

```
class NPTTokenizer:
    def __init__(self, vocab_size=10000, min_freq=5, seq_length=8):
        self.vocab_size = vocab_size
        self.min_freq = min_freq
        self.seq_length = seq_length
        self.tokenizer = AutoTokenizer.from_pretrained("gpt2")

        self.special_tokens = {
            "PAD": "<PAD>",
            "UNK": "<UNK>",
            "BOS": "<BOS>",
            "EOS": "<EOS>"
        }

        self.token2idx = {}
        self.idx2token = {}
```

```
def prepare_sequence_data(self, texts):  
  
    inputs = []  
    targets = []  
  
    for text in tqdm(texts, desc="Preparing sequences"):  
        # Encode full text  
        token_ids = self.encode(text)  
  
        # Create sequences of length seq_length with next token as target  
        for i in range(len(token_ids) - self.seq_length):  
            seq = token_ids[i:i + self.seq_length]  
            next_token = token_ids[i + self.seq_length]  
  
            inputs.append(seq)  
            targets.append(next_token)  
  
    return inputs, targets
```

```
class BaselineNTPModel(nn.Module):

    def __init__(self, vocab_size, embedding_dim=128, hidden_dims=[256, 128]):

        super(BaselineNTPModel, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim)

        layers = []

        input_dim = embedding_dim # If using mean/max pooling
        # input_dim = embedding_dim * seq_length # If using concatenation

        # Add hidden layers
        prev_dim = input_dim
        for hidden_dim in hidden_dims:
            layers.append(nn.Linear(prev_dim, hidden_dim))
            layers.append(nn.ReLU())
            prev_dim = hidden_dim

        # Output layer
        layers.append(nn.Linear(prev_dim, vocab_size))

        self.feedforward = nn.Sequential(*layers)
```

```
def forward(self, x):  
  
    embeddings = self.embedding(x) # [batch_size, seq_length, embedding_dim]  
  
    # Combine embeddings by averaging, similar to Continuous Bag of Words (CBOW)  
    combined = torch.mean(embeddings, dim=1)  
  
    logits = self.feedforward(combined) # [batch_size, vocab_size]  
  
    return logits
```

```
def train_epoch(model, dataloader, optimizer, criterion, device):  
    """Train the model for one epoch."""  
    model.train()  
    total_loss = 0  
    correct = 0  
    total = 0  
  
    for batch in dataloader:  
        inputs = batch['input'].to(device)  
        targets = batch['target'].to(device)  
  
        # Forward pass  
        logits = model(inputs)  
        loss = criterion(logits, targets)  
  
        # Backward pass and optimization  
        optimizer.zero_grad()  
        loss.backward()  
        optimizer.step()  
  
        total_loss += loss.item()  
  
        # Calculate accuracy  
        _, predicted = torch.max(logits, 1)  
        correct += (predicted == targets).sum().item()  
        total += targets.size(0)  
  
    return total_loss / len(dataloader), correct / total
```

```

class TransformerNTPModel(nn.Module):

    def __init__(self, vocab_size, embedding_dim=128, nhead=4, num_layers=2,
                  dim_feedforward=512, dropout=0.1, max_seq_length=5000):

        super(TransformerNTPModel, self).__init__()
        # Token embedding
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        # Positional encoding
        self.positional_encoding = PositionalEncoding(
            embedding_dim, max_seq_length, dropout
        )
        # Transformer encoder layer
        encoder_layer = nn.TransformerEncoderLayer(
            d_model=embedding_dim,
            nhead=nhead,
            dim_feedforward=dim_feedforward,
            dropout=dropout,
            batch_first=True
        )
        # Transformer encoder
        self.transformer_encoder = nn.TransformerEncoder(
            encoder_layer,
            num_layers=num_layers
        )
        # Output layer
        self.output_layer = nn.Linear(embedding_dim, vocab_size)

```

```

class PositionalEncoding(nn.Module):

    def __init__(self, d_model, max_seq_length=5000, dropout=0.1):
        """
        Args:
            d_model: Embedding dimension
            max_seq_length: Maximum sequence length
            dropout: Dropout rate
        """
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        # Create positional embedding matrix
        pe = torch.zeros(max_seq_length, d_model)
        position = torch.arange(0, max_seq_length, dtype=torch.float).unsqueeze(1)

        # Here generate frequencies from 1 to 1/10000 decreasing exponentially
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model))

        # Apply sine to even idx, cosine to odd idx
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)

        # Add batch dimension
        pe = pe.unsqueeze(0)

        # Register buffer — for parameters that are stored but not updated like positional encodings
        self.register_buffer('pe', pe)

