Granite-3.3:8B — Code Completion Evaluation

Comparison of Hole Filler Template vs. Fill In the Middle Template

📒 Appendix: Prompt Design (for Reference)

- **Hole Filler Template:**
  Prefix only; cursor at end of code block.
  Prompts used: https://github.com/IBM-OSS-Support/Continue.dev-Granite-manual-test-cases/blob/main/tab-autocomplete/granite3.3-8b/usecases/prefix-only/prefix-only-usecases.py
- **FIM Template:**
  Prefix (before hole), Suffix (after hole), cursor at the gap.
  Prompts used: https://github.com/IBM-OSS-Support/Continue.dev-Granite-manual-test-cases/blob/main/tab-autocomplete/granite3.3-8b/usecases/fim/fim-usecases.py

✅ Executive Summary

- **Hole Filler Template (HFT):**
  - *Strengths:* Reliable for sequential code completions, strong with Python syntax and functional code, robust for simple to moderately complex code.
  - *Weaknesses:* Prone to hallucinations in structured/multi-branch logic, verbose in certain scenarios.
- **Fill In the Middle (FIM):**
  - *Strengths:* Excels at filling isolated code holes (e.g., lambdas, recursion, exception logic), ideal for small focused completions, integrates well in refactoring workflows.
  - *Weaknesses:* Struggles with complex, multi-branch logic and structural completions (e.g., tax slabs, pandas aggregations, OOP methods); sometimes produces incomplete or vague output.

📊 Evaluation Summary Table

Test Case	Hole Filler Verdict	FIM Verdict
1. Nested Conditions (Tax Slabs)	⚠️ Partial: Syntax ok, logic error in slab calc	❌ Broken: Dead code, misses slab, syntax err
2. Lambda + Filter	✅ Excellent: Clean, correct, idiomatic	✅ Excellent: Clean, correct, idiomatic

3. Pandas Chaining	✅ Good: Uses NamedAgg, minor format issue	❌ Vague: "all columns" non-Python output
4. Exception Handling with Custom Message	✅ Solid: Logical, correct, verbose	✅ Excellent: Clean, concise, correct
5. Class with Dunder/Bonus	✅ Good: Dunder and bonus logic are correct	❌ Broken: Incomplete, missing return, logic
6. Recursive Function (Factorial)	✅ Excellent: Canonical recursion, clean	✅ Excellent: Canonical recursion, clean

📝 Detailed Comparison by Use Case

1️⃣ Nested Conditions (Tax Slabs)

- **Hole Filler:**
  Partial logic, hardcodes slab values, does not fully respect business logic, but structure is valid Python.
- **FIM:**
  Misses a whole branch, inserts dead code, and outputs syntactically invalid Python.

2️⃣ Lambda + Filter

- **Hole Filler:**
  Generates a correct, idiomatic lambda filter for even numbers.
- **FIM:**
  Also generates correct lambda. Both modes perform equally well.

3️⃣ Pandas Chaining

- **Hole Filler:**
  Uses advanced API (NamedAgg), correct overall structure, only minor formatting issues.
- **FIM:**
  Produces non-Python output ("all columns"), does not provide a real aggregation statement.

4️⃣ Exception Handling with Custom Messages

- **Hole Filler:**
  Robust, verbose, but logic is correct.

- **FIM:**
  Clean, concise, and correct — in fact, more succinct than HFT.

5️⃣ Class Definitions with Dunder/Bonus

- **Hole Filler:**
  - Generates correct __init__ and __str__.
  - Implements calculate_bonus logic as intended (salary check, bonus calculation, returns correct value).
  - Minor verbosity at end, but not incorrect.
  - **Result: Solid performance** for both OOP syntax and business logic.
- **FIM:**
  - Output is incomplete, missing both condition and return.
  - **Result: Fails to provide usable code for this scenario.**

6️⃣ Recursive Logic

- **Hole Filler:**
  Outputs canonical recursive implementation; perfect.
- **FIM:**
  Same as HFT; perfect recursion.

🟢 Strengths & 🔴 Weaknesses

Hole Filler Template (HFT)

Strengths:
- Good at classic code completion and generating boilerplate or sequential code.
- Handles simple functions, recursion, and basic functional programming patterns well.
- Can generate complex API calls if context is linear.
- Excels at both classic and OOP code completions, including dunder methods and conditional logic inside methods.

Weaknesses:
- Hallucinates or fumbles structured/branching logic.
- Tends to generate verbose code (especially with print statements).

Fill In the Middle (FIM) Template

Strengths:
- Excels at targeted, in-place code hole filling (e.g., filling in a lambda, a recursive call, or a single exception raise).
- More concise for isolated code blocks.

- Integrates well in a refactoring workflow or patch-based code editing.

Weaknesses:

- Unreliable for multi-branch, multi-step, or structural completions (e.g., tax slab logic, pandas aggregations, full method bodies).
- Sometimes produces vague or invalid output when the required logic is complex.
- Suffers on OOP patterns and logic-heavy completions.

⚖️ Overall Verdict

- **For most real-world developer workflows:**
  - Use **Hole Filler Template** for general code completion, larger blocks, or full function/method generation.
  - Use **FIM Template** for targeted patching, simple insertions, or automated refactoring tasks that require context before and after the hole.
- **Model Quality:**
  - Granite-3.3:8B is strong on simple/medium code completion, recursion, lambdas, and straightforward exception logic in both modes.
  - It needs improvement in class structure, complex branching, and context-heavy or "abstract" tasks (like custom aggregation).

📌 Recommendations

- **Use Case Fit:**
  - **Hole Filler:** For new function writing, classic autocomplete, or when the task is to finish a code block given the start.
  - **FIM:** For editing/filling gaps in the middle of existing code, quick patching, or tight refactoring loops.
