IPA-Verified Singable Lyrics Translation with LLM Agents

Anonymous ACL submission

Abstract

Translating song lyrics while preserving singability requires satisfying musical constraints, including syllable counts, rhyme schemes, and syllable patterns, that standard machine translation ignores. We present a framework that exposes IPA-based phonetic analysis as a suite of composable Agent Skills, each defined by a natural-language description plus deterministic verification scripts that the orchestrating LLM invokes on demand. The agent extracts constraints from source lyrics, then translates through a multi-phase process, calling skills to verify all constraints and iteratively refine syllable-count mismatches. We formalize the three musical constraints, describe the IPA skill suite and agent architecture, and propose three evaluation metrics: Syllable Error Rate (SER), Syllable Count Relative Error (SCRE), and Adjusted Rand Index (ARI). Ablation experiments across three frontier orchestrators confirm that iterative tool-verified refinement dramatically improves constraint satisfaction and preserves rhyme structure substantially better than a supervised baseline, while requiring no task-specific training and no parallel corpora, and transferring to Spanish and Japanese targets with only a small language-specific syllabification adapter.

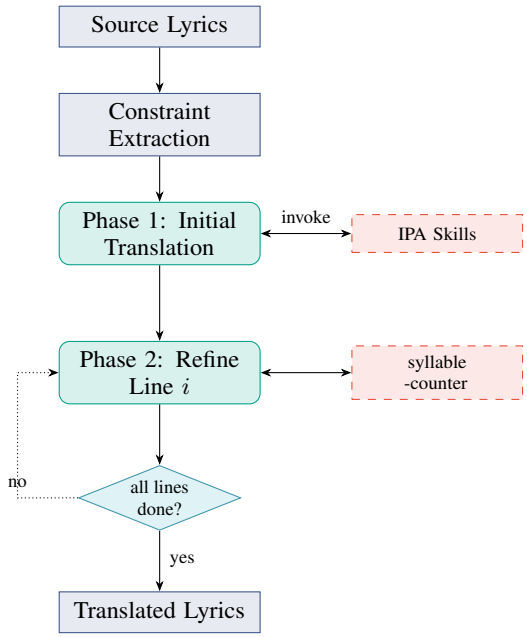

Figure 1: Skill framework overview. Phase 1 translates all lines using an internal tool-use loop that invokes the IPA skills. Phase 2 refines each line sequentially (up to 10 attempts per line); the dotted arrow shows the orchestration loop that iterates through lines.

1 Introduction

A translated song lyric must conform to the musical structure of the original so that it remains *singable* to the same melody (Low, 2005). This requires satisfying hard structural constraints: syllable counts must match melodic phrases, rhyme schemes should be preserved, and word-level syllable distributions should mirror the original phrasing. Despite theoretical grounding (Low, 2005; Franzon, 2008), computational approaches remain scarce. Neural MT systems have no mechanism for phonetic constraints, and constrained decoding methods (Hokamp and Liu, 2017; Post and Vilar, 2018) address lexical but not phonetic constraints. A fundamental obstacle is that *counting syllables, detecting rhyme, and analyzing phonetic patterns are not tasks that LLMs perform reliably through text generation alone* (Gao et al., 2023).

We address this by exposing IPA-based phonetic analysis as a suite of Agent Skills, where each capability is packaged as a directory of natural-language instructions plus deterministic scripts that the orchestrating LLM discovers and invokes during translation (Yao et al., 2023). Unlike supervised approaches, the framework requires no parallel lyric corpora, supports any language pair via eSpeak-NG, and scales with LLM capability. Our framework is open-source for reproducibility and

practical use.[1]

Our contributions are:

1. A formalization of three musical constraints (syllable counts, rhyme schemes, and syllable patterns) and IPA-based methods for their extraction and verification (§3).

2. An IPA-augmented Skill suite and multi-phase orchestration architecture for constraint-preserving lyrics translation (§4).

3. Three automatic evaluation metrics (SER, SCRE, and ARI) designed to measure structural constraint preservation in singable translation (§5).

To our knowledge, this is the first framework to package IPA-based phonetic analysis as Agent Skills for singable lyrics translation.

2 Related Work

2.1 Song Translation

Theoretical accounts treat song translation as balancing singability, sense, naturalness, rhythm, and rhyme (Low, 2005; Franzon, 2008; Low, 2022). Computationally, prior work spans constrained sequence-to-sequence learning (Ou et al., 2023a), tonal-language constraints (Guo et al., 2022), constrained decoding for poetry (Ghazvininejad et al., 2018), musical-feature modeling (Ye et al., 2024), curriculum reinforcement learning (Ren et al., 2025), melody-constrained editing (Zhao et al., 2025), melody-to-lyric generation (Ou et al., 2023b), joint lyric–melody generation (Ding et al., 2025), multilingual datasets and evaluation (Cho et al., 2025; Kim et al., 2023, 2024), scansion- and prosody-aware generation (Chen and Teufel, 2024; Liang et al., 2024; Yu et al., 2025), and multimodal singability (Yoo et al., 2025). Closest to our setting, Liu et al. (2026) study zero-shot prompting strategies for singable lyric translation on a new en/zh/jp dataset, comparing seven prompts of increasing complexity (including verification-guided and multi-round refinement) and finding that moderate complexity is sufficient. Their refinement step asks the LLM to count syllables *within* the prompt; we instead expose syllable counting as a deterministic external skill, which keeps line-level counts exact rather than approximate (§6.2). These approaches rely on specialized fine-tuning, constrained decoding, reinforcement learning, or prompt-only verification; we instead augment a general-purpose LLM with *external* IPA-verification skills, requiring no task-specific training, no new model, and no new dataset. We therefore compare directly to the supervised constraint-token baseline closest in task setting (Ou et al., 2023a; §6) rather than retraining the others.

2.2 Constrained Text Generation

Lexically constrained decoding has been studied via Grid Beam Search and Dynamic Beam Allocation (Hokamp and Liu, 2017; Post and Vilar, 2018), while computational poetry has used finite-state machinery and neural models for meter and rhyme (Ghazvininejad et al., 2016; Hopkins and Kiela, 2017; Lau et al., 2018). These methods enforce constraints at the decoding level or learn them implicitly; ours lets the LLM generate freely and verifies via external phonetic skills.

2.3 Tool-Augmented Language Models

Prior work has shown that LLMs can be trained or prompted to invoke external tools (Schick et al., 2023), interleave reasoning with tool calls (Yao et al., 2023), and offload computation to program execution (Gao et al., 2023), the latter directly relevant here since syllable counting is unreliable in text generation but deterministic via IPA tools. More recently, frontier LLM platforms have introduced *Agent Skills*: capabilities packaged as a directory with a natural-language description (`SKILL.md`) plus optional helper scripts loaded on demand; we adopt this design to factor phonetic analysis into independent, reusable skills. While LLMs achieve strong general translation quality (Jiao et al., 2023), they do not address musical or structural constraints.

