IPA-Verified Singable Lyrics Translation with LLM Agents

Guan-Ming Chiu
National Taiwan University
gmchiu@arbor.ee.ntu.edu.tw

Chiao-Chih Cheng
National Taiwan University
r14921040@ntu.edu.tw

Kuan-Wei Lee
National Taiwan University
r14921101@ntu.edu.tw

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 syllable counts and rhyme 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). Experiments across three frontier orchestrators, with component ablations, confirm that tool-verified translation 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.

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* (§6.2); following program-aided reasoning (Gao et al., 2023), we offload them to deterministic scripts.

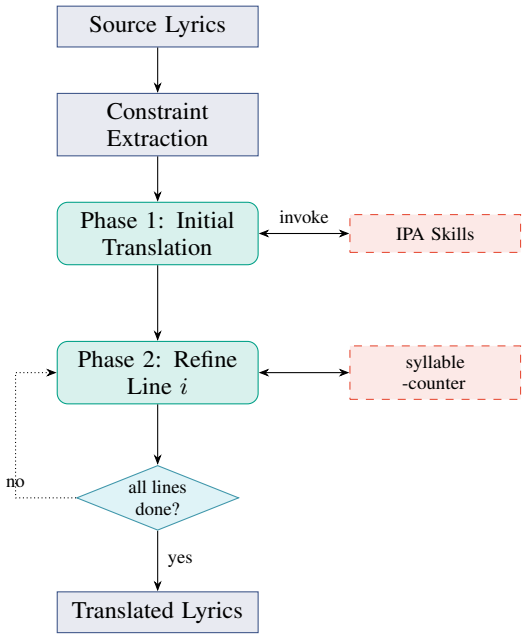


Figure 1: Skill framework overview. Phase 1 translates all lines using an internal tool-use loop that invokes the IPA skills. Phase 2 refines each line sequentially (up to 10 attempts per line); the dotted arrow shows the orchestration loop that iterates through lines.

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.[1] Unlike supervised approaches, the framework requires no parallel lyric corpora, supports any language pair via eSpeak-NG, and

[1] https://agentskills.io/specification

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 24826–24841
October 24-29, 2026 ©2026 Association for Computational Linguistics

scales with LLM capability. Our framework is open-source for reproducibility and practical use.[2]

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 interactive sign-language translation for song signing (Yoo et al., 2025). Closest to our setting, Liu et al. (2026) compare zero-shot prompting strategies for singable lyric translation, including verification-guided and multi-round refinement. 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 fine-tuning, constrained decoding, reinforcement learning, or prompt-only verification; we instead augment a general-purpose LLM with *external* IPA-verification skills and compare to the supervised constraint-token baseline closest in task setting (Ou et al., 2023a; §6).

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 decoding time; ours lets the LLM generate freely and verifies via external phonetic skills.

2.3 Tool-Augmented Language Models

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); 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.

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 (§4); Chen et al. (2025) likewise use IPA as a language-agnostic interface for LLM translation of spoken-only languages.

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. In a syllabic setting, where each note carries one syllable, a phrase set to three notes must contain exactly three syllables; fewer leave notes unmatched, while extra syllables force

[2]Code and Skills available at `https://github.com/guan404ming/blt-skills`.

cramming that disrupts the rhythmic feel. Melismatic settings let one syllable span several notes, but preserving the source alignment of syllables to notes still requires matching the source count, which is the constraint we adopt. 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 Han character constitutes exactly one syllable under our counting convention, 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 $\text{nuclei}(\cdot)$ 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 gəʊ/ 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.[3]

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 (a diphthong counts as one 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-ing" 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 `jieba` segmentation for Chinese), then compute each word's syllable count via Equation 1. We measure pattern similarity using a normalized alignment score. Given patterns of potentially different lengths, we pad the shorter one with zeros and compute:

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

[3] We validate this vowel-nuclei counting against independent references in Appendix B.4 (within-±1 agreement of 97.5% vs. CMUdict for English, 92.0% vs. pyphen for Spanish); Chinese counts one syllable per Han character. Throughout, "exact" syllable matching is defined relative to this eSpeak-NG counting convention rather than to citation-form syllabification.

