AI & SOCIETY
https://doi.org/10.1007/s00146-025-02689-w

OPEN FORUM

Vernacular computing as encoded aesthetics for decolonial code intervention

Koundinya Dhulipalla[1]

Received: 18 April 2025 / Accepted: 7 October 2025

Abstract
This paper examines programming languages as cultural and political artifacts embedded within colonial systems of power. Drawing from critical code studies and decolonial theory, it explores how dominant programming paradigms encode assumptions inherited from military-industrial infrastructures, capitalist productivity models, and Western epistemologies. While often framed as neutral tools, programming languages function as infrastructures that quietly structure knowledge, exclude alternate forms of reasoning, and naturalise particular logics of abstraction. In response, the paper introduces vernacular computing as a conceptual and methodological framework for decolonial code intervention. This approach reimagines programming not through general-purpose utility, but through culturally situated logics drawn from oral, poetic, and embodied traditions. As a practice-based articulation of this framework, the paper presents *Prāsa*, an esoteric programming language based on the Telugu poetic system of *chandassu*. Rather than relying on conventional syntax or procedural logic, *Prāsa* encodes metrical constraints as computational rules, enabling a form of programming grounded in rhythm, repetition, and positional form. By situating *Prāsa* within the lineage of esoteric languages, the paper demonstrates how programming can emerge from alternate epistemic traditions. *Prāsa* does not offer a universal replacement for existing languages, but proposes a method for composing otherwise, where code becomes an expressive, situated act of cultural memory and aesthetic reasoning. This reframing contributes to emerging discourses on decolonial computing by showing how vernacular practices might inform programming language design without being flattened into utility or performance.

Keywords Decolonial computing · Programming languages · Vernacular knowledge systems · Esoteric programming · Cultural computing

1 Introduction

Programming languages form the foundational layer of modern computing systems. They create layers of abstraction between our actions and interactions, shaping how we engage with machines and how new mediums of communication and their infrastructure are produced and circulated. Despite their centrality, the cultural and political dimensions of programming languages remain underexplored in critical discourses. This paper addresses this gap by interrogating programming languages through a critical decolonial lens, examining the historical legacies, institutional ideologies, and power structures embedded within them (Nofre et al. 2014).

The historical development of programming languages is inseparable from systems of colonialism, patriarchy, and institutional control. This includes the ENIAC computer, developed during World War II for artillery calculations, whose design and use exemplify early entanglements of computational systems with military goals (Ceruzzi 2003). From Ada Lovelace's foundational contributions to computation (Fuegi and Francis 2003) to the militarised development of programming, these languages have evolved within contexts that reflect and perpetuate global hierarchies. Today, dominant programming paradigms continue to privilege Western-centric logics and epistemologies, marginalising alternative ways of thinking about computation. This paper situates programming languages as political artefacts that encode not only technical functionality but also cultural values and exclusions.

✉ Koundinya Dhulipalla
k.dhulipalla@ucl.ac.uk

[1] University College London, London, UK

Published online: 19 October 2025

Situated within the interdisciplinary domains of software studies and critical code studies, this paper draws on theoretical frameworks that foreground the cultural specificity of code. Mark Marino's (2020) critical code studies encourage analyzing code as a cultural text, not just a functional artifact, which supports this paper's examination of syntax for embedded ideologies. Walter Mignolo's (2011) decolonial theory, which critiques the colonial matrix of power and argues for the epistemic authority of local knowledges, frames this project's challenge to computation's dominant logics. His work supports the view that vernacular grammars should be recognised as valid sites of knowledge production. These perspectives unsettle normative assumptions about the neutrality of programming languages by positioning code as a space where institutional ideologies are both encoded and contested.

Central to this inquiry is the framework of *vernacular computing*, which focuses on integrating culturally specific knowledge systems into computational practices as a mode of decolonial intervention. This framework is operationalised through the development of *Prāsa*, an esoteric programming language inspired by Telugu poetic grammar. By embedding cultural specificity into its syntactic and structural design, *Prāsa* exemplifies vernacular computing as both an artistic artefact and a political gesture. It offers a material manifestation of computation from a vernacular epistemic position.

Overall, this research addresses three central questions: How do programming languages function as political artefacts? What role do vernacular knowledge systems play in challenging dominant computational paradigms? And how can esoteric programming languages serve as decolonial interventions? To answer these questions, this paper uses a combination of theoretical analysis and practice-based research. The theoretical framework situates programming languages within broader systems of power. The practice-based component focuses on the design and implementation of *Prāsa*, analysing its potential as both a functional language and an infrastructure for cultural expression.