2.4 Phonetic Resources

Our skill suite builds on Phonemizer (Bernard and Titeux, 2021) for text-to-IPA conversion and PanPhon (Mortensen et al., 2016) for phonetic distance computation; both are described in §4. Concurrently, Chen et al. (2025) use IPA as an intermediary for LLM translation of spoken-only languages; though aimed at low-resource preservation rather than musical constraints, they share our insight that IPA offers a language-agnostic phonetic interface for LLMs.

[1] Code and Skills available at `https://github.com/anonymous` (anonymized for review).

3 Musical Constraints in Lyrics Translation

A singable translation must satisfy three structural constraints, each arising from a different aspect of the relationship between lyrics and melody.

3.1 Syllable Count Constraint

In sung performance, each syllable maps to one or more musical notes. When a melody assigns three notes to a phrase, the lyric must contain exactly three syllables; fewer leave notes unmatched, while extra syllables force cramming that disrupts the rhythmic feel. This makes syllable count the most rigid of the three constraints: rhyme and phrasing mismatches may be tolerable in some styles, but syllable count mismatches are almost always audible. Formally, given source lyrics $L_s = (l_1, \ldots, l_n)$ in language λ_s, the constraint requires that translated lyrics $L_t = (l'_1, \ldots, l'_n)$ in target language λ_t satisfy $\text{syll}(l'_i, \lambda_t) = \text{syll}(l_i, \lambda_s)$ for all lines i.

Syllable counting is language-dependent. For Chinese, each character constitutes exactly one syllable, so counting is trivial. For other languages, we convert text to IPA via Phonemizer (Bernard and Titeux, 2021) and count *vowel nuclei*, the vocalic centers of syllables:

$$\text{syll}(l_i, \lambda_s) = \left|\text{nuclei}\big(\text{IPA}(l_i, \lambda_s)\big)\right| \qquad (1)$$

where nuclei$(\cdot)$ matches a predefined set of IPA vowel symbols and diphthongs (e.g., a, i, u, ə, aɪ, eɪ, oʊ), with adjacent vowels forming diphthongs counted as single nuclei. For example, "*Let it go*" yields IPA /lɛt ɪt ɡəʊ/ with three vowel nuclei [ɛ, ɪ, əʊ], giving a count of 3. The resulting sequence $\mathbf{s} = (s_1, \ldots, s_n)$ serves as the hard constraint for translation.[2]

3.2 Rhyme Scheme Constraint

Rhyme provides structural cohesion in lyrics, creating sonic expectations that are satisfying when met and jarring when violated. A song with an $AABB$ scheme feels fundamentally different from one with $ABAB$; preserving the rhyme scheme ensures the translation maintains the same pattern of sonic echoes as the original. Formally, a rhyme scheme is a labeling $R = (r_1, \ldots, r_n)$ where $r_i \in \{A, B, C, \ldots\}$ assigns each line to a rhyme class, with $r_i = r_j$ if and only if lines i and j rhyme. The constraint requires that the translation's scheme R_t match the source scheme R_s.

We detect rhyme by comparing phonetic endings rather than orthographic forms. For each line, we extract the *rhyme ending*: the IPA substring from the last vowel nucleus to the end of the transcription. For Chinese, we extract the pinyin final of the last character instead. The scheme is constructed by labeling the first line A, and assigning each subsequent line the label of any earlier line it rhymes with, or a new label otherwise. Consider the following example:

> *The stars shine bright with silver* ***light*** → /laɪt/ → ending: /aɪt/
> *And covers all in purest* ***white*** → /waɪt/ → ending: /aɪt/
> *I stand and watch the world* ***below*** → /bɪləʊ/ → ending: /əʊ/
> *As gentle winds begin to* ***blow*** → /bləʊ/ → ending: /əʊ/

Lines 1–2 share ending /aɪt/ (class A) and lines 3–4 share /əʊ/ (class B), yielding $AABB$. This IPA-based approach correctly identifies rhymes that orthographic comparison would miss (e.g., "weight" and "late") and rejects false rhymes based on spelling alone (e.g., "cough" and "through").

3.3 Syllable Pattern Constraint

Even when two lines share the same total syllable count, different word-level distributions create different rhythmic feels. Consider two eight-syllable lines: "un-for-get-ta-ble mel-o-dy" with pattern $[5, 3]$ versus "I can hear the mu-sic play" with pattern $[1, 1, 1, 1, 2, 2]$. The first has two long words producing a flowing feel, while the second has many short words producing a staccato rhythm. Word boundaries interact with note groupings and breaths in sung performance, so preserving the syllable pattern ensures the translation fits the melody at the phrasing level, not just the syllable level. Formally, the syllable pattern of a line l_i is a vector $\mathbf{p}_i = (p_{i,1}, \ldots, p_{i,k_i})$ recording the syllable count of each word. The constraint asks that the translation's pattern $\mathbf{p}'_i$ be similar to $\mathbf{p}_i$ while maintaining the same total: $\sum_j p'_{i,j} = \sum_j p_{i,j} = s_i$.

To extract patterns, we apply language-specific word segmentation (space-based tokenization for English and other space-delimited languages, and a neural tokenizer for Chinese), then compute each word's syllable count via Equation 1. We measure pattern similarity using a normalized align-

[2] We validate this vowel-nuclei counting against independent references in Appendix B.4 (within-±1 agreement of 92% vs. CMUdict for English, 89% vs. pyphen for Spanish); Chinese counts are exact by construction. Throughout, "exact" syllable matching is defined relative to this eSpeak-NG counting convention rather than to citation-form syllabification.

ment score. Given patterns of potentially different lengths, we pad the shorter one with zeros and compute:

$$\mathrm{sim}(\mathbf{p}, \mathbf{p}') = 1 - \frac{\sum_{j=1}^{m} |p_j - p'_j|}{\sum_{j=1}^{m} \max(p_j, p'_j)} \quad (2)$$

where $m = \max(|\mathbf{p}|, |\mathbf{p}'|)$. A score of 1.0 indicates an exact match; we treat ≥ 0.8 as acceptable. For finer-grained per-word feedback, the deployed `syllable-pattern-analyzer` skill computes a weighted composite of three sub-scores (position, length, and total similarity); see Appendix B.3 for the exact form. Equation 2 is the simplified core that all three sub-scores reduce to when patterns share equal length and total syllables. For example, "Let it go" has pattern $[1, 1, 1]$. A Chinese translation tokenized as two words with pattern $[2, 1]$ yields $\mathrm{sim}([1, 1, 1], [2, 1, 0]) = 0.5$ (poor alignment), whereas a translation tokenized as three single-character words gives pattern $[1, 1, 1]$ with $\mathrm{sim} = 1.0$.

4 IPA-Augmented Skill Framework

Figure 1 overviews the skill orchestration architecture.

4.1 IPA as Universal Phonetic Interface

The International Phonetic Alphabet provides a standardized, language-independent representation of speech sounds, making it an ideal intermediary for cross-lingual phonetic analysis: regardless of the source or target language, all phonetic operations (syllable counting, rhyme comparison, pattern analysis) can be performed on IPA transcriptions using the same algorithms. Our system uses Phonemizer (Bernard and Titeux, 2021) as the text-to-IPA backend, wrapping the eSpeak NG speech synthesizer with support for over 100 languages. Language codes are normalized internally (e.g., `zh`, `zh-cn`, `cmn` all map to Mandarin Chinese) to provide a uniform interface. For phonetic distance computation, we use PanPhon (Mortensen et al., 2016), which maps each IPA segment to a vector of articulatory features, enabling fine-grained similarity measurement.

