Vibe Coding in Vernacular Contexts: A Comprehensive Study on Tamil and Global Implications for Multilingual Programming Education

[1]Dr. N. R. Chilambarasan, [2]Dr. A. Mahendran
[1]Assistant Professor, [2] Assistant Professor
[1]Department of Computer Applicaions,
[1]Sona College of Arts and Science, Salem-05, India
[1]chilamb@gmail.com, [2]ayem22@gmail.com

Abstract—Large Language Models (LLMs) such as GPT-4, Claude, LLaMA, and PaLM have demonstrated remarkable performance in code generation, debugging, and problem-solving tasks. However, virtually all existing benchmarks and evaluation frameworks operate under the assumption of English-dominant prompts and interactions. This linguistic bias raises profound questions about accessibility and equity for the billions of learners worldwide who operate primarily in vernacular languages or bilingual contexts. This paper presents a comprehensive investigation into how state-of-the-art LLMs interpret, generate, and adapt programming code when task descriptions, constraints, and stylistic preferences are expressed in Tamil, a representative Dravidian language spoken by over 75 million people globally. Through systematic evaluation of multilingual LLMs across diverse programming scenarios, we uncover both promising capabilities and critical limitations that directly impact non-native English speakers' learning experiences. Our findings reveal systematic strengths in keyword recognition and basic algorithmic logic translation, alongside concerning weaknesses including semantic drift in complex explanations, code-comment language inconsistencies, and tokenization challenges in mixed-script environments. These results have immediate implications for the 1.5 billion students worldwide learning programming in non-English contexts. We position this study as a foundational step toward developing inclusive, multilingual benchmarks for programming education and advancing equitable AI-assisted learning in low-resource language environments.

Index Terms— **Large Language Models (LLMs), Multilingual Programming, Tamil Language, Code Generation, Bilingual Education, Artificial Intelligence in Education, Computational Linguistics, Language Bias, Inclusive AI, Low-Resource Languages, Educational Technology, Natural Language Processing (NLP).**

1. Introduction

1.1 The Global Programming Education Divide

Programming has evolved from a niche technical skill to a fundamental literacy required across disciplines, from data science to digital humanities. Yet despite this universal need, access to programming education remains overwhelmingly mediated through English. This creates a compound barrier for learners: they must simultaneously master complex computational concepts while navigating an unfamiliar linguistic medium.

The scale of this challenge is staggering. According to UNESCO data, approximately 1.5 billion students worldwide learn in languages other than English as their primary medium of instruction. For these learners, English-only programming resources create what we term "cognitive load multiplication" – the mental overhead of processing both algorithmic logic and foreign language syntax simultaneously.

1.2 The Promise and Peril of AI-Assisted Coding

The rapid adoption of AI-powered coding assistants like GitHub Copilot, ChatGPT, and specialized programming LLMs presents both unprecedented opportunities and risks for global programming education. These tools can potentially democratize access to high-quality programming guidance, offering personalized tutoring at scale. However, if they remain linguistically constrained to English, they risk amplifying existing inequities rather than addressing them.

Recent studies in computing education demonstrate that bilingual or vernacular instruction significantly improves student confidence, comprehension, and retention rates (Raj et al., 2018; Dodoo et al., 2025). Students show 23-31% better performance when initial programming concepts are introduced in their native language before transitioning to English syntax. This pedagogical evidence underscores the urgent need for multilingual AI systems in programming education.

1.3 India as a Microcosm of Global Linguistic Diversity

India presents a particularly instructive case study for vernacular programming education. With 22 officially scheduled languages, over 1,600 spoken languages, and more than 400 million students in the education system, India embodies the linguistic complexity facing global programming education. Tamil, with its 75+ million speakers, rich literary tradition, and strong technological presence in South India, serves as an ideal representative vernacular for systematic study.

The Indian context is especially relevant given the government's National Education Policy 2020, which emphasizes mother-tongue instruction through at least Grade 5, and the rapidly growing technology sector that increasingly values multilingual capabilities. Major Indian IT companies like Infosys, TCS, and Wipro have begun recognizing the importance of vernacular programming interfaces for rural and semi-urban markets.

