EMBINO

Grammar-Constrained Code Generation for Resource-Limited Embedded Systems

Tiny Intelligence for Tiny Devices

Technology & Opportunity Overview

Academic Research · Intellectual Property · Commercial Strategy

Kernel Keys LLC
2025

Provisional Patent Application Filed

Systems and Methods for Grammar-Constrained Code Generation and Execution on Resource-Limited Embedded Devices

Contact

david@kernelkeys.com

www.kernelkeys.com

Contents

Executive Summary **5**

1 Technology Overview **6**
1.1 The Abstraction Gap . . . 6
1.2 Grammar-Constrained Small Language Models . . . 6
1.2.1 Small Language Models . . . 6
1.2.2 Grammar-Guided Decoding . . . 7
1.2.3 Search Space Reduction . . . 7
1.3 Sparse Low-Rank Adaptation (S-LoRA) . . . 7
1.4 Micro-Interpreter . . . 7

2 Intellectual Property **8**
2.1 Patent Overview . . . 8
2.2 Independent Claims . . . 8
2.3 Key Dependent Claims . . . 8
2.4 Freedom to Operate . . . 8

3 Commercial Opportunity **10**
3.1 Market Landscape . . . 10
3.2 The Problem We Solve . . . 10
3.3 Competitive Landscape . . . 10
3.4 Business Model . . . 10

4 Hardware Roadmap & Funding **11**
4.1 Three-Stage Hardware Strategy . . . 11
4.1.1 Stage 1: Add-On Module (12–24 months) . . . 11
4.1.2 Stage 2: System-on-PCB (24–36 months) . . . 11
4.1.3 Stage 3: On-Chip / Custom Silicon (36+ months) . . . 11
4.2 Funding Requirements . . . 12
4.2.1 Phase 1: Proof of Concept ($500K, 12 months) . . . 12
4.2.2 Phase 2: Product Development ($2M, 18 months) . . . 12
4.2.3 Phase 3: Scale to Breakeven ($5–6M, 24–36 months) . . . 12
4.3 Funding Summary . . . 12
4.4 Use of Funds . . . 13

5 Technical Specifications **14**
5.1 System Requirements 14
5.1.1 Generation System (Edge/Cloud) 14
5.1.2 Target Hardware 14
5.2 DSL Grammar (Excerpt) 14
5.3 Bytecode Instruction Set 15
5.4 Safety Guarantees 15

6 Theoretical Foundations **16**
6.1 Complexity Analysis 16
6.2 Capacity Lower Bounds 16
6.3 Sample Complexity 16

A Patent Claims Reference **17**

B Contact Information **18**

Executive Summary

The Opportunity

EMBINO bridges the gap between human intent and embedded implementation. Using small language models with grammar-constrained decoding, we enable natural-language programming of microcontrollers—without the resource requirements of large language models.

The Problem

Microcontrollers power 30–40 billion devices annually, yet programming them remains stuck in 1985: hand-written C/C++, manual memory management, device-specific registers, and zero abstraction for intent.

Our Solution

EMBINO provides a complete stack:

- **GC-SLM Framework:** Transformers under 50M params with grammar-guided decoding ensuring 100% syntactic validity
- **Domain-Specific Language:** Expressive rule-based constructs for embedded control
- **Micro-Interpreter:** Deterministic bytecode execution in under 16KB RAM

Intellectual Property

Provisional patent filed: 35 claims across 5 independent inventions.

Key Metrics

Metric	LLM	MicroPython	Embino
Model size	1–175B	N/A	1–50M
RAM required	8–128GB	256KB+	<16KB
Syntax validity	70–90%	N/A	100%
MCU deployment	No	Limited	Yes

Chapter 1

Technology Overview

1.1 The Abstraction Gap

The history of programming abstraction is one of steady ascent: machine code gave way to assembly; assembly to C; C to higher-level languages. Yet embedded systems remain stuck at C/C++.

Large language models demonstrate code generation capabilities, but their size (billions of parameters) and latency render them unsuitable for microcontrollers with kilobytes of memory.

EMBINO addresses this gap.

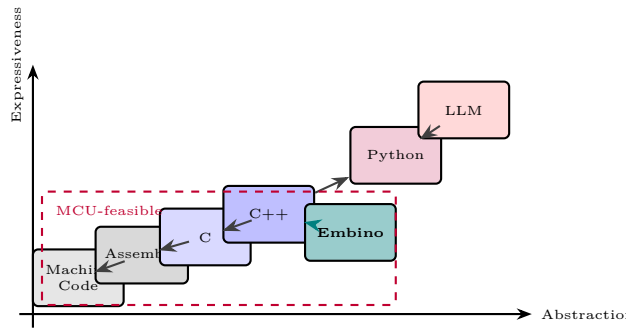

Figure 1.1: Evolution of programming abstraction. EMBINO occupies the upper-right of the MCU-feasible region.