4.2 Skill Suite

The orchestrator has access to a suite of Agent Skills (Table 1), each packaged as a `SKILL.md` description plus a deterministic verification script. The four IPA-based analytic skills (`syllable-counter`, `phonetic-analyzer`, `rhyme-analyzer`, `syllable-pattern-analyzer`) wrap functions that provide capabilities the LLM cannot reliably perform through text generation alone (Gao et al., 2023); `lyrics-validator` and `lyrics-translator` sit on top to compose them.

Skill	Description
`syllable-counter`	Count syllables via IPA vowel nuclei (or character count for Chinese)
`phonetic-analyzer`	Convert text to IPA via eSpeak NG and compute phonetic similarity
`rhyme-analyzer`	Detect the rhyme scheme by comparing IPA endings
`syllable-pattern-analyzer`	Compare target vs. current syllable patterns with structured feedback
`lyrics-validator`	Validate all three constraints jointly on a translated line set
`lyrics-translator`	Top-level orchestration skill coordinating the five sub-skills above through the multi-phase loop in §4.4

Table 1: Agent Skills exposed to the orchestrating LLM. Each skill is a directory with a `SKILL.md` description plus deterministic verification scripts; all skills accept a language parameter for language-specific processing.

The `syllable-counter` skill is the most frequently invoked: given a text string and language code, it converts to IPA and returns an integer syllable count (§3.1). The `phonetic-analyzer` skill exposes the raw IPA transcription, allowing the orchestrator to inspect phonetic realizations and reason about which syllables to modify, a form of chain-of-thought reasoning (Wei et al., 2022) grounded in phonetic observation. The `rhyme-analyzer` skill analyzes the IPA endings of a set of lines and returns the rhyme scheme (e.g., "$AABB$"), ensuring judgments are phonetic rather than orthographic. The `syllable-pattern-analyzer` skill compares a target syllable pattern against the current one, returning the similarity score (Equation 2), per-word differences, and natural-language suggestions for improvement. All skills are discovered by the orchestrating LLM through their `SKILL.md` descriptions and invoked via the host's tool-use interface.

4.3 Constraint Extraction

Before translation begins, the system extracts all three constraints from the source lyrics L_s in language λ_s: (1) **syllable counts** $\mathbf{s} = (s_1, \ldots, s_n)$ via `syllable-counter` on each line, (2) **rhyme**

Line	Text	Syll.	Pattern
1	The snow glows white	4	[1,1,1,1]
2	On the mountain tonight	6	[1,1,2,2]
3	Not a footprint to be seen	7	[1,1,2,1,1,1]
4	A kingdom of isolation	8	[1,2,1,4]

Rhyme scheme: $AABC$ (*white/tonight* rhyme via shared IPA ending /aɪt/; lines 3–4 are unmatched)

Table 2: Example constraint extraction from English source lyrics. Syllable counts are computed via IPA vowel nuclei; patterns via space-based word segmentation.

scheme R_s via `rhyme-analyzer` on all lines, and (3) **syllable patterns** $P_s = (\mathbf{p}_1, \ldots, \mathbf{p}_n)$ via `syllable-pattern-analyzer`. These constraints are passed to the orchestrator as part of the translation prompt, providing explicit targets for each line. Table 2 shows an example.

4.4 Multi-Phase Skill Orchestration

Translation proceeds through two phases, encoded as procedural instructions in the top-level `lyrics-translator` `SKILL.md` that the orchestrating LLM follows, invoking the sub-skills at each step. The design reflects the priority ordering identified by Low (2005): semantic content and rhythm (syllable counts) take precedence, with rhyme considered throughout but not given a dedicated rewrite phase.

Phase 1: Initial Translation with Skill-Guided Reasoning. The orchestrator receives the source lyrics, target language, and all extracted constraints from the `lyrics-translator` skill instructions. It generates an initial translation of all lines, interleaving reasoning, skill invocations to verify output, and adjustments within the same generation turn (a tool-use pattern in the spirit of Yao et al. (2023)). The skill instructions direct the orchestrator to consider all three constraints simultaneously, producing a "best-effort" initial translation.

Typically, the orchestrator reasons about the target syllable count for a line, drafts a translation, invokes `syllable-counter` to verify, observes the result, and adjusts as needed, continuing until all lines are translated.

Phase 2: Line-by-Line Syllable Refinement. The initial translation rarely satisfies all syllable constraints exactly. In this phase, the orchestrator processes each line individually, invoking `syllable-counter` and comparing the result to the target. If there is a mismatch, the skill returns explicit feedback (e.g., "Line i has s'_i syllables but the target is s_i; please revise while preserving meaning") and the orchestrator generates a revised translation. The skill is re-invoked, continuing for up to 10 iterations; if no exact match is found, the closest attempt is retained. This phase is critical because even a single syllable mismatch can make a line unsingable.

Priority, rhyme handling, and termination. The two-phase design prioritizes syllable counts (the hardest constraint), then exits. Rhyme is monitored throughout via `rhyme-analyzer` and `lyrics-validator` but receives no dedicated refinement phase: rhyme violations are generally less disruptive to singability than syllable mismatches (Low, 2005). The process terminates after Phase 2 with a final `lyrics-validator` pass.

4.5 Worked Example

To illustrate, consider translating "*Not a footprint to be seen*" (7 syllables, pattern $[1, 1, 2, 1, 1, 1]$) into Chinese. In Phase 1, the orchestrator produces a 7-character translation; `syllable-counter` confirms the match, so Phase 2 skips this line. For lines that come out off-count, Phase 2 iterates up to 10 times with explicit count feedback until `syllable-counter` reports a match or the closest candidate is retained.

5 Evaluation Metrics

Standard MT metrics such as BLEU measure n-gram overlap and do not capture whether a translation preserves musical structure. Kim et al. (2023) proposed a computational evaluation framework for singable lyric translation; building on this direction, we propose three complementary metrics, each targeting one of the three musical constraints.

5.1 Syllable Error Rate (SER)

SER measures the overall alignment between the source and translated syllable count sequences using the Levenshtein edit distance (Levenshtein, 1966):

$$\text{SER} = \frac{\text{Lev}(\mathbf{s}, \mathbf{s}')}{\max(|\mathbf{s}|, |\mathbf{s}'|)} \tag{3}$$

where $\mathbf{s} = (s_1, \ldots, s_n)$ and $\mathbf{s}' = (s'_1, \ldots, s'_m)$ are the source and translated syllable count sequences, and Lev operates over sequences of integers (with substitution cost equal to 1 regardless of the magnitude of the difference).

SER captures both *substitution errors* (a line has the wrong syllable count) and *structural errors* (lines are missing or added). SER $= 0$ indicates perfect preservation; higher values indicate greater divergence. Unlike per-line metrics, SER penalizes translations that drop or add lines, which is important because line structure directly corresponds to melodic phrases.