```

# Analysis of Results

---

## Model Architectures

---

### Baseline Model

- Simple feed-forward architecture
- Parameters: 1,280,000 ( $10,000 * 128 + 128 * 256 + 256 * 128 + 128 * 10,000$ )
- Embedding dimension: 128
- Hidden layers: [256, 128]
- Loss function: Cross-entropy
- Optimizer: Adam (lr=0.0001)

### Transformer Model

- Transformer encoder architecture
- Parameters: 2,560,000 ( $10,000 * 128 + 128 * 128 * 4 + 128 * 512 + 512 * 128 + 128 * 10,000$ )
- Embedding dimension: 128
- Attention heads: 4
- Transformer layers: 2
- Feed-forward dimension: 512
- Dropout: 0.1



## Results

Baseline Model Results:

- Test Perplexity: 360.5
- Test Accuracy: 15.59%
- Final Training Loss: 5.675
- Final Validation Loss: 5.974
- Parameters: 1.28M

Transformer Model Results:

- Test Perplexity: 120.5
- Test Accuracy: 24.16%
- Final Training Loss: 4.676
- Final Validation Loss: 4.801
- Parameters: 2.56M

## Training Dynamics

Both models were trained with:

- Batch size: 64
- Sequence length: 8
- Vocabulary size: 10,000
- WikiText-2 dataset



How should I design a commercial hazelnut planting in Ohio?

## 1. Choose the Right Cultivars

Ohio's humid continental climate (zones 5–6) can be challenging due to Eastern Filbert Blight (EFB). Look for:

- **Blight-resistant hybrids:** Often crosses of *Corylus americana* × *Corylus avellana*.
  - Recommended cultivars: 'Jefferson', 'Gamma', 'Delta', 'Yamhill', or newer Rutgers hybrids like 'Gratitude' or 'Celebration'.
- **Pollinizers:** Plant 10–15% as compatible pollinizers that shed pollen at different times (early, mid, late).

I live in Ohio and want to grow European hazelnuts. What varieties should I plant?

## Recommended Varieties for Ohio (Zone 5/6)

### EFB-Resistant Hybrids (Best Bet)

- **'Jefferson'** – High-quality European-style nuts; needs a pollinizer and is **moderately resistant** to EFB.
- **'Geneva'** – Bred in New York; more cold-hardy and EFB-tolerant.
- **'Slate' and 'Gasaway'** – Older but still useful resistant pollinizers.
- **'Beast' and 'Grand Traverse'** – Hybrid hazelnuts with good nut size and strong resistance (Badgersett selections).

### Pollinizers

Pollinators are essential. You'll want at least two genetically distinct varieties:

- **'Yamhill', 'Eta', and 'Theta'** – Good early-season pollinators.
- Plant at least **2–3 varieties with overlapping bloom** times for proper wind pollination.



## Recommended Varieties for Ohio (Zone 5/6)

### EFB-Resistant in the Eastern United State


- **'Raritan'** – High-quality European-style nuts with very high level of tolerance to EFB. From Rutgers University breeding program. An early season bloomer. S-alleles S3 and S22.
- **'Monmouth'** – Another Rutgers selection, with different S-alleles (S1 & S12) than Raritan and also an early bloomer, making it a good pollinizer partner for Raritan.
- **'Somerset'** – Also from Rutgers. A smaller, early-producing tree that produces abundant thin-shelled nuts. Expresses S-allele S3 in pollen and so will not cross-pollinate with Raritan.
- **'Beast'** – A release from the Hybrid Hazelnut Consortium. An extremely vigorous, high-yielding tree with abundant pollen and S-alleles S8 and S23, making it compatible with all the Rutgers releases. Abundant producer of small, yet commercially viable nuts.
- **'Grand Traverse'** – An older selection (1989) from Michigan by Cecil Farris. It's EFB resistance comes from the Turkish Tree hazel. It produces large nuts (1.3 grams) that blanch well and alleles S11 and S25, making it a good pollinizer pairing for all the other varieties on this list.