By critically engaging with programming languages as sites where epistemic frameworks are reproduced and challenged, this paper contributes to emerging debates on decolonial computing (Ali 2016). Rather than focusing computation solely within the history of the digital computer, it proposes an expanded frame. It recognises that computation has long existed in vernacular systems, oral traditions, and poetic forms. Through *Prāsa*, the paper explores how computational thinking can be reframed through cultural grammars that precede and exceed machinic logics.

2 Contexts of control: command, productivity, and programming language design

This section turns to the structuring role of programming languages within computation, not only as tools for writing code, but as systems that shape how knowledge is organised, expressed, and made actionable. Programming languages are often positioned as neutral instruments: vehicles for writing logic, structuring processes, or interacting with machines. A tool to build; a means to an end. But they do more than this. They organise knowledge, embed values, and shape how problems are formulated. They act as infrastructures: not only supporting computational systems, but quietly conditioning what 'computation' itself can mean.

To consider programming languages as infrastructural is to ask how they have come to hold such structuring power, and what assumptions they carry. Their history is not just one of technical refinement but of alignment with institutional demands. From their early formation, programming languages were developed in response to State, industrial, and bureaucratic requirements; each privileging precision, repeatability, and machinic legibility. These qualities became foundational to what programming was imagined to be, even as their origins were rooted in historically contingent systems of control and standardisation.

Languages, such as *FORTRAN* (1957), *ALGOL* (1958), and later *C* (1972), were developed to formalise thought into machine-readable instructions. Their design encoded assumptions not only about how computation should function, but also about how knowledge ought to be structured. Syntax, control-flow, and abstraction became tools to discipline in linguistic, procedural, and logical terms. As Nofre et al. (2014) outline, the institutionalisation of programming languages coincided with efforts to regulate thought into syntactical units optimised for computational execution. Notably, these developments occurred primarily in Western contexts and were exported globally, effectively marginalising non-Western computing practices and reinforcing a colonial hierarchy of knowledge (Ali 2016).

The idea of abstraction plays a central role in this trajectory. It is often framed as an enabling concept—one that allows for generalisation, modularity, and conceptual clarity. Yet abstraction also performs exclusions. It separates process from context and meaning from material form. The abstraction models privileged in dominant languages tend to emphasise efficiency, universality, and separation. These are qualities that align with modes of knowledge production inherited from industrial and computational rationalism. What gets abstracted is often what cannot be measured, controlled, or neatly structured within the available syntax.

These values have become naturalised within programming infrastructures. The aesthetics of 'good code', such as clarity, modularity, and legibility, are tethered to historical notions of correctness and design that reflect particular cultural logics. Syntax becomes not only a technical constraint, but also a normative one, establishing what is possible or desirable within a programming language. The idea of a 'well-written' program is shaped not only by how it functions, but by how it aligns with assumptions about readability, structure, and utility.

This section considers programming languages as infrastructures, as systems that quietly organise practice, thought, and expression. They are not only mediums through which programmers communicate with machines, but also frameworks that shape what kinds of problems appear solvable, and what forms of articulation are permitted. Their structures, once sedimented, rarely make themselves visible. Instead, they operate in the background, reinforcing ways of thinking while excluding others from being legible as computation.

The aim here is not to reject abstraction outright. It is to trace how abstraction, in its form, has been stabilised around particular values, and how these values structure the design and uptake of programming languages. There are other forms of logic, expression, and epistemology that do not sit easily within this model. These forms may be rhythmic, poetic, oral, embodied, or situated. They often fall outside the scope of dominant programming paradigms, not because they are less systematic, but because they are structured otherwise.

The following section turns to esoteric programming languages. These are languages that experiment with syntax and execution in ways that challenge normative ideas of programming. From there, the discussion moves to *Prāsa*, a language shaped by the Telugu poetic tradition. It departs from both dominant programming paradigms and esoteric languages by grounding its computational logic in vernacular and metrical structures.

3 Esoteric languages as critical code practice

To question the dominant structures of programming languages requires a space in which those structures can be deformed, exaggerated, or rendered strange. Esoteric programming languages (esolangs hereafter) occupy this space. Emerging in tension with mainstream programming norms, esolangs have become experimental sites where programming paradigms are no longer treated as fixed or neutral. Instead, they are manipulated, questioned, or even made nonsensical (Temkin 2017). Their refusal to conform offers a way to surface what dominant languages often suppress: the cultural, formal, and epistemic assumptions behind programming itself.