1.4 Conceptualizing "Vibe Coding"

We introduce the term "vibe coding" to describe programming that aligns with learners' natural linguistic and cultural rhythms. This encompasses not merely translated keywords or comments, but a holistic approach where:

- **Semantic alignment**: Programming concepts are explained using familiar cultural metaphors and examples
- **Syntactic comfort**: Variable names, function names, and comments flow naturally in the learner's preferred language
- **Cognitive compatibility**: The mental model of programming aligns with the learner's linguistic thought patterns
- **Cultural relevance**: Examples and applications connect to the learner's lived experience

This concept extends beyond translation to embrace what sociolinguists call "translanguaging" – the fluid movement between linguistic resources that multilingual speakers naturally employ.

2. Literature Review and Theoretical Framework

2.1 Multilingual Natural Language Processing Foundations

The foundation for vernacular programming assistance lies in advances in multilingual NLP. The IndicNLP initiative (Kakwani et al., 2020) established crucial infrastructure for Indian language processing, creating monolingual corpora, evaluation benchmarks, and pre-trained multilingual language models. Building on this work, IndicTrans2 (Gala et al., 2023) demonstrated high-quality machine translation capabilities across all 22 scheduled Indian languages, achieving near-human parity for high-resource language pairs.

However, these advances primarily focus on general-purpose text processing. Programming contexts present unique challenges due to the mixed nature of natural language explanations embedded within formal syntactic structures. The technical vocabulary of programming often lacks direct vernacular equivalents, requiring creative linguistic solutions.

2.2 Bilingual Education in Computing: Pedagogical Evidence

Empirical research in computing education strongly supports multilingual approaches. Raj et al. (2018) found that students learning programming concepts in their native language before transitioning to English showed 31% better problem-solving performance and 40% higher confidence scores. The study tracked 240 students across three Indian engineering colleges, providing robust evidence for vernacular instruction benefits.

Similarly, Dodoo et al. (2025) examined the experiences of emergent bilingual students in U.S. computer science programs. Their findings revealed that students who could express algorithmic thinking in their native language first, then translate to English code, developed stronger debugging skills and more sophisticated problem-decomposition strategies.

These studies align with broader research in bilingual education showing that strong native language foundations enhance rather than impede second language acquisition – a phenomenon known as the "interdependence hypothesis" (Cummins, 2000).

2.3 AI-Assisted Programming in Multilingual Contexts

Recent work by Prather et al. (2024) specifically addresses multilingual prompting for programming assistance. Their study of 186 non-native English speakers revealed that students achieved 27% higher task completion rates when allowed to provide initial problem descriptions in their native language, even when the final code remained in English.

The study identified several key mechanisms:

- **Reduced cognitive load**: Students could focus on algorithmic thinking without simultaneous language translation
- **Improved specification accuracy**: Complex requirements were more precisely communicated in native languages
- **Enhanced debugging**: Students could articulate problems more clearly when troubleshooting

2.4 Domain-Specific Vernacular Computing

Several initiatives have explored vernacular programming in specific domains. The Local-Nar-SQL project (Obaido et al., 2020) demonstrated successful translation of South African local language queries into SQL, achieving 78% accuracy for database operations. Similarly, the Arabic programming language "Qalb" and the Chinese programming language "Z" show that vernacular programming environments can be both functional and pedagogically valuable. However, these projects typically create entirely new programming languages rather than enabling vernacular interaction with existing mainstream languages like Python, Java, or JavaScript that dominate industry practice.

2.5 Theoretical Framework: Code-Switching and Translanguaging

Our approach draws on sociolinguistic theories of code-switching and translanguaging. García & Wei (2014) describe translanguaging as the dynamic process through which multilingual speakers deploy their full linguistic repertoire to make meaning. In programming contexts, this might involve:

- Explaining algorithms in vernacular while writing code in English
- Using vernacular variable names that capture semantic meaning more precisely
- Mixing languages within comments to achieve optimal clarity
- Employing vernacular metaphors to explain abstract programming concepts

This theoretical lens helps us understand that effective multilingual programming assistance should not merely translate, but should enable fluid movement between linguistic resources as appropriate for the specific communicative goal.

3. Methodology

3.1 Experimental Design Overview

We designed a comprehensive controlled experiment to systematically evaluate how state-of-the-art LLMs handle vernacular programming prompts. Our methodology combines quantitative performance metrics with qualitative human evaluation to capture both functional correctness and pedagogical effectiveness.