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 position-level core; the released implementation combines it with length and total-count sub-scores into a weighted composite (Appendix B). For example, "Let it go" has pattern $[1, 1, 1]$. A Chinese translation tokenized as two words with pattern $[2, 1]$ yields $\text{sim}([1, 1, 1], [2, 1, 0]) = 0.5$ (poor alignment), whereas a translation tokenized as three single-character words gives pattern $[1, 1, 1]$ with $\text{sim} = 1.0$.

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. Four of them are IPA-based analytic skills that wrap deterministic computations the LLM cannot perform reliably in text (§6.2), in the spirit of program-aided reasoning (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 (characters 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 syllable counts and rhyme scheme on a line set (patterns optional)
`lyrics-translator`	Top-level skill coordinating the five sub-skills through the 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, letting the orchestrator inspect phonetic realizations and reason about which syllables to modify. 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 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.

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.

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. Prioritizing syllable counts is our engineering choice, informed by Low (2005), who balances singability, sense, naturalness, rhythm, and rhyme and treats rhyme as the most negotiable: rhyme is considered throughout but has no 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 agent trajectory (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, only post-validation rewrites of lines that break the detected scheme, constrained to keep their syllable counts and repeated until validation passes or the agent gives up (median one validator pass, at most four in our runs): we treat rhyme as the more negotiable constraint, following the flexibility Low (2005) allows it. The process terminates after Phase 2 with a final `lyrics-validator` pass that checks syllable counts and the rhyme scheme (lines that share a label must rhyme; lines with different labels must not).

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: SER and SCRE for syllable counts and ARI for the rhyme scheme; the syllable pattern is used as soft guidance during drafting and is not scored.

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 $\in [-0.5, 1]$ (Pedregosa et al., 2011): 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. The rhyme scheme feeding ARI and the pair test inside `lyrics-validator` both use exact equality of the extracted ending (diphthong-aware, Appendix B.2), so the validator optimizes the same criterion that ARI measures.

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); note that they define CCVO as a distance yet report a positive correlation with MOS, so we treat only the magnitude of that finding as evidence and report CCVO as a distance throughout. Per-language syllabification follows §3.1.

Validity of SER, SCRE, and ARI. Within a single strong system these metrics have almost no variance, so we validate them across systems: pooling item-level scores of Vanilla, CLT, and BLT on en→zh (Haiku, n=296 completed items) against CCVO, all three correlate in the expected direction, SCRE most strongly (Pearson r=0.71, Spearman ρ=0.56), then SER (r=0.47, ρ=0.48), then ARI (r=−0.20, ρ=−0.25; higher ARI, lower distance). Item-cluster bootstrap 95% CIs for Pearson r (the 100 items recur across systems): SCRE $[0.61, 0.80]$, SER $[0.39, 0.55]$, ARI $[-0.31, -0.09]$. SER and SCRE measure syllable structure, which CCVO encodes; ARI measures rhyme, which CCVO captures only indirectly.

System	LLM	n	SER ↓	SCRE ↓	ARI ↑	CCVO ↓	Turns	Time
CLT (supervised)	–	100	0.284	0.033	0.422	**0.318**	–	1.4
Vanilla	Haiku 4.5	100	0.790	0.216	0.343	0.414	1	14.8
Vanilla	Sonnet 5	100	0.788	0.220	0.261	0.417	1	7.3
Vanilla	Opus 5	87	0.802	0.209	0.279	0.409	1	10.6
Prompt-only	Haiku 4.5	100	0.234	0.032	0.518	0.322	3	67.3
BLT	Haiku 4.5	96	0.008	0.003	0.990	0.320	18	106.6
BLT	Sonnet 5	99	0.008	0.002	0.996	0.325	9	71.0
BLT	Opus 5	89	**0.000**	**0.000**	**1.000**	0.314	8	63.4

Table 3: Main results on the Ou 2023 en→zh test windows (n=100 items, 5 lines each; orchestrators `claude-haiku-4-5-20251001`, `claude-sonnet-5`, `claude-opus-5`). n is the number of items that produced output: Prompt-only denies every tool, so the orchestrator self-counts (Liu et al., 2026). BLT lost 3 / 1 / 11 items (Haiku / Sonnet / Opus) to the API content filter and one Haiku item to an unparsable reply; Opus Vanilla declined 13 items as copyrighted. Scoring lost items as fully unmatched gives BLT SER 0.048 / 0.018 / 0.110 and ARI 0.950 / 0.986 / 0.890 (Vanilla SER 0.790 / 0.788 / 0.828). CLT (Ou et al., 2023a) receives the same per-line syllable targets as BLT as its length tokens, plus the dataset's own rhyme and boundary tokens. **Turns**: median assistant turns per item; **Time**: median seconds per item for the LLM conditions (via API) and the mean over a 30-case sample for CLT (Apple Silicon MPS). Human A/B study (§6.3, Opus, n=10): BLT preferred in 93% of 60 judgments, 10 of 10 items.