5.2 Syllable Count Relative Error (SCRE)

SCRE provides a normalized, per-line view of syllable accuracy, assuming the translation preserves the number of lines ($m = n$):

$$\text{SCRE} = \frac{1}{n} \sum_{i=1}^{n} \frac{|s_i - s'_i|}{s_i} \tag{4}$$

SCRE weights errors relative to line length: a two-syllable error on a four-syllable line ($\text{SCRE}_i = 0.5$) is penalized more heavily than on a twelve-syllable line ($\text{SCRE}_i = 0.17$). This reflects the intuition that short lines leave less room for syllabic deviation; a single extra syllable in a three-syllable phrase is far more noticeable than in a long verse. SCRE $= 0$ indicates all lines match exactly.

SER and SCRE are complementary: SER captures structural alignment including line count mismatches, while SCRE provides percentage-based accuracy that is more interpretable for line-by-line comparison.

5.3 Adjusted Rand Index (ARI)

To measure rhyme scheme preservation, we treat the rhyme scheme as a *clustering* of lines and compute the Adjusted Rand Index (Hubert and Arabie, 1985) between the source and translated clusterings.

Let $R_s = (r_1, \ldots, r_n)$ and $R_t = (r'_1, \ldots, r'_n)$ be the rhyme scheme labelings. The Rand Index (RI) counts the fraction of line pairs that are either in the same cluster in both schemes or in different clusters in both schemes. ARI adjusts RI for the expected value under random label assignments:

$$\text{ARI} = \frac{\text{RI} - \mathbb{E}[\text{RI}]}{\max(\text{RI}) - \mathbb{E}[\text{RI}]} \tag{5}$$

ARI $\in [-1, 1]$: 1 indicates the translation perfectly preserves the rhyme structure, 0 indicates chance-level agreement, and negative values indicate systematic disagreement. Crucially, ARI is robust to label permutation: a source with scheme $AABB$ and a translation with $BBAA$ receives ARI $= 1$, since the grouping structure is identical. We note that the rhyme scheme feeding ARI is built from *strict* exact-ending equality, whereas the system's `lyrics-validator` optimizes against a more permissive substring rule; reported ARI is therefore computed on a stricter criterion than the one the pipeline targets and is not inflated by that permissiveness (we quantify the gap in Appendix B.2).

Together, SER, SCRE, and ARI cover the structural dimensions of singable translation. They do not measure semantic fidelity or naturalness and are complementary to standard MT metrics such as BLEU and COMET.

5.4 CCVO Distance

To cross-validate against an external singability proxy, we additionally report Consonant Cluster and Vowel Openness Distance (CCVO; Štěpánková and Rosa, 2025): each syllable contributes a consonant-cluster marker (C/N) and a vowel-openness marker ($OO/VO/VV$), and CCVO is the per-line normalized Levenshtein distance between the source and translation label strings, averaged across lines (lower is better). We adopt it because Liu et al. (2026) find it the strongest automatic predictor of human MOS for singable translation (r=0.84), far above rhythm distance (r=−0.16). Per-language syllabification follows §3.1.

6 Experiments

6.1 Setup

We evaluate on English-to-Chinese (en → cmn) song translation using 100 test cases (5 lines each) from the publicly available test split of the parallel lyric corpus released by Ou et al. (2023a).[3] The dataset covers pop, ballad, rock, and musical-theatre genres; we draw the test items from the public split as released, without subsampling or balancing by genre, and report aggregate metrics over the full set rather than per-genre subsets. To probe language-pair generalization, we additionally run the framework on English-to-Spanish (en → es) and English-to-Japanese (en → ja) with Haiku (n=30 each, same English source lines retargeted to the new output language); for Spanish, syllabification uses `pyphen` before IPA conversion, and for Japanese, syllable counting uses mora counts via `pykakasi` after kanji-to-kana normalization

[3] https://huggingface.co/datasets/LongshenOu/lyric-trans-en2zh-data

Orchestrator	SER ↓	SCRE ↓	ARI ↑	CCVO ↓	LLM	Time
Supervised baseline						
CLT	0.686	0.135	0.314	0.353	–	1.4
Vanilla (no skills, single prompt)						
Haiku 4.5	0.814	0.245	0.306	0.468	1.0	51.3
Sonnet 4.6	0.810	0.251	0.229	0.480	1.0	24.2
Opus 4.7	0.824	0.282	0.264	0.485	1.0	7.5
BLT (full pipeline)						
Haiku 4.5	**0.000**	**0.000**	**0.767**	0.366	1.3	94.6
Sonnet 4.6	0.002	0.0001	0.729	0.368	1.2	121.8
Opus 4.7	0.002	0.0001	0.733	**0.364**	1.4	32.1

Table 3: Multi-orchestrator ablation on Ou 2023 en→zh test split (seed=42, n=100). CLT (Ou et al., 2023a) uses source-derived length constraints like BLT; its high SER but low SCRE means it gets lengths approximately, rarely exactly. **LLM**: mean calls per case. **Time**: median seconds per case.

with a kana-to-IPA mapping for CCVO. Our open-source release (Apache-2.0) ships the full Skill suite (six `SKILL.md` directories with their verification scripts), the IPA tool implementations, and evaluation scripts under the `blt-skills` repository, enabling reproduction on any user-supplied lyrics in any language pair supported by eSpeak-NG.

Implementation. The framework is implemented as Agent Skills: each phase is encoded as procedural instructions in the orchestrating `lyrics-translator SKILL.md`, which references the five sub-skills (§4.2) and invokes their verification scripts as needed (full prompt templates in Appendix A). For LLM inference, we evaluate three Claude orchestrators, Haiku 4.5, Sonnet 4.6, and Opus 4.7, invoked via the Claude Code CLI with `-effort medium`; the Skill suite is identical across orchestrators. The IPA verification scripts are exposed to the orchestrator through each skill's `SKILL.md` description.

Conditions. We compare three conditions on the Ou 2023 en→zh test split (seed=42, n=100):

- **Vanilla**: Single-prompt translation with no skills, providing a baseline for what a frontier LLM produces without IPA-based verification.
- **BLT**: The full two-phase pipeline (Phase 1 initial translation followed by Phase 2 line-by-line syllable-count refinement, up to 10 iterations per line), using the complete Skill suite. We refer to this condition as BLT throughout, mirroring the supervised CLT baseline.
- **CLT** (Ou et al., 2023a): A supervised mBART baseline fine-tuned with explicit length, rhyme, and word-boundary constraint tokens, representing a strongly length-controlled task-specific training approach (reproduction details in Appendix E).

We report SER, SCRE, ARI, and CCVO as defined in §5.