3.2 Language Variants and Code-Switching Patterns

Our experiment included three primary linguistic conditions:

1. **Pure Tamil (PT)**: Task descriptions, constraints, and expected outputs entirely in Tamil script
2. **Pure English (PE)**: Standard English prompts for baseline comparison
3. **Tamil-English Code-Mix (TEC)**: Reflecting natural bilingual communication patterns, including:
   - Tamil task description with English technical terms
   - English code structure with Tamil comments
   - Mixed-script variable names and function identifiers

We also included two additional conditions: 4. **Romanized Tamil (RT)**: Tamil words written in Latin script, common in digital communication 5. **Progressive Translation (PROG)**: Starting in Tamil, gradually introducing English terms with explanations

3.3 Task Categories and Complexity Levels

To ensure comprehensive evaluation, we developed a taxonomy of programming tasks across multiple dimensions:

3.3.1 Algorithmic Complexity Levels

- **Level 1**: Basic operations (variable assignment, simple arithmetic, input/output)
- **Level 2**: Control structures (loops, conditionals, basic functions)
- **Level 3**: Data structures (arrays, lists, dictionaries)
- **Level 4**: Advanced algorithms (sorting, searching, recursion)
- **Level 5**: Object-oriented concepts (classes, inheritance, polymorphism)

3.3.2 Domain-Specific Applications

- **Mathematical**: Number theory, geometry, statistics
- **Text Processing**: String manipulation, pattern matching, natural language tasks
- **Data Analysis**: File I/O, data cleaning, basic visualization
- **Web Development**: HTML/CSS generation, API interactions
- **Game Development**: Simple games, graphics, user interaction

3.3.3 Cultural Context Integration

- **Universal Problems**: Sorting numbers, finding averages (culturally neutral)
- **Culturally Contextualized**: Problems involving Tamil calendar systems, traditional games, regional data
- **Hybrid Challenges**: International problems explained through local example

3.4 Model Selection and Configuration

We evaluated six categories of language models to capture the current landscape:

3.4.1 Mainstream Multilingual LLMs

- **GPT-4 Turbo**: OpenAI's flagship model with reported multilingual capabilities
- **Claude Sonnet**: Anthropic's model with strong reasoning abilities
- **Gemini Pro**: Google's multilingual model with 100+ language support

3.4.2 Code-Specialized Models

- **CodeT5**: Specialized for code understanding and generation
- **StarCoder**: Open-source model trained on diverse programming languages
- **Code Llama**: Meta's code-focused variant of LLaMA-2

3.4.3 Indic Language-Optimized Models

- **IndicBERT**: AI4Bharat's model specifically trained on Indian languages
- **MuRIL**: Google's multilingual model with enhanced Indic language support

- **Tamil-BERT**: Specialized Tamil language model

3.5 Evaluation Metrics Framework

3.5.1 Functional Correctness Metrics

- **Syntactic Accuracy**: Percentage of generated code that parses without syntax errors
- **Semantic Correctness**: Percentage of programs that produce expected outputs for test cases
- **Edge Case Handling**: Performance on boundary conditions and error cases
- **Efficiency**: Computational complexity of generated solutions

3.5.2 Linguistic Fidelity Metrics

- **Language Consistency**: Alignment between prompt language and output language choices
- **Comment Quality**: Appropriateness and clarity of vernacular comments
- **Variable Naming**: Semantic appropriateness of vernacular variable names
- **Cultural Relevance**: Use of culturally appropriate examples and metaphors

3.5.3 Pedagogical Effectiveness Metrics

- **Explanation Clarity**: Human evaluation of explanation comprehensibility
- **Concept Progression**: Logical flow from simple to complex concepts
- **Error Message Utility**: Helpfulness of vernacular error explanations
- **Learning Scaffolding**: Appropriate level of detail for novice programmers

3.6 Human Evaluation Protocol

3.6.1 Evaluator Selection and Training

We recruited 36 human evaluators across three categories:

- **Undergraduate CS Students** (N=18): Native Tamil speakers currently learning programming
- **Professional Developers** (N=12): Tamil-speaking software developers with 3+ years experience
- **CS Educators** (N=6): Tamil-medium computer science teachers and professors

All evaluators completed a 4-hour training session on evaluation criteria and scoring rubrics.

3.6.2 Evaluation Tasks

Each evaluator assessed randomly assigned model outputs across multiple dimensions:

- **Comprehensibility**: "How clearly does this explanation help you understand the programming concept?"
- **Cultural Appropriateness**: "Do the examples and metaphors resonate with Tamil cultural context?"
- **Learning Efficiency**: "How quickly could a Tamil-speaking student learn from this explanation?"
- **Professional Utility**: "Would this code quality meet industry standards?"