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).[4] Each test item is a window of five consecutive non-empty lines from the public test split (4,006 lines), with window starts sampled uniformly (seed 42); windows do not respect song or stanza boundaries and 8 pairs overlap (479 distinct source lines in 500 slots); we report aggregate metrics over the full set. 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); Spanish syllable counts use eSpeak-NG vowel nuclei like English (`pyphen` is used only inside the CCVO adapter), and Japanese counts are morae via `pykakasi` after kanji-to-kana normalization, so the Japanese SER and SCRE are mora-count errors. Our open-source code (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.

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

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 tells the agent which verification script to run at each step (Appendix A). The harness sends one prompt per test item through the Claude Code CLI (`claude -p, -effort medium`) containing only the source lines and the instruction to use the `lyrics-translator` skill; the agent discovers the skills from their `SKILL.md` descriptions, extracts the constraints itself by running the scripts, translates, and verifies, with `Skill`, `Read`, and `Bash` as the only allowed tools. Every run is logged as a `stream-json` trace, from which we count skill loads, script invocations, turns, tokens, and cost (Appendix F); the harness never counts syllables or injects targets into the prompt. Items fail in three ways: the API content filter rejects the request (three attempts), the model declines in text to translate copyrighted lyrics (not retried), or the final message cannot be parsed into n lines. Table 3 reports completed-item metrics in its columns and failure-adjusted metrics, scoring each failed item as fully unmatched, in its caption.

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

- **Vanilla**: One plain translation prompt through the same traced harness with tools disallowed (no trace contains a tool call), the baseline for what a frontier LLM produces alone.

- **Prompt-only**: The same workflow with every tool denied, so the orchestrator counts syllables and judges rhyme in its own reasoning, as in the verification-guided prompting of Liu et al. (2026) (Haiku only; Figure 2).

- **BLT**: The agent-driven two-phase pipeline (constraint extraction, Phase 1 initial translation, Phase 2 line-by-line syllable-count refinement with up to 10 attempts per line, final validation of counts and rhyme), with the agent invoking the skills itself. 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. Without tools, all three orchestrators leave about four lines in five off-count (SER 0.79–0.80) and preserve almost no rhyme on rhymed sources (ARI 0.09–0.15), confirming that syllable counting and rhyme matching are unreliable in-prompt even for frontier models. BLT cuts SER by 99% to 0.008 / 0.008 / 0.000 (Haiku / Sonnet / Opus; 476 of 480 Haiku lines exact) and lifts ARI to 0.990 / 0.996 / 1.000, because the agent runs `lyrics-validator` on its draft and rewrites lines that break the detected scheme (Table 3). On the completed items whose source actually rhymes (at least one repeated label; 42 for Haiku BLT, 43 for Vanilla and CLT), Haiku BLT scores ARI 0.977 versus 0.099 for Vanilla and 0.029 for CLT; the remaining unrhymed items are trivially satisfiable and all three score higher there (1.000, 0.526, 0.719). Sonnet reaches the same accuracy in half the turns by batching script runs, and Opus completes all 89 of its items perfectly, at 18 / 9 / 8 turns and a median 107 / 71 / 63 s per item versus one call and 7–15 s for Vanilla (Appendices D and F).

Comparison with CLT. CLT, given the same syllable targets as length tokens, matches 72% of lines exactly and overshoots by one syllable on most of the rest (SER 0.284, SCRE 0.033), so its length control is approximate rather than exact; its ARI of 0.422 (0.029 on rhymed items) shows that its rhyme token does not transfer the source scheme. BLT beats it on SER, SCRE, and ARI without task-specific training, with nearly identical CCVO.