1.2 Grammar-Constrained Small Language Models

The GC-SLM framework combines three innovations:

1.2.1 Small Language Models

Transformer models in the 1–50M parameter range:

Size	Layers	Hidden	Params
Tiny	4	256	4M
Small	6	384	15M
Base	8	512	45M

1.2.2 Grammar-Guided Decoding

At each generation step, the decoder: (1) computes logits, (2) queries parser for valid continuations, (3) masks invalid tokens, (4) samples from constrained distribution. This guarantees 100% syntactic validity.

1.2.3 Search Space Reduction

For a typical DSL with $k \approx 50$ valid tokens per state and vocabulary $|\mathcal{V}| = 32{,}000$:

$$\text{Reduction} = \left(\frac{|\mathcal{V}|}{k}\right)^L = 640^L$$

For a 50-token program: $640^{50} \approx 10^{140}$ reduction.

1.3 Sparse Low-Rank Adaptation (S-LoRA)

Standard LoRA introduces adapters $\Delta W = AB$. S-LoRA adds structured sparsity: $\Delta W = (A \odot M_A)(B \odot M_B)$ where M_A, M_B are binary masks.

Benefits: 60% fewer adapter parameters, compatible with 8-bit quantization.

1.4 Micro-Interpreter

Property	**Specification**
Program memory	<16 KB
Data memory	<4 KB
Stack depth	32 elements (fixed)
Memory allocation	None (static only)

Chapter 2

Intellectual Property

2.1 Patent Overview

A provisional patent application has been filed covering the core innovations.

Title: Systems and Methods for Grammar-Constrained Code Generation and Execution on Resource-Limited Embedded Devices

Claims: 35 total (5 independent, 30 dependent)

Status: Provisional application filed

2.2 Independent Claims

The patent covers five independent inventions:

1. **Generation System:** GC-SLM + grammar decoder + compiler + micro-interpreter
2. **Programming Method:** NL → DSL → bytecode → MCU execution
3. **Embedded Apparatus:** MCU with interpreter, fixed stack, hardware interfaces
4. **Decoding Method:** Grammar-constrained token generation
5. **Adaptation System:** S-LoRA with sparse low-rank matrices

2.3 Key Dependent Claims

Claim	Coverage
6–7	Model size (<50M, <10M quantized)
8	MCU targets (ESP32, RP2040, ATmega, STM32)
9–10	Memory constraints, no dynamic allocation
26–30	S-LoRA specifics (masks, sparsity, quantization)
31–32	On-device generation, FPGA implementation

2.4 Freedom to Operate

The invention occupies a novel intersection of three domains:

Prior art addresses individual components but not the integrated system.

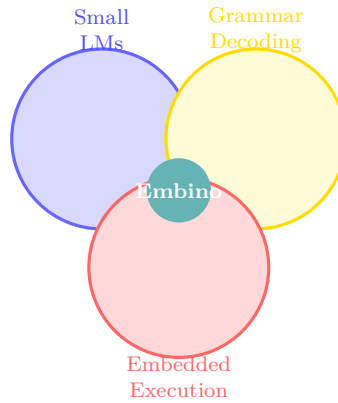

Figure 2.1: Patent coverage at the intersection of three domains.

Chapter 3

Commercial Opportunity

3.1 Market Landscape

Metric	Value
MCU annual shipments	30–40 billion units
Dev board volume	Tens of millions/year
Addressable (0.1%)	30–40 million units/year

3.2 The Problem We Solve

Programming microcontrollers requires C/C++ expertise, memory management, register knowledge, timing/interrupts, and specialized debug toolchains.

Alternatives fall short:

- **MicroPython:** 256KB+ RAM, unpredictable timing
- **Block programming:** Limited expressiveness, cloud-dependent
- **LLMs:** Cannot run on-device, produce invalid code

3.3 Competitive Landscape

Solution	Abstraction	MCU-Safe	Deterministic	AI
C/C++	Low	Yes	Yes	No
MicroPython	High	Partial	No	No
Block coding	Medium	Yes	Yes	No
LLM APIs	High	No	No	Yes
Embino	**High**	**Yes**	**Yes**	**Yes**

3.4 Business Model

1. **Hardware sales:** Dev boards and modules
2. **Software licensing:** Toolchain subscriptions
3. **Enterprise licensing:** Custom integrations
4. **IP licensing:** Per-unit royalties for OEMs

Chapter 4

Hardware Roadmap & Funding

4.1 Three-Stage Hardware Strategy

4.1.1 Stage 1: Add-On Module (12–24 months)

Product: External board via UART/I^{2}C/SPI
Target: Makers, educators, robotics hobbyists
Revenue: Dev board sales, starter kits
Price: $15–35 per unit
Advantage: Zero changes to existing boards

4.1.2 Stage 2: System-on-PCB (24–36 months)

Product: Integrated board with EMBINO runtime
Target: Robotics startups, automation integrators
Revenue: Volume board sales, subscriptions
Price: $8–20 per unit (volume)
Advantage: Standard Arduino/Pico pinout

4.1.3 Stage 3: On-Chip / Custom Silicon (36+ months)

Product: MCU with ROM-resident interpreter
Target: Appliance manufacturers, industrial OEMs
Revenue: Licensing fees, per-unit royalties
Price: $0.10–0.50 per chip (licensing)
Advantage: Lowest BOM, highest scale