Esolangs are not designed for efficiency, clarity, or ease of use. Languages like *INTERCAL* (1972, Don Woods and James Lyon) parody the seriousness of formal syntax by introducing needlessly obscure commands. *Brainfuck* (1993, Urban Müller) reduces computation to a minimal symbolic core, stripping programming to its barest operational logic. *Piet* (2001, David Morgan-Mar) rewrites logic through colour, treating code as image rather than text. *Whitespace* (2003, Edwin Brady and Chris Morris) writes programs using only space, tab, and newline characters, rendering the code visually empty. These languages often complicate rather than simplify expression, turning programming into a space of conceptual or aesthetic exploration rather than practical function (Mateas & Montfort 2005).

In doing so, they perform a kind of formal subversion. They challenge what programming languages are assumed to be for, and what constitutes 'proper' code. This resistance is useful not only for its playfulness but for its ability to surface the norms embedded within programming paradigms. Many esolangs experiment with form, syntax, and logic in ways that expose the assumptions of mainstream programming. They open computation as a space for aesthetic, conceptual, and critical play. These languages stretch what code can be, making room for ambiguity, contradiction, and strangeness. Their existence exposes how values like efficiency, readability, and logic are not universal but are shaped by specific contexts.

3.1 *Prāsa* as situated esolang

Prāsa builds within this experimental lineage but takes a different path. Its refusal is not focused on technical abstraction or syntactic minimalism. Instead, it turns toward a structure grounded in Telugu poetic metre. The logic behind *Prāsa* is shaped by *chandassu* (Medicherla 1981), a prosodic system that has long guided literary and oral composition. This is not a rejection of structure, but an engagement with a different kind of structure, one that emerges from rhythm, repetition, and positional patterning. Where esolangs highlight the strangeness of code through disruption, *Prāsa* attempts to introduce an alternate logic drawn from composition.

By embedding positional and rhythmic constraints into its design, *Prāsa* moves beyond critique toward proposition. Computation, in this model, is not merely reinterpreted but potentially restructured. This shift aligns with a broader concern of the paper: how to imagine programming languages that operate outside industrial, militarised, and formalist epistemologies. The aim is not to discard structure, but to foreground different ones. It does not ask what computation can be without clarity or logic; it asks what computation might look like when rooted in other systems of rhythm,

structure, and meaning—other traditions of knowledge. It operationalizes vernacular logic as a form of decolonial intervention.

Rather than relying on absurdity, constraint, or parody, as many esolangs do, it turns to tradition. This is not tradition as something fixed, but as an active aesthetic grounded in the historical poetics of Telugu verse. It offers a way to think about computation beyond machine logic, English-based syntax, and industrial expectations. It is informed by the metrical technique of the same name, *Prāsa*, a form of rhythmic echo and alliteration that structures expression through constraint. Here, constraint is not disciplinary but generative.

To situate *Prāsa* is to locate it against the epistemic foundations of what is often called 'traditional programming.' This tradition, born in the military-industrial complex of the mid-twentieth century, privileges abstraction, control, and optimisation. Its knowledge systems are derived from logics of formalism and mathematics, where clarity, legibility, and efficiency are treated as universal virtues. As Foucault suggests, regimes of knowledge are not neutral but structured by power (Foucault 1980). They determine what is sayable, writable, and thinkable. *Prāsa* intervenes in this regime not by opposing it from the outside, but by composing within another episteme. This episteme draws from literary craft, orality, and rhythmic aesthetics.

Where most programming languages use keywords and syntax and mathematical formalism, *Prāsa* draws from Telugu poetic structures. It is not a metaphorical gesture; it is a formal, syntactic one. This positions *Prāsa* not just as a language, but as an alternate approach to inscription. It understands computation as expressive, rhythmic, and situated. The stakes here are not cultural representation, but epistemic reconfiguration. What does it mean to write code when its structure echoes poetic metre rather than procedural logic? What kinds of knowledge does it prioritise, and what modes of attention does it require?

Prāsa, then, is not only a programming language but a proposition for how code might be inscribed, transmitted, and inhabited differently. It does not seek to replace existing paradigms, but to question them by introducing forms of composition that emerge from other traditions of thought and expression.

4 Operationalising vernacular poetics: technical implementation of *Prāsa*

This section details the technical design of *Prāsa* as a programming language grounded in metrical poetics. Building on the critical and conceptual foundation established in the previous sections, *Prāsa*'s implementation functions as a material articulation of vernacular computing, where computational structure emerges from cultural form rather than symbolic logic. The subsections that follow explain the syntax, evaluation model, and broader implications of this approach, demonstrating how *Prāsa* reconceives code not as instruction but as constraint-driven composition.