6.2 Results

Findings. Vanilla Claude SER stays at 0.81–0.82 across the three orchestrators, confirming that without the IPA skill suite even a frontier LLM cannot honor syllable counts. Under BLT all three orchestrators reach near-perfect syllable accuracy (Haiku SER=SCRE=0; Sonnet and Opus miss one line in 500), a >99% SER cut over Vanilla that confirms iterative skill-verified refinement, not better translation alone, drives constraint satisfaction. ARI is high and stable across orchestrators (0.73–0.77), well above CLT's 0.31, and average LLM calls stay between 1.2 and 1.4 (Table 3), as most lines satisfy their count on the initial pass.

Comparison with CLT. Given source-derived length constraints (as BLT receives), CLT reaches moderate syllable control (SCRE 0.135) and rhyme (ARI 0.314), but BLT beats it on both (SCRE≈0, ARI 0.77) without task-specific training; CLT's SCRE only improves to 0.028 when given oracle gold-translation lengths that BLT never receives.

CCVO cross-validates the gain. On the singability-correlated CCVO metric (§5.4), BLT reduces distance by 22–25% over Vanilla across all three orchestrators, which converge to within 0.005 of each other (Table 3), indicating a pipeline effect rather than a model-specific artifact. Since CCVO is the strongest automatic predictor of

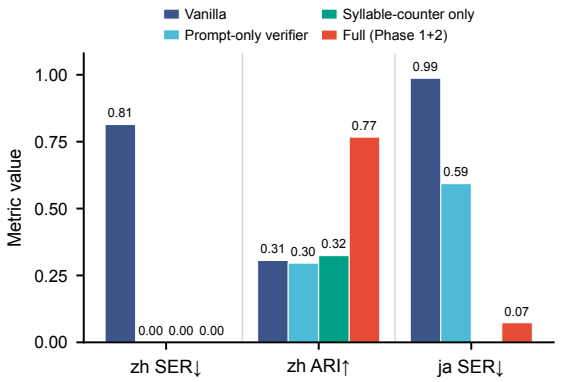

Figure 2: Component ablation (Haiku; en→zh n=100, en→ja n=30). The prompt-only verifier self-counts in-prompt with no tool; syllable-counter-only uses the tool but withholds the rhyme scheme (no ja run). Two readings: the tool is unnecessary for Chinese character counting (zh SER) but essential for Japanese mora (ja SER), and exposing the rhyme scheme is what drives rhyme preservation (zh ARI).

human MOS in Liu et al. (2026) (r=0.84), this gain corroborates the human study below.

Cross-lingual generalization. The unchanged pipeline transfers to two further target languages with Haiku (Table 4), requiring only an eSpeak-NG language-code swap and a small syllabification adapter (`pyphen` for Spanish, a `pykakasi` mora counter for Japanese). On Spanish, BLT reaches SER=SCRE=0 on all 30 cases with ARI rising to 0.634 (Spanish, like Chinese, has rich end-rhyme); on Japanese, SER falls 93%, and the low Japanese ARI reflects the scarcity of structural end-rhyme in Japanese pop rather than a pipeline failure (Appendix F).

Component ablation. To isolate each gain, we compare BLT against two ablations on Haiku (Figure 2): a *prompt-only verifier* that self-counts in-prompt with no tool call, and a *syllable-counter-only* variant that uses the tool but withholds the rhyme scheme from the Phase 1 prompt. The counter is decisive only where in-prompt counting fails: on en→zh the prompt-only verifier reaches SER=0 just like the tool (character counting is reliable in-prompt), but on en→ja it stalls at SER=0.59 while the tool reaches 0.07 (8×). Separately, withholding the rhyme scheme keeps SER near zero but collapses ARI from 0.77 to 0.32, so exposing the detected scheme is what preserves rhyme.

Condition	SER ↓	SCRE ↓	ARI ↑	CCVO ↓	Time
English-to-Spanish (en→es)					
Vanilla	0.882	0.368	0.093	0.517	7.2
BLT	**0.000**	**0.000**	**0.634**	**0.333**	147.4
English-to-Japanese (en→ja)					
Vanilla	0.987	0.894	0.032	0.974	12.3
BLT	**0.073**	**0.041**	**0.177**	**0.346**	163.2

Table 4: Cross-lingual generalization (Haiku, n=30 each, same Ou 2023 English source lines retargeted to the new output language). **Time**: median wall-clock seconds per case.

6.3 Qualitative Analysis

On the *Let It Go* excerpt (Table 2), Vanilla overshoots every target (7–9 characters), whereas BLT matches three of four exactly via Phase 2 counter checks, with the last line one syllable short after the refinement budget is exhausted (Table 7).

6.4 Human Evaluation

The structural metrics do not directly measure perceived singability. We run a blind A/B study on n=10 en→zh cases with six fluent-Mandarin raters (60 judgments, Opus): raters pick the more *singable* of two anonymized versions (Vanilla vs. BLT, randomized) and rate each 1–5 on singability and naturalness. BLT is preferred in 68% of pairs (41/60; binomial sign test p=0.006) and scores higher on both singability (mean MOS 4.02 vs. 3.35; p=0.04) and naturalness (3.98 vs. 3.22; p=0.03), with moderate inter-rater agreement (Krippendorff's α=0.53), confirming the gains are perceptible rather than metric artifacts, with no fluency penalty.

7 Conclusion

We presented BLT Skills, a training-free framework that packages IPA-based phonetic analysis as composable Agent Skills, orchestrated in two phases that prioritize syllable counts while monitoring rhyme. On English-to-Chinese (n=100) it cuts syllable error by over 99% over Vanilla, preserves rhyme better than a supervised baseline, and transfers to Spanish and Japanese. Ablations show deterministic counting is decisive where in-prompt counting fails, while exposing the rhyme scheme preserves rhyme structure. Offloading verifiable subtasks to deterministic, composable skills is a general recipe for constraint-bound generation; natural next steps include rhyme refinement and singing-voice synthesis (Ren et al., 2020).

Limitations

Our approach has several limitations. First, the framework's correctness depends on Phonemizer / eSpeak-NG for IPA transcription, which has uneven quality across languages; Appendix B.4 reports an intrinsic check (within-± 1 agreement of 92% for English and 89% for Spanish against independent references), but absolute syllable counts remain calibrated to eSpeak-NG rather than to citation-form syllabification. Second, the framework lacks a dedicated rhyme refinement phase: rhyme is monitored throughout but not explicitly rewritten. Third, our main evaluation is English-to-Chinese (n=100) with three Claude orchestrators; cross-lingual validation covers en→es and en→ja (n=30 each, Haiku only), and broader coverage across language pairs and orchestrators is left to future work. All runs use a single fixed seed (42); given the stochasticity of agentic orchestration, multi-seed runs would better characterize variance, which we leave to future work. Fourth, BLT's constraint satisfaction depends on the capability of the orchestrating LLM, whereas a supervised model such as CLT bakes length control into its weights; although our BLT pipeline matches or exceeds CLT on syllable metrics and ARI with all three orchestrators tested, weaker models may not. Fifth, the iterative Phase 2 refinement introduces latency (median 32–122 s for the Claude orchestrators via the API, vs. 1.4 s for CLT on Apple Silicon MPS per test case), which may limit practical deployment. Sixth, our automatic metrics measure structural constraint preservation rather than semantic fidelity; the human study (§6.4) covers singability and naturalness but only on a small sample, and a larger study with semantic-adequacy judgments would strengthen the evaluation. Finally, the technical approach of invoking deterministic phonetic skills from an orchestrating LLM applies established patterns (Yao et al., 2023; Gao et al., 2023; Schick et al., 2023) to a new domain; the primary contribution is the formalization of musical constraints and the IPA-based Skill suite rather than a novel agent architecture.