CCVO cross-validates the gain. On the external CCVO distance (§5.4) BLT scores 0.320 and CLT 0.318, both a 23% reduction over Vanilla (0.414); CCVO encodes consonant clusters and vowel openness, which both constrained systems improve equally, whereas rhyme, where they differ most, is outside its scope.

Cross-lingual generalization. The unchanged pipeline transfers to two further target languages with Haiku (Table 4), requiring only an eSpeak-NG language-code swap for Spanish and a `pykakasi` mora counter for Japanese. On Spanish, BLT reaches SER=SCRE=0 and ARI=1.0 on all 30 items (Vanilla: SER 0.913, ARI 0.091); on Japanese, SER falls from 0.980 to 0.013 (28 of 30 items perfect) and ARI rises from -0.022 to 0.615. The lower Japanese ARI is a property of the language: Japanese rhyme endings reduce to one of five vowels, so a five-line window whose source has no repeated label often cannot avoid a repeated vowel in the translation (ARI 0.444 on unrhymed sources versus 0.872 on rhymed ones, Appendix F). Japanese also costs the most agent effort (median 44 turns, 265 s), because kanji readings cannot be counted in-prompt and every line goes through the counter.

Component ablation. To isolate each gain, we run three skill variants with Haiku on the same items (Figure 2): a *prompt-only* variant with all tools denied, whose `SKILL.md` tells the agent to count syllables and judge rhyme in its own reasoning, analogous to the verification-guided prompting of Liu et al. (2026); a *counter-only* variant that keeps `syllable-counter` but never extracts or checks the rhyme scheme; and the *Phase 1 only* variant of the next paragraph. Prompt-only self-verification leaves 66 of 100 items imperfect and rhyme well short of BLT because the model's own counts are wrong; the deterministic counter alone drives SER to 0.006 (counter-only, 98 of 100 exact). Withholding the rhyme scheme costs nothing on syllables but collapses ARI to 0.250 (0.048 on rhymed sources), so exposing the detected scheme and validating against it is what preserves rhyme.

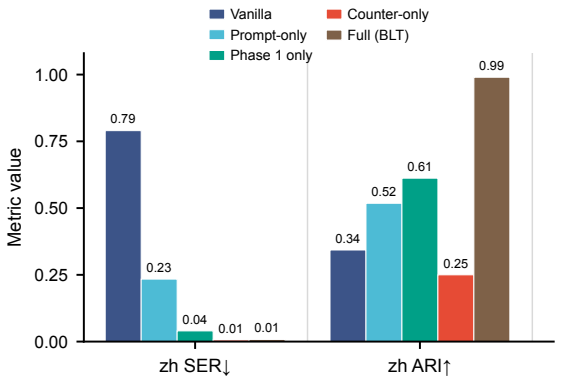


Figure 2: Component ablation (Haiku, en→zh, same 100 items; refused items excluded). Prompt-only denies all tools and lets the agent self-count; Phase 1 only extracts constraints with the scripts and translates once; counter-only keeps the syllable counter but hides the rhyme scheme. The deterministic counter is what fixes syllables, and the exposed scheme plus validation is what fixes rhyme.

Phase ablation. A `phase1-only` skill variant extracts the constraints with the scripts and translates once, with no re-counting, refinement, or validation (Haiku, same items). It already reaches SER 0.040 (81 of 96 items perfect), so skill-extracted targets carry most of the syllable gain; the full pipeline removes the rest (paired SER difference 0.030, 95% CI $[0.013, 0.052]$, better on 13 items, worse on 2). Rhyme is split: Phase 1 alone lifts ARI from 0.343 (Vanilla) to 0.612 by drafting against the extracted scheme, and the jump to 0.990 comes from the final `lyrics-validator` pass and the rhyme rewrites it triggers (49 of 96 items re-validated at least once), so verified refinement, not constraint-aware drafting, is what preserves the scheme.

Semantic cost. Table 11 (Appendix F) scores all systems with COMET-22 (Rei et al., 2022) and BLEU (Papineni et al., 2002) against the gold translations of Ou et al. (2023a). Meeting the constraints costs Haiku 0.064 COMET relative to Vanilla (paired over the 96 items both completed, 95% CI $[0.053, 0.075]$) and 9.0 BLEU; stronger orchestrators pay less: 0.027 for Sonnet (CI $[0.019, 0.035]$) and 0.022 for Opus (CI $[0.013, 0.031]$). Among the constrained systems, BLT beats CLT on COMET ($+0.045$, CI $[0.032, 0.059]$) as well as on SER, SCRE, and ARI, while their CCVO is nearly identical (0.320 vs. 0.318); CLT scores higher on BLEU but lower on COMET, reflecting n-gram overlap with the references it was trained on.