Table 1 Chandassu gana and syllable sequences (Medicherla 1981)

Sequence	Notation	*Ganam* (group)
guruvu-laghuvu-laghuvu	U I I	*Bha-ganam*
laghuvu-guruvu-laghuvu	I U I	*Ja-ganam*
laghuvu-laghuvu-guruvu	I I U	*Sa-ganam*
laghuvu-guruvu-guruvu	I U U	*Ya-ganam*
guruvu-laghuvu-guruvu	U I U	*Ra-ganam*
guruvu-guruvu-laghuvu	U U I	*Ta-ganam*
guruvu-guruvu-guruvu	U U U	*Ma-ganam*
laghuvu-laghuvu-laghuvu	I I I	*Na-ganam*

4.1 Language design

Prāsa's syntax derives from Telugu *chandassu*, a poetic system governing verse structure through rhythmic syllable sequences (*ganas*). *Chandassu*, which is a set of grammatical rules for writing poetry bases its foundation on syllables. A poem is traditionally four lines long, and *chandassu* rules apply to each one of them. Syllables can be categorised into long-syllables (*guruvu*) or short-syllables (*laghuvu*), denoted by U and I respectively. A set of three possible sequences of these syllables is called a *ganam* (group) (Medicherla 1981). This poetic tradition employs a complex system of syllabic patterns that has evolved over centuries in the Telugu-speaking regions of Southern India.

The syllable combinations in Table 1 define the metrical logic of Telugu verse and structure *Prāsa*'s rule system. Depending on the arrangement of these patterns, poems can be categorised into three major umbrella groups: *jaati*, *upajaati*, and *vruttam*. Of these, *vruttam* is the commonly used and requires both a caesura (*yati*) and rhythmic alliteration (*Prāsa*[1]). Within *vruttam*, there are seven subcategories of poems that follow the same core rules with slight variations in their metrics. This rich structure enables over 134 million[2] unique syllabic combinations and metric variations (Tables 2, 3, 4).

For example, the following excerpt from the epic poem Andhra Mahabhagavatam (fifteenth century) by Potana (1987) illustrates such constraints:

[1] The metrical device after which this language is named

[2] Based on possible permutations of *gana* structures across standard poem formats; estimate by author. Based on $Total\ combinations = 8^n$ (where n = number of *ganas*).

Table 2 Stress-to-syllable approximation

Word	Syllables	Stress	Pattern
BEAUTIFUL	BEAU	1	U
	TI	0	I
	FUL	2	I

Table 3 Tokens used in Prāsa syntax

Token Type	Example	Function
INDENT	4 spaces	Move tape head + 1
WORD	"HERE BE DRAGONS"	Syllable accumulation
BRACKET	()	Metric multiplier

Table 4 Cell value to ASCII encoding

Cell	Syllables	ASCII Value	Character
0	72	72	H
1	69	69	E

పదముల బట్టినం దలకుబా టొకయింతయు లేక శూరతన్
మదగజవల్లభుండు మతిమంతుడు దంతయు గాంత ఘట్టనం
జెదరగ జిమ్మె నమ్మకరిచిప్పలు పాదులు దప్పనొప్పఱన్
వదలి జలగ్రహంబు కరివాలముమూలముజీరె గోఱలన్.

(padamula baṭṭinaṁ dalakubā ṭokayintayu lēka śūratan.
madagajavallabhuṇḍu matimantuḍu dantayu gānta ghaṭṭanaṁ.
jedaraga jimme nammakaricippalu pādulu dappanopparaṁ.
vadali jalagrahambu karivālamumūlamujīre gōraḷan).[3]

It satisfies:

1. Each line contains a total of 21 syllables.
2. The second syllable in each line across the poem shares a common phonetic sound.
3. The first and eleventh syllables match across lines.
4. Syllable pattern on each line are in the following order: I I I I U I U I I I U I I U I I U I U I U

[3] *Translation*: Judging by words alone, one may seem brave, though not truly so. Even a wise elephant, when in rut, charges and crashes into obstacles. Ships break apart, trusted cargo spills and scatters. The sea's pull and waves drag everything into the deep. Translation by the author.

These rules serve as executable constraints in *Prāsa*, shifting code from procedural logic to cultural composition.

The implementation leverages the CMU Pronouncing Dictionary[4] to approximate Telugu meter using English phonetics. In this system, a primary-stressed syllable (marker '1') is treated as a long syllable (*guruvu*, denoted U), while an unstressed or secondary-stressed syllable (marker '0' or '2') is treated as short (laghuvu, denoted I). A *Python* script parses words according to these stress markers and produces a JSON dictionary that maps English words onto *chandassu*-style syllable patterns.

This allows a dictionary of syllables to be constructed that mimics *chandassu* metrics, creating a phonological bridge between the writing systems.