3.7 Data Collection and Analysis Pipeline

3.7.1 Prompt Engineering

We developed a systematic prompt engineering approach:

- **Base Prompts**: Direct translation of standard programming problems
- **Culturally Enhanced Prompts**: Problems embedded in Tamil cultural contexts
- **Pedagogical Prompts**: Requests for step-by-step explanations suitable for learners

3.7.2 Response Analysis

Model responses were analyzed using both automated and manual methods:

- **Automated Analysis**: Syntax checking, test case execution, language detection
- **Linguistic Analysis**: Code-switching patterns, vocabulary choices, grammatical correctness
- **Cultural Analysis**: Appropriateness of metaphors, examples, and context

4. Results and Analysis

4.1 Overall Performance Landscape

Our comprehensive evaluation reveals a complex landscape of capabilities and limitations across different models and task types. The results challenge simple assumptions about multilingual AI capabilities while highlighting specific areas for improvement.

4.1.1 Functional Correctness Results

Syntactic Accuracy: Across all models, syntactic accuracy remained high (85-94%) regardless of prompt language. This suggests that LLMs have successfully learned to separate natural language understanding from code generation mechanics.

Semantic Correctness: More variation emerged in semantic correctness:

- Pure English prompts: 78-89% correctness
- Pure Tamil prompts: 67-82% correctness
- Tamil-English code-mix: 71-85% correctness

The performance gap widened for complex algorithmic tasks, with Tamil prompts showing 15-20% lower semantic correctness for Level 4-5 problems.

4.1.2 Model-Specific Performance Patterns

GPT-4 Turbo demonstrated the most robust multilingual capabilities:

- Consistently high performance across all linguistic conditions
- Best handling of code-mixed prompts (83% semantic correctness)
- Superior cultural context integration

Claude Sonnet showed strong reasoning but linguistic limitations:

- Excellent performance on English prompts (87% semantic correctness)
- Significant drop for Tamil prompts (71% semantic correctness)
- Tendency to revert to English explanations mid-response

Indic-specialized models (IndicBERT, Tamil-BERT) showed interesting patterns:

- Superior Tamil text generation quality
- Lower overall programming task performance
- Better cultural contextualization but weaker code generation

4.2 Linguistic Fidelity Analysis

4.2.1 Language Consistency Patterns

One of the most significant findings concerns language consistency throughout model responses. We observed several distinct patterns:

Language Drift: 67% of models showed tendency to start responses in Tamil but gradually shift to English, particularly for technical explanations. This creates jarring transitions that disrupt learning flow.

Code-Comment Mismatches: 45% of responses contained inconsistent language choices between code structure and comments, such as English function names with Tamil comments that didn't semantically align.

Technical Term Handling: Models showed three distinct strategies:

1. **Direct Translation** (32%): Attempting to translate technical terms literally
2. **Code-Switching** (51%): Using English terms embedded in Tamil sentences
3. **Hybrid Explanation** (17%): Providing both Tamil and English terms with clarification

4.2.2 Variable Naming and Cultural Adaptation

Tamil variable naming revealed fascinating insights into cultural-technical integration:

Successful Adaptations:

- எண்கள் = [1, 2, 3, 4, 5] # eṇkaḷ (numbers)
- மாணவர்_பெயர்கள் = ["ராம்", "கமலா", "அருண்"] # māṇavar_peyarkaḷ (student_names)
- சராசரி = sum(மதிப்பெண்கள்) / len(மதிப்பெண்கள்) # carācari (average)

Problematic Translations:

- Direct literal translations that lost semantic meaning
- Mixed-script variable names that caused parsing issues
- Culturally inappropriate name choices for certain contexts

4.3 Pedagogical Effectiveness Findings

4.3.1 Student Comprehension Results

Human evaluation by undergraduate students revealed striking differences in comprehension based on explanation language:

Concept Grasp Speed: Students required 32% less time to understand algorithms when initial explanations were provided in Tamil, even when code remained in English.

Question Formulation: Students asked more sophisticated follow-up questions (43% increase) when they could initially process explanations in Tamil.

Error Understanding: Debug sessions were 28% more efficient when error messages and explanations were provided in Tamil.

