Here's a brief explanation of Cursor AI limitations with practical examples:1. \*\*Code Understanding Limitations\*\*  
```python  
# Cursor AI might struggle with complex business logic  
class OrderProcessor:  
    def calculate\_discount(self, order):  
        # AI might not understand company-specific discount rules  
        if order.is\_premium\_customer and order.total > 1000:  
            return self.apply\_special\_holiday\_discount()  # Custom business logic  
```  
\*Mitigation\*: Add detailed comments explaining business rules2. \*\*Language Support\*\*  
```javascript  
// AI might suggest outdated syntax  
// Old way  
var x = function() { ... }// Modern way  
const x = () => { ... }  
```  
\*Mitigation\*: Always verify with current language documentation3. \*\*Real-time Updates\*\*  
```python  
# AI might not know latest framework features  
# Old Django syntax  
path('profile/', views.profile, name='profile')# New Django syntax with type hints  
path('profile/', views.profile, name='profile') -> HttpResponse  
```  
\*Mitigation\*: Check framework's latest documentation4. \*\*Security Considerations\*\*  
```python  
# AI might suggest vulnerable code  
# Unsafe  
user\_input = request.GET['id']  
query = f"SELECT \* FROM users WHERE id = {user\_input}"# Safe  
from django.db.models import User  
user = User.objects.get(id=user\_input)  
```  
\*Mitigation\*: Always review for security best practices5. \*\*Code Quality\*\*  
```python  
# AI might generate verbose code  
# Less optimal  
def get\_active\_users(users):  
    active\_users = []  
    for user in users:  
        if user.is\_active:  
            active\_users.append(user)  
    return active\_users# Better  
def get\_active\_users(users):  
    return [user for user in users if user.is\_active]  
```  
\*Mitigation\*: Refactor for better readability and efficiency6. \*\*Context Window\*\*  
```python  
# AI might miss context from other files  
# In models.py  
class User(models.Model):  
    role = models.CharField(max\_length=100)# In views.py - AI might not know about User model's fields  
def get\_admins():  
    return User.objects.filter(role='admin')  # AI needs context about User model  
```  
\*Mitigation\*: Provide necessary context in queries7. \*\*Dependency Management\*\*  
```python  
# AI might suggest incompatible versions  
# requirements.txt  
django==2.2  # AI might not know this is EOL  
requests==2.26.0# Should be  
django>=4.2  # Latest LTS version  
requests>=2.31.0  
```  
\*Mitigation\*: Verify compatibility with project requirements8. \*\*Performance Optimization\*\*  
```python  
# AI might generate inefficient queries  
# Inefficient  
users = User.objects.all()  
for user in users:  
    print(user.profile.name)  # N+1 query problem# Optimized  
users = User.objects.select\_related('profile').all()  
for user in users:  
    print(user.profile.name)  
```  
\*Mitigation\*: Profile and optimize critical code paths9. \*\*Error Handling\*\*  
```python  
# AI might miss edge cases  
# Incomplete  
def divide(a, b):  
    return a / b# Better  
def divide(a, b):  
    if b == 0:  
        raise ValueError("Cannot divide by zero")  
    if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):  
        raise TypeError("Arguments must be numbers")  
    return a / b  
```  
\*Mitigation\*: Add comprehensive error handling10. \*\*Documentation\*\*  
```python  
# AI might generate generic docs  
# Generic  
def process\_order(order):  
    """Process the order."""# Better  
def process\_order(order):  
    """  
    Process customer order and apply business rules.    Args:  
        order (Order): Order object containing items and customer info    Raises:  
        InvalidOrderError: If order validation fails  
        PaymentError: If payment processing fails  
    """  
```  
\*Mitigation\*: Enhance with specific details and examples11. \*\*Integration Limitations\*\*  
```python  
# AI might not understand system architecture  
# Service integration without proper error handling  
def send\_notification(user, message):  
    requests.post('<http://notification-service/send>',  
                 json={'user': user, 'message': message})# Better integration  
def send\_notification(user, message):  
    try:  
        response = requests.post(  
            '<http://notification-service/send>',  
            json={'user': user, 'message': message},  
            timeout=5  
        )  
        response.raise\_for\_status()  
    except requests.exceptions.RequestException as e:  
        logger.error(f"Notification failed: {e}")  
        raise NotificationError(f"Failed to send notification: {e}")  
```  
\*Mitigation\*: Add proper error handling and logging\*\*Key Takeaways:\*\*  
- Always review AI-generated code  
- Test thoroughly, especially edge cases  
- Add proper error handling  
- Consider security implications  
- Optimize for performance when needed  
- Document clearly and specifically