4.2 a. Lexical analysis: stress-based syllabification

Prāsa's lexical engine includes two interdependent systems:

1 CMU Dictionary Processing

As described above, a *Python* script parses phonetic data from the CMU Pronouncing Dictionary to identify syllable boundaries using vowel clustering heuristics. For instance, "RIVER" is syllabified as RI (U), VER (I), approximating Telugu's *guruvu*/*laghuvu* syllables through stress markers. This process enables the generation of a syllable dictionary that maps English words to *chandassu*-inspired metrical forms.

2 Indentation as tape navigation

Inspired by the conceptual model of the Turing machine, *Prāsa* reimagines indentation as tape movement. Each 4-space indent represents a move to the next memory cell, and line breaks accumulate syllables into each cell. This spatial structure mirrors the layout of a Turing tape, in which a head moves across an infinite tape to read and write values.

A Turing Machine is a theoretical model of computation in which a read/write head moves across an infinite tape divided into discrete cells, each capable of holding a symbol (Fig. 1). It describes computation in terms of sequential symbolic manipulation and is foundational to computer science.

Prāsa draws on the structure of the Turing tape to shape how programs unfold across space. Indentation becomes a way of moving between memory cells, with each level marking a shift in position (Fig. 2). Syllables placed within these cells build meaning through their sequence and alignment. This model lets poetic metre guide computation through

[4] Dictionary file available at https://www.speech.cs.cmu.edu/cgi-bin/cmudict

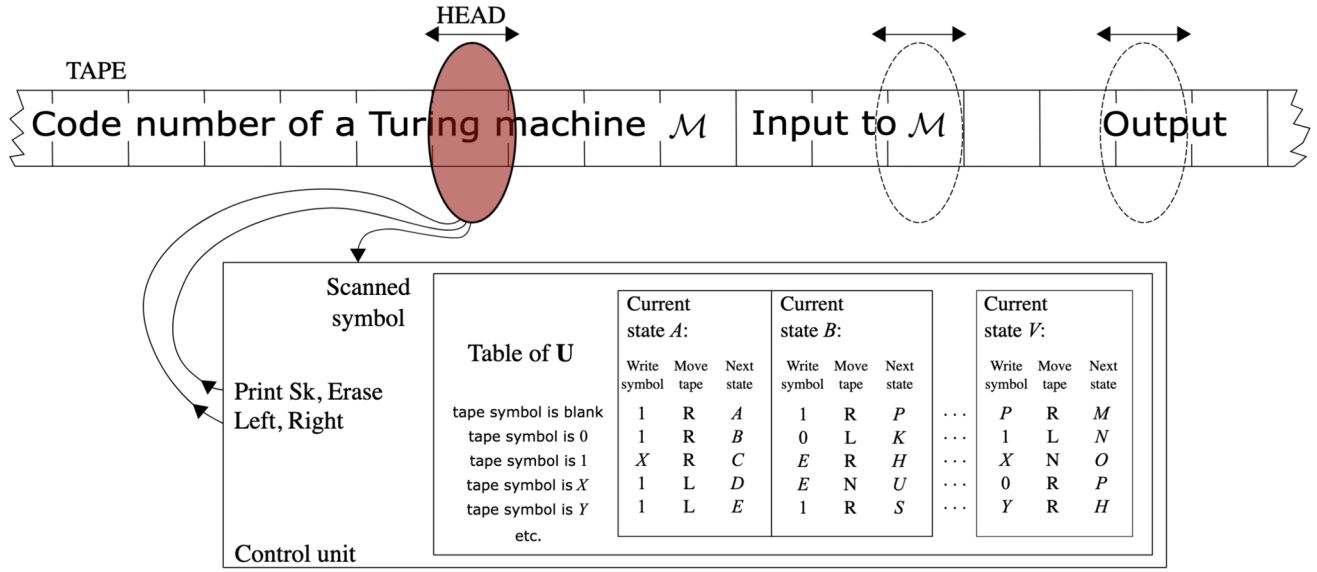

Fig. 1 Representation of the tape in a Universal Turing Machine (Image by Cbuckley, licensed under CC BY-SA 3.0, via Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Universal_Turing_machine.svg)

Fig. 2 Cell movement with indentations

For instance:

```
␣␣␣␣HERE BE DRAGONS // 1 indentation -> cell 1

␣␣␣␣AND BUTTERFLIES // 1 indentation -> cell 1

FROM ONE EAR TO THE OTHER // no indentations -> cell 0
```

spatial rhythm rather than procedural logic. By treating movement of a cell as functional, *Prāsa* links the structure of verse with the mechanics of execution, allowing form itself to drive how a program behaves.

This spatial approach to syntax draws from both computational theory and experimental poetry, creating a visual grammar where the physical arrangement of text on the page becomes functionally meaningful. The indentation system transforms the typically aesthetic use of whitespace in poetry into a computational mechanism, allowing programmers to navigate the conceptual 'tape' of memory while maintaining and experimenting with poetic form.