Ethical considerations. Song lyrics are protected by copyright in most jurisdictions. Our quantitative evaluation uses the en→zh test split released by Ou et al. (2023a), redistributed under the terms of that release; the qualitative *Let It Go* excerpts in Tables 2 and 7 are reproduced as short fragments for academic illustration under fair-use principles. Our open-source code does not bundle any lyric corpora; users supply their own input lyrics and are responsible for ensuring they hold the rights to translate and redistribute them. The framework's reliance on Phonemizer/eSpeak-NG also inherits that backend's uneven coverage across languages: dialectal, low-resource, or non-standard varieties may receive lower-quality IPA, which would propagate into both syllable counts and rhyme detection; users translating into such varieties should treat the structural metrics as advisory rather than authoritative. As with any high-fidelity translation tool, BLT Skills could in principle be used to produce singable derivative versions without rights-holder consent; we view rights-clearance as the user's responsibility and recommend that downstream services pair the framework with provenance tracking or licensing checks. The six raters in the human study (§6.4) were fluent-Mandarin volunteers recruited from the authors' acquaintances who participated without monetary compensation; they were informed in advance that their anonymized judgments would be used for research evaluation and consented prior to participating. The study collected only anonymized pairwise preferences and 1–5 ratings with no personal data, and was determined exempt from ethics-board review under our institution's guidelines.

References

Mathieu Bernard and Hadrien Titeux. 2021. Phonemizer: Text to phones transcription for multiple languages in Python. *Journal of Open Source Software*, 6(68):3958.

Jiale Chen, Xuelian Dong, Qihao Yang, Wenxiu Xie, and Tianyong Hao. 2025. Can large language models translate spoken-only languages through international phonetic transcription? In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 23431–23446.

Yiwen Chen and Simone Teufel. 2024. Scansion-based lyrics generation. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 14370–14381.

Woohyun Cho, Youngmin Kim, Sunghyun Lee, and Youngjae Yu. 2025. MAVL: A multilingual audio-video lyrics dataset for animated song translation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 13640–13668.

Shuangrui Ding, Zihan Liu, Xiaoyi Dong, Pan Zhang, Rui Qian, Junhao Huang, Conghui He, Dahua Lin,

and Jiaqi Wang. 2025. SongComposer: A large language model for lyric and melody generation in song composition. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7108–7127.

Johan Franzon. 2008. Choices in song translation: Singability in print, subtitles and sung performance. *The Translator*, 14(2):373–399.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, pages 10764–10799.

Marjan Ghazvininejad, Yejin Choi, and Kevin Knight. 2018. Neural poetry translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 67–71.

Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. 2016. Generating topical poetry. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1191.

Fenfei Guo, Chen Zhang, Zhirui Zhang, Qixin He, Kejun Zhang, Jun Xie, and Jordan Boyd-Graber. 2022. Automatic song translation for tonal languages. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 729–743.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546.

Jack Hopkins and Douwe Kiela. 2017. Automatically generating rhythmic verse with neural networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 168–178.

Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *Journal of Classification*, 2(1):193–218.

Wenxiang Jiao, Wenxuan Wang, Jen-tse Huang, Xing Wang, Shuming Shi, and Zhaopeng Tu. 2023. Is ChatGPT a good translator? Yes with GPT-4 as the engine. *arXiv preprint arXiv:2301.08745*.

Haven Kim, Jongmin Jung, Dasaem Jeong, and Juhan Nam. 2024. K-pop lyric translation: Dataset, analysis, and neural-modelling. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 9974–9987.

Haven Kim, Kento Watanabe, Masataka Goto, and Juhan Nam. 2023. A computational evaluation framework for singable lyric translation. In *Proceedings of the 24th International Society for Music Information Retrieval Conference*.

Jey Han Lau, Trevor Cohn, Timothy Baldwin, Julian Brooke, and Adam Hammond. 2018. Deep-speare: A joint neural model of poetic language, meter and rhyme. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1948–1958.

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Qihao Liang, Xichu Ma, Finale Doshi-Velez, Brian Lim, and Ye Wang. 2024. XAI-lyricist: Improving the singability of AI-generated lyrics with prosody explanations. In *Proceedings of the 33rd International Joint Conference on Artificial Intelligence*, pages 7877–7885.

Hanze Liu, Yusuke Sakai, and Taro Watanabe. 2026. Towards singable lyrics translation using large language models. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 544–554.

Peter Low. 2005. The pentathlon approach to translating songs. In Dinda L. Gorlée, editor, *Song and Significance: Virtues and Vices of Vocal Translation*, pages 185–212. Rodopi, Amsterdam.

Peter Low. 2022. Translating the texts of songs and other vocal music. In Kirsten Malmkjær, editor, *The Cambridge Handbook of Translation*. Cambridge University Press.

David R. Mortensen, Patrick Littell, Akash Bharadwaj, Kartik Goyal, Chris Dyer, and Lori Levin. 2016. PanPhon: A resource for mapping IPA segments to articulatory feature vectors. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3475–3484.

Longshen Ou, Xichu Ma, Min-Yen Kan, and Ye Wang. 2023a. Songs across borders: Singable and controllable neural lyric translation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 447–467.

Longshen Ou, Xichu Ma, and Ye Wang. 2023b. LOAF-M2L: Joint learning of wording and formatting for singable melody-to-lyric generation. *arXiv preprint arXiv:2307.02146*.

Matt Post and David Vilar. 2018. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1314–1324.

Le Ren, Xiangjian Zeng, Qingqiang Wu, and Ruoxuan Liang. 2025. LyriCAR: A difficulty-aware curriculum reinforcement learning framework for

controllable lyric translation. *arXiv preprint arXiv:2510.19967*.

Yi Ren, Xu Tan, Tao Qin, Jian Luan, Zhou Zhao, and Tie-Yan Liu. 2020. DeepSinger: Singing voice synthesis with data mined from the web. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1979–1989.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36.

Barbora Štěpánková and Rudolf Rosa. 2025. Song lyrics adaptations: Computational interpretation of the pentathlon principle. In *Proceedings of the 5th International Conference on Natural Language Processing for Digital Humanities*, pages 117–128.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the Eleventh International Conference on Learning Representations*.

Zhuorui Ye, Jinhan Li, and Rongwu Xu. 2024. Sing it, narrate it: Quality musical lyrics translation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 5498–5520.

Suhyeon Yoo, Khai N. Truong, and Young-Ho Kim. 2025. ELMI: Interactive and intelligent sign language translation of lyrics for song signing. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*.

Jiaxing Yu, Xinda Wu, Yunfei Xu, Tieyao Zhang, Songruoyao Wu, Le Ma, and Kejun Zhang. 2025. SongGLM: Lyric-to-melody generation with 2D alignment encoding and multi-task pre-training. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence*, pages 25742–25750.

