EPL Language Development Roadmap (Domain-Focused)

This roadmap outlines how to extend the EPL language from its **core foundation** into five major domains: **Robotics, Machine Learning & AI, Web Development, Automation, and Application Development.**

Phase 0: Core Foundation (Prerequisite for All Domains)

- **Language runtime**: memory model, type system (int, float, char, string, objects).
- **Execution modes**:
- Interpreter (fast prototyping).
- Compiler (EPL → LLVM → native machine code).
- JIT (for performance-critical loops).
- **Virtual Machine (EPL-VM)**:
- Platform-independent bytecode.
- Object model similar to JVM/CLR.
- **FFI (Foreign Function Interface)**:
- Call into C/C++ and link against system libraries.
- **Package manager (`eplpkg`)**:
- Dependency resolution.
- Publishing and sharing packages.

Phase 1: Robotics

Goals

- Real-time control and deterministic execution.
- Hardware interaction (GPIO, motors, sensors).

Features & Tasks

- EPL-RT (real-time safe runtime subset).
- Bindings to **ROS 2** (Robot Operating System).
- Direct GPIO, UART, I²C, SPI libraries.
- Scheduler API for task timing.
- Robotics stdlib package: `epl-robotics`.

Example

```
robot arm = Motor(pin: 21);
arm.move(90);
```

Phase 2: Machine Learning & AI

Goals

- High-performance numeric computing.
- GPU/accelerator integration.

Features & Tasks

- Tensor type in runtime.
- BLAS/LAPACK bindings via FFI.
- ONNX Runtime integration.
- CUDA/OpenCL backend for GPU.
- ML stdlib package: `epl-ml` (linear algebra, neural nets).
- JIT support for tensor kernels.

Example

```
Tensor X = Tensor([[1,2],[3,4]]);
Tensor Y = X.matmul(X);
print(Y);
```

Phase 3: Web Development

Goals

- Full-stack capabilities.
- WebAssembly support.

Features & Tasks

- Async runtime (event loop, coroutines).
- HTTP server/client library (`epl-web`).
- JSON, WebSocket, HTML parsers.
- EPL → WASM compiler backend.
- Web framework (like Flask/Express in EPL).

Example

```
server s = http.listen(8080);
s.route("/hello", (req,res) => {
    res.send("Hello, Web!");
});
```

Phase 4: Automation & Scripting

Goals

- EPL as a scripting/automation tool like Python.

Features & Tasks

- Process management (`os.exec`, `os.pipe`).
- File system stdlib (`fs`, `path`).
- Shell scripting API.
- Embeddable interpreter (EPL inside apps).

Example

```
files = fs.list("./logs");
for f in files {
    if (f.endsWith(".txt")) print(f);
}
```

Phase 5: Application Development

Goals

- Cross-platform desktop & mobile apps.

Features & Tasks

- GUI bindings (Qt, GTK, Webview).
- EPL → native mobile compiler (iOS/Android via LLVM).
- Packaging tools for apps (desktop installers, APK/IPA).
- Stdlib `epl-ui` for GUI widgets.

Example

```
app = ui.App();
btn = ui.Button("Click Me");
btn.onClick(() => print("Hello from EPL UI!"));
app.run(btn);
```

Long-Term Extensions

- **Security sandboxing** for untrusted EPL code.
- **Cloud runtime** (serverless EPL functions).
- **GPU-native VM** (run bytecode directly on GPUs).

Suggested Development Order

1. **Core foundation** (VM, runtime, FFI, package manager).
2. **Automation & scripting** (quick adoption).
3. **Web development** (async + WASM).
4. **ML/AI** (FFI to ONNX, CUDA, JIT optimizations).
5. **Robotics** (real-time subset, ROS bindings).
6. **Application dev** (GUI, mobile support).

Key Principle

EPL doesn’t need to **reinvent all libraries**. Instead: - Use FFI to wrap **existing ecosystems** (C/C++, Python libs). - Gradually replace bottlenecks with **native EPL implementations**. - Keep VM + bytecode stable to support all domains consistently.