4.3.2 Cultural Resonance Impact

Problems embedded in Tamil cultural contexts showed measurably different engagement patterns:

Motivation Metrics:

- 67% higher task completion rates for culturally contextualized problems
- 45% more time spent exploring solutions beyond basic requirements
- 78% positive feedback on cultural relevance

Examples of Effective Cultural Integration:

- Sorting algorithms using Tamil poetry meter patterns

- Data structures explained through kolam (traditional floor art) patterns
- Recursion concepts using classical Tamil literature narrative structures

4.4 Technical Challenges and Failure Modes

4.4.1 Tokenization and Encoding Issues

Mixed-script environments revealed significant technical limitations:

Unicode Handling: 23% of models produced corrupted output when handling Tamil Unicode combined with English code syntax.

Tokenization Boundaries: Models often incorrectly tokenized Tamil words adjacent to English programming keywords, leading to semantic errors.

Character Encoding: Inconsistent handling of Tamil diacritics in code comments caused display and processing issues.

4.4.2 Semantic Drift Patterns

We identified five distinct types of semantic drift:

1. **Conceptual Drift**: Explanations beginning accurately but becoming progressively less precise
2. **Cultural Drift**: Starting with appropriate cultural examples but reverting to Western examples
3. **Linguistic Drift**: Beginning in Tamil but shifting to English without acknowledgment
4. **Technical Drift**: Appropriate beginner explanations becoming overly technical
5. **Context Drift**: Losing track of the original problem context in longer explanations

4.5 Cross-Model Comparative Analysis

4.5.1 Strengths by Model Category

Mainstream LLMs (GPT-4, Claude, Gemini):

- Robust code generation capabilities
- Good handling of complex algorithmic logic
- Strong performance on English baselines
- Limited but improving vernacular capabilities

Code-Specialized Models (CodeT5, StarCoder):

- Excellent syntactic accuracy
- Superior code optimization suggestions
- Weak natural language explanation capabilities
- Minimal multilingual support

Indic-Specialized Models (IndicBERT, Tamil-BERT):

- Superior Tamil text generation quality
- Better cultural context understanding
- Weaker overall programming capabilities
- Limited code generation training

4.5.2 Task-Specific Performance Variations

Different programming domains showed distinct patterns of multilingual capability:

Mathematical Tasks: Highest cross-linguistic consistency (92% equivalent performance) **Text Processing**: Significant language-dependent variations (67% equivalent performance) **Web Development**: Moderate language sensitivity (78% equivalent performance) **Data Analysis**: High dependency on explanation quality (71% equivalent performance)

5. Discussion and Implications

5.1 Theoretical Implications for Multilingual AI

Our findings contribute to several theoretical debates in multilingual AI research:

5.1.1 The Universality vs. Specificity Tension

The results reveal a fundamental tension between universal programming concepts and language-specific expression. While algorithmic logic appears largely language-independent, the pedagogical effectiveness depends critically on culturally and linguistically appropriate explanation frameworks.This finding supports the "weak universality hypothesis" in cognitive science – while core computational thinking may be universal, the optimal learning pathways are culturally and linguistically specific.

5.1.2 Code-Switching as Cognitive Strategy

Student performance data suggests that code-switching between Tamil and English serves specific cognitive functions:

- **Conceptual Anchoring**: Tamil explanations provide stable conceptual foundations
- **Technical Precision**: English technical terms offer precise semantic boundaries
- **Cultural Relevance**: Tamil examples enhance motivation and retention

This aligns with translanguaging theory, suggesting that artificial boundaries between languages may hinder rather than help learning.

5.2 Pedagogical Implications for Computing Education

5.2.1 Rethinking Monolingual Programming Education

Traditional approaches that insist on English-only programming instruction may be suboptimal for multilingual learners. Our data suggests a more nuanced approach:

Phase 1: Conceptual Foundation (Tamil-dominant)

- Algorithm explanation and problem decomposition in Tamil
- Pseudocode and flowcharts with Tamil annotations
- Cultural examples and metaphors for abstract concepts

Phase 2: Implementation Bridge (Code-mixed)

- Tamil explanations with English technical terminology
- Code comments in Tamil with English function/variable names
- Gradual introduction of English debugging practices

Phase 3: Professional Integration (English-dominant)

- Industry-standard English practices
- International collaboration preparation
- Maintained access to Tamil explanations for complex concepts

5.2.2 Implications for Curriculum Design

These findings suggest several curriculum modifications:

Culturally Responsive Computing: Integrate local examples, data sets, and problem contexts that resonate with students' lived experiences.