Songyan Zhao, Bingxuan Li, Yufei Tian, and Nanyun Peng. 2025. REFFLY: Melody-constrained lyrics editing model. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11295–11315.

A Prompt Templates

We provide the full prompt templates used in each phase of the two-phase orchestration described in §4.4. All prompts live inside `skills/lyrics-translator/SKILL.md` and the sub-skill descriptions, which the orchestrating LLM loads on demand.

System prompt (all phases, from `lyrics-translator/SKILL.md`).

```
You are a lyrics translator. Translate
ONE line at a time while matching the
target syllable count.

Available skills:
- syllable-counter: count syllables of a
candidate translation
- lyrics-validator: verify a line
against all three constraints

WORKFLOW:
1. Translate the source line
2. Invoke syllable-counter to check your
translation
3. If the count does not match, revise
and re-check
4. Once correct, output the final
translation with no punctuation

IMPORTANT: Always verify with the skills
before finalizing your answer.
```

Phase 1: Initial translation. The orchestrator receives all source lines with their target counts, rhyme scheme, and patterns, and translates *all* lines at once while invoking skills to verify constraints:

```
Translate ALL N lines of the following
lyrics to {target_lang} while meeting
ALL 3 musical constraints.

CONSTRAINT 1: SYLLABLE COUNTS (MUST
MATCH EXACTLY)
CONSTRAINT 2: RHYME SCHEME:
{rhyme_scheme}
CONSTRAINT 3: SYLLABLE PATTERNS:
{patterns}

Invoke the available skills to verify:
(1) syllable counts match exactly, (2)
rhyme scheme is correct, (3) syllable
patterns are maintained.

Output exactly N translated lines,
numbered 1-N.
```

Phase 2: Syllable refinement. For each mismatched line, the orchestrator receives the current best translation and explicit corrective feedback from `syllable-counter`:

```
Adjust this translation to have EXACTLY
{target} syllables. Focus ONLY on
syllable count. Minimize meaning
changes.

Original: "{source_line}"
Current: "{best_translation}"
```

```
Current syllables: {actual}/{target}
Action: {add/remove k syllables}

STRATEGIES:
- If too long: remove adjectives, use
shorter words, merge concepts
- If too short: add descriptive words,
use longer characters

Output ONLY the adjusted translation.
```

B Skill Algorithms

We describe the core algorithms underlying each IPA skill.

B.1 `syllable-counter`

1. Remove punctuation from input text.

2. **Chinese:** Return the character count directly (each character = one syllable).

3. **Other languages:** Convert text to IPA via Phonemizer / eSpeak-NG.

4. Match all diphthong and monophthong nuclei against the inventories below; diphthongs are tried first so adjacent vowels forming a single nucleus are not double-counted.

   Diphthongs (longest-match, in order):

   aɪ, eɪ, ɔɪ, aʊ, oʊ, ɪə, eə, ʊə, aɪə, aʊə

   Monophthong nuclei: a fixed class of 27 IPA vowel symbols spanning cardinal (e.g. i, ɛ, a, u), central (e.g. ə, ɜ), rounded-front (e.g. y, ø, œ), and back-unrounded plus rhotacised vowels; the exact codepoints are enumerated in `src/blt_skills/phonetics.py:13-16` of the release.

   The matcher tolerates trailing combining marks (length, nasalization, tone) and uses only this vowel set, never matching consonant symbols.

5. Return the number of matches as the syllable count (one match per vowel nucleus).

We treat eSpeak-NG's IPA output as ground truth for vowel nuclei.

B.2 `rhyme-analyzer`

1. For each line, extract the *rhyme ending*:

   - **Chinese:** Use PyPinyin to extract the final (韻母) of the last character.
   - **Other languages:** Convert to IPA, find the last vowel nucleus, return the substring from that position to end-of-string.

2. Build the rhyme scheme used for ARI by iterating through lines and grouping by *exact* ending equality: maintain a hash map from ending strings to labels; for each line, if its ending key is already in the map, reuse the label, otherwise assign the next unused label and insert it.

3. Pair-level rhyme judgments inside the `lyrics-validator` skill use a more permissive rule: two lines rhyme if $\texttt{ending}_1 == \texttt{ending}_2 \vee \texttt{ending}_1 \subset \texttt{ending}_2 \vee \texttt{ending}_2 \subset \texttt{ending}_1$.

The substring rule in step 3 lets the validator accept a short ending like /aɪ/ as rhyming with a longer one like /aɪt/, mirroring the audible vowel match that drives perceived rhyme in singing. Crucially, the substring rule is *not* used when constructing the scheme that feeds into the ARI metric: ARI is computed on the strict, exact-match scheme produced in step 2, so reported ARI values are not inflated by the validator's permissiveness. We quantify this gap on the en→zh BLT outputs (three orchestrators, 3,000 line pairs): the permissive substring rule labels 26.8% of line pairs as rhyming, whereas the strict exact-equality scheme used for ARI labels only 9.7%; 64% of the pairs the validator accepts fail the exact test. This also explains why ARI improves over Vanilla yet remains well below 1.0. A natural extension would replace exact equality in step 2 with a PanPhon (Mortensen et al., 2016) feature-distance threshold on the rhyming nuclei; we leave a sensitivity analysis of this threshold to future work.

B.3 `syllable-pattern-analyzer`

Given target pattern $\mathbf{p}$ and current pattern $\mathbf{p}'$:

1. Compute position-by-position differences for each word position j: $d_j = p'_j - p_j$ (zero-padded to the longer length).

2. Compute three sub-scores:

   - *Position similarity* (weight 0.5): $1 - \sum_j |d_j| / \sum_j p_j$
   - *Length similarity* (weight 0.25): $1 - ||\mathbf{p}| - |\mathbf{p}'|| / \max(|\mathbf{p}|, |\mathbf{p}'|)$
   - *Total similarity* (weight 0.25): $1 - |\sum p_j - \sum p'_j| / \sum p_j$

3. Return the weighted sum, clipped to $[0, 1]$, along with per-word suggestions (e.g., "word 3: has 3 syllables, target is 1").

Note: the similarity in the main paper (Equation 2) uses the simplified formulation; the implementation adds length and total sub-scores for finer-grained feedback.

B.4 Intrinsic Validation of the Phonetic Layer

Because the framework's central claim is exact satisfaction of syllable constraints, we validate the counting layer against independent references rather than treating it as self-evidently correct. Chinese counts are exact by construction (one character maps to one syllable), and Japanese uses a deterministic mora count over the kana transcription (via `pykakasi`); neither uses IPA vowel nuclei. For the languages actually counted through eSpeak-NG vowel nuclei, we measure agreement with established references:

Language	Reference	Exact	Within ± 1
English	CMUdict	62%	92%
Spanish	pyphen	54%	89%

Table 5: eSpeak-NG vowel-nuclei syllable counts vs. independent references (English: 475 source lines vs. CMUdict vowel-phoneme counts; Spanish: 285 output lines vs. pyphen syllabification).