Condition	SER ↓	SCRE ↓	ARI ↑	CCVO ↓	Time
English-to-Spanish (en→es)					
Vanilla (n=30)	0.913	0.482	0.091	0.509	10.9
BLT (n=30)	**0.000**	**0.000**	**1.000**	**0.291**	195.8
English-to-Japanese (en→ja)					
Vanilla (n=30)	0.980	1.004	-0.022	0.972	15.3
BLT (n=30)	**0.013**	**0.003**	**0.615**	**0.287**	265.2

Table 4: Cross-lingual generalization (Haiku, 30 items each, the same English source windows retargeted; all items produced output). Japanese SER and SCRE are mora-count errors. **Time**: median wall-clock seconds per item.

6.3 Human Evaluation

The structural metrics do not directly measure perceived singability. We run a blind A/B study on n=10 en→zh items with six fluent-Mandarin raters (60 judgments) on the Opus 5 outputs of Table 3, on items both systems completed with non-overlapping windows: raters pick the more *singable* of two anonymized versions (Vanilla vs. BLT, side randomized) and rate each 1–5 on singability, naturalness, and meaning preservation. BLT is preferred in 93% of judgments (56/60) and wins all 10 items by majority (sign test p=0.002), and it scores higher on singability (MOS 4.38 vs. 3.00), naturalness (4.17 vs. 3.42), and meaning (4.23 vs. 3.27); two-way cluster bootstrap 95% CIs over raters and items exclude zero for all three differences ($[+0.85, +1.97]$, $[+0.08, +1.45]$, $[+0.30, +1.68]$). Agreement is moderate on singability (Krippendorff's α=0.46) and low elsewhere (0.13–0.16); raters judged text only, and the meaning ratings suggest the 0.02 COMET cost is not perceived as lost sense.

7 Conclusion

We presented BLT Skills, a training-free framework that packages IPA-based phonetic analysis as composable Agent Skills that an LLM agent invokes to extract, verify, and refine constraints. On English-to-Chinese (n=100) it cuts syllable error by 99% over single-prompt translation on completed items (SER 0.008; 87–98% counting lost items as failures) and lifts rhyme-scheme preservation to ARI 0.99, far above a supervised baseline (0.42), and transfers to Spanish and Japanese. Ablations show that the deterministic counter fixes syllables and the validated rhyme scheme fixes rhyme; offloading verifiable subtasks to composable skills is a general recipe for constraint-bound generation.

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 97.5% for English and 92.0% for Spanish against independent references), but absolute syllable counts remain calibrated to eSpeak-NG rather than to citation-form syllabification. Language coverage is bounded by eSpeak-NG, and Japanese mora counting relies on `pykakasi`; dialect-level and non-standard varieties are untested. Items that produced no translation are scored as failures; they come from API content-filter rejections (retried up to three attempts), from the model declining in text to translate copyrighted lyrics (Opus Vanilla, not retried), and in one case from an unparsable final message. Second, the framework has no dedicated rhyme refinement phase: rhyme is only rewritten when the final validation fails, and the counter-only ablation (ARI 0.990 to 0.250 when the scheme is withheld) shows how much rests on that check. Phase 2 also never feeds its corrections back into a global re-draft of Phase 1; the phase ablation (§6.2) shows the current split already saturates the syllable constraint on the main en→zh task, but a global re-drafting loop could trade latency for rhyme. Third, our main evaluation is English-to-Chinese (n=100 five-line windows, not 100 independent songs, §6) 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. The seed (42) fixes only the test-window sample; LLM decoding is not seeded, so repeated runs vary, and multi-seed runs would better characterize that 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 exceeds CLT on syllable metrics and ARI with all three orchestrators tested, weaker models may not. Fifth, the agent loop introduces latency (median 63–107 s per item for the Claude orchestrators via the API, vs. 1.4 s for CLT on Apple Silicon MPS), which may limit practical deployment. Sixth, our automatic metrics measure structural constraint preservation rather than semantic fidelity; the human study (§6.3) covers singability, naturalness, and meaning but only on a small sample (six raters, ten items, one orchestrator); COMET and BLEU (Table 11) quantify the semantic cost automatically, but a larger study with professionally experienced raters remains future work. 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 release does not bundle the full lyric corpus; for reproducibility, it includes the sampled evaluation windows and generated traces and results. Source lyric excerpts are distributed under the original dataset's CC BY-NC-SA 4.0 license, separately from our Apache-2.0 software. Users applying the framework to new material 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 (§6.3) 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 deter-

mined 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 23420–23435.