4.2 Funding Requirements

4.2.1 Phase 1: Proof of Concept ($500K, 12 months)

Deliverables:
- DSL specification and grammar
- Working GC-SLM prototype (15M params)
- Micro-interpreter on ESP32/Arduino
- 3 demo applications with metrics
- User study with 8–10 participants

4.2.2 Phase 2: Product Development ($2M, 18 months)

Deliverables:
- Production-ready add-on module
- Cloud toolchain and IDE
- 5+ paying pilot customers
- Multi-MCU support (ESP32, RP2040, STM32)
- Patent prosecution and IP strengthening

4.2.3 Phase 3: Scale to Breakeven ($5–6M, 24–36 months)

Deliverables:
- System-on-PCB product line
- Enterprise licensing agreements
- 50+ enterprise customers
- Custom silicon partnerships
- Path to profitability / breakeven

4.3 Funding Summary

Phase	Funding	Timeline	Milestone
1. PoC	$500K	12 mo	Working prototype
2. Product	$2M	+18 mo	Paying customers
3. Scale	$5–6M	+24–36 mo	Breakeven company
Total	**$7.5–8.5M**	**4–5 years**	

4.4 Use of Funds

Category	Phase 1	Phase 2	Phase 3
Engineering (R&D)	60%	50%	35%
Hardware/Prototypes	15%	20%	25%
Operations	15%	15%	20%
Sales/Marketing	5%	10%	15%
Legal/IP	5%	5%	5%

Chapter 5

Technical Specifications

5.1 System Requirements

5.1.1 Generation System (Edge/Cloud)

Component	Requirement
Model size	4M–45M parameters
Inference hardware	CPU (x86/ARM), GPU optional
RAM	256MB–2GB
Latency	<500ms per program
Syntax validity	100% (guaranteed)

5.1.2 Target Hardware

MCU	Flash	RAM	Clock
ESP32	4MB	520KB	240MHz
RP2040	2MB	264KB	133MHz
ATmega328	32KB	2KB	16MHz
STM32F4	512KB	128KB	168MHz

5.2 DSL Grammar (Excerpt)

```
program     ::= statement+
statement   ::= assignment | conditional | loop | action
conditional ::= 'when' condition ':' action+
loop        ::= 'every' duration ':' action+
action      ::= 'set' target 'to' value | 'wait' duration
```

Listing 5.1: DSL grammar (BNF)

5.3 Bytecode Instruction Set

Opcode	Description	Bytes
`PUSH/POP`	Stack operations	1–5
`LOAD/STORE`	Variable access	2
`READ_SENSOR`	Sensor input	2
`SET_PIN`	GPIO output	3
`JUMP/JUMP_IF`	Control flow	3
`WAIT/HALT`	Timing/termination	1–3

5.4 Safety Guarantees

1. **Syntactic validity:** Grammar constraints ensure valid programs
2. **Resource bounds:** Static analysis verifies memory/stack
3. **Timing bounds:** WCET analysis for real-time guarantees
4. **Pin safety:** Configurable write protection

Chapter 6

Theoretical Foundations

6.1 Complexity Analysis

Proposition (Decoding Complexity). Grammar-guided decoding adds $\mathcal{O}(|G| + |\mathcal{V}|)$ time per token.

For typical DSLs ($|G| \approx 100$, $|\mathcal{V}| \approx 32{,}000$): <1ms per token.

6.2 Capacity Lower Bounds

Theorem. A transformer generating programs for grammar $\mathcal{G}$ with n rules, depth ℓ, and c semantic classes requires:

$$|\theta| \geq \Omega\left(\frac{c \log(n\ell)}{\epsilon}\right)$$

parameters to achieve loss $< \epsilon$.

Implication: For $n = 50$, $\ell = 10$, $c = 1000$, $\epsilon = 0.1$: lower bound $\approx$ 600K params. Models with 1–50M are well above this.

6.3 Sample Complexity

Theorem. S-LoRA fine-tuning requires

$$N = \mathcal{O}\left(\frac{sm \cdot c}{\epsilon^2}\right)$$

training examples for ϵ-accuracy.

For typical values: $\approx$ 10K curated examples with augmentation.

Appendix A

Patent Claims Reference

#	Summary
1	System: GC-SLM + grammar decoder + compiler + interpreter
2	Method: NL → DSL → bytecode → MCU
3	Apparatus: MCU with interpreter, fixed stack
4	Decoding: Grammar-constrained token generation
5	Adaptation: S-LoRA with sparse matrices
6–7	Model size constraints
8	Target MCUs (ESP32, RP2040, etc.)
9–13	Memory, safety, verification
14–20	Decoding and training specifics
21–25	Timing, scheduling, safety
26–30	S-LoRA implementation details
31–35	On-device, FPGA, multi-grammar

Appendix B

Contact Information

Kernel Keys LLC

Primary Contact
David H. Silver, Founder

Email: david@kernelkeys.com

Web: www.kernelkeys.com / www.embino.com

For investment inquiries, partnership opportunities, or technical collaboration.