<https://nationalnutgrower.com/article/rutgers-eastern-hazelnut-research/>

<https://njaes.rutgers.edu/e368/>



1. **Focus on EFB Resistance:** This is the most critical factor for long-term commercial viability.

- **Rutgers University Releases:** Rutgers has released several new European hazelnut cultivars with resistance to EFB. These include:
  - 'Raritan'
  - 'Monmouth'
  - 'Hunterdon'
  - 'Somerset'
- These are recommended for USDA hardiness zones 6 and 7 and have not been extensively trialed in the Upper Midwest. However, they are worth considering, especially in the warmer parts of Ohio. Be aware that their winter hardiness in Ohio needs to be carefully evaluated. 





Is it better to plant grafted or seedling chinese chestnut varieties?  
Give a brief and thoughtful recommendation for someone starting a chestnut orchard

For someone starting a Chinese chestnut orchard, **grafted varieties are generally the better choice**. Grafted trees offer consistency: they come from proven cultivars with known traits such as large nut size, early ripening, good flavor, and disease resistance (especially to chestnut blight). This predictability is crucial for commercial or semi-commercial operations aiming for reliable yields and marketable quality. Seedling chestnuts, while more genetically diverse and potentially more adaptable to site-specific conditions, are highly variable in their nut quality, productivity, and tree form. They can be a good option for breeding, ecological restoration, or experimental plantings but are riskier for uniform nut production.



# Should We Plant Grafted or Seedling Trees for Chestnut Production Orchards?

Gregory Miller

Graft failure in chestnuts has been a longstanding problem, and has hindered development of the chestnut industry. Especially frustrating is delayed graft failure wherein grafts initially grow, but subsequently die weeks, months, or years later. And, even in the absence of scion death, grafted trees often exhibit reduced growth, production, and nut size compared to the ortet from which the scion came. This might be called “graft stress”. The causes of graft failure or graft stress are not well understood. The most often assumed cause is incompatibility, *i.e.*, genetic differences between stock and scion.

While incompatibility remains a likely contributing factor, no one has yet identified reliably compatible scion/rootstock pairings. Even when scions are grafted onto their offspring, graft failure occurs at an unacceptably high rate. It has been consistently observed that graft failure is exacerbated by stress conditions, including water stress, freeze injury, and transplant shock. Graft failure and graft stress, however, still occur in

grafted trees remains. It is noteworthy that there exists very little experimental data comparing seedlings and grafted trees from a production or nut quality standpoint. Thus, in this article we present such data.

## PLANTING ESTABLISHMENT

In 2014, we planted alternating rows of seedling Chinese chestnuts and grafted Chinese chestnut cultivars on a good chestnut site in east central Ohio (near Carrollton). The seedlings were bare-root trees, 1 year old and about 30 to 40 inches (75-100 cm) tall. The grafted trees were chip budded onto container-grown rootstocks in the spring of 2012, and grown in 2-gal pots in 2013 to about 30 to 40 inches (75-100 cm) tall; they were more branched than the seedlings. The spacing of the trees was 26 feet (8 m) within row and 40 feet (12 m) between rows (104 trees/ha or 43 trees/A). For the data presented here, we are considering two rows of PQ seedling full-sib progeny of the cultivars ‘Peach’ and ‘Qing’ planted

seedling trees grew in size between the seedling trees and grafted trees has become in over time (fig. 1).

Survival of seedlings has been better than survival of grafted trees (28 out of 38). The trees were each of the experimental treatments and evaluated in 2014. The main purpose of the experiment was to assess the genetic differences between individuals in the seedling and grafted trees and compare them to the grafted trees. Harvests from the grafted trees were tallied and evaluated. Harvests from individual cultivars were just tallied separately by tree.

Data collected in 2014 included total weight of nuts, number of nuts, and number of culls (splits, anthracnose, other defects). Samples from each tree were peeled and tasted. Considering all data collected, seedling trees were each

## CONCLUSIONS

For this planting, which is at the beginning of its commercial production, trees from seedlings of named cultivars, on average, have substantially outperformed their grafted counterparts in terms of yields of commercially acceptable chestnuts and in terms of nut size. The seedling trees have grown faster and survived better than their grafted counterparts.

<https://nutgrowing.org/wp-content/uploads/2024/04/78-1.pdf>