For instance:

Indentation levels dictate tape position, while syllable counts generate output-merging poetic structure with machine execution. This visual-spatial approach to code organisation represents a significant departure from conventional programming paradigms, where indentation typically serves readability rather than functional purposes. In essence, moving one indent level to the right in code corresponds to moving one cell forward on an imagined tape (memory), and vice versa – thus the poem's layout in space determines program state changes.

4.3 b. Parsing and evaluation: enacting cultural constraints

Prāsa's parser validates programs against *chandassu* rules, constructing an Abstract Syntax Tree (AST) that prioritizes cultural fidelity:

Prāsa's parser checks each line for correct syllable counts, repeated sounds at required positions, and structural alignment based on *chandassu* metrics. The code is tokenised according to indentation and syllabic structure, and the resulting sequence is validated against the expected metrical pattern. If these constraints are not satisfied, the parser simply rejects the validity of the poem as a *chandassu* composition. It does not produce an error or halt execution in a conventional sense. There is no *crash*, only a refusal to recognise the input as code. The result remains a poem, just not one that meets the structural requirements for interpretation within *Prāsa*.

Following parsing, where the input code-poetry is translated to measure, syllable counts in each tape cell are translated into their corresponding ASCII values:

This encoding structure transforms cultural metrics into machine-readable output, bridging oral tradition with computational function. The convergence of poetic form and computation here demonstrates that technical rules can emerge from aesthetic and culturally situated systems, rather than from industrial or utilitarian priorities. Because ASCII accommodates a broad range of character sets, the output generated by *Prāsa* need not be linguistically uniform; any language may be represented, provided the cell values correspond to the appropriate numerical codes. This flexibility decouples the language of output from the structure of input, allowing vernacular composition to coexist with multilingual expression.

4.4 Situated computation and vernacular infrastructure

Prāsa's technical implementation is not only a demonstration of poetic constraint as logic but also an articulation of vernacular computing as a conceptual and material framework. By centering a metrical tradition like Telugu *chandassu*, *Prāsa* foregrounds a mode of computation that does not rely on general-purpose utility. Instead, it proposes a grammar rooted in rhythm, in repetition, in positional constraints, and in vernacular knowledge systems.

This approach reframes what programming can be. In most contemporary models, programming languages are built around presumed universality: their syntax draws from formal logic, their functions are abstracted from specific cultural practices, and their aims often prioritise efficiency and scalability. These are not neutral defaults but reflect historical legacies shaped by institutional, military, and commercial infrastructures. In contrast, vernacular computing treats code as situated, expressive, and often oral or embodied. It resists the idea of universality by showing that computation can emerge from specific traditions of making, writing, and knowing.

Prāsa enacts this refusal through form. Its syntax does not generalise or optimise. It enforces poetic rules with precision, requiring that programs adhere to syllabic length, rhythmic alignment, and alliterative echoes. These are not aesthetic flourishes layered onto code. They are the conditions of execution. If a line fails to meet metrical constraints, it does not run. Programming becomes an act of attunement rather than abstraction. The programmer is not issuing commands to a machine but composing verse within a formal structure that predates computational logic.

This changes the role of the programmer. In *Prāsa*, the programmer becomes a performer, one who internalises the rhythm and structure of *chandassu* to write code-poetry that is *valid*. This is not merely an analogy. The process involves counting syllables, ensuring repetition, and aligning with a prosodic pattern. Execution is not outsourced to a compiler alone. It requires the programmer's embodied attention. This is particularly evident in the use of indentation as navigation across a Turing tape, where spatial form governs memory allocation and syllable accumulation drives output. The tape does not merely store values but becomes a surface inscribed with poetic form.

Prāsa's machine independence extends this logic further. Its structure is designed to work with any syllabic language. The current implementation uses a custom-built English dictionary based on syllable approximation. It also allows for modularity in a way where the language used to write the code-poetry into *Prāsa* can be replaced with any language, provided a suitable dictionary file. This makes it, in principle language-agnostic, not by flattening cultural distinctions but by offering a framework into which other writing systems can be inscribed. A program could be written in any language and script if its phonological structures and constraints can be defined within *Prāsa*'s syllable system as a dictionary. The output, encoded through ASCII values, can similarly represent any language or symbolic system, as long as the numerical values correspond to valid characters. English was used for the current implementation due to resource availability, specifically the CMU English phonetic dictionary. This choice is a practical convenience, not a cultural preference. Any language with defined syllable or stress patterns could be integrated. For example, a Telugu or Hindi dictionary file could be developed to write *Prāsa* code-poems in those languages. The use of ASCII output is a concession to interoperability, but it does not privilege English in the input. This allows for an expansive vision of vernacular computing, not as a fixed linguistic reference but as a methodology for building situated, culturally informed systems of logic and expression.