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.

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*, pages 774–781.

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 (Volume 4: 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*, pages 499–518. 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*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

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.

Ricardo Rei, José G. C. de Souza, Duarte Alves, Chrysoula Zerva, Ana C Farinha, Taisiya Glushkova, Alon Lavie, Luisa Coheur, and André F. T. Martins. 2022. COMET-22: Unbabel-IST 2022 submission for the metrics shared task. In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 578–585.

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*.

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, pages 68539–68551.

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.

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*, pages 1–21.

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

The harness sends a single user prompt per test item; every other instruction comes from the skill files that the agent loads itself.

User prompt (from the harness, `scripts/run_agent.py`).

> `Use the lyrics-translator skill to translate these` N `lines from {source_lang} to {target_lang}.`
>
> `Source lines:`
> `1. {line_1}`
> $\ldots$
> N`. {line_N}`
>
> `Follow the skill workflow exactly and run its scripts for every count. Your final message must contain only the numbered translated lines.`

`lyrics-translator/SKILL.md` (abridged; the released file is the source of truth).

> `Never count syllables or judge rhyme yourself. Always run the sub-skill scripts and use their output.`
>
> `Step 0: Extract constraints. Run on the source lines:`

```
 python skills/syllable-counter/
scripts/count_syllables.py "<line>"
"<source_lang>"
 python skills/rhyme-analyzer/scripts/
rhyme_analysis.py -scheme
"<source_lang>" "<line1>" . . .
 python skills/
syllable-pattern-analyzer/scripts/
pattern_analysis.py "<source_lang>"
"<line1>" . . .
Record the target syllable counts, the
rhyme scheme, and the patterns.

Phase 1: Initial translation. Translate
all lines at once. Each line must have
EXACTLY the target syllable count.
Lines that share a rhyme label must
rhyme; lines with different labels must
not. Keep the word-level pattern close
to the source when possible. Keep the
meaning and the emotion. Chinese
targets: one Han character is one
syllable; write only Han characters.

Phase 2: Syllable refinement. For each
translated line, run
count_syllables.py. If the count
differs, rewrite the line (too long:
remove modifiers, shorter words; too
short: add descriptive words) and count
again. Stop after 10 attempts and keep
the closest attempt.

Final validation. Run
 python skills/lyrics-validator/
scripts/validate.py "<target_lang>"
'[c1, . . . , cN]' "<line1>" . . . -rhyme
<scheme>
If syllables_match is false, return to
Phase 2 for the failing lines. If
rhymes_valid is false, try one rewrite
of the failing lines that keeps the
syllable counts, then count again.

Output: return only the numbered
translated lines.
```

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 the same exact-equality rule; the validator additionally rejects a translation when lines with different scheme labels share an ending.

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 one syllable per Han character (embedded Latin words are counted through eSpeak-NG), 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	77.9%	97.5%
Spanish	pyphen	58.7%	92.0%

Table 5: eSpeak-NG vowel-nuclei syllable counts vs. independent references (English: 475 CMUdict-covered source lines vs. CMUdict vowel-phoneme counts; Spanish: 150 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 4.5, Sonnet 5, Opus 5
Model	Interface	Claude Code CLI
Model	Reasoning effort	`medium`
Agent	Allowed tools	`Skill`, `Read`, `Bash`
Agent	Max attempts	3
Phase 2	Max attempts / line	10
Orchestr.	Max lines / test	5
Segment.	English	Whitespace
Segment.	Chinese	`jieba`

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 matched *success* case from the en→zh test split (Haiku, agent run): every line matched its count after Phase 2, and the agent rewrote lines that broke the ABABB scheme twice after validation until the scheme held (ARI=1.0).

