The Stack: Enriched Programming Dataset

Executive Summary

The Stack Enriched Dataset represents a significant advancement in programming language datasets, offering 1.3TB of highly curated, annotated, and enriched code spanning 888 categories. This dataset has been specifically enriched to address key gaps in existing code datasets, with particular focus on emerging languages, modern frameworks, security patterns, and code optimization techniques. Built upon the foundation of "The Stack," our dataset provides AI developers with a balanced, high-quality corpus that can meaningfully improve code generation capabilities across multiple dimensions.

Dataset Statistics and Composition

Size and Scale

Total Size: 1.3 Terabytes

Total Files: ~2.1 million

Categories: 888 primary directories

Languages: 45+ programming languages

Language Distribution

Category	File Count	Percentage
Python	310,000+	14.8%
JavaScript	295,000+	14.0%
Java	250,000+	11.9%
C/C++	225,000+	10.7%
TypeScript	160,000+	7.6%
PHP	145,000+	6.9%
C#	140,000+	6.7%
Ruby	105,000+	5.0%
Go	100,000+	4.8%

Rust	85,000+	4.0%
Other	285,000+	13.6%

Special Collections

Enriched Language Collections:
Rust: 20,000 extensively annotated files
Go: 20,000 extensively annotated files
TypeScript: 20,000 extensively annotated files

Framework-Specific Collections:
React: 10,000 files showing modern web development patterns
TensorFlow: 3,130 files covering ML/AI implementation
PyTorch: 218 files demonstrating deep learning applications

Heavily Annotated Code:
Python: 10,000 files with 15%+ comment density
JavaScript: 10,000 files with 15%+ comment density
Java: 10,000 files with 15%+ comment density

Special Pattern Collections:
Refactoring Patterns: 25 complete examples across languages
Security Vulnerabilities & Fixes: 150+ paired examples
Performance Optimization: 200+ examples with benchmarks

Enrichment Methodology

Our enrichment process followed a systematic approach to ensure maximum value for AI training:
Gap Analysis: We analyzed existing code datasets to identify underrepresented languages, frameworks, and coding patterns.
Collection & Curation: Additional code samples were gathered from public repositories under permissible licenses, with strict quality filters (functional completeness, minimal dependency issues, consistent style within files, representative of contemporary practices).
Annotation Enhancement: We increased comment density where appropriate, ensuring explanations of function purpose, parameter meanings, return value specifications, edge case handling.
Quality Assurance: Each enriched file underwent syntax validation, static analysis, duplicate detection, comment quality assessment.
Categorization: Files were methodically organized into a coherent taxonomy that facilitates targeted training.

Sample Content Showcase

Example 1: Rust Error Handling (Enriched)

```rust
/// Attempts to read user data from the provided file path
///
/// # Arguments
/// * `path` - A string slice containing the file path
///
/// # Returns
/// * `Result<UserData, FileError>` - UserData if successful, appropriate error otherwise
///
/// # Error Cases
/// * Returns FileError::NotFound if the file doesn't exist
/// * Returns FileError::Permission if access is denied
/// * Returns FileError::Format if the content is invalid
pub fn read_user_data(path: &str) -> Result<UserData, FileError> {
    if !Path::new(path).exists() {
        return Err(FileError::NotFound(path.to_string()));
    }
    let file = match File::open(path) {
        Ok(file) => file,
        Err(e) if e.kind() == ErrorKind::PermissionDenied => {
            return Err(FileError::Permission(path.to_string()));
        }
        Err(e) => {
            eprintln!("Unexpected error opening {}: {}", path, e);
            return Err(FileError::Unknown(e.to_string()));
        }
    };
    let reader = BufReader::new(file);
    match serde_json::from_reader(reader) {
        Ok(data) => Ok(data),
        Err(_) => Err(FileError::Format(path.to_string())),
    }
}
```

Example 2: Python Optimization Pattern (Before/After)

Before Optimization:

```python
def find_common_elements(list1, list2):
    """
    Find elements that appear in both lists.
    """
    common = []
```

```
    for item in list1:
        if item in list2:
            common.append(item)
    return common
```

After Optimization:

```
def find_common_elements(list1, list2):
    """
    Find elements that appear in both lists.
    Args:
        list1: First input list
        list2: Second input list
    Returns:
        List of elements found in both input lists
    Performance Note:
        Uses set intersection for O(n+m) time complexity instead of the O(n*m) of nested
iteration.
    """
    set2 = set(list2)
    return list(set(list1).intersection(set2))
```

Example 3: React Component with Security Considerations

```
/**
 * UserInput component for handling user-provided content
 * with built-in XSS protection and input validation
 * @param {Object} props - Component props
 * @param {string} props.initialValue - Initial input value
 * @param {function} props.onChange - Change handler callback
 * @param {string} props.placeholder - Input placeholder text
 * @param {boolean} props.required - Whether input is required
 */
const UserInput = ({ initialValue = '', onChange, placeholder = 'Enter text', required = false })
=> {
  const [value, setValue] = useState(initialValue);
  const [error, setError] = useState('');
  const sanitizeInput = (input) => {
    return input.replace(/<script\b[^<]*(?:(?!<\/script>)<[^<]*)*<\/script>/gi,
'').replace(/on\w+="[^"]*"/gi, '');
  };
  const handleChange = (e) => {
    const rawInput = e.target.value;
    const sanitized = sanitizeInput(rawInput);
    if (required && !sanitized.trim()) {
```

```
    setError('This field is required');
   } else if (sanitized.length > 100) {
    setError('Input exceeds maximum length of 100 characters');
   } else {
    setError('');
   }
   setValue(sanitized);
   if (onChange) onChange(sanitized);
  };
  return (
   <div className="input-container">
    <input type="text" value={value} onChange={handleChange} placeholder={placeholder}
classNameerror ? 'input-error' : ''} aria-invalid={!!error} />
    {error && <div className="error-message" role="alert">{error}</div>}
   </div>
  );
};
```

Licensing Options

Our dataset is available for licensing under flexible terms tailored to your organization's needs:

Standard License: Full dataset access, One-time fee, Use for internal model training, 1-year support and updates.

Enterprise License: Full dataset access, Custom enrichment services, Integration support, Multi-year updates, Priority support.

Research License: Available for academic research, Limited commercial use, Reduced fee.

Contact Information

Vincenzo Gallo
vincenzo.gallo77@hotmail.com
+39 3477539060