In these ways, *Prāsa* serves as a proof-of-concept for vernacular computing, not a universal solution, but a situated, limited, and culturally embedded infrastructure. Its constraints are not limitations to be overcome but propositions that reveal the assumptions of other programming systems. Computation, through this lens, becomes not only what machines execute, but what bodies compose, what cultures refine, and what histories carry forward.

The appendix includes a *HELLO, WORLD!* program written in Prāsa. Rather than a minimal proof of functionality, it unfolds as a poetic composition that meets the constraints required for execution. Its inclusion reflects both a formal gesture—fulfilling a common convention in programming—and a conceptual one, demonstrating how *Prāsa* reimagines even the most canonical exercises through metrical and vernacular logic.

Prāsa language itself is written in *JavaScript* and *Python*; it is distributed through a GitHub repository.[5] This is not a contradiction but a reflection of its technical dependencies. Its reliance on existing infrastructures is a limitation. While it proposes an alternative approach to computation, it must still navigate the architectures it critiques. These infrastructures remain deeply embedded across technical, institutional, and cultural domains, making it difficult to build outside them without also depending on them.

5 Conclusion

This paper has proposed vernacular computing as a conceptual and methodological framework for rethinking how programming languages can be designed, interpreted, and practiced. Rather than treating computation as a culturally neutral domain defined by abstraction and logic, vernacular computing foregrounds the cultural, rhythmic, and epistemic structures that already inform how people think, speak, and reason. It frames code not as a universal language, but as one possible articulation among many: each embedded in particular ways of knowing and making.

Through the design and implementation of *Prāsa*, the paper has shown how the syllabic constraints of Telugu *chandassu* can form the basis for a programming language. This is not merely metaphorical or illustrative: the language functions by enforcing metrical and alliterative patterns as syntactic and semantic constraints. In doing so, it replaces the usual control-flow paradigm with one of rhythmic, constraint-driven composition that demands the programmer's embodied attention. *Prāsa* is intentionally limited, culturally specific, and structurally awkward. Rather than viewing these as limitations, we can see them as conditions that reveal different ways of thinking about computation.

Across the paper's sections, this intervention has been positioned against the backdrop of programming language infrastructure, esolang traditions, oral knowledge systems, and embodied computation. Together, these elements make the case that programming languages are not neutral or universal, but ideological and constructed. *Prāsa* does not solve this; it simply redirects the terms. By encoding cultural form as logic, it opens space for code that expresses rather than commands, and for systems that calculate through aesthetic form rather than formal abstraction.

What this paper contributes is not a tool, but a way of thinking about code. Vernacular computing is not offered as a category of languages, but as a lens for understanding how computation always reflects cultural and historical assumptions. It invites treating programming languages as expressive forms shaped by the worlds they emerge from. *Prāsa* is one instance of this—one that demonstrates how computation can be rooted in rhythm, poetics, and linguistic memory.

This framework points toward further questions: What other epistemic traditions might structure code differently? How might computation look if built from the logics of chant, weaving, oral formulae, or storytelling? And how might infrastructures of software and learning shift if code were treated not as a neutral abstraction, but as an extension of specific cultural practices and ways of reasoning?

Vernacular computing is not an answer to these questions. It is a proposal that they are worth asking.

Appendix

HELLO, WORLD! program in *Prāsa*. By Janani Venkateswaran.

```
                A FIBONACCI CONUNDRUM

        HERE, HERE

        A DISTANT SOUND

        TRAVELS (A) DA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

FAINTLY,

VISCERALLY

    AND THEN ALL AT ONCE.

            FROM ONE EAR

                TO THE OTHER

                (ON THE RIGHT)

                    A GENTLE GIANT

                    IN PEACEFUL SLUMBER

                      EARS FANNING THE WIND

                      TAIL STROKING THE SOIL

                  STOMACH COOING

                  MOUTH DROOLING

                  CHEST HEAVING

    RISE

                    AND
```

[5] An online collaboration and version control platform for working with code. Also where the full source code for *Prāsa* is available—https://github.com/koundinyad/prasa-lang

FALL

RISE

AND

FALL

RISE

AND

FALL

A SONG.

(AND A MIRACLE)

(ON ITS CHEEK)

AN EYELASH!

(MAKE A WISH MAKE A WISH MAKE A WISH MAKE A WISH) DA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

A WISP OF DNA

DELICATE, BROWN

DARK CHOCOLATE MILO

HER BROTHER'S EYES

YOUR SISTER'S HAIR

MY FATHER'S ANGER

THE TRUNK OF A TREE.