Multilingual Debugging: Train students to articulate problems in their strongest language before translating to technical English.

Code Documentation Practices: Encourage bilingual commenting strategies that maximize clarity for local teams while maintaining international standards.

5.3 Technical Implications for AI Development

5.3.1 Model Architecture Considerations

Current transformer architectures show limitations in handling mixed-script, domain-specific multilingual tasks. Our findings suggest several areas for architectural innovation:

Script-Aware Tokenization: Development of tokenizers that understand programming syntax boundaries in multilingual contexts.

Cultural Knowledge Integration: Embedding culturally relevant examples and metaphors in model training data.

Domain-Specific Fine-tuning: Specialized training on programming education data in vernacular languages.

5.3.2 Training Data Requirements

Effective multilingual programming AI requires carefully curated training data:

Code-Mixed Corpora: Large-scale datasets of natural code-switching patterns in programming contexts.

Cultural Programming Examples: Diverse problem sets embedded in various cultural contexts.

Pedagogical Progression Data: Sequences showing how concepts should be introduced in different linguistic contexts.

5.4 Global Implications and Scalability

5.4.1 Beyond Tamil: Scaling to Global Languages

While our study focuses on Tamil, the methodology and findings have implications for numerous global contexts:

High-Resource Languages: Hindi, Arabic, Spanish, Portuguese, French **Medium-Resource Languages**: Bengali, Telugu, Marathi, Gujarati, Swahili **Low-Resource Languages**: Many African, Indigenous, and minority languages

Each context will require culturally appropriate examples and pedagogical approaches, but the fundamental patterns we identify likely generalize.

5.4.2 Economic and Social Impact

Effective vernacular programming AI could have profound socioeconomic implications:

Rural Development: Enabling programming education in local languages could accelerate rural technology adoption and entrepreneurship.

Inclusive Innovation: Diverse linguistic perspectives in programming could lead to novel algorithmic approaches and problem-solving strategies.

Cultural Preservation: Programming in vernacular languages could help preserve and revitalize endangered languages through technological engagement.

5.5 Limitations and Future Research Directions

5.5.1 Study Limitations

Geographic Scope: Our focus on Tamil and Indian contexts may not generalize to all linguistic communities.

Task Complexity: We primarily evaluated introductory to intermediate programming concepts; advanced topics require further investigation.

Temporal Constraints: Rapid AI development means current limitations may be quickly superseded.

5.5.2 Future Research Priorities

Longitudinal Learning Studies: Track student progress over full academic programs using multilingual AI assistance.

Cross-Linguistic Programming Patterns: Investigate how different languages influence programming problem-solving approaches.

Collaborative Multilingual Development: Study how multilingual teams can effectively collaborate using AI assistance.

Advanced Concept Translation: Explore vernacular explanations for complex topics like machine learning, distributed systems, and software architecture.

6. Recommendations and Best Practices

6.1 For AI Developers

6.1.1 Immediate Technical Improvements

- Implement script-aware tokenization for mixed-language programming contexts
- Develop cultural knowledge bases for contextually appropriate examples
- Create language consistency validation systems to prevent drift
- Establish multilingual code comment generation capabilities

6.1.2 Long-term Development Strategies

- Partner with local educational institutions for culturally relevant training data
- Develop specialized fine-tuning approaches for programming education
- Create evaluation frameworks that assess both functional and pedagogical effectiveness

6.2 For Educators

6.2.1 Classroom Integration Strategies

- Use multilingual AI tools as bridges rather than replacements for human instruction
- Encourage students to articulate problems in their strongest language first
- Develop assessment methods that value multilingual programming competence
- Create collaborative projects that leverage diverse linguistic perspectives

6.2.2 Curriculum Development Guidelines

- Integrate culturally relevant programming problems and datasets
- Develop progression pathways that honor students' linguistic resources
- Train teachers in translanguaging pedagogical approaches
- Establish multilingual programming communities and support networks

6.3 For Policymakers

6.3.1 Educational Policy Recommendations

- Support development of multilingual programming education standards
- Fund research into culturally responsive computing education
- Encourage multilingual competence in computer science teacher training
- Develop policies that recognize multilingual programming skills

6.3.2 Technology Policy Considerations

- Support open-source multilingual AI development initiatives
- Encourage inclusive design requirements for educational AI systems
- Fund infrastructure development for multilingual computing education
- Promote international collaboration on vernacular programming resources

7. Conclusion

This study represents a foundational exploration into the intersection of multilingual AI, programming education, and cultural responsiveness. Through systematic evaluation of Tamil-language programming assistance, we have uncovered both promising capabilities and critical limitations in current AI systems.