For English, exact agreement rises from 62% to 80% (and within-± 1 from 92% to 97%) when the GOAT vowel is transcribed with the American close-mid onset [oʊ] rather than the mid-central (schwa) onset of the British [əʊ] that eSpeak-NG's `en-gb` voice emits; almost the entire residual is this single systematic vowel-transcription choice rather than random error. The remaining Spanish disagreements stem from eSpeak-NG's context-dependent diphthong-versus-hiatus decisions (e.g., *cantautor*) and cross-word vowel merging in connected speech, neither of which is a counting bug per se. Two points bound the practical impact. First, the same counting procedure defines the per-line target for *every* condition, so this calibration offset shifts Vanilla and BLT identically and does not affect the relative comparison that drives our conclusions; reported counts are calibrated to eSpeak-NG's transcription rather than to citation-form syllabification. Second, the Chinese target side of our main en→zh results is counted exactly, so the headline constraint-satisfaction results do not depend on eSpeak-NG accuracy at all.

C Hyperparameters

Table 6 lists all hyperparameters used in the experiments.

Category	Parameter	Value
Model	Orchestrator	Haiku/Sonnet/Opus
Model	Interface	Claude Code CLI
Model	Reasoning effort	`medium`
Phase 1	Max skill invocations	10
Phase 1	Skill set	All 6 skills
Phase 2	Max attempts / line	10
Phase 2	Skill calls / attempt	5
Orchestr.	Recursion limit	50
Orchestr.	Max lines / test	5
Segment.	English	Whitespace
Segment.	Chinese	HanLP

Table 6: Hyperparameters for all experiments.

D Extended Qualitative Examples

Line	Text	Syll.	Tgt
Source (English):			
1	The snow glows white	4	–
2	On the mountain tonight	6	–
3	Not a footprint to be seen	7	–
4	A kingdom of isolation	8	–
Vanilla (Chinese, no tools):			
1	白雪在夜裡閃耀	7	4 ×
2	今夜在那座山上	7	6 ×
3	看不見任何一個腳印	9	7 ×
4	一個與世隔絕的王國	9	8 ×
BLT (Chinese, with IPA skills):			
1	皚皚白雪	4	4 ✓
2	籠罩今夜山巔	6	6 ✓
3	看不到一絲足跡	7	7 ✓
4	這是孤獨的國度	7	8 ×

Table 7: Qualitative example: English-to-Chinese translation of the opening lines of *Let It Go*. "Syll." is the actual character count; "Tgt" is the source syllable count that the translation should match. BLT matches 3/4 targets; Vanilla matches 0/4.

Table 8 shows a *failure* case: line 4 requires 8 syllables but the agent converges on 7 after exhausting 10 attempts.

This oscillation accounts for many Phase 2 failures, particularly when the target count falls between two "natural" translation lengths.

E CLT Baseline Details

The Controllable Lyric Translation (CLT) baseline (Ou et al., 2023a) uses a fine-tuned mBART model

L	Source	Tgt	P1	P2
1	The snow glows white	4	4 ✓	4 ✓
2	On the mountain tonight	6	6 ✓	6 ✓
3	Not a footprint to be seen	7	9 ×	7 ✓
4	A kingdom of isolation	8	9 ×	7 ×
5	And it looks like I'm the queen	8	8 ✓	8 ✓

Table 8: Failure case (en → cmn). Line 4 targets 8 syllables; Phase 2 reduces from 9 to 7 but overshoots, and subsequent attempts oscillate between 7 and 9 without hitting 8.

with explicit constraint tokens prepended to the encoder input. We use the publicly released checkpoint `LongshenOu/lyric-trans-en2zh`.

Constraint encoding. CLT encodes three constraints as special tokens: (1) a *length token* specifying the target character count, (2) a *rhyme token* specifying the target rhyme class (e.g., `a/ia/ua`), and (3) a *word boundary token* vector indicating which positions should be word boundaries. The model generates characters in *reverse order* (right-to-left) and the output is flipped to produce the final translation.

Reported metrics. The original paper reports 99.85% length accuracy on their test set using *reference* (gold-translation) length tokens. On our n=100 sample, feeding CLT those oracle reference lengths reproduces strong length control (SCRE 0.028); under *source*-derived length tokens, the setting comparable to BLT, CLT reaches SCRE 0.135 and ARI 0.314 (Table 3). We reproduce CLT with the authors' released checkpoint and their forked `transformers`; on stock `transformers` the constraint decoding degenerates, so the fork is required.

Inference time. On a 30-case sample of the en→zh test split, CLT averages 1.4 s per case (median 1.5 s, range 0.6–1.9 s) on Apple Silicon MPS with `num_beams=5` and `max_length=36`. This places CLT one to two orders of magnitude faster than the Claude orchestrators under the BLT pipeline (median 32–122 s via API; Table 3). The gap is architectural: CLT is a 600M-parameter mBART decoded once per song, whereas BLT issues multiple LLM calls when Phase 2 is triggered, so even on identical hardware CLT would remain substantially faster.

Key differences from BLT. CLT requires: (1) a parallel lyric corpus for fine-tuning, (2) per-language-pair training, and (3) pre-specified constraint values at inference time. BLT requires none of these: it extracts constraints automatically from source lyrics and operates with any general-purpose LLM. The trade-off is that CLT runs far faster once trained, while BLT matches or exceeds it on syllable metrics and additionally preserves rhyme structure and language flexibility.

F Orchestration and Error Analysis

Phase 2 trigger rate and skill usage. Across the en→zh BLT runs (three orchestrators, 300 cases), only 13% of cases required any Phase 2 refinement; the rest satisfied all syllable counts in Phase 1. The mean number of LLM calls per case is 1.29, with a heavily skewed distribution (261 of 300 cases use a single call; the maximum observed is 12). Skill invocations are dominated by `syllable-counter` (89% of all calls), with `rhyme-analyzer` (9%) and `syllable-pattern-analyzer` (2%) used more sparingly, consistent with the pipeline's syllable-count priority.

Metric sensitivity. SER uses a uniform substitution cost, ignoring the magnitude of a per-line count mismatch. Recomputing with a magnitude-weighted substitution cost (normalized by total source syllables) leaves the conclusion unchanged: Vanilla scores 0.806 (uniform) / 0.206 (magnitude-weighted) and BLT scores 0.0013 / 0.0001, so both variants show the same near-total error reduction. SCRE (§5) already provides the magnitude-sensitive, per-line view.

Cross-lingual rhyme (en→ja). The low Japanese ARI (Table 4) is a genuine linguistic property rather than a metric artifact. On the en→ja outputs the permissive validator rule and the strict ARI scheme agree exactly (0% divergence), because Japanese rhyme endings reduce to one of five vowels and leave no proper-substring relationships. The 25.9% of line pairs sharing a final vowel is close to the chance rate for a five-vowel system, confirming that Japanese pop lyrics carry little structural end-rhyme for the pipeline to preserve. Spanish, with richer syllable codas, shows the expected gap: the validator accepts 22.0% of pairs as rhyming versus 11.3% under the exact ARI scheme.