(DEAFENING EXCITEMENT) LA DA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

(AND HANDS TREMBLING BUT FINGERS STEADY) DA LA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

(FEET SLOWLY INCHING CLOSER) LA DA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

AND THEN

IN THE BLINK OF AN EYE

IT HOVERS (AIRBORNE)

TEASING,

THEN

FLIES,

DRIFTS

(AWAY) DA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

INTO THE UNKNOWN

AWAY FROM THE VAST EXPANSE

OF GREY, GOO AND GLUE

AND

INTO THE VAST EXPANSE

OF OLIVES AND PISTACHIOS,

OF WATER AND LIGHT,

OF HANDSOME HUNKS

(TO LAY ON LAPS OF) DA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

TALL AND DARK

SHORT AND THICK

(AND OTHERS) LA DA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

BUT

(NOT HERE THOUGH, NOT YET) DA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

SO,

BACK TO

FLOATING,

DRIFTING,

SCOURING,

AND SEARCHING,

AND JUST AROUND THE CORNER

(LIKE A BLESSING IN A PALM)

A HEART.

VEINS RUNNING ACROSS

EMBRACING AND WELCOMING

AND GRAZING AND FLIRTING

SOME REST (AND)

(A WORTHY SPOT TO SETTLE) LA DA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

THE MOON LOOKS OVER

(A SIGHT, BEHOLD)

A BIG FAT SIGH.

AND THEN

SUDDENLY

A FAMILIAR SOUND

RISE

AND

FALL

RISE

AND

FALL

RISE

AND

FALL

A SONG.

THE SONG.

(UGH) DA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

THE LONE EYELASH

BACK IN THE AIR

DRIFTING,

SCOUTING,

SCOURING.

BLUE AND BROWN AND BLUE AND BROWN

WATER AGAINST SAND

AND SAND AGAINST WATER

BLUE AND BROWN AND BLUE AND BROWN

FINALLY,

THE BEACH!

A SMOOTH (SMOOTH) PEBBLE

BRICK RED, IN A BLANKET OF MOSS

(PERFECT!) DA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

THE MOON IS IVORY

IT'S BEEN A LONG DAY

(IT'S TIME, IT'S TIME) DA DI ALA A DA A LA A A A LA DI DA A DA LALA DI

A BIG FAT YAWN.

BUT WHY IS

THE TIDE RESTLESS

THE HORIZON EVASIVE

THE WATER COLD

AND THEN OUT OF NOWHERE

A FAMILIAR SOUND SNEAKS IN

FAINTLY,

VISCERALLY,

THEN ALL AT ONCE.

EMERGING FROM THE HEART OF THE SEA,

(RISE)

(AND)

(FALL)

(RISE)

(AND)

(FALL)

(RISE)

(AND)

(FALL)

THE SONG.

Acknowledgements I thank Janani Venkateswaran for contributing the "HELLO WORLD!" text to the Prāsa, and helping bring it to life.

Author contributions K.D. wrote the main manuscript text and developed the technical artefact. K.D. reviewed the manuscript.

Data availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

Ali SM (2016) A brief introduction to decolonial computing. XRDS 22(4):16–21. https://doi.org/10.1145/2930886

Ceruzzi, P. E. (2003). A history of modern computing. Mit Press.

Foucault, M. (1980). Power/knowledge: Selected Interviews and Other writings, 1972–1977 (C. Gordon, Eds). Pantheon Books.

Fuegi J, Francis J (2003) Lovelace & babbage and the creation of the 1843 "notes." IEEE Ann Hist Comput 25(4):16–26. https://doi.org/10.1109/mahc.2003.1253887

Marino, M. C. (2020). Critical code studies. The Mit Press.

Mateas, M., & Montfort, N. (2005). A Box, Darkly: Obfuscation, Weird Languages, and Code Aesthetics.

Medicherla, A. (1981). *Chando Vyākaranamu* [Grammar of Prosody] (2nd eds). Vidyarthi Publications.

Mignolo W (2011) The Darker Side of Western Modernity: Global Futures. Duke University Press, Decolonial Options

Nofre D, Priestley M, Alberts G (2014) When technology became language: the origins of the linguistic conception of computer programming, 1950–1960. Technol Cult 55(1):40–75. https://doi.org/10.1353/tech.2014.0031

Potana. (1987). *Potana Bhagavatamu* [Andhra Mahabhagavatam; Telugu translation of the Srimad Bhagavatam]. Tirumala Tirupati Devasthanams. (Original work composed ca. 15th century)

Temkin D (2017) Language without code: intentionally unusable, uncomputable, or conceptual programming languages. J Sci Technol Arts 9(3):83. https://doi.org/10.7559/citarj.v9i3.432