L	Source / BLT	Tgt	Syll.
1	Amazing grace how sweet the sound	8	
	奇妙恩典甘美之芳		8 ✓
2	That saved a wretch like me	6	
	拯救如我的灵		6 ✓
3	I once was lost, but now I'm found	8	
	曾失落如今已听唱		8 ✓
4	Was blind but now I see	6	
	曾盲目今已明		6 ✓
5	My chains are gone, I've been set free.	8	
	我的锁链已脱落清		8 ✓

Table 8: Success case (en → cmn, test item `ou_006`, scheme ABABB). The agent ran `syllable-counter` 15 times and `lyrics-validator` 4 times; the final lines rhyme as *-ang* (1, 3) and *-ing* (2, 4, 5).

Table 9 shows a *failure* case: line 4 requires 8 syllables but the agent converges on 7 after exhausting 10 attempts.

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 9: 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.

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 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 we feed CLT our source-derived syllable targets as length tokens, the setting comparable to BLT, giving SCRE 0.033 and ARI 0.422 (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 63–107 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. From the agent traces of the Haiku en→zh run (96 completed items), every item loaded `lyrics-translator` and ran the three extraction scripts on the source; counting script executions in the tool results, the agent then ran `syllable-counter` 12.4 times per item on average, where the workflow prescribes 10 baseline source and target checks: 61 of 96 items exceeded that baseline, and the agent ran `lyrics-validator` 1.6 times per item, re-running it after a rhyme rewrite on 49 items. The median Haiku item takes 18 assistant turns and 107 s wall-clock, and costs $0.108 (8.1k output tokens, 393k input tokens including cache reads); Sonnet needs only 9 turns and 71 s because it batches several script executions into one tool call (11.9 counter runs, 1.4 validator runs per item), at $0.149 per item, and Opus batches further (10.8 counter runs,

1.3 validator runs, 8 turns, 63 s) at $0.297 per item (Table 10).

A second Haiku run of the full pipeline on the same items (a configuration mistake that left the scripts enabled during a prompt-only attempt) reproduced the result: SER 0.004, ARI 0.990 on 98 completed items, which bounds the run-to-run variance of the unseeded orchestrator.

Condition	Turns	Cnt.	Val.	$	Time
Vanilla (Haiku)	1	0	0	0.013	14.8
BLT (Haiku)	18	12.4	1.6	0.108	107
BLT (Sonnet)	9	11.9	1.4	0.149	71
BLT (Opus)	8	10.8	1.2	0.297	63

Table 10: Per-item cost of the en→zh runs, from the agent traces: median assistant turns, mean `syllable-counter` (Cnt.) and `lyrics-validator` (Val.) script executions counted from tool results, mean API cost in USD, and median wall-clock seconds.

System	COMET-22 ↑	BLEU ↑	SER ↓	ARI ↑
CLT	0.706	17.7	0.284	0.422
Vanilla (Haiku)	**0.815**	**23.9**	0.790	0.343
BLT (Haiku)	0.741	15.0	**0.008**	0.990
Vanilla (Sonnet)	0.811	22.8	0.788	0.261
BLT (Sonnet)	0.779	18.7	0.008	0.996
Vanilla (Opus)	0.764	17.7	0.802	0.279
BLT (Opus)	0.752	16.6	**0.000**	**1.000**

Table 11: Semantic fidelity vs. structural constraints (en→zh, 100 items / 500 lines, gold references from Ou et al. (2023a)). COMET and BLEU are over all 500 lines with lost items scored as empty (Haiku: 0 Vanilla and 20 BLT lines; Sonnet: 0 and 5; Opus: 65 and 55); SER and ARI are over completed items as in Table 3. BLEU uses `sacrebleu` with the `zh` tokenizer.

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: Haiku Vanilla scores 0.790 (uniform) / 0.207 (magnitude-weighted) and Haiku BLT 0.008 / 0.002, 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 lower Japanese ARI (Table 4) is a linguistic property rather than a metric artifact. Japanese rhyme endings reduce to one of five vowels, so exact-ending equality leaves no room for near-rhyme, and a five-line window whose source has five distinct labels rarely admits a Japanese translation with five distinct final vowels. Splitting the 30 BLT items by source scheme makes this visible: on the 12 items with a repeated source label ARI is 0.872, on the 18 with five distinct labels it is 0.444, and the agent spends its extra turns on this unwinnable constraint.