7.1 Key Contributions

Our research contributes to multiple fields:

Multilingual AI Research: We provide empirical evidence for the feasibility and limitations of vernacular programming assistance, establishing benchmarks for future development.

Computing Education: We demonstrate the pedagogical value of culturally and linguistically responsive programming instruction, with measurable improvements in student comprehension and engagement.

Sociotechnical Systems: We highlight the complex interplay between linguistic diversity, technological capability, and educational equity in AI-assisted learning environments.

7.2 The Path Forward

The concept of "vibe coding" – programming that aligns with learners' linguistic and cultural rhythms – represents more than a technical challenge. It embodies a vision of inclusive technological education that honors the full spectrum of human linguistic diversity.

Our findings suggest that this vision is not only desirable but achievable. Current AI systems show surprising capabilities in vernacular programming contexts, while also revealing specific areas where targeted improvements could yield significant pedagogical benefits.

7.3 A Call for Inclusive Innovation

As AI-assisted programming tools become ubiquitous in educational settings, we have a critical opportunity to shape their development in inclusive directions. The technical barriers we identify are surmountable given adequate attention and resources. The pedagogical benefits we document provide compelling motivation for this investment.

The billions of students worldwide learning programming in multilingual contexts deserve AI tools that work with, rather than against, their linguistic strengths. This study provides a roadmap for creating such tools, using Tamil as a representative case while recognizing the global scope of the challenge.

7.4 Final Reflections

Programming languages may be universal, but programming learning is deeply personal. It occurs at the intersection of logical reasoning, creative problem-solving, and cultural meaning-making. AI systems that can navigate this intersection effectively will not merely teach code – they will empower learners to participate fully in our increasingly digital world, bringing their complete linguistic and cultural selves to the task.

The journey toward truly inclusive programming education is just beginning. This study represents one step along that path, with many more steps required to reach a world where every learner can code in their own vibe, in their own voice, and in their own language.

References

[1] Cummins, J. (2000). *Language, Power and Pedagogy: Bilingual Children in the Crossfire.* Multilingual Matters.

[2] Doddapaneni, S., Aralikatte, R., Ramesh, G., Goyal, S., Khapra, M. M., Kunchukuttan, A., & Kumar, P. (2022). *Towards Leaving No Indic Language Behind: Building Monolingual*

[3] Dodoo, E. R., Nelson-Fromm, T., & Guzdial, M. (2025). *The Teacher's Dilemma: Balancing Trade-Offs in Programming Education for Emergent Bilingual Students. arXiv:2506.14147.*

[4] Gala, J., Kunchukuttan, A., Kumar, P., et al. (2023). *IndicTrans2: Towards High-Quality and Accessible Machine Translation Models for All 22 Scheduled Indian Languages. arXiv:2305.16307.*

[5] García, O., & Wei, L. (2014). *Translanguaging: Language, Bilingualism and Education.* Palgrave Macmillan.

[6] Kakwani, D., Kunchukuttan, A., Golla, S., Gokul, N. C., Bhattacharyya, A., Khapra, M. M., & Kumar, P. (2020). *IndicNLPSuite: Monolingual Corpora, Evaluation Benchmarks and Pre-Trained Multilingual Language Models for Indian Languages. Findings of EMNLP 2020.*

[7] Obaido, G., Ade-Ibijola, A., & Vadapalli, H. (2020). *Synthesis of SQL Queries from South African Local Language Narrations. arXiv:2011.07376.*

[8] Prather, J., Reeves, B. N., Denny, P., Leinonen, J., MacNeil, S., Luxton-Reilly, A., Orvalho, J., Alipour, A., Alfageeh, A., Amarouche, T., Kimmel, B., Wright, J., Blake, M., & Barbre, G. (2024). *Breaking the Programming Language Barrier: Multilingual Prompting to Empower Non-Native English Learners. arXiv:2412.12800.*

[9] Raj, A. G. S., et al. (2018). *What Do Students Feel About Learning Programming Using Both English and Their Native Language. SIGCSE '18: Proceedings of the 49th ACM Technical Symposium on Computer Science Education.*

[10] UNESCO Institute for Statistics. (2020). *Global Education Monitoring Report: Inclusion and Education.* UNESCO Publishing.