumed exactly once. The second builds the new segmentation from those copies and merge outputs.

To express eligibility, row $s = (i,h,j,r)$ of $\boldsymbol{\Pi}^{(m)}$ selects merge r, while rows s of $\boldsymbol{X}_{\mathrm{L}}^{(m)}$ and $\boldsymbol{X}_{\mathrm{R}}^{(m)}$ select its left and right input arcs. Thus the three quantities tested at this site are simply

$$(\boldsymbol{\Pi}^{(m)} \boldsymbol{\rho}_t)_s = \rho_{tr}, \qquad (\boldsymbol{X}_{\mathrm{L}}^{(m)} \boldsymbol{x}^{(m,t-1)})_s = x_{ih,L(r)}^{(m,t-1)}, \qquad (\boldsymbol{X}_{\mathrm{R}}^{(m)} \boldsymbol{x}^{(m,t-1)})_s = x_{hj,R(r)}^{(m,t-1)}. \tag{26}$$

The upper and lower bounds on $\boldsymbol{e}$ in Program (24) are the standard linear encoding of the conjunction of these three binary values. All matrix entries and subscripts are fixed by the candidates, just as in DVS.

To encode overlaps, define $\boldsymbol{W}_{s,s'}^{(m)} = 1$ when site s' consumes the left input arc of s as its right input, and 0 otherwise. The quantity $\boldsymbol{p}^{(m,t)} = \boldsymbol{W}^{(m)} \boldsymbol{z}^{(m,t)}$ then records preceding overlapping replacements. The consumption equality makes each p_s binary: it is 1 exactly when the left input of site s has already been assigned to the preceding replacement. The bounds

$$e_s - p_s \le z_s \le e_s \tag{27}$$

force an eligible site to merge unless that preceding replacement blocks it. This is the left-to-right scan written as simultaneous linear constraints. Neither $\bar{a}$ nor p is an independent choice.

Worked variable assignment. For the aaab example above, the scheduled merges are $r_1 = (\mathtt{a}, \mathtt{a} \to \mathtt{aa})$ and $r_2 = (\mathtt{a}, \mathtt{b} \to \mathtt{ab})$. At stage 1, $\rho_{1r_1} = 1$ and $n_1 = 0$. Both $s_0 = (0, 1, 2, r_1)$ and $s_1 = (1, 2, 3, r_1)$ are eligible, so $e_{s_0} = e_{s_1} = 1$ (suppressing m, t). The leftmost convention requires $z_{s_0} = 1$ and $z_{s_1} = 0$, because they share the a on $[1, 2)$. The last a and the b are copied: $q_{23,\mathtt{a}} = q_{34,\mathtt{b}} = 1$. The next snapshot therefore has exactly $x_{02,\mathtt{aa}} = x_{23,\mathtt{a}} = x_{34,\mathtt{b}} = 1$. At stage 2, the aa is copied and the remaining a, b are replaced by ab. A later no-op would copy both final tokens. The preceding-overlap indicators at stage 1 are $p_{s_0} = 0$ and $p_{s_1} = 1$.

A no-op selects no merge, making every e_s and z_s zero; the balance equations then copy the entire previous snapshot.

Proposition 3 (OMS ILP)**.** *The integer feasible points of Program* (24) *correspond to the sequences of at most K allowed merges, padded with no-ops to length K, in which each merge is applied exhaustively with the left-to-right overlap convention; corresponding points have objective value equal to the final multiplicity-weighted token count.*

Proof of Proposition 3. Given an operational OMS sequence, pad it with no-ops to length K and set ρ and n to its schedule. Set $x^{(m,t)}$ to the occurrence indicators of the string after stage t, set $z^{(m,t)}$ to the merges performed by the deterministic left-to-right scan, and set $q^{(m,t)}$ to the copied occurrences. The schedule obeys (24d)–(24g); the stage strings obey (24b) and (24c). Setting e to the eligible pairs makes (24j)–(24k) exact. Every old occurrence is copied or consumed once and every new occurrence is a copy or merge output, giving (24h)–(24i). The operational overlap convention is precisely (24m), so all ILP rows hold and (24a) counts the final tokens.

Conversely, take a feasible integer point and induct on t. Equation (24b) gives the character-level initial string. Equation (24d) chooses one merge or a no-op, and (24e)–(24g) make that choice a valid merge sequence. Suppose $x^{(m,t-1)}$ describes the operational state before stage t. The eligibility rows set e to one exactly at adjacent occurrences of the chosen pair. The consumption balance makes the preceding-consumption sum (24l) binary, and (24m) therefore merges every eligible pair except when its left occurrence was consumed by the immediately preceding overlapping merge. The consume and produce equations then set $x^{(m,t)}$ to exactly the result of that left-to-right scan. Induction gives the same state at every stage, and the final objectives agree. ■

B.5 Matched Token-Count Hierarchy

On the same corpus, let DVS and OPE use the same candidate strings Ω, containing the yield of every OMS candidate token. Require the OPE merge set $\mathcal{R}$ to contain the string projection $(\mathrm{yield}(L(r)), \mathrm{yield}(R(r)) \to \mathrm{yield}(O(r)))$ of each $r \in \mathcal{R}_{\mathrm{OMS}}$, and give all three problems the same budget K. If L_{BPE} is the final token count of any at-most-K-merge BPE run using the allowed OMS merges with the same overlap convention, then the hierarchy in (7) holds. Each inclusion follows directly from the definitions: an OPE vocabulary is feasible for DVS, an exhaustive merge sequence is a legal OPE sequence (normalized by Lemma 1), and a BPE run is one feasible OMS sequence.

C Dual-Decomposition Details

C.1 Direct-Selection Bounds

Separable blocks and linking rows. The local feasible sets are the path sets $X^{(m)}$. The global feasible set is

$$Y \stackrel{\text{def}}{=} \left\{ \boldsymbol{y} \in \{0,1\}^{\Omega} : y_\sigma = 1 \ (\sigma \in \Sigma),\ \sum_{\delta \in \Omega^-} y_\delta \le K \right\}. \tag{28}$$

The only rows joining these blocks are

$$x_{ij}^{(m)} - y_{\sigma_{i:j}^{(m)}} \le 0, \qquad \forall m,\ (i,j) \in E^{(m)}. \tag{29}$$

Dualizing these scalar residuals gives

$$L(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\lambda}) = \sum_m \sum_{(i,j) \in E^{(m)}} (w_{ij}^{(m)} + \lambda_{ij}^{(m)}) x_{ij}^{(m)} - \sum_{\delta \in \Omega} \Lambda_\delta y_\delta, \tag{30}$$

whose minimum over $\prod_m X^{(m)} \times Y$ separates into the path and vocabulary oracles of § 4.1.

C.2 Algorithms and Implementation

Both DD systems use the Viterbi and global-selection oracles of §4. We deduplicate pretokens, sort them by length, and evaluate the corpus oracle in large GPU batches. A naive occurrence builder would compare each candidate string independently. Instead, a vectorized trie walk shares work among candidates with common prefixes, so its state is the trie rather than a separate representation of every candidate string. Batched on-device traceback accumulates token usage without materializing decoded segmentations, and CUDA-graph replay removes repeated host launches.

For OPE, the global recovery oracle additionally enforces merge closure over the fixed token strings and creator merges. Resident mode keeps corpus state on the GPU. Streamed mode instead keeps persistent multipliers in pinned host memory and rebuilds one batch of occurrence tensors at a time, leaving the dual updates and certificates unchanged.

All experiments use NVIDIA B200 GPUs with PyTorch 2.13.0 and CUDA 13.0. DVS DD uses occurrence-specific multipliers, a $1/\sqrt{t}$ step schedule, primal evaluation every 10 rounds, and ergodic recovery every 25 rounds. The main runs use an active set as described below. OPE uses BPE prefix vocabularies as feasible warm starts and Polyak updates for creator-input multipliers with coefficient 0.25. The DVS-LP baseline uses cuOpt PDLP with relative gap tolerance 10^{-5} and no crossover.

Algorithm 1: Viterbi oracle for one string

$\alpha_0 \leftarrow 0$; $\alpha_j \leftarrow \infty$ for $j > 0$
for $j = 1, \ldots, N$ **do**
  $(\alpha_j, \beta_j) \leftarrow \min_{i:(i,j)\in E}(\alpha_i + q_{ij},\, i)$
Backtrack $\beta_N, \beta_{\beta_N}, \ldots$ to recover the selected arcs

Algorithm 2: Dual decomposition for DVS

$\boldsymbol{\lambda} \leftarrow \mathbf{0}$; LB $\leftarrow -\infty$; UB $\leftarrow \infty$
for $t = 1, \ldots, T$ **do**
  run Algorithm 1 for every string at cost $w + \boldsymbol{\lambda}$
  aggregate Λ_δ and select the top-K tokens with positive Λ_δ
  update LB $\leftarrow \max\{\text{LB}, g(\boldsymbol{\lambda})\}$
  re-run Viterbi under that vocabulary and update UB
  update multipliers by (11)
return *best vocabulary,* UB*,* LB

For the full-oracle loop in Algorithm 2, bounded supergradients and projected subgradient ascent using a horizon-tuned step give $O(1/\sqrt{T})$ best-iterate error (Polyak, 1969; Nemirovsky & Yudin, 1983).

Active set. The main runs reduce multiplier memory using the following active set representation. In DVS DD, each licensing row $x_{ij}^{(m)} \le y_{\sigma_{i:j}^{(m)}}$ has an occurrence-specific multiplier. An active set representation retains explicit multipliers only for a working set $\mathcal{A}$ of these rows. Writing $a = (m, i, j)$ for a row, it stores $\{\lambda_a : a \in \mathcal{A}\}$ and interprets $\lambda_a = 0$ for $a \notin \mathcal{A}$. Rows may be grouped by token type for admission and removal, while retaining a separate multiplier for each occurrence. Memory for explicit multipliers therefore scales with $|\mathcal{A}|$. The set Ω and licensing constraints remain fixed; membership in $\mathcal{A}$ controls stored dual state, not token eligibility in the original ILP.

Changing the working set. Newly admitted rows start with zero multipliers, and a row can be removed without changing the represented multiplier vector only when its multiplier is zero. Removing a whole token's group requires every occurrence multiplier in that group to be zero. Retained multipliers are updated by the nonnegative projection in (11). These rules preserve a nonnegative

vector over the full occurrence space. They state the conditions needed for validity, without prescribing a particular admission heuristic, pruning criterion, or sweep interval.

Certification with the full oracle. A restricted path oracle retains the mandatory alphabet arcs and only the optional occurrences represented in the working set. Its minimum can exceed the unrestricted minimum, so its value cannot automatically serve as a lower bound for the original problem. An unrestricted sweep considers every eligible occurrence with missing multipliers read as zero, and solves the original vocabulary block, thereby computing the exact $g(\boldsymbol{\lambda})$ in (10). If $\mathcal{S}$ is the set of unrestricted-sweep iterations, the certified running bound is

$$D_t = \max_{s \in \mathcal{S},\, s \le t} g(\boldsymbol{\lambda}_s). \tag{31}$$

We take $D_t = -\infty$ before the first unrestricted sweep. Between such sweeps, the previous certified lower bound is retained; restricted-oracle values do not enter this maximum. Feasible vocabularies can still improve the upper bound through Viterbi evaluation without multipliers. This separates the working set used to guide optimization from the fixed problem addressed by the certificate.

Since missing multipliers are read as zero, the stored multipliers always represent a nonnegative vector over the full occurrence space, so every full-sweep value is a valid lower bound by weak duality, and every recovered vocabulary is feasible for the DVS ILP.

C.3 OPE Dual and Recovery

Proposition 4 (Exact OPE subproblem oracles)**.** *For fixed multipliers, the OPE dual is computed by (i) Viterbi with arc cost $c^{(m)} + \lambda^{(m)}_{ij}$, (ii) one merge scan for each output token's cheapest creator, and (iii) top-K selection by net token profit.*

Proof of Proposition 4. Use the string candidates and merges of § 3.2, with the same optional-token set Ω^- and fixed alphabet entries $y_\sigma = 1$. Relax nonalphabet occurrence licensing with multipliers $\lambda^{(m)}_{ij} \ge 0$, with $\lambda^{(m)}_{ij} = 0$ on alphabet arcs, and the creator-input links (5b) by $\alpha_r, \beta_r \ge 0$. Retain the budget, alphabet, and one-creator equations in the dictionary block. The OPE Lagrangian is

$$\begin{aligned} L_{\mathrm{OPE}} = \sum_m \sum_{(i,j) \in E^{(m)}} \Big[c^{(m)} x^{(m)}_{ij} + \lambda^{(m)}_{ij} (x^{(m)}_{ij} - y_{\boldsymbol{\sigma}^{(m)}_{i:j}}) \Big] \\ + \sum_{r \in \mathcal{R}} \alpha_r (u_r - y_{L(r)}) + \sum_{r \in \mathcal{R}} \beta_r (u_r - y_{R(r)}). \end{aligned} \tag{32}$$

For $g \in \Omega^-$, aggregate occurrence multipliers as in (9), writing them as Λ_g. Define its cheapest creator cost and net profit by

$$\begin{aligned} d_g &\stackrel{\text{def}}{=} \min_{r : O(r) = g} (\alpha_r + \beta_r), \\ P_g &\stackrel{\text{def}}{=} \Lambda_g + \sum_{r : L(r) = g} \alpha_r + \sum_{r : R(r) = g} \beta_r - d_g. \end{aligned} \tag{33}$$

If g has no candidate creator, set $d_g = +\infty$ and $P_g = -\infty$, so it cannot be selected. The constant contributed by mandatory alphabet inputs is $c_\Sigma \stackrel{\text{def}}{=} -\sum_{r : L(r) \in \Sigma} \alpha_r - \sum_{r : R(r) \in \Sigma} \beta_r$. Retaining the vocabulary budget, fixed alphabet entries, and creator equations, and grouping terms by path and output token, gives

$$\begin{aligned} g_{\mathrm{OPE}} = c_\Sigma + \sum_m \min_{\boldsymbol{x}^{(m)} \in X^{(m)}} \sum_{(i,j) \in E^{(m)}} (c^{(m)} + \lambda^{(m)}_{ij}) x^{(m)}_{ij} \\ - \sum_{g \in \mathrm{TopK}(P_+)} P_g. \end{aligned} \tag{34}$$

Here $\mathrm{TopK}(P_+)$ takes at most K optional tokens with positive profit. Selecting an optional token also selects a creator attaining d_g. The first term is computed by Viterbi with arc costs $c^{(m)} + \lambda^{(m)}_{ij}$, d_g by one scan over each output token's merges, and the last term by top-K selection on P_g. ■

The supergradient consists of the relaxed residuals $x_{ij}^{(m)} - y_{\sigma_{i:j}^{(m)}}$ on nonalphabet arcs, $u_r - y_{L(r)}$, and $u_r - y_{R(r)}$.

For feasible recovery, discard merges with recursively unconstructible inputs. One recovery procedure processes tokens in decreasing P_g: recursively collect their chosen creators and required optional input tokens, and accept a closure only when its union with the accepted tokens still fits K. Inputs are shorter than outputs, so recursion terminates; ordering accepted merges by output length is valid, and Viterbi over the recovered vocabulary therefore gives a feasible OPE upper bound. With $A = \sum_m |E^{(m)}|$, $G = |\Omega^-|$, and $R = |\mathcal{R}|$, dual evaluation and multiplier updates take $O(A + R + G)$ expected time with linear-time top-K selection and $O(A + R + G)$ storage. This bound excludes the sorting and recursive closure work used for feasible recovery.

Corollary 1 requires only one candidate per distinct substring to cover unrestricted OPE, but all substring occurrences and compatible splits may still be too large to materialize.

D Selecting Vocabulary Size by Two-Part Coding

Token count alone is insufficient for choosing vocabulary size: adding vocabulary entries incurs no cost in the objective. We therefore minimize corpus bits plus vocabulary and codebook storage under a computational cap, so extra tokens must justify their storage cost through compression savings.

Figure 7 compares BPE with heuristic DVS on ClimbMix. Both use Huffman coding (Huffman, 1952) and prune unused tokens; the DVS search also edits the vocabulary and uses bit-weighted segmentation. Vocabularies and codes are fitted on training data and frozen for development evaluation.

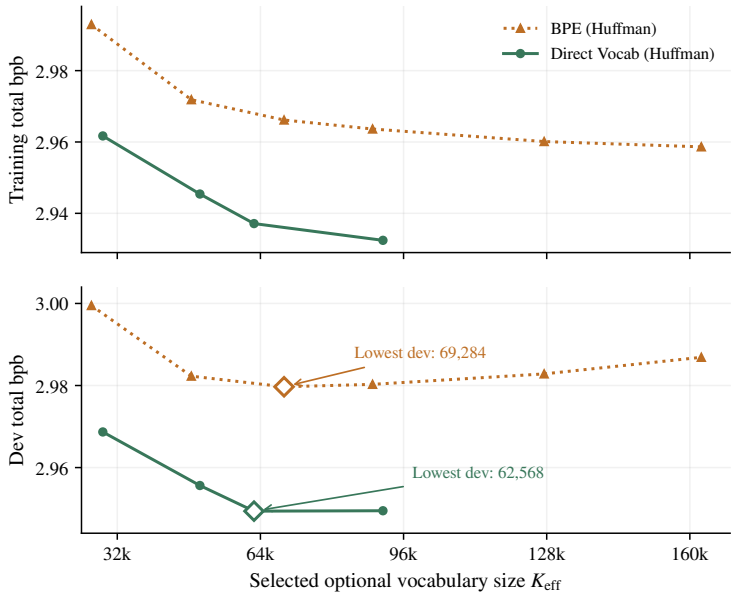


Figure 7: Two-part bits per byte in the larger-vocabulary region of the sweep; K_{eff} counts selected optional types. Diamonds mark each method's lowest observed development cost.

DVS achieves lower development cost in the plotted range, while increasing vocabulary size does not always improve it. The marked minima are best observed checkpoints, without a certificate of an optimal vocabulary size.

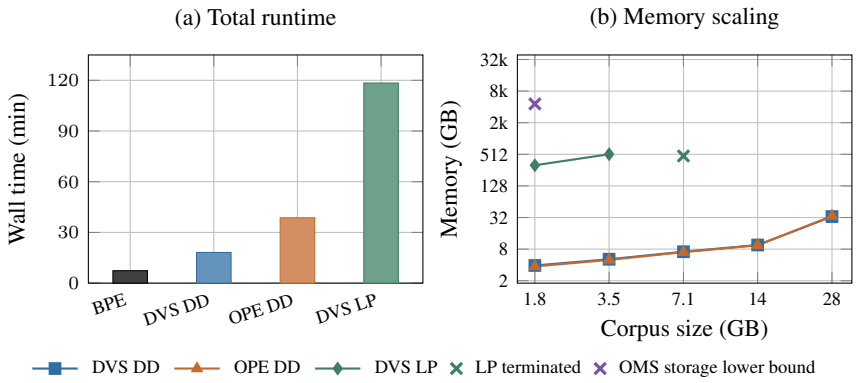


Figure 8: Runtime and memory scaling. (a) Total runtime across five vocabulary sizes on 1.766B bytes: $K \in \{4096, 8192, 16384, 32768, 65536\}$. (b) Peak host memory at $K = 32768$: lines are measurements; crosses are lower bounds from LP termination and OMS storage requirements. BPE memory was not measured. LP time excludes vocabulary rescoring.

E Additional Experimental Results

E.1 Runtime and Memory Scaling

On the 1.766B-byte corpus, the five-budget sweep takes 7.35 minutes for BPE training plus Viterbi evaluation, 18.14 for DVS DD, 38.68 for OPE DD, and 118.39 for the measured monolithic-LP components (Figure 8). BPE is thus 2.5× and 5.3× faster than the decomposed methods. At $K = 32768$, both DD systems reach 28.232B bytes with roughly 34 GB peak host memory. The DVS LP is terminated during its 7.064B-byte build after reaching at least 476 GB. No full OMS sweep is attempted: its merge–stage block requires at least 4.67 TB for three float64 vectors alone at the smallest rung, exceeding the 3.0 TiB host before corpus trajectories or constraints are included.

E.2 Controlled Solver Check

Table 1 compares the solvers on the 32 most frequent ClimbMix chunks (332.3M weighted bytes), a small instance chosen for validation. DVS DD matches the DVS LP to reported precision, with floating-point discrepancies of at most 19 weighted tokens (less than 10^{-7} relative). The OMS LP is tight and its ILP matches BPE; the OPE bounds do not certify the returned BPE vocabulary as optimal.

Table 1: Controlled solver check, in millions of weighted tokens. BPE uses Viterbi; paired entries give feasible/lower-bound (OPE) and LP/ILP (OMS) objectives.

K	BPE	DVS DD/LP	OPE primal/LB	OPE LP	OMS LP/ILP
4	258.662	234.269/234.269	258.662/245.719	247.865	258.662/258.662
8	219.860	191.313/191.313	219.860/209.462	210.877	219.860/219.860

E.3 Held-Out Compression

We evaluate token-count vocabularies trained on ClimbMix shards 0–6 on held-out shard 200 (252635578 bytes). Huffman codes use add-one-smoothed training Viterbi counts; vocabularies, minimum-token encoders, and codes are frozen for evaluation. This retains the fixed-budget token-count objective, unlike §D. DD reduces held-out bit cost only at the larger budgets, despite reducing token count throughout the sweep.

Both methods use the same one-time header: gamma-coded optional-type count plus one, token byte lengths, and code lengths, plus literal optional-token bytes. Pretoken-boundary metadata is excluded.

Table 2: Held-out bits per byte (bpb). Positive reductions favor DVS DD; the last column also charges the lexicon and code-length header.

K	DD (bpb)	BPE (bpb)	Reduction	Incl. header
4096	3.213619	3.193677	-0.62%	-0.62%
8192	3.137006	3.135728	-0.04%	-0.04%
16384	3.079841	3.088572	$+0.28\%$	$+0.28\%$
32768	3.034380	3.053304	$+0.62\%$	$+0.62\%$
65536	3.006676	3.033664	$+0.89\%$	$+0.88\%$

E.4 Long-Horizon Checkpoints

On one uninterrupted DVS DD trajectory at $K = 32768$, the gap $(P - D)/P$ falls from 0.304% after 300 rounds (295 s) to 0.129% after 1000 rounds (990 s) and 0.054% after 4000 rounds (3967 s). The main sweep uses 300 rounds to balance bound tightness